From the MariaDB Knowledge Base

2023-11

MariaDB Server

Documentation

MariaDB Foundation

Ian Gilfillan (Editor)

Dedicated to bazaar builders everywhere.

Licence

If you find any errors, please see Reporting Documentation Bugs

Generated from the MariaDB Knowledge Base on 2023-11-28

Preface

If you're contemplating whether to devote some time to this book, read this:

- MariaDB Server is a general-purpose, open source, relational database management system, optimised for performance and easy usabiility; it has its roots in MySQL Server, and is an alternative to Postgres, Oracle Database and other relational and NoSQL databases
- This book is the full documentation on MariaDB Server, a "Reference Manual Plus" which includes aspects of a User's Guide; it is based on the contents of the MariaDB Knowledge base (https://mariadb.com/kb/), an open, community-edited site contributed to since the inception of MariaDB in 2009
- This edition is not specific to any version of MariaDB Server, but includes functionality up to the latest version of MariaDB at the time of generation

This preface describes the goals, structure and contents of the documentation. Reading it is intended as a helpful step in understanding how to best use the manual to improve productivity in using MariaDB Server.

This Book's "Prehistory"

As noted, MariaDB Server has its roots in MySQL Server. It started as a fork of MySQL Server, using the same GPLv2 license. However, although the MySQL Server documentation was always publicly available, it was never released using a free documentation license. This means that the documentation of MariaDB Server was created from scratch. Or rather, from the online help texts, which had a compatible open licence that made them usable as a starting point.

The place to which documentation was written was labelled the "Knowledge Base", by MySQL and MariaDB creator Michael "Monty" Widenius. The Knowledge Base was – and remains – a community effort. As with many community efforts, there are core contributors around whom the work is centered. This is where Daniel Bartholomew loaded the online help text, as a first seed. For roughly the last ten years, the core editor of the MariaDB Knowledge Base has been Ian Gilfillan, working for MariaDB Foundation and based in South Africa. Hence, his name is on the cover of the book. However, there are a large number of other contributors, many of whom come from MariaDB Corporation – both as developers of code and as documentation writers. They are listed on https://mariadb.com/kb/stats/users/.

With now some 3000 pages in this book, most of the initial holes in the documentation have been filled. There should now be no reason to do as in the very early days of MariaDB Server – namely look up MariaDB features in the MySQL documentation. On the contrary, the functionality of the two databases have diverged considerably, so you would be ill advised not to use this MariaDB Server specific documentation.

The First Edition

The first edition of the MariaDB Server Documentation as a PDF file was released in April 2022. Prior to this, the contents were accessible as individual Knowledge Base (KB) articles. But already in 2014 – over seven years before – the user base requested a PDF version, as seen by MariaDB's Jira entry https://jira.mariadb.org/browse/MDEV-6881 MariaDB Documentation improvements. There, user Stoykov pointed out that MariaDB documentation already had search capabilities and a way to mirror the KB in an offline version – but lacked downloadable PDF and EPUB versions,

Fast forward some seven years and a number of upvotes and watchers, we decided to devote resources to it. Creating a PDF from an HTML file is something Python is good at, and Dorje Gilfillan did all the tweaking necessary to merge the individual KB pages into one huge HTML file for PDF conversion.

This Book's Structure

We had to impose a chapter structure on the book which is only indirectly visible from a collection of KB articles on the web. This means that the work in compiling the PDF isn't just about merging many KB pages in an order that could be derived from the hierarchical pointers between the articles. It also involves cleaning up that structure.

As a result, you will see two tables of contents. One is a one-pager overview with just the two top levels of hierarchy. The other is over 30 pages long. True to the Open Source mantra of "release early, release often", we believe that the structure can still be improved upon – but it is a good starting point. We have seven overall chapters, and the structures below them all make sense at some level.

To get the most out of the book, we recommend you to spend time making yourself familiar with the table of contents. It will give you an idea of existing functionality. Just browsing it through may give you ideas of commands you didn't know existed.

This Book's Format

There is currently just one version of the book. It's delivered in the PDF format, and in the Golden Ratio aspect ratio – meaning, A4. As we envision it to be read mostly on-screen anyway, we wanted to avoid the additional complexity of also providing a US Letter format. If we meet demand for further versions, doing US Letter is of course an option; however, given there are many ways to improve the documentation, we would also like to understand how adding another aspect ratio of the PDF would benefit the users in practice.

We don't yet provide the ePub format. Again, if you desire ePub, please educate us as to what added benefits you expect of ePub on top of PDF.

Use Cases For This Book

We expect the main use case for the PDF version of the book to be offline access. Offline may be imposed by a flaky or non-existent internet, but also by self-imposed abstinence from the many distractions of being online.

We expect that browsing the PDF will enable concentrated time to be spent on learning about MariaDB Server. The search functionality of PDF browsers helps in finding out about commands and syntax you already know of; browsing through a PDF – in particular the clickable Table of Contents – will hopefully provide you with an educational overview better than the online KB does.

We expect downloading the manual into laptops, tablets and phones will make sense. If you have the MariaDB Server Documentation on your phone, you can turn waiting time into something productive, perhaps even fun.

What we should work on

We have lots of room for improvement. That said, our foremost goal now is to get the book out, to get it used. User feedback will help us determine the right priority for our already existing ideas for improvements. We will likely get other requests beyond what we currently have in mind.

In the area of basic usability, an index has been spoken about. Looking up commands through searches or through browsing the table of contents is ok, but an index also has use cases. Our plan here starts from automatic indexing based on keywords of the headers of individual articles.

In the area of layout, we are looking at finding icons that make the PDF look more like a book, and less like a web page. We already solved the first issue, which was to find a clearer visual distinction between links within the PDF and links to the web.

In the area of structure, the length of individual chapters varies a lot. It may make sense to move around chapters in the TOC tree, to be more balanced. It may be that the reader expects another ordering based on experiences from other databases. It may even be that we lack entire topics. For instance, we eliminated the Release Notes for unsupported versions of MariaDB, even though these are still accessible on the KB.

In the area of accessibility, there may be places we should publish the PDF to make it easier to find, download, and use.

The common denominator for all of the above is that we need your feedback on what makes sense for you as a user of MariaDB Server.

Give Us Feedback

We would like to pick the brains of individual users. At conferences, asking open-ended questions is easy and feels productive for both parties, when meeting in the corridors between talks. Replicating the same productive discussion on-line is much harder. It takes effort from both parties. It feels like work.

We are still looking for the best way for you to give us meaningful feedback. Feel free to approach us over Zulip (https://mariadb.zulipchat.com/ – the Documentation topic). Also email to foundation@mariadb.org will find its way to us.

When you find individual bugs, please enter them into Jira using the guidelines mentioned in the KB article https://mariadb.com/kb/en/reporting-documentation-bugs/.

Acknowledgements

Compiling any book requires more effort than expected by the authors, and more than visible to the readers. This book is no exception. It has been over ten years in the making.

The primary thanks go to Ian Gilfillan, as the overall editor of the book and as the individually most productive author.

Close to Ian, we have Daniel Bartholomew. Daniel even beats Ian when it comes to articles created, and comes second on articles edited.

Among the community contributors, we want to highlight Federico Razzoli. He has two accounts, totalling 4488, at the time of writing – making him rank third amongst personal contributors.

When it comes to organisational contributors, the largest one is MariaDB Corporation. With them coding most of the features, they also stand for the lion's share of their documentation. As writers, besides Daniel Bartholomew whom we already mentioned several times, we want to highlight Russell Dyer, Kenneth Dyer, Geoff Montee, and Jacob Moorman.

As the developer of the KB software itself, Bryan Alsdorf deserves special acknowledgement.

A special thanks goes to Michael "Monty" Widenius, the creator of MariaDB. Monty has always understood the importance of documentation. He is leading by example, with a large number of personal edits. In fact, Monty has the second highest number of edits amongst developers, after Sergei Golubchik and followed by Sergey Petrunia – all of which have over a thousand edits.

Amongst the prolific contributors within the MariaDB Corporation Engineering team, the Connectors team stands out, with Diego Dupin, Georg Richter, and Lawrin Novitzky ranking near the top. However, we have decided not to include Connectors documentation in this first edition; we are contemplating whether it should be a separate PDF manual.

Other past and present Engineering team members, in decreasing order of number of edits, are David Hill, Dipti Joshi, David

Thompson, Massimiliano Pinto, Kolbe Kegel, Vladislav Vaintroub, Ralf Gebhardt, Markus Mäkelä, Sunanda Menon, the late Rasmus Johansson, Todd Stoffel, Elena Stepanova, Julien Fritsch, and Alexander Barkov. They all have more than one hundred edits, which is a lot.

As a true Open Source project, MariaDB Server documentation attracts attention and plentiful contributions also from outside the MariaDB Corporation Documentation and Engineering teams. We want to highlight those with over a hundred edits: Colin Charles and Stephane Varoqui, both of MariaDB Corporation, and Daniel Black, of MariaDB Foundation.

Amongst community contributors in the over-a-hundred-edits category, we want to mention especially Alena Subotina, with edits related to the dbforge documentation tool, and Juan Telleria, with edits often related to R Statistical Programming. Prolific contributors whose contributions are not visible in this English manual are Esper Ecyan (Japanese) and Hector Stredel (French); Federico Razzoli (Italian) has many edits also in English.

We also want to extend a thank you to the code developers who make work easy for the documentation team through thoroughly prepared, reusable texts in Jira; in this category, Marko Mäkelä and Oleksandr Byelkin come to mind.

As for the PDF manual, it has been teamwork between Ian and his son Dorje Gilfillan. Ian has done what editors do, Dorje has coded the Python code that compiles the KB pages into one.

All in all, thank you to everyone who has contributed to this book! We hope compiling it into one volume is of use for you, and we would love to hear what you think about the end result.

Munich, Germany, October 2022

Kaj Arnö, CEO, MariaDB Foundation

Chapter Contents

Table of Contents

1 Using MariaDB Server

Documentation on using MariaDB Server.

SQL Statements & Structure

SQL statements, structure, and rules.

Built-in Functions

Functions and procedures in MariaDB.

Clients & Utilities

Client and utility programs for MariaDB.

1.1 SQL Statements & Structure

The letters *SQL* stand for Structured Query Language. As with all languages—even computer languages—there are grammar rules. This includes a certain structure to statements, acceptable punctuation (i.e., operators and delimiters), and a vocabulary (i.e., reserve words).

SQL Statements

Explanations of all of the MariaDB SQL statements.

SQL Language Structure

Explanation of SQL grammar rules, including reserved words and literals.

Geographic & Geometric Features

Spatial extensions for geographic and geometric features.

NoSQL

NoSQL-related commands and interfaces

Operators

Operators for comparing and assigning values.

Sequences

Sequence objects, an alternative to AUTO_INCREMENT.

Temporal Tables

MariaDB supports system-versioning, application-time periods and bitemporal tables.

Unleashing the Power of Advanced SQL: Joins, Subqueries, and Set Operations

SQL (Structured Query Language) is a highly potent language used in the rea...

There are 19 related questions.

1.1.1 SQL Statements

Complete list of SQL statements for data definition, data manipulation, etc.

Account Management SQL Commands

CREATE/DROP USER, GRANT, REVOKE, SET PASSWORD etc.

Administrative SQL Statements

SQL statements for setting, flushing and displaying server variables and resources.

Data Definition

SQL commands for defining data, such as ALTER, CREATE, DROP, RENAME etc.

Data Manipulation

SQL commands for querying and manipulating data, such as SELECT, UPDATE, DELETE etc.

Prepared Statements

Prepared statements from any client using the text based prepared statement interface.

Programmatic & Compound Statements

Compound SQL statements for stored routines and in general.

Stored Routine Statements

SQL statements related to creating and using stored routines.

Table Statements

Documentation on creating, altering, analyzing and maintaining tables.

Transactions

Sequence of statements that are either completely successful, or have no effect on any schemas

HELP Command

The HELP command will retrieve syntax and help within the mariadb client.

Comment Syntax

Comment syntax and style.

Built-in Functions

Functions and procedures in MariaDB.

There are 24 related questions.

1.1.1.1 Account Management SQL Commands

CREATE/DROP USER, GRANT, REVOKE, SET PASSWORD etc.

CREATE USER

Create new MariaDB accounts.

ALTER USER

Modify an existing MariaDB account.

DROP USER

Remove one or more MariaDB accounts.

GRANT

Create accounts and set privileges or roles.

RENAME USER

Rename user account.

REVOKE

Remove privileges or roles.

SET PASSWORD

Assign password to an existing MariaDB user.

CREATE ROLE

Add new roles.

DROP ROLE

Drop a role.

SET ROLE

Enable a role.

SET DEFAULT ROLE
Sets a default role for a specified (or current) user.

SHOW GRANTS
View GRANT statements.

SHOW CREATE USER
Show the CREATE USER statement for a specified user.

There are 3 related questions.

1.1.1.1.1 CREATE USER

Syntax

```
CREATE [OR REPLACE] USER [IF NOT EXISTS]
 user_specification [,user_specification ...]
  [REQUIRE {NONE | tls_option [[AND] tls_option ...] }]
  [WITH resource_option [resource_option ...] ]
  [lock_option] [password_option]

user_specification:
  username [authentication_option]

authentication_option:
  IDENTIFIED BY 'password'
  | IDENTIFIED BY PASSWORD 'password_hash'
  | IDENTIFIED {VIA|WITH} authentication_rule [OR authentication_rule  ...]

authentication_rule:
    authentication_plugin
  | authentication_plugin {USING|AS} 'authentication_string'
  | authentication_plugin {USING|AS} PASSWORD('password')

tls_option:
  SSL
  | X509
  | CIPHER 'cipher'
  | ISSUER 'issuer'
  | SUBJECT 'subject'

resource_option:
  MAX_QUERIES_PER_HOUR count
  | MAX_UPDATES_PER_HOUR count
  | MAX_CONNECTIONS_PER_HOUR count
  | MAX_USER_CONNECTIONS count
  | MAX_STATEMENT_TIME time

password_option:
  PASSWORD EXPIRE
  | PASSWORD EXPIRE DEFAULT
  | PASSWORD EXPIRE NEVER
  | PASSWORD EXPIRE INTERVAL N DAY

lock_option:
    ACCOUNT LOCK
  | ACCOUNT UNLOCK
}
```

Contents

Description

The `CREATE USER` statement creates new MariaDB accounts. To use it, you must have the global CREATE USER privilege or the INSERT privilege for the mysql database. For each account, `CREATE USER` creates a new row in mysql.user (until MariaDB 10.3 this is a table, from MariaDB 10.4 it's a view) or mysql.global_priv_table (from MariaDB 10.4) that has no privileges.

If any of the specified accounts, or any permissions for the specified accounts, already exist, then the server returns `ERROR 1396 (HY000)`. If an error occurs, `CREATE USER` will still create the accounts that do not result in an error. Only one error is produced for all users which have not been created:

```
ERROR 1396 (HY000):
  Operation CREATE USER failed for 'u1'@'%','u2'@'%'
```

`CREATE USER`, DROP USER, CREATE ROLE, and DROP ROLE all produce the same error code when they fail.

See Account Names below for details on how account names are specified.

OR REPLACE

If the optional `OR REPLACE` clause is used, it is basically a shortcut for:

```
DROP USER IF EXISTS name;
CREATE USER name ...;
```

For example:

```
CREATE USER foo2@test IDENTIFIED BY 'password';
ERROR 1396 (HY000): Operation CREATE USER failed for 'foo2'@'test'

CREATE OR REPLACE USER foo2@test IDENTIFIED BY 'password';
Query OK, 0 rows affected (0.00 sec)
```

IF NOT EXISTS

When the `IF NOT EXISTS` clause is used, MariaDB will return a warning instead of an error if the specified user already exists.

For example:

```
CREATE USER foo2@test IDENTIFIED BY 'password';
ERROR 1396 (HY000): Operation CREATE USER failed for 'foo2'@'test'

CREATE USER IF NOT EXISTS foo2@test IDENTIFIED BY 'password';
Query OK, 0 rows affected, 1 warning (0.00 sec)

SHOW WARNINGS;
+-------+------+-----------------------------------------------------+
| Level | Code | Message                                             |
+-------+------+-----------------------------------------------------+
| Note  | 1973 | Can't create user 'foo2'@'test'; it already exists |
+-------+------+-----------------------------------------------------+
```

Authentication Options

IDENTIFIED BY 'password'

The optional `IDENTIFIED BY` clause can be used to provide an account with a password. The password should be specified in plain text. It will be hashed by the PASSWORD function prior to being stored in the mysql.user/mysql.global_priv_table table.

For example, if our password is `mariadb`, then we can create the user with:

```
CREATE USER foo2@test IDENTIFIED BY 'mariadb';
```

If you do not specify a password with the `IDENTIFIED BY` clause, the user will be able to connect without a password. A blank password is not a wildcard to match any password. The user must connect without providing a password if no password is set.

The only authentication plugins that this clause supports are mysql_native_password and mysql_old_password.

IDENTIFIED BY PASSWORD 'password_hash'

The optional `IDENTIFIED BY PASSWORD` clause can be used to provide an account with a password that has already been hashed. The password should be specified as a hash that was provided by the PASSWORD function. It will be stored in the mysql.user/mysql.global_priv_table table as-is.

For example, if our password is `mariadb`, then we can find the hash with:

```
SELECT PASSWORD('mariadb');
+-------------------------------------------+
| PASSWORD('mariadb')                       |
+-------------------------------------------+
| *54958E764CE10E50764C2EECBB71D01F08549980 |
+-------------------------------------------+
1 row in set (0.00 sec)
```

And then we can create a user with the hash:

```
CREATE USER foo2@test IDENTIFIED BY PASSWORD '*54958E764CE10E50764C2EECBB71D01F08549980';
```

If you do not specify a password with the `IDENTIFIED BY` clause, the user will be able to connect without a password. A blank password is not a wildcard to match any password. The user must connect without providing a password if no password is set.

The only authentication plugins that this clause supports are mysql_native_password and mysql_old_password.

IDENTIFIED {VIA|WITH} authentication_plugin

The optional `IDENTIFIED VIA authentication_plugin` allows you to specify that the account should be authenticated by a specific authentication plugin. The plugin name must be an active authentication plugin as per SHOW PLUGINS. If it doesn't show up in that output, then you will need to install it with INSTALL PLUGIN or INSTALL SONAME.

`VIA` and `WITH` are synonyms.

For example, this could be used with the PAM authentication plugin:

```
CREATE USER foo2@test IDENTIFIED VIA pam;
```

Some authentication plugins allow additional arguments to be specified after a `USING` or `AS` keyword. For example, the PAM authentication plugin accepts a service name:

```
CREATE USER foo2@test IDENTIFIED VIA pam USING 'mariadb';
```

The exact meaning of the additional argument would depend on the specific authentication plugin.

MariaDB starting with 10.4.0

The `USING` or `AS` keyword can also be used to provide a plain-text password to a plugin if it's provided as an argument to the PASSWORD() function. This is only valid for authentication plugins that have implemented a hook for the PASSWORD() function. For example, the ed25519 authentication plugin supports this:

```
CREATE USER safe@'%' IDENTIFIED VIA ed25519 USING PASSWORD('secret');
```

MariaDB starting with 10.4.3

One can specify many authentication plugins, they all work as alternatives ways of authenticating a user:

```
CREATE USER safe@'%' IDENTIFIED VIA ed25519 USING PASSWORD('secret') OR unix_socket;
```

By default, when you create a user without specifying an authentication plugin, MariaDB uses the mysql_native_password plugin.

TLS Options

By default, MariaDB transmits data between the server and clients without encrypting it. This is generally acceptable when the server and client run on the same host or in networks where security is guaranteed through other means. However, in cases where the server and client exist on separate networks or they are in a high-risk network, the lack of encryption does introduce security concerns as a malicious actor could potentially eavesdrop on the traffic as it is sent over the network between them.

To mitigate this concern, MariaDB allows you to encrypt data in transit between the server and clients using the Transport Layer Security (TLS) protocol. TLS was formerly known as Secure Socket Layer (SSL), but strictly speaking the SSL protocol is a predecessor to TLS and, that version of the protocol is now considered insecure. The documentation still uses the term SSL often and for compatibility reasons TLS-related server system and status variables still use the prefix ssl_, but internally, MariaDB only supports its secure successors.

See Secure Connections Overview for more information about how to determine whether your MariaDB server has TLS support.

You can set certain TLS-related restrictions for specific user accounts. For instance, you might use this with user accounts that require access to sensitive data while sending it across networks that you do not control. These restrictions can be enabled for a user account with the CREATE USER, ALTER USER, or GRANT statements. The following options are available:

Option	Description
`REQUIRE NONE`	TLS is not required for this account, but can still be used.
`REQUIRE SSL`	The account must use TLS, but no valid X509 certificate is required. This option cannot be combined with other TLS options.
`REQUIRE X509`	The account must use TLS and must have a valid X509 certificate. This option implies `REQUIRE SSL`. This option cannot be combined with other TLS options.
`REQUIRE ISSUER 'issuer'`	The account must use TLS and must have a valid X509 certificate. Also, the Certificate Authority must be the one specified via the string `issuer`. This option implies `REQUIRE X509`. This option can be combined with the `SUBJECT`, and `CIPHER` options in any order.
`REQUIRE SUBJECT 'subject'`	The account must use TLS and must have a valid X509 certificate. Also, the certificate's Subject must be the one specified via the string `subject`. This option implies `REQUIRE X509`. This option can be combined with the `ISSUER`, and `CIPHER` options in any order.

`REQUIRE CIPHER 'cipher'`	The account must use TLS, but no valid X509 certificate is required. Also, the encryption used for the connection must use a specific cipher method specified in the string `cipher`. This option implies `REQUIRE SSL`. This option can be combined with the `ISSUER`, and `SUBJECT` options in any order.

The `REQUIRE` keyword must be used only once for all specified options, and the `AND` keyword can be used to separate individual options, but it is not required.

For example, you can create a user account that requires these TLS options with the following:

```
CREATE USER 'alice'@'%'
 REQUIRE SUBJECT '/CN=alice/O=My Dom, Inc./C=US/ST=Oregon/L=Portland'
 AND ISSUER '/C=FI/ST=Somewhere/L=City/ O=Some Company/CN=Peter
Parker/emailAddress=p.parker@marvel.com'
 AND CIPHER 'SHA-DES-CBC3-EDH-RSA';
```

If any of these options are set for a specific user account, then any client who tries to connect with that user account will have to be configured to connect with TLS.

See Securing Connections for Client and Server for information on how to enable TLS on the client and server.

Resource Limit Options

It is possible to set per-account limits for certain server resources. The following table shows the values that can be set per account:

Limit Type	Decription
`MAX_QUERIES_PER_HOUR`	Number of statements that the account can issue per hour (including updates)
`MAX_UPDATES_PER_HOUR`	Number of updates (not queries) that the account can issue per hour
`MAX_CONNECTIONS_PER_HOUR`	Number of connections that the account can start per hour
`MAX_USER_CONNECTIONS`	Number of simultaneous connections that can be accepted from the same account; if it is 0, `max_connections` will be used instead; if `max_connections` is 0, there is no limit for this account's simultaneous connections.
`MAX_STATEMENT_TIME`	Timeout, in seconds, for statements executed by the user. See also Aborting Statements that Exceed a Certain Time to Execute.

If any of these limits are set to `0`, then there is no limit for that resource for that user.

Here is an example showing how to create a user with resource limits:

```
CREATE USER 'someone'@'localhost' WITH
    MAX_USER_CONNECTIONS 10
    MAX_QUERIES_PER_HOUR 200;
```

The resources are tracked per account, which means `'user'@'server'`; not per user name or per connection.

The count can be reset for all users using FLUSH USER_RESOURCES, FLUSH PRIVILEGES or mariadb-admin reload.

Per account resource limits are stored in the user table, in the mysql database. Columns used for resources limits are named `max_questions`, `max_updates`, `max_connections` (for `MAX_CONNECTIONS_PER_HOUR`), and `max_user_connections` (for `MAX_USER_CONNECTIONS`).

Account Names

Account names have both a user name component and a host name component, and are specified as `'user_name'@'host_name'`.

The user name and host name may be unquoted, quoted as strings using double quotes (`"`) or single quotes (`'`), or quoted as identifiers using backticks (`` ` ``). You must use quotes when using special characters (such as a hyphen) or wildcard characters. If you quote, you must quote the user name and host name separately (for example `'user_name'@'host_name'`).

Host Name Component

If the host name is not provided, it is assumed to be `'%'`.

Host names may contain the wildcard characters `%` and `_`. They are matched as if by the LIKE clause. If you need to use a wildcard character literally (for example, to match a domain name with an underscore), prefix the character with a backslash. See `LIKE` for more information on escaping wildcard characters.

Host name matches are case-insensitive. Host names can match either domain names or IP addresses. Use `'localhost'` as the host name to allow only local client connections. On Linux, the loopback interface (127.0.0.1) will not match 'localhost' as it is not considered a local connection: this means that only connections via UNIX-domain sockets will match 'localhost'.

You can use a netmask to match a range of IP addresses using `'base_ip/netmask'` as the host name. A user with an IP address *ip_addr* will be allowed to connect if the following condition is true:

```
ip_addr & netmask = base_ip
```

For example, given a user:

```
CREATE USER 'maria'@'247.150.130.0/255.255.255.0';
```

the IP addresses satisfying this condition range from 247.150.130.0 to 247.150.130.255.

Using `255.255.255.255` is equivalent to not using a netmask at all. Netmasks cannot be used for IPv6 addresses.

Note that the credentials added when creating a user with the `'%'` wildcard host will not grant access in all cases. For example, some systems come with an anonymous localhost user, and when connecting from localhost this will take precedence.

Before MariaDB 10.6, the host name component could be up to 60 characters in length. Starting from MariaDB 10.6, it can be up to 255 characters.

User Name Component

User names must match exactly, including case. A user name that is empty is known as an anonymous account and is allowed to match a login attempt with any user name component. These are described more in the next section.

For valid identifiers to use as user names, see Identifier Names.

It is possible for more than one account to match when a user connects. MariaDB selects the first matching account after sorting according to the following criteria:

- Accounts with an exact host name are sorted before accounts using a wildcard in the host name. Host names using a netmask are considered to be exact for sorting.
- Accounts with a wildcard in the host name are sorted according to the position of the first wildcard character. Those with a wildcard character later in the host name sort before those with a wildcard character earlier in the host name.
- Accounts with a non-empty user name sort before accounts with an empty user name.
- Accounts with an empty user name are sorted last. As mentioned previously, these are known as anonymous accounts. These are described more in the next section.

The following table shows a list of example account as sorted by these criteria:

```
+---------+-------------+
| User    | Host        |
+---------+-------------+
| joffrey | 192.168.0.3 |
|         | 192.168.0.% |
| joffrey | 192.168.%   |
|         | 192.168.%   |
+---------+-------------+
```

Once connected, you only have the privileges granted to the account that matched, not all accounts that could have matched. For example, consider the following commands:

```
CREATE USER 'joffrey'@'192.168.0.3';
CREATE USER 'joffrey'@'%';
GRANT SELECT ON test.t1 to 'joffrey'@'192.168.0.3';
GRANT SELECT ON test.t2 to 'joffrey'@'%';
```

If you connect as joffrey from `192.168.0.3`, you will have the `SELECT` privilege on the table `test.t1`, but not on the table `test.t2`. If you connect as joffrey from any other IP address, you will have the `SELECT` privilege on the table `test.t2`, but not on the table `test.t1`.

Usernames can be up to 80 characters long before 10.6 and starting from 10.6 it can be 128 characters long.

Anonymous Accounts

Anonymous accounts are accounts where the user name portion of the account name is empty. These accounts act as special catch-all accounts. If a user attempts to log into the system from a host, and an anonymous account exists with a host name portion that matches the user's host, then the user will log in as the anonymous account if there is no more specific account match for the user name that the user entered.

For example, here are some anonymous accounts:

```
CREATE USER ''@'localhost';
CREATE USER ''@'192.168.0.3';
```

Fixing a Legacy Default Anonymous Account

On some systems, the mysql.db table has some entries for the `''@'%'` anonymous account by default. Unfortunately, there is no matching entry in the mysql.user/mysql.global_priv_table table, which means that this anonymous account doesn't exactly exist, but it does have privileges--usually on the default `test` database created by mariadb-install-db. These account-less privileges are a legacy that is leftover from a time when MySQL's privilege system was less advanced.

This situation means that you will run into errors if you try to create a `''@'%'` account. For example:

```
CREATE USER ''@'%';
ERROR 1396 (HY000): Operation CREATE USER failed for ''@'%'
```

The fix is to DELETE the row in the mysql.db table and then execute FLUSH PRIVILEGES:

```
DELETE FROM mysql.db WHERE User='' AND Host='%';
FLUSH PRIVILEGES;
```

Note that `FLUSH PRIVILEGES` is only needed if one modifies the `mysql` tables directly. It is not needed when using `CREATE USER`, `DROP USER`, `GRANT` etc.

And then the account can be created:

```
CREATE USER ''@'%';
Query OK, 0 rows affected (0.01 sec)
```

See MDEV-13486 for more information.

Password Expiry

MariaDB starting with 10.4.3

Besides automatic password expiry, as determined by default_password_lifetime, password expiry times can be set on an individual user basis, overriding the global setting, for example:

```
CREATE USER 'monty'@'localhost' PASSWORD EXPIRE INTERVAL 120 DAY;
```

See User Password Expiry for more details.

Account Locking

MariaDB starting with 10.4.2

Account locking permits privileged administrators to lock/unlock user accounts. No new client connections will be permitted if an account is locked (existing connections are not affected). For example:

```
CREATE USER 'marijn'@'localhost' ACCOUNT LOCK;
```

See Account Locking for more details.

From MariaDB 10.4.7 and MariaDB 10.5.8, the *lock_option* and *password_option* clauses can occur in either order.

1.1.1.1.2 ALTER USER

Syntax

```
ALTER USER [IF EXISTS]
 user_specification [,user_specification] ...
  [REQUIRE {NONE | tls_option [[AND] tls_option] ...}]
  [WITH resource_option [resource_option] ...]
  [lock_option] [password_option]

user_specification:
  username [authentication_option]

authentication_option:
  IDENTIFIED BY 'password'
  | IDENTIFIED BY PASSWORD 'password_hash'
  | IDENTIFIED {VIA|WITH} authentication_rule [OR authentication_rule] ...

authentication_rule:
  authentication_plugin
  | authentication_plugin {USING|AS} 'authentication_string'
  | authentication_plugin {USING|AS} PASSWORD('password')

tls_option
  SSL
  | X509
  | CIPHER 'cipher'
  | ISSUER 'issuer'
  | SUBJECT 'subject'

resource_option
  MAX_QUERIES_PER_HOUR count
  | MAX_UPDATES_PER_HOUR count
  | MAX_CONNECTIONS_PER_HOUR count
  | MAX_USER_CONNECTIONS count
  | MAX_STATEMENT_TIME time

password_option:
  PASSWORD EXPIRE
  | PASSWORD EXPIRE DEFAULT
  | PASSWORD EXPIRE NEVER
  | PASSWORD EXPIRE INTERVAL N DAY

lock_option:
    ACCOUNT LOCK
  | ACCOUNT UNLOCK
}
```

Contents

Description

The `ALTER USER` statement modifies existing MariaDB accounts. To use it, you must have the global CREATE USER privilege or the UPDATE privilege for the mysql database. The global SUPER privilege is also required if the read_only system variable is enabled.

If any of the specified user accounts do not yet exist, an error results. If an error occurs, `ALTER USER` will still modify the accounts that do not result in an error. Only one error is produced for all users which have not been modified.

For renaming an existing account (user name and/or host), see RENAME USER.

IF EXISTS

When the `IF EXISTS` clause is used, MariaDB will return a warning instead of an error for each specified user that does not exist.

Account Names

For `ALTER USER` statements, account names are specified as the `username` argument in the same way as they are for CREATE USER statements. See account names from the `CREATE USER` page for details on how account names are specified.

CURRENT_USER or `CURRENT_USER()` can also be used to alter the account logged into the current session. For example, to change the current user's password to `mariadb`:

```
ALTER USER CURRENT_USER() IDENTIFIED BY 'mariadb';
```

Authentication Options

MariaDB starting with 10.4

From MariaDB 10.4, it is possible to use more than one authentication plugin for each user account. For example, this can be useful to slowly migrate users to the more secure ed25519 authentication plugin over time, while allowing the old mysql_native_password authentication plugin as an alternative for the transitional period. See Authentication from MariaDB 10.4 for more.

When running `ALTER USER`, not specifying an authentication option in the IDENTIFIED VIA clause will remove that authentication method. (However this was not the case before MariaDB 10.4.13, see MDEV-21928)

For example, a user is created with the ability to authenticate via both a password and unix_socket:

```
CREATE USER 'bob'@'localhost'
  IDENTIFIED VIA mysql_native_password USING PASSWORD('pwd')
  OR unix_socket;

SHOW CREATE USER 'bob'@'localhost'\G
*************************** 1. row ***************************
CREATE USER for bob@localhost: CREATE USER `bob`@`localhost`
  IDENTIFIED VIA mysql_native_password
  USING '*975B2CD4FF9AE554FE8AD33168FBFC326D2021DD'
  OR unix_socket
```

If the user's password is updated, but unix_socket authentication is not specified in the `IDENTIFIED VIA` clause, unix_socket authentication will no longer be permitted.

```
ALTER USER 'bob'@'localhost' IDENTIFIED VIA mysql_native_password
  USING PASSWORD('pwd2');

SHOW CREATE USER 'bob'@'localhost'\G
*************************** 1. row ***************************
CREATE USER for bob@localhost: CREATE USER `bob`@`localhost`
  IDENTIFIED BY PASSWORD '*38366FDA01695B6A5A9DD4E428D9FB8F7EB75512'
```

IDENTIFIED BY 'password'

The optional `IDENTIFIED BY` clause can be used to provide an account with a password. The password should be specified in plain text. It will be hashed by the PASSWORD function prior to being stored to the mysql.user table.

For example, if our password is `mariadb`, then we can set the account's password with:

```
ALTER USER foo2@test IDENTIFIED BY 'mariadb';
```

If you do not specify a password with the `IDENTIFIED BY` clause, the user will be able to connect without a password. A blank password is not a wildcard to match any password. The user must connect without providing a password if no password is set.

The only authentication plugins that this clause supports are mysql_native_password and mysql_old_password.

IDENTIFIED BY PASSWORD 'password_hash'

The optional `IDENTIFIED BY PASSWORD` clause can be used to provide an account with a password that has already been hashed. The password should be specified as a hash that was provided by the PASSWORD#function. It will be stored to the mysql.user table as-is.

For example, if our password is `mariadb`, then we can find the hash with:

```
SELECT PASSWORD('mariadb');
+-------------------------------------------+
| PASSWORD('mariadb')                       |
+-------------------------------------------+
| *54958E764CE10E50764C2EECBB71D01F08549980 |
+-------------------------------------------+
```

And then we can set an account's password with the hash:

```
ALTER USER foo2@test
  IDENTIFIED BY PASSWORD '*54958E764CE10E50764C2EECBB71D01F08549980';
```

If you do not specify a password with the `IDENTIFIED BY` clause, the user will be able to connect without a password. A blank password is not a wildcard to match any password. The user must connect without providing a password if no password is set.

The only authentication plugins that this clause supports are mysql_native_password and mysql_old_password.

IDENTIFIED {VIA|WITH} authentication_plugin

The optional `IDENTIFIED VIA authentication_plugin` allows you to specify that the account should be authenticated by a specific authentication plugin. The plugin name must be an active authentication plugin as per SHOW PLUGINS. If it doesn't show up in that output, then you will need to install it with INSTALL PLUGIN or INSTALL SONAME.

For example, this could be used with the PAM authentication plugin:

```
ALTER USER foo2@test IDENTIFIED VIA pam;
```

Some authentication plugins allow additional arguments to be specified after a `USING` or `AS` keyword. For example, the PAM authentication plugin accepts a service name:

```
ALTER USER foo2@test IDENTIFIED VIA pam USING 'mariadb';
```

The exact meaning of the additional argument would depend on the specific authentication plugin.

In MariaDB 10.4 and later, the `USING` or `AS` keyword can also be used to provide a plain-text password to a plugin if it's provided as an argument to the PASSWORD() function. This is only valid for authentication plugins that have implemented a hook for the PASSWORD() function. For example, the ed25519 authentication plugin supports this:

```
ALTER USER safe@'%' IDENTIFIED VIA ed25519 USING PASSWORD('secret');
```

TLS Options

By default, MariaDB transmits data between the server and clients without encrypting it. This is generally acceptable when the server and client run on the same host or in networks where security is guaranteed through other means. However, in cases where the server and client exist on separate networks or they are in a high-risk network, the lack of encryption does introduce security concerns as a malicious actor could potentially eavesdrop on the traffic as it is sent over the network between them.

To mitigate this concern, MariaDB allows you to encrypt data in transit between the server and clients using the Transport Layer Security (TLS) protocol. TLS was formerly known as Secure Socket Layer (SSL), but strictly speaking the SSL protocol is a predecessor to TLS and, that version of the protocol is now considered insecure. The documentation still uses

the term SSL often and for compatibility reasons TLS-related server system and status variables still use the prefix ssl_, but internally, MariaDB only supports its secure successors.

See Secure Connections Overview for more information about how to determine whether your MariaDB server has TLS support.

You can set certain TLS-related restrictions for specific user accounts. For instance, you might use this with user accounts that require access to sensitive data while sending it across networks that you do not control. These restrictions can be enabled for a user account with the CREATE USER, ALTER USER, or GRANT statements. The following options are available:

Option	Description
`REQUIRE NONE`	TLS is not required for this account, but can still be used.
`REQUIRE SSL`	The account must use TLS, but no valid X509 certificate is required. This option cannot be combined with other TLS options.
`REQUIRE X509`	The account must use TLS and must have a valid X509 certificate. This option implies `REQUIRE SSL`. This option cannot be combined with other TLS options.
`REQUIRE ISSUER 'issuer'`	The account must use TLS and must have a valid X509 certificate. Also, the Certificate Authority must be the one specified via the string `issuer`. This option implies `REQUIRE X509`. This option can be combined with the `SUBJECT`, and `CIPHER` options in any order.
`REQUIRE SUBJECT 'subject'`	The account must use TLS and must have a valid X509 certificate. Also, the certificate's Subject must be the one specified via the string `subject`. This option implies `REQUIRE X509`. This option can be combined with the `ISSUER`, and `CIPHER` options in any order.
`REQUIRE CIPHER 'cipher'`	The account must use TLS, but no valid X509 certificate is required. Also, the encryption used for the connection must use a specific cipher method specified in the string `cipher`. This option implies `REQUIRE SSL`. This option can be combined with the `ISSUER`, and `SUBJECT` options in any order.

The `REQUIRE` keyword must be used only once for all specified options, and the `AND` keyword can be used to separate individual options, but it is not required.

For example, you can alter a user account to require these TLS options with the following:

```
ALTER USER 'alice'@'%'
 REQUIRE SUBJECT '/CN=alice/O=My Dom, Inc./C=US/ST=Oregon/L=Portland' AND
 ISSUER '/C=FI/ST=Somewhere/L=City/ O=Some Company/CN=Peter Parker/emailAddress=p.parker@marvel.c
 AND CIPHER 'SHA-DES-CBC3-EDH-RSA';
```

If any of these options are set for a specific user account, then any client who tries to connect with that user account will have to be configured to connect with TLS.

See Securing Connections for Client and Server for information on how to enable TLS on the client and server.

Resource Limit Options

It is possible to set per-account limits for certain server resources. The following table shows the values that can be set per account:

Limit Type	Description
`MAX_QUERIES_PER_HOUR`	Number of statements that the account can issue per hour (including updates)
`MAX_UPDATES_PER_HOUR`	Number of updates (not queries) that the account can issue per hour
`MAX_CONNECTIONS_PER_HOUR`	Number of connections that the account can start per hour
`MAX_USER_CONNECTIONS`	Number of simultaneous connections that can be accepted from the same account; if it is 0, `max_connections` will be used instead; if `max_connections` is 0, there is no limit for this account's simultaneous connections.
`MAX_STATEMENT_TIME`	Timeout, in seconds, for statements executed by the user. See also Aborting Statements that Exceed a Certain Time to Execute.

If any of these limits are set to `0`, then there is no limit for that resource for that user.

Here is an example showing how to set an account's resource limits:

```
ALTER USER 'someone'@'localhost' WITH
    MAX_USER_CONNECTIONS 10
    MAX_QUERIES_PER_HOUR 200;
```

The resources are tracked per account, which means `'user'@'server'`; not per user name or per connection.

The count can be reset for all users using FLUSH USER_RESOURCES, FLUSH PRIVILEGES or mysqladmin reload.

Per account resource limits are stored in the user table, in the mysql database. Columns used for resources limits are named `max_questions`, `max_updates`, `max_connections` (for `MAX_CONNECTIONS_PER_HOUR`), and `max_user_connections` (for `MAX_USER_CONNECTIONS`).

Password Expiry

MariaDB starting with 10.4.3

Besides automatic password expiry, as determined by default_password_lifetime, password expiry times can be set on an individual user basis, overriding the global setting, for example:

```
ALTER USER 'monty'@'localhost' PASSWORD EXPIRE INTERVAL 120 DAY;
ALTER USER 'monty'@'localhost' PASSWORD EXPIRE NEVER;
ALTER USER 'monty'@'localhost' PASSWORD EXPIRE DEFAULT;
```

See User Password Expiry for more details.

Account Locking

MariaDB starting with 10.4.2

Account locking permits privileged administrators to lock/unlock user accounts. No new client connections will be permitted if an account is locked (existing connections are not affected). For example:

```
ALTER USER 'marijn'@'localhost' ACCOUNT LOCK;
```

See Account Locking for more details.

From MariaDB 10.4.7 and MariaDB 10.5.8, the *lock_option* and *password_option* clauses can occur in either order.

1.1.1.1.3 DROP USER

Syntax

```
DROP USER [IF EXISTS] user_name [, user_name] ...
```

Contents

Description

The `DROP USER` statement removes one or more MariaDB accounts. It removes privilege rows for the account from all grant tables. To use this statement, you must have the global CREATE USER privilege or the DELETE privilege for the mysql database. Each account is named using the same format as for the `CREATE USER` statement; for example, `'jeffrey'@'localhost'`. If you specify only the user name part of the account name, a host name part of `'%'` is used. For additional information about specifying account names, see CREATE USER.

Note that, if you specify an account that is currently connected, it will not be deleted until the connection is closed. The connection will not be automatically closed.

If any of the specified user accounts do not exist, `ERROR 1396 (HY000)` results. If an error occurs, `DROP USER` will still drop the accounts that do not result in an error. Only one error is produced for all users which have not been dropped:

```
ERROR 1396 (HY000): Operation DROP USER failed for 'u1'@'%','u2'@'%'
```

Failed `CREATE` or `DROP` operations, for both users and roles, produce the same error code.

IF EXISTS

If the `IF EXISTS` clause is used, MariaDB will return a note instead of an error if the user does not exist.

Examples

```
DROP USER bob;

DROP USER foo2@localhost,foo2@'127.%';
```

`IF EXISTS`:

```
DROP USER bob;
ERROR 1396 (HY000): Operation DROP USER failed for 'bob'@'%'

DROP USER IF EXISTS bob;
Query OK, 0 rows affected, 1 warning (0.00 sec)

SHOW WARNINGS;
+-------+------+-----------------------------------------+
| Level | Code | Message                                 |
+-------+------+-----------------------------------------+
| Note  | 1974 | Can't drop user 'bob'@'%'; it doesn't exist |
+-------+------+-----------------------------------------+
```

1.1.1.1.4 GRANT

Contents

Syntax

```
GRANT
    priv_type [(column_list)]
      [, priv_type [(column_list)]] ...
    ON [object_type] priv_level
    TO user_specification [ user_options ...]

user_specification:
  username [authentication_option]
  | PUBLIC
authentication_option:
  IDENTIFIED BY 'password'
  | IDENTIFIED BY PASSWORD 'password_hash'
  | IDENTIFIED {VIA|WITH} authentication_rule [OR authentication_rule  ...]

authentication_rule:
    authentication_plugin
  | authentication_plugin {USING|AS} 'authentication_string'
  | authentication_plugin {USING|AS} PASSWORD('password')

GRANT PROXY ON username
    TO user_specification [, user_specification ...]
    [WITH GRANT OPTION]

GRANT rolename TO grantee [, grantee ...]
    [WITH ADMIN OPTION]

grantee:
    rolename
    username [authentication_option]

user_options:
    [REQUIRE {NONE | tls_option [[AND] tls_option] ...}]
    [WITH with_option [with_option] ...]

object_type:
    TABLE
  | FUNCTION
  | PROCEDURE
  | PACKAGE
  | PACKAGE BODY

priv_level:
    *
  | *.*
  | db_name.*
  | db_name.tbl_name
  | tbl_name
  | db_name.routine_name

with_option:
    GRANT OPTION
  | resource_option

resource_option:
  MAX_QUERIES_PER_HOUR count
  | MAX_UPDATES_PER_HOUR count
  | MAX_CONNECTIONS_PER_HOUR count
  | MAX_USER_CONNECTIONS count
  | MAX_STATEMENT_TIME time

tls_option:
  SSL
  | X509
  | CIPHER 'cipher'
  | ISSUER 'issuer'
  | SUBJECT 'subject'
```

Description

The `GRANT` statement allows you to grant privileges or roles to accounts. To use `GRANT`, you must have the `GRANT OPTION` privilege, and you must have the privileges that you are granting.

Use the REVOKE statement to revoke privileges granted with the `GRANT` statement.

Use the SHOW GRANTS statement to determine what privileges an account has.

Account Names

For `GRANT` statements, account names are specified as the `username` argument in the same way as they are for CREATE USER statements. See account names from the `CREATE USER` page for details on how account names are specified.

Implicit Account Creation

The `GRANT` statement also allows you to implicitly create accounts in some cases.

If the account does not yet exist, then `GRANT` can implicitly create it. To implicitly create an account with `GRANT`, a user is required to have the same privileges that would be required to explicitly create the account with the `CREATE USER` statement.

If the `NO_AUTO_CREATE_USER` SQL_MODE is set, then accounts can only be created if authentication information is specified, or with a CREATE USER statement. If no authentication information is provided, `GRANT` will produce an error when the specified account does not exist, for example:

```
show variables like '%sql_mode%' ;
+---------------+---------------------------------------------+
| Variable_name | Value                                       |
+---------------+---------------------------------------------+
| sql_mode      | NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION |
+---------------+---------------------------------------------+

GRANT USAGE ON *.* TO 'user123'@'%' IDENTIFIED BY '';
ERROR 1133 (28000): Can't find any matching row in the user table

GRANT USAGE ON *.* TO 'user123'@'%'
  IDENTIFIED VIA PAM using 'mariadb' require ssl ;
Query OK, 0 rows affected (0.00 sec)

select host, user from mysql.user where user='user123' ;

+------+----------+
| host | user     |
+------+----------+
| %    | user123 |
+------+----------+
```

Privilege Levels

Privileges can be set globally, for an entire database, for a table or routine, or for individual columns in a table. Certain privileges can only be set at certain levels.

Global privileges do not take effect immediately and are only applied to connections created after the `GRANT` statement was executed.

- Global privileges *priv_type* are granted using `*.*` for *priv_level*. Global privileges include privileges to administer the database and manage user accounts, as well as privileges for all tables, functions, and procedures. Global privileges are stored in the mysql.user table prior to MariaDB 10.4, and in mysql.global_priv table afterwards.
- Database privileges *priv_type* are granted using `db_name.*` for *priv_level*, or using just `*` to use default database. Database privileges include privileges to create tables and functions, as well as privileges for all tables, functions, and procedures in the database. Database privileges are stored in the mysql.db table.
- Table privileges *priv_type* are granted using `db_name.tbl_name` for *priv_level*, or using just `tbl_name` to specify a table in the default database. The `TABLE` keyword is optional. Table privileges include the ability to select and change data in the table. Certain table privileges can be granted for individual columns.
- Column privileges *priv_type* are granted by specifying a table for *priv_level* and providing a column list after the privilege type. They allow you to control exactly which columns in a table users can select and change.
- Function privileges *priv_type* are granted using `FUNCTION db_name.routine_name` for *priv_level*, or using just `FUNCTION routine_name` to specify a function in the default database.
- Procedure privileges *priv_type* are granted using `PROCEDURE db_name.routine_name` for *priv_level*, or using just `PROCEDURE routine_name` to specify a procedure in the default database.

The `USAGE` Privilege

The `USAGE` privilege grants no real privileges. The SHOW GRANTS statement will show a global `USAGE` privilege for a newly-created user. You can use `USAGE` with the `GRANT` statement to change options like `GRANT OPTION` and `MAX_USER_CONNECTIONS` without changing any account privileges.

The `ALL PRIVILEGES` Privilege

The `ALL PRIVILEGES` privilege grants all available privileges. Granting all privileges only affects the given privilege level. For example, granting all privileges on a table does not grant any privileges on the database or globally.

Using `ALL PRIVILEGES` does not grant the special `GRANT OPTION` privilege.

You can use `ALL` instead of `ALL PRIVILEGES`.

The `GRANT OPTION` Privilege

Use the `WITH GRANT OPTION` clause to give users the ability to grant privileges to other users at the given privilege level. Users with the `GRANT OPTION` privilege can only grant privileges they have. They cannot grant privileges at a higher privilege level than they have the `GRANT OPTION` privilege.

The `GRANT OPTION` privilege cannot be set for individual columns. If you use `WITH GRANT OPTION` when specifying column privileges, the `GRANT OPTION` privilege will be granted for the entire table.

Using the `WITH GRANT OPTION` clause is equivalent to listing `GRANT OPTION` as a privilege.

Global Privileges

The following table lists the privileges that can be granted globally. You can also grant all database, table, and function privileges globally. When granted globally, these privileges apply to all databases, tables, or functions, including those created later.

To set a global privilege, use `*.*` for *priv_level*.

BINLOG ADMIN

Enables administration of the binary log, including the PURGE BINARY LOGS statement and setting the system variables:

- binlog_annotate_row_events
- binlog_cache_size
- binlog_commit_wait_count
- binlog_commit_wait_usec
- binlog_direct_non_transactional_updates
- binlog_expire_logs_seconds
- binlog_file_cache_size
- binlog_format
- binlog_row_image
- binlog_row_metadata
- binlog_stmt_cache_size
- expire_logs_days
- log_bin_compress
- log_bin_compress_min_len
- log_bin_trust_function_creators
- max_binlog_cache_size
- max_binlog_size
- max_binlog_stmt_cache_size
- sql_log_bin and
- sync_binlog.

Added in MariaDB 10.5.2.

BINLOG MONITOR

New name for REPLICATION CLIENT from MariaDB 10.5.2, (`REPLICATION CLIENT` still supported as an alias for compatibility purposes). Permits running SHOW commands related to the binary log, in particular the SHOW BINLOG STATUS and SHOW BINARY LOGS statements. Unlike REPLICATION CLIENT prior to MariaDB 10.5, SHOW REPLICA STATUS isn't included in this privilege, and REPLICA MONITOR is required.

BINLOG REPLAY

Enables replaying the binary log with the BINLOG statement (generated by mariadb-binlog), executing SET timestamp

when secure_timestamp is set to `replication`, and setting the session values of system variables usually included in BINLOG output, in particular:

- gtid_domain_id
- gtid_seq_no
- pseudo_thread_id
- server_id.

Added in MariaDB 10.5.2

CONNECTION ADMIN

Enables administering connection resource limit options. This includes ignoring the limits specified by:

- max_connections
- max_user_connections and
- max_password_errors.

The statements specified in init_connect are not executed, killing connections and queries owned by other users is permitted. The following connection-related system variables can be changed:

- connect_timeout
- disconnect_on_expired_password
- extra_max_connections
- init_connect
- max_connections
- max_connect_errors
- max_password_errors
- proxy_protocol_networks
- secure_auth
- slow_launch_time
- thread_pool_exact_stats
- thread_pool_dedicated_listener
- thread_pool_idle_timeout
- thread_pool_max_threads
- thread_pool_min_threads
- thread_pool_oversubscribe
- thread_pool_prio_kickup_timer
- thread_pool_priority
- thread_pool_size, and
- thread_pool_stall_limit.

Added in MariaDB 10.5.2.

CREATE USER

Create a user using the CREATE USER statement, or implicitly create a user with the `GRANT` statement.

FEDERATED ADMIN

Execute CREATE SERVER, ALTER SERVER, and DROP SERVER statements. Added in MariaDB 10.5.2.

FILE

Read and write files on the server, using statements like LOAD DATA INFILE or functions like LOAD_FILE(). Also needed to create CONNECT outward tables. MariaDB server must have the permissions to access those files.

GRANT OPTION

Grant global privileges. You can only grant privileges that you have.

PROCESS

Show information about the active processes, for example via SHOW PROCESSLIST or mariadb-admin processlist. If you have the PROCESS privilege, you can see all threads. Otherwise, you can see only your own threads (that is, threads associated with the MariaDB account that you are using).

READ_ONLY ADMIN

User can set the read_only system variable and allows the user to perform write operations, even when the `read_only` option is active. Added in MariaDB 10.5.2.

From MariaDB 10.11.0, the `READ_ONLY ADMIN` privilege has been removed from SUPER. The benefit of this is that one can remove the READ_ONLY ADMIN privilege from all users and ensure that no one can make any changes on any non-temporary tables. This is useful on replicas when one wants to ensure that the replica is kept identical to the primary.

RELOAD

Execute FLUSH statements or equivalent mariadb-admin commands.

REPLICATION CLIENT

Execute SHOW MASTER STATUS and SHOW BINARY LOGS informative statements. Renamed to BINLOG MONITOR in MariaDB 10.5.2 (but still supported as an alias for compatibility reasons). SHOW SLAVE STATUS was part of REPLICATION CLIENT prior to MariaDB 10.5.

REPLICATION MASTER ADMIN

Permits administration of primary servers, including the SHOW REPLICA HOSTS statement, and setting the gtid_binlog_state, gtid_domain_id, master_verify_checksum and server_id system variables. Added in MariaDB 10.5.2.

REPLICA MONITOR

Permit SHOW REPLICA STATUS and SHOW RELAYLOG EVENTS. From MariaDB 10.5.9.

When a user would upgrade from an older major release to a MariaDB 10.5 minor release prior to MariaDB 10.5.9, certain user accounts would lose capabilities. For example, a user account that had the REPLICATION CLIENT privilege in older major releases could run SHOW REPLICA STATUS, but after upgrading to a MariaDB 10.5 minor release prior to MariaDB 10.5.9, they could no longer run SHOW REPLICA STATUS, because that statement was changed to require the REPLICATION REPLICA ADMIN privilege.

This issue is fixed in MariaDB 10.5.9 with this new privilege, which now grants the user the ability to execute SHOW [ALL] (SLAVE | REPLICA) STATUS.

When a database is upgraded from an older major release to MariaDB Server 10.5.9 or later, any user accounts with the REPLICATION CLIENT or REPLICATION SLAVE privileges will automatically be granted the new REPLICA MONITOR privilege. The privilege fix occurs when the server is started up, not when mariadb-upgrade is performed.

However, when a database is upgraded from an early 10.5 minor release to 10.5.9 and later, the user will have to fix any user account privileges manually.

REPLICATION REPLICA

Synonym for REPLICATION SLAVE. From MariaDB 10.5.1.

REPLICATION SLAVE

Accounts used by replica servers on the primary need this privilege. This is needed to get the updates made on the master. From MariaDB 10.5.1, REPLICATION REPLICA is an alias for `REPLICATION SLAVE`.

REPLICATION SLAVE ADMIN

Permits administering replica servers, including START REPLICA/SLAVE, STOP REPLICA/SLAVE, CHANGE MASTER, SHOW REPLICA/SLAVE STATUS, SHOW RELAYLOG EVENTS statements, replaying the binary log with the BINLOG statement (generated by mariadb-binlog), and setting the system variables:

- gtid_cleanup_batch_size
- gtid_ignore_duplicates
- gtid_pos_auto_engines
- gtid_slave_pos
- gtid_strict_mode
- init_slave
- read_binlog_speed_limit
- relay_log_purge
- relay_log_recovery
- replicate_do_db
- replicate_do_table
- replicate_events_marked_for_skip
- replicate_ignore_db
- replicate_ignore_table
- replicate_wild_do_table
- replicate_wild_ignore_table

- slave_compressed_protocol
- slave_ddl_exec_mode
- slave_domain_parallel_threads
- slave_exec_mode
- slave_max_allowed_packet
- slave_net_timeout
- slave_parallel_max_queued
- slave_parallel_mode
- slave_parallel_threads
- slave_parallel_workers
- slave_run_triggers_for_rbr
- slave_sql_verify_checksum
- slave_transaction_retry_interval
- slave_type_conversions
- sync_master_info
- sync_relay_log, and
- sync_relay_log_info.

Added in MariaDB 10.5.2.

SET USER

Enables setting the `DEFINER` when creating triggers, views, stored functions and stored procedures. Added in MariaDB 10.5.2.

SHOW DATABASES

List all databases using the SHOW DATABASES statement. Without the `SHOW DATABASES` privilege, you can still issue the `SHOW DATABASES` statement, but it will only list databases containing tables on which you have privileges.

SHUTDOWN

Shut down the server using SHUTDOWN or the mariadb-admin shutdown command.

SUPER

Execute superuser statements: CHANGE MASTER TO, KILL (users who do not have this privilege can only `KILL` their own threads), PURGE LOGS, SET global system variables, or the mariadb-admin debug command. Also, this permission allows the user to write data even if the read_only startup option is set, enable or disable logging, enable or disable replication on replica, specify a `DEFINER` for statements that support that clause, connect once reaching the `MAX_CONNECTIONS`. If a statement has been specified for the init-connect `mysqld` option, that command will not be executed when a user with `SUPER` privileges connects to the server.

The SUPER privilege has been split into multiple smaller privileges from MariaDB 10.5.2 to allow for more fine-grained privileges (MDEV-21743). The privileges are:

- SET USER
- FEDERATED ADMIN
- CONNECTION ADMIN
- REPLICATION SLAVE ADMIN
- BINLOG ADMIN
- BINLOG REPLAY
- REPLICA MONITOR
- BINLOG MONITOR
- REPLICATION MASTER ADMIN
- READ_ONLY ADMIN

However, the smaller privileges are still a part of the SUPER grant in MariaDB 10.5.2. From MariaDB 11.0.1 onwards, these grants are no longer a part of SUPER and need to be granted separately (MDEV-29668).

From MariaDB 10.11.0, the READ_ONLY ADMIN privilege has been removed from `SUPER`. The benefit of this is that one can remove the READ_ONLY ADMIN privilege from all users and ensure that no one can make any changes on any non-temporary tables. This is useful on replicas when one wants to ensure that the replica is kept identical to the primary (MDEV-29596).

Database Privileges

The following table lists the privileges that can be granted at the database level. You can also grant all table and function privileges at the database level. Table and function privileges on a database apply to all tables or functions in that database,

including those created later.

To set a privilege for a database, specify the database using `db_name.*` for *priv_level*, or just use `*` to specify the default database.

Privilege	Description
`CREATE`	Create a database using the CREATE DATABASE statement, when the privilege is granted for a database. You can grant the `CREATE` privilege on databases that do not yet exist. This also grants the `CREATE` privilege on all tables in the database.
`CREATE ROUTINE`	Create Stored Programs using the CREATE PROCEDURE and CREATE FUNCTION statements.
`CREATE TEMPORARY TABLES`	Create temporary tables with the CREATE TEMPORARY TABLE statement. This privilege enable writing and dropping those temporary tables
`DROP`	Drop a database using the DROP DATABASE statement, when the privilege is granted for a database. This also grants the `DROP` privilege on all tables in the database.
`EVENT`	Create, drop and alter `EVENT` s.
`GRANT OPTION`	Grant database privileges. You can only grant privileges that you have.
`LOCK TABLES`	Acquire explicit locks using the LOCK TABLES statement; you also need to have the `SELECT` privilege on a table, in order to lock it.
`SHOW CREATE ROUTINE`	Permit viewing the `SHOW CREATE` definition statement of a routine, for example SHOW CREATE FUNCTION, even if not the routine owner. From MariaDB 11.3.0.

Table Privileges

Privilege	Description
`ALTER`	Change the structure of an existing table using the ALTER TABLE statement.
`CREATE`	Create a table using the CREATE TABLE statement. You can grant the `CREATE` privilege on tables that do not yet exist.
`CREATE VIEW`	Create a view using the CREATE_VIEW statement.
`DELETE`	Remove rows from a table using the DELETE statement.
`DELETE HISTORY`	Remove historical rows from a table using the DELETE HISTORY statement. Displays as `DELETE VERSIONING ROWS` when running SHOW GRANTS until MariaDB 10.3.15 and until MariaDB 10.4.5 (MDEV-17655), or when running SHOW PRIVILEGES until MariaDB 10.5.2, MariaDB 10.4.13 and MariaDB 10.3.23 (MDEV-20382). From MariaDB 10.3.4. From MariaDB 10.3.5, if a user has the `SUPER` privilege but not this privilege, running mariadb-upgrade will grant this privilege as well.
`DROP`	Drop a table using the DROP TABLE statement or a view using the DROP VIEW statement. Also required to execute the TRUNCATE TABLE statement.
`GRANT OPTION`	Grant table privileges. You can only grant privileges that you have.
`INDEX`	Create an index on a table using the CREATE INDEX statement. Without the `INDEX` privilege, you can still create indexes when creating a table using the CREATE TABLE statement if the you have the `CREATE` privilege, and you can create indexes using the ALTER TABLE statement if you have the `ALTER` privilege.
`INSERT`	Add rows to a table using the INSERT statement. The `INSERT` privilege can also be set on individual columns; see Column Privileges below for details.
`REFERENCES`	Unused.
`SELECT`	Read data from a table using the SELECT statement. The `SELECT` privilege can also be set on individual columns; see Column Privileges below for details.
`SHOW VIEW`	Show the CREATE VIEW statement to create a view using the SHOW CREATE VIEW statement.

`TRIGGER`	Required to run the CREATE TRIGGER, DROP TRIGGER, and SHOW CREATE TRIGGER statements. When another user activates a trigger (running INSERT, UPDATE, or DELETE statements on the associated table), for the trigger to execute, the user that defined the trigger should have the TRIGGER privilege for the table. The user running the INSERT, UPDATE, or DELETE statements on the table is not required to have the TRIGGER privilege.
`UPDATE`	Update existing rows in a table using the UPDATE statement. `UPDATE` statements usually include a `WHERE` clause to update only certain rows. You must have `SELECT` privileges on the table or the appropriate columns for the `WHERE` clause. The `UPDATE` privilege can also be set on individual columns; see Column Privileges below for details.

Column Privileges

Some table privileges can be set for individual columns of a table. To use column privileges, specify the table explicitly and provide a list of column names after the privilege type. For example, the following statement would allow the user to read the names and positions of employees, but not other information from the same table, such as salaries.

```
GRANT SELECT (name, position) on Employee to 'jeffrey'@'localhost';
```

Privilege	Description
`INSERT (column_list)`	Add rows specifying values in columns using the INSERT statement. If you only have column-level `INSERT` privileges, you must specify the columns you are setting in the `INSERT` statement. All other columns will be set to their default values, or `NULL`.
`REFERENCES (column_list)`	Unused.
`SELECT (column_list)`	Read values in columns using the SELECT statement. You cannot access or query any columns for which you do not have `SELECT` privileges, including in `WHERE`, `ON`, `GROUP BY`, and `ORDER BY` clauses.
`UPDATE (column_list)`	Update values in columns of existing rows using the UPDATE statement. `UPDATE` statements usually include a `WHERE` clause to update only certain rows. You must have `SELECT` privileges on the table or the appropriate columns for the `WHERE` clause.

Function Privileges

Privilege	Description
`ALTER ROUTINE`	Change the characteristics of a stored function using the ALTER FUNCTION statement.
`EXECUTE`	Use a stored function. You need `SELECT` privileges for any tables or columns accessed by the function.
`GRANT OPTION`	Grant function privileges. You can only grant privileges that you have.

Procedure Privileges

Privilege	Description
`ALTER ROUTINE`	Change the characteristics of a stored procedure using the ALTER PROCEDURE statement.
`EXECUTE`	Execute a stored procedure using the CALL statement. The privilege to call a procedure may allow you to perform actions you wouldn't otherwise be able to do, such as insert rows into a table.
`GRANT OPTION`	Grant procedure privileges. You can only grant privileges that you have.

```
GRANT EXECUTE ON PROCEDURE mysql.create_db TO maintainer;
```

Proxy Privileges

Privilege	Description

PROXY	Permits one user to be a proxy for another.

The PROXY privilege allows one user to proxy as another user, which means their privileges change to that of the proxy user, and the CURRENT_USER() function returns the user name of the proxy user.

The PROXY privilege only works with authentication plugins that support it. The default mysql_native_password authentication plugin does not support proxy users.

The pam authentication plugin is the only plugin included with MariaDB that currently supports proxy users. The PROXY privilege is commonly used with the pam authentication plugin to enable user and group mapping with PAM.

For example, to grant the PROXY privilege to an anonymous account that authenticates with the pam authentication plugin, you could execute the following:

```
CREATE USER 'dba'@'%' IDENTIFIED BY 'strongpassword';
GRANT ALL PRIVILEGES ON *.* TO 'dba'@'%' ;

CREATE USER ''@'%' IDENTIFIED VIA pam USING 'mariadb';
GRANT PROXY ON 'dba'@'%' TO ''@'%';
```

A user account can only grant the PROXY privilege for a specific user account if the granter also has the PROXY privilege for that specific user account, and if that privilege is defined WITH GRANT OPTION. For example, the following example fails because the granter does not have the PROXY privilege for that specific user account at all:

```
SELECT USER(), CURRENT_USER();
+-----------------+-----------------+
| USER()          | CURRENT_USER()  |
+-----------------+-----------------+
| alice@localhost | alice@localhost |
+-----------------+-----------------+

SHOW GRANTS;
+-------------------------------------------------------------------------------------------------
--------------------------+
| Grants for alice@localhost
|
+-------------------------------------------------------------------------------------------------
--------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'alice'@'localhost' IDENTIFIED BY PASSWORD
'*2470C0C06DEE42FD1618BB99005ADCA2EC9D1E19' |
+-------------------------------------------------------------------------------------------------
--------------------------+

GRANT PROXY ON 'dba'@'localhost' TO 'bob'@'localhost';
ERROR 1698 (28000): Access denied for user 'alice'@'localhost'
```

And the following example fails because the granter does have the PROXY privilege for that specific user account, but it is not defined WITH GRANT OPTION:

```
SELECT USER(), CURRENT_USER();
+----------------+----------------+
| USER()         | CURRENT_USER() |
+----------------+----------------+
| alice@localhost | alice@localhost |
+----------------+----------------+

SHOW GRANTS;
+-----------------------------------------------------------------------------------------------
--------------------------+
| Grants for alice@localhost
|
+-----------------------------------------------------------------------------------------------
--------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'alice'@'localhost' IDENTIFIED BY PASSWORD
'*2470C0C06DEE42FD1618BB99005ADCA2EC9D1E19' |
| GRANT PROXY ON 'dba'@'localhost' TO 'alice'@'localhost'
|
+-----------------------------------------------------------------------------------------------
--------------------------+

GRANT PROXY ON 'dba'@'localhost' TO 'bob'@'localhost';
ERROR 1698 (28000): Access denied for user 'alice'@'localhost'
```

But the following example succeeds because the granter does have the `PROXY` privilege for that specific user account, and it is defined `WITH GRANT OPTION`:

```
SELECT USER(), CURRENT_USER();
+----------------+----------------+
| USER()         | CURRENT_USER() |
+----------------+----------------+
| alice@localhost | alice@localhost |
+----------------+----------------+

SHOW GRANTS;
+-----------------------------------------------------------------------------------------------
-------------------------------------------+
| Grants for alice@localhost
|
+-----------------------------------------------------------------------------------------------
-------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'alice'@'localhost' IDENTIFIED BY PASSWORD
'*2470C0C06DEE42FD1618BB99005ADCA2EC9D1E19' WITH GRANT OPTION |
| GRANT PROXY ON 'dba'@'localhost' TO 'alice'@'localhost' WITH GRANT OPTION
|
+-----------------------------------------------------------------------------------------------
-------------------------------------------+

GRANT PROXY ON 'dba'@'localhost' TO 'bob'@'localhost';
```

A user account can grant the `PROXY` privilege for any other user account if the granter has the `PROXY` privilege for the `''@'%'` anonymous user account, like this:

```
GRANT PROXY ON ''@'%' TO 'dba'@'localhost' WITH GRANT OPTION;
```

For example, the following example succeeds because the user can grant the `PROXY` privilege for any other user account:

```
SELECT USER(), CURRENT_USER();
+----------------+----------------+
| USER()         | CURRENT_USER() |
+----------------+----------------+
| alice@localhost | alice@localhost |
+----------------+----------------+

SHOW GRANTS;
+-------------------------------------------------------------------------------------------------
-----------------------------------------+
| Grants for alice@localhost
|
+-------------------------------------------------------------------------------------------------
-----------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'alice'@'localhost' IDENTIFIED BY PASSWORD
'*2470C0C06DEE42FD1618BB99005ADCA2EC9D1E19' WITH GRANT OPTION |
| GRANT PROXY ON ''@'%' TO 'alice'@'localhost' WITH GRANT OPTION
|
+-------------------------------------------------------------------------------------------------
-----------------------------------------+

GRANT PROXY ON 'app1_dba'@'localhost' TO 'bob'@'localhost';
Query OK, 0 rows affected (0.004 sec)

GRANT PROXY ON 'app2_dba'@'localhost' TO 'carol'@'localhost';
Query OK, 0 rows affected (0.004 sec)
```

The default `root` user accounts created by mariadb-install-db have this privilege. For example:

```
GRANT ALL PRIVILEGES ON *.* TO 'root'@'localhost' WITH GRANT OPTION;
GRANT PROXY ON ''@'%' TO 'root'@'localhost' WITH GRANT OPTION;
```

This allows the default `root` user accounts to grant the `PROXY` privilege for any other user account, and it also allows the default `root` user accounts to grant others the privilege to do the same.

Authentication Options

The authentication options for the `GRANT` statement are the same as those for the CREATE USER statement.

IDENTIFIED BY 'password'

The optional `IDENTIFIED BY` clause can be used to provide an account with a password. The password should be specified in plain text. It will be hashed by the PASSWORD function prior to being stored.

For example, if our password is `mariadb` , then we can create the user with:

```
GRANT USAGE ON *.* TO foo2@test IDENTIFIED BY 'mariadb';
```

If you do not specify a password with the `IDENTIFIED BY` clause, the user will be able to connect without a password. A blank password is not a wildcard to match any password. The user must connect without providing a password if no password is set.

If the user account already exists and if you provide the `IDENTIFIED BY` clause, then the user's password will be changed. You must have the privileges needed for the SET PASSWORD statement to change a user's password with `GRANT` .

The only authentication plugins that this clause supports are mysql_native_password and mysql_old_password.

IDENTIFIED BY PASSWORD 'password_hash'

The optional `IDENTIFIED BY PASSWORD` clause can be used to provide an account with a password that has already been hashed. The password should be specified as a hash that was provided by the PASSWORD function. It will be stored as-is.

For example, if our password is `mariadb` , then we can find the hash with:

```
SELECT PASSWORD('mariadb');
+-------------------------------------------+
| PASSWORD('mariadb')                       |
+-------------------------------------------+
| *54958E764CE10E50764C2EECBB71D01F08549980 |
+-------------------------------------------+
1 row in set (0.00 sec)
```

And then we can create a user with the hash:

```
GRANT USAGE ON *.* TO foo2@test IDENTIFIED BY
  PASSWORD '*54958E764CE10E50764C2EECBB71D01F08549980';
```

If you do not specify a password with the `IDENTIFIED BY` clause, the user will be able to connect without a password. A blank password is not a wildcard to match any password. The user must connect without providing a password if no password is set.

If the user account already exists and if you provide the `IDENTIFIED BY` clause, then the user's password will be changed. You must have the privileges needed for the SET PASSWORD statement to change a user's password with `GRANT`.

The only authentication plugins that this clause supports are mysql_native_password and mysql_old_password.

IDENTIFIED {VIA|WITH} authentication_plugin

The optional `IDENTIFIED VIA authentication_plugin` allows you to specify that the account should be authenticated by a specific authentication plugin. The plugin name must be an active authentication plugin as per SHOW PLUGINS. If it doesn't show up in that output, then you will need to install it with INSTALL PLUGIN or INSTALL SONAME.

For example, this could be used with the PAM authentication plugin:

```
GRANT USAGE ON *.* TO foo2@test IDENTIFIED VIA pam;
```

Some authentication plugins allow additional arguments to be specified after a `USING` or `AS` keyword. For example, the PAM authentication plugin accepts a service name:

```
GRANT USAGE ON *.* TO foo2@test IDENTIFIED VIA pam USING 'mariadb';
```

The exact meaning of the additional argument would depend on the specific authentication plugin.

MariaDB starting with 10.4.0

The `USING` or `AS` keyword can also be used to provide a plain-text password to a plugin if it's provided as an argument to the PASSWORD() function. This is only valid for authentication plugins that have implemented a hook for the PASSWORD() function. For example, the ed25519 authentication plugin supports this:

```
CREATE USER safe@'%' IDENTIFIED VIA ed25519
  USING PASSWORD('secret');
```

MariaDB starting with 10.4.3

One can specify many authentication plugins, they all work as alternatives ways of authenticating a user:

```
CREATE USER safe@'%' IDENTIFIED VIA ed25519
  USING PASSWORD('secret') OR unix_socket;
```

By default, when you create a user without specifying an authentication plugin, MariaDB uses the mysql_native_password plugin.

Resource Limit Options

It is possible to set per-account limits for certain server resources. The following table shows the values that can be set per account:

Limit Type	Decription
`MAX_QUERIES_PER_HOUR`	Number of statements that the account can issue per hour (including updates)

MAX_UPDATES_PER_HOUR	Number of updates (not queries) that the account can issue per hour
MAX_CONNECTIONS_PER_HOUR	Number of connections that the account can start per hour
MAX_USER_CONNECTIONS	Number of simultaneous connections that can be accepted from the same account; if it is 0, max_connections will be used instead; if max_connections is 0, there is no limit for this account's simultaneous connections.
MAX_STATEMENT_TIME	Timeout, in seconds, for statements executed by the user. See also Aborting Statements that Exceed a Certain Time to Execute.

If any of these limits are set to 0, then there is no limit for that resource for that user.

To set resource limits for an account, if you do not want to change that account's privileges, you can issue a GRANT statement with the USAGE privilege, which has no meaning. The statement can name some or all limit types, in any order.

Here is an example showing how to set resource limits:

```
GRANT USAGE ON *.* TO 'someone'@'localhost' WITH
    MAX_USER_CONNECTIONS 0
    MAX_QUERIES_PER_HOUR 200;
```

The resources are tracked per account, which means 'user'@'server'; not per user name or per connection.

The count can be reset for all users using FLUSH USER_RESOURCES, FLUSH PRIVILEGES or mariadb-admin reload.

Users with the CONNECTION ADMIN privilege (in MariaDB 10.5.2 and later) or the SUPER privilege are not restricted by max_user_connections, max_connections, or max_password_errors.

Per account resource limits are stored in the user table, in the mysql database. Columns used for resources limits are named max_questions, max_updates, max_connections (for MAX_CONNECTIONS_PER_HOUR), and max_user_connections (for MAX_USER_CONNECTIONS).

TLS Options

By default, MariaDB transmits data between the server and clients without encrypting it. This is generally acceptable when the server and client run on the same host or in networks where security is guaranteed through other means. However, in cases where the server and client exist on separate networks or they are in a high-risk network, the lack of encryption does introduce security concerns as a malicious actor could potentially eavesdrop on the traffic as it is sent over the network between them.

To mitigate this concern, MariaDB allows you to encrypt data in transit between the server and clients using the Transport Layer Security (TLS) protocol. TLS was formerly known as Secure Socket Layer (SSL), but strictly speaking the SSL protocol is a predecessor to TLS and, that version of the protocol is now considered insecure. The documentation still uses the term SSL often and for compatibility reasons TLS-related server system and status variables still use the prefix ssl_, but internally, MariaDB only supports its secure successors.

See Secure Connections Overview for more information about how to determine whether your MariaDB server has TLS support.

You can set certain TLS-related restrictions for specific user accounts. For instance, you might use this with user accounts that require access to sensitive data while sending it across networks that you do not control. These restrictions can be enabled for a user account with the CREATE USER, ALTER USER, or GRANT statements. The following options are available:

Option	Description
REQUIRE NONE	TLS is not required for this account, but can still be used.
REQUIRE SSL	The account must use TLS, but no valid X509 certificate is required. This option cannot be combined with other TLS options.
REQUIRE X509	The account must use TLS and must have a valid X509 certificate. This option implies REQUIRE SSL. This option cannot be combined with other TLS options.
REQUIRE ISSUER 'issuer'	The account must use TLS and must have a valid X509 certificate. Also, the Certificate Authority must be the one specified via the string issuer. This option implies REQUIRE X509. This option can be combined with the SUBJECT, and CIPHER options in any order.

`REQUIRE SUBJECT 'subject'`	The account must use TLS and must have a valid X509 certificate. Also, the certificate's Subject must be the one specified via the string `subject`. This option implies `REQUIRE X509`. This option can be combined with the `ISSUER`, and `CIPHER` options in any order.
`REQUIRE CIPHER 'cipher'`	The account must use TLS, but no valid X509 certificate is required. Also, the encryption used for the connection must use a specific cipher method specified in the string `cipher`. This option implies `REQUIRE SSL`. This option can be combined with the `ISSUER`, and `SUBJECT` options in any order.

The `REQUIRE` keyword must be used only once for all specified options, and the `AND` keyword can be used to separate individual options, but it is not required.

For example, you can create a user account that requires these TLS options with the following:

```
GRANT USAGE ON *.* TO 'alice'@'%'
  REQUIRE SUBJECT '/CN=alice/O=My Dom, Inc./C=US/ST=Oregon/L=Portland'
  AND ISSUER '/C=FI/ST=Somewhere/L=City/ O=Some Company/CN=Peter
Parker/emailAddress=p.parker@marvel.com'
  AND CIPHER 'SHA-DES-CBC3-EDH-RSA';
```

If any of these options are set for a specific user account, then any client who tries to connect with that user account will have to be configured to connect with TLS.

See Securing Connections for Client and Server for information on how to enable TLS on the client and server.

Roles

Syntax

```
GRANT role TO grantee [, grantee ... ]
[ WITH ADMIN OPTION ]

grantee:
    rolename
    username [authentication_option]
```

The GRANT statement is also used to grant the use of a role to one or more users or other roles. In order to be able to grant a role, the grantor doing so must have permission to do so (see WITH ADMIN in the CREATE ROLE article).

Specifying the `WITH ADMIN OPTION` permits the grantee to in turn grant the role to another.

For example, the following commands show how to grant the same role to a couple different users.

```
GRANT journalist TO hulda;

GRANT journalist TO berengar WITH ADMIN OPTION;
```

If a user has been granted a role, they do not automatically obtain all permissions associated with that role. These permissions are only in use when the user activates the role with the SET ROLE statement.

TO PUBLIC

MariaDB starting with 10.11

Syntax

```
GRANT <privilege> ON <database>.<object> TO PUBLIC;
REVOKE <privilege> ON <database>.<object> FROM PUBLIC;
```

GRANT ... TO PUBLIC grants privileges to all users with access to the server. The privileges also apply to users created after the privileges are granted. This can be useful when one only wants to state once that all users need to have a certain set of privileges.

When running SHOW GRANTS, a user will also see all privileges inherited from PUBLIC. SHOW GRANTS FOR PUBLIC will only show TO PUBLIC grants.

Grant Examples

Granting Root-like Privileges

You can create a user that has privileges similar to the default `root` accounts by executing the following:

```
CREATE USER 'alexander'@'localhost';
GRANT ALL PRIVILEGES ON  *.* to 'alexander'@'localhost' WITH GRANT OPTION;
```

1.1.1.1.5 RENAME USER

Syntax

```
RENAME USER old_user TO new_user
    [, old_user TO new_user] ...
```

Description

The RENAME USER statement renames existing MariaDB accounts. To use it, you must have the global CREATE USER privilege or the `UPDATE` privilege for the `mysql` database. Each account is named using the same format as for the CREATE USER statement; for example, `'jeffrey'@'localhost'` . If you specify only the user name part of the account name, a host name part of `'%'` is used.

If any of the old user accounts do not exist or any of the new user accounts already exist, `ERROR 1396 (HY000)` results. If an error occurs, `RENAME USER` will still rename the accounts that do not result in an error.

For modifying an existing account, see ALTER USER.

Examples

```
CREATE USER 'donald', 'mickey';
RENAME USER 'donald' TO 'duck'@'localhost', 'mickey' TO 'mouse'@'localhost';
```

Renaming the host component of a user

```
RENAME USER 'foo'@'1.2.3.4' TO 'foo'@'10.20.30.40';
```

1.1.1.1.6 REVOKE

Contents

Privileges

Syntax

```
REVOKE
    priv_type [(column_list)]
      [, priv_type [(column_list)]] ...
    ON [object_type] priv_level
    FROM user [, user] ...

REVOKE ALL PRIVILEGES, GRANT OPTION
    FROM user [, user] ...
```

Description

The `REVOKE` statement enables system administrators to revoke privileges (or roles - see section below) from MariaDB accounts. Each account is named using the same format as for the `GRANT` statement; for example, '`jeffrey'@'localhost`'. If you specify only the user name part of the account name, a host name part of '`%`' is used. For details on the levels at which privileges exist, the allowable `priv_type` and `priv_level` values, and the syntax for specifying users and passwords, see GRANT.

To use the first `REVOKE` syntax, you must have the `GRANT OPTION` privilege, and you must have the privileges that you are revoking.

To revoke all privileges, use the second syntax, which drops all global, database, table, column, and routine privileges for the named user or users:

```
REVOKE ALL PRIVILEGES, GRANT OPTION FROM user [, user] ...
```

To use this `REVOKE` syntax, you must have the global CREATE USER privilege or the UPDATE privilege for the mysql database. See GRANT.

Examples

```
REVOKE SUPER ON *.* FROM 'alexander'@'localhost';
```

Roles

Syntax

```
REVOKE role  [, role ...]
    FROM grantee [, grantee2 ... ]

REVOKE ADMIN OPTION FOR role FROM grantee [, grantee2]
```

Description

`REVOKE` is also used to remove a role from a user or another role that it's previously been assigned to. If a role has previously been set as a default role, `REVOKE` does not remove the record of the default role from the mysql.user table. If the role is subsequently granted again, it will again be the user's default. Use SET DEFAULT ROLE NONE to explicitly remove this.

Before MariaDB 10.1.13, the `REVOKE role` statement was not permitted in prepared statements.

Example

```
REVOKE journalist FROM hulda
```

1.1.1.1.7 SET PASSWORD

Syntax

```
SET PASSWORD [FOR user] =
    {
        PASSWORD('some password')
      | OLD_PASSWORD('some password')
      | 'encrypted password'
    }
```

Contents

Description

The `SET PASSWORD` statement assigns a password to an existing MariaDB user account.

If the password is specified using the `PASSWORD()` or `OLD_PASSWORD()` function, the literal text of the password should be given. If the password is specified without using either function, the password should be the already-encrypted password value as returned by `PASSWORD()`.

`OLD_PASSWORD()` should only be used if your MariaDB/MySQL clients are very old (< 4.0.0).

With no `FOR` clause, this statement sets the password for the current user. Any client that has connected to the server using a non-anonymous account can change the password for that account.

With a `FOR` clause, this statement sets the password for a specific account on the current server host. Only clients that have the `UPDATE` privilege for the `mysql` database can do this. The user value should be given in `user_name@host_name` format, where `user_name` and `host_name` are exactly as they are listed in the User and Host columns of the `mysql.user` table (or view in MariaDB-10.4 onwards) entry.

The argument to `PASSWORD()` and the password given to MariaDB clients can be of arbitrary length.

Authentication Plugin Support

MariaDB starting with 10.4

In MariaDB 10.4 and later, `SET PASSWORD` (with or without `PASSWORD()`) works for accounts authenticated via any authentication plugin that supports passwords stored in the `mysql.global_priv` table.

The `ed25519`, `mysql_native_password`, and `mysql_old_password` authentication plugins store passwords in the `mysql.global_priv` table.

If you run `SET PASSWORD` on an account that authenticates with one of these authentication plugins that stores passwords in the `mysql.global_priv` table, then the `PASSWORD()` function is evaluated by the specific authentication plugin used by the account. The authentication plugin hashes the password with a method that is compatible with that specific authentication plugin.

The `unix_socket`, `named_pipe`, `gssapi`, and `pam` authentication plugins do **not** store passwords in the `mysql.global_priv` table. These authentication plugins rely on other methods to authenticate the user.

If you attempt to run `SET PASSWORD` on an account that authenticates with one of these authentication plugins that doesn't store a password in the `mysql.global_priv` table, then MariaDB Server will raise a warning like the following:

```
SET PASSWORD is ignored for users authenticating via unix_socket plugin
```

See Authentication from MariaDB 10.4 for an overview of authentication changes in MariaDB 10.4.

MariaDB until 10.3

In MariaDB 10.3 and before, `SET PASSWORD` (with or without `PASSWORD()`) only works for accounts authenticated via `mysql_native_password` or `mysql_old_password` authentication plugins

Passwordless User Accounts

User accounts do not always require passwords to login.

The `unix_socket` , `named_pipe` and `gssapi` authentication plugins do not require a password to authenticate the user.

The `pam` authentication plugin may or may not require a password to authenticate the user, depending on the specific configuration.

The `mysql_native_password` and `mysql_old_password` authentication plugins require passwords for authentication, but the password can be blank. In that case, no password is required.

If you provide a password while attempting to log into the server as an account that doesn't require a password, then MariaDB server will simply ignore the password.

MariaDB starting with 10.4

In MariaDB 10.4 and later, a user account can be defined to use multiple authentication plugins in a specific order of preference. This specific scenario may be more noticeable in these versions, since an account could be associated with some authentication plugins that require a password, and some that do not.

Example

For example, if you had an entry with User and Host column values of ' `bob` ' and ' `%.loc.gov` ', you would write the statement like this:

```
SET PASSWORD FOR 'bob'@'%.loc.gov' = PASSWORD('newpass');
```

If you want to delete a password for a user, you would do:

```
SET PASSWORD FOR 'bob'@localhost = PASSWORD("");
```

1.1.1.1.8 CREATE ROLE

Syntax

```
CREATE [OR REPLACE] ROLE [IF NOT EXISTS] role
  [WITH ADMIN
    {CURRENT_USER | CURRENT_ROLE | user | role}]
```

Contents

Description

The `CREATE ROLE` statement creates one or more MariaDB roles. To use it, you must have the global CREATE USER privilege or the INSERT privilege for the mysql database. For each account, `CREATE ROLE` creates a new row in the mysql.user table that has no privileges, and with the corresponding `is_role` field set to `Y` . It also creates a record in the mysql.roles_mapping table.

If any of the specified roles already exist, `ERROR 1396 (HY000)` results. If an error occurs, `CREATE ROLE` will still create the roles that do not result in an error. The maximum length for a role is 128 characters. Role names can be quoted, as explained in the Identifier names page. Only one error is produced for all roles which have not been created:

```
ERROR 1396 (HY000): Operation CREATE ROLE failed for 'a','b','c'
```

Failed `CREATE` or `DROP` operations, for both users and roles, produce the same error code.

`PUBLIC` and `NONE` are reserved, and cannot be used as role names. `NONE` is used to unset a role and `PUBLIC` has a special use in other systems, such as Oracle, so is reserved for compatibility purposes.

For valid identifiers to use as role names, see Identifier Names.

WITH ADMIN

The optional `WITH ADMIN` clause determines whether the current user, the current role or another user or role has use of the newly created role. If the clause is omitted, `WITH ADMIN CURRENT_USER` is treated as the default, which means that the current user will be able to GRANT this role to users.

OR REPLACE

If the optional `OR REPLACE` clause is used, it acts as a shortcut for:

```
DROP ROLE IF EXISTS name;
CREATE ROLE name ...;
```

IF NOT EXISTS

When the `IF NOT EXISTS` clause is used, MariaDB will return a warning instead of an error if the specified role already exists. Cannot be used together with the `OR REPLACE` clause.

Examples

```
CREATE ROLE journalist;

CREATE ROLE developer WITH ADMIN lorinda@localhost;
```

Granting the role to another user. Only user `lorinda@localhost` has permission to grant the `developer` role:

```
 SELECT USER();
+-------------------+
| USER()            |
+-------------------+
| henning@localhost |
+-------------------+
...
GRANT developer TO ian@localhost;
Access denied for user 'henning'@'localhost'

 SELECT USER();
+-------------------+
| USER()            |
+-------------------+
| lorinda@localhost |
+-------------------+

GRANT m_role TO ian@localhost;
```

The `OR REPLACE` and `IF NOT EXISTS` clauses. The `journalist` role already exists:

```
CREATE ROLE journalist;
ERROR 1396 (HY000): Operation CREATE ROLE failed for 'journalist'

CREATE OR REPLACE ROLE journalist;
Query OK, 0 rows affected (0.00 sec)

CREATE ROLE IF NOT EXISTS journalist;
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

```
SHOW WARNINGS;
+-------+------+-----------------------------------------------------+
| Level | Code | Message                                             |
+-------+------+-----------------------------------------------------+
| Note  | 1975 | Can't create role 'journalist'; it already exists |
+-------+------+-----------------------------------------------------+
```

1.1.1.1.9 DROP ROLE

Syntax

```
DROP ROLE [IF EXISTS] role_name [,role_name ...]
```

Contents

Description

The `DROP ROLE` statement removes one or more MariaDB roles. To use this statement, you must have the global CREATE USER privilege or the DELETE privilege for the mysql database.

`DROP ROLE` does not disable roles for connections which selected them with SET ROLE. If a role has previously been set as a default role, `DROP ROLE` does not remove the record of the default role from the mysql.user table. If the role is subsequently recreated and granted, it will again be the user's default. Use SET DEFAULT ROLE NONE to explicitly remove this.

If any of the specified user accounts do not exist, `ERROR 1396 (HY000)` results. If an error occurs, `DROP ROLE` will still drop the roles that do not result in an error. Only one error is produced for all roles which have not been dropped:

```
ERROR 1396 (HY000): Operation DROP ROLE failed for 'a','b','c'
```

Failed `CREATE` or `DROP` operations, for both users and roles, produce the same error code.

IF EXISTS

If the `IF EXISTS` clause is used, MariaDB will return a warning instead of an error if the role does not exist.

Examples

```
DROP ROLE journalist;
```

The same thing using the optional `IF EXISTS` clause:

```
DROP ROLE journalist;
ERROR 1396 (HY000): Operation DROP ROLE failed for 'journalist'

DROP ROLE IF EXISTS journalist;
Query OK, 0 rows affected, 1 warning (0.00 sec)

Note (Code 1975): Can't drop role 'journalist'; it doesn't exist
```

1.1.1.1.10 SET ROLE

Syntax

```
SET ROLE { role | NONE }
```

Contents

Description

The `SET ROLE` statement enables a role, along with all of its associated permissions, for the current session. To unset a role, use `NONE` .

If a role that doesn't exist, or to which the user has not been assigned, is specified, an `ERROR 1959 (OP000): Invalid role specification` error occurs.

An automatic SET ROLE is implicitly performed when a user connects if that user has been assigned a default role. See SET DEFAULT ROLE.

Example

```
SELECT CURRENT_ROLE;
+--------------+
| CURRENT_ROLE |
+--------------+
| NULL         |
+--------------+

SET ROLE staff;

SELECT CURRENT_ROLE;
+--------------+
| CURRENT_ROLE |
+--------------+
| staff        |
+--------------+

SET ROLE NONE;

SELECT CURRENT_ROLE();
+----------------+
| CURRENT_ROLE() |
+----------------+
| NULL           |
+----------------+
```

1.1.1.1.11 SET DEFAULT ROLE

Contents

Syntax

```
SET DEFAULT ROLE { role | NONE } [ FOR user@host ]
```

Description

The `SET DEFAULT ROLE` statement sets a **default role** for a specified (or current) user. A default role is automatically enabled when a user connects (an implicit SET ROLE statement is executed immediately after a connection is established).

To be able to set a role as a default, the role must already have been granted to that user, and one needs the privileges to enable this role (if you cannot do `SET ROLE X` , you won't be able to do `SET DEFAULT ROLE X`). To set a default role for another user one needs to have write access to the `mysql` database.

To remove a user's default role, use `SET DEFAULT ROLE NONE [ FOR user@host ]` . The record of the default role is not removed if the role is dropped or revoked, so if the role is subsequently re-created or granted, it will again be the user's default role.

The default role is stored in the `default_role` column in the mysql.user table/view, as well as in the Information Schema APPLICABLE_ROLES table, so these can be viewed to see which role has been assigned to a user as the default.

Examples

Setting a default role for the current user:

```
SET DEFAULT ROLE journalist;
```

Removing a default role from the current user:

```
SET DEFAULT ROLE NONE;
```

Setting a default role for another user. The role has to have been granted to the user before it can be set as default:

```
CREATE ROLE journalist;
CREATE USER taniel;

SET DEFAULT ROLE journalist FOR taniel;
ERROR 1959 (OP000): Invalid role specification `journalist`

GRANT journalist TO taniel;
SET DEFAULT ROLE journalist FOR taniel;
```

Viewing mysql.user:

```
select * from mysql.user where user='taniel'\G
*************************** 1. row ***************************
                  Host: %
                  User: taniel
...
               is_role: N
          default_role: journalist
...
```

Removing a default role for another user

```
SET DEFAULT ROLE NONE FOR taniel;
```

1.1.1.1.12 SHOW GRANTS

Contents

Syntax

```
SHOW GRANTS [FOR user|role]
```

Description

The `SHOW GRANTS` statement lists privileges granted to a particular user or role.

Users

The statement lists the GRANT statement or statements that must be issued to duplicate the privileges that are granted to a MariaDB user account. The account is named using the same format as for the `GRANT` statement; for example, '`jeffrey'@'localhost`'. If you specify only the user name part of the account name, a host name part of '`%`' is used. For additional information about specifying account names, see GRANT.

```
SHOW GRANTS FOR 'root'@'localhost';
+---------------------------------------------------------------------+
| Grants for root@localhost                                           |
+---------------------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'root'@'localhost' WITH GRANT OPTION |
+---------------------------------------------------------------------+
```

To list the privileges granted to the account that you are using to connect to the server, you can use any of the following statements:

```
SHOW GRANTS;
SHOW GRANTS FOR CURRENT_USER;
SHOW GRANTS FOR CURRENT_USER();
```

If `SHOW GRANTS FOR CURRENT_USER` (or any of the equivalent syntaxes) is used in `DEFINER` context (such as within a stored procedure that is defined with `SQL SECURITY DEFINER`), the grants displayed are those of the definer and not the invoker.

Note that the `DELETE HISTORY` privilege, introduced in MariaDB 10.3.4, was displayed as `DELETE VERSIONING ROWS` when running `SHOW GRANTS` until MariaDB 10.3.15 (MDEV-17655).

Roles

`SHOW GRANTS` can also be used to view the privileges granted to a role.

Example

```
SHOW GRANTS FOR journalist;
+-----------------------------------------+
| Grants for journalist                   |
+-----------------------------------------+
| GRANT USAGE ON *.* TO 'journalist'      |
| GRANT DELETE ON `test`.* TO 'journalist' |
+-----------------------------------------+
```

FOR PUBLIC

MariaDB starting with 10.11

GRANT ... TO PUBLIC was introduced in MariaDB 10.11 to grant privileges to all users. `SHOW GRANTS FOR PUBLIC` shows all these grants.

```
SHOW GRANTS FOR public;
+-------------------------------------------------+
| Grants for PUBLIC                               |
+-------------------------------------------------+
| GRANT ALL PRIVILEGES ON `dev_db`.* TO `PUBLIC` |
+-------------------------------------------------+
```

1.1.1.1.13 SHOW CREATE USER

Syntax

```
SHOW CREATE USER [user]
```

Description

Shows the CREATE USER statement that created the given user. The statement requires the SELECT privilege for the mysql database, except for the current user. The CREATE USER statement for the current user is shown where no user is specified.

Examples

```
CREATE USER foo4@test require cipher 'text'
  issuer 'foo_issuer' subject 'foo_subject';

SHOW CREATE USER foo4@test\G
*************************** 1. row ***************************
CREATE USER 'foo4'@'test'
  REQUIRE ISSUER 'foo_issuer'
  SUBJECT 'foo_subject'
  CIPHER 'text'
```

User Password Expiry:

```
CREATE USER 'monty'@'localhost' PASSWORD EXPIRE INTERVAL 120 DAY;

SHOW CREATE USER 'monty'@'localhost';
+-----------------------------------------------------------------------+
| CREATE USER for monty@localhost                                       |
+-----------------------------------------------------------------------+
| CREATE USER 'monty'@'localhost' PASSWORD EXPIRE INTERVAL 120 DAY |
+-----------------------------------------------------------------------+
```

1.1.1.2 Administrative SQL Statements

SQL statements for administering MariaDB.

Table Statements

Documentation on creating, altering, analyzing and maintaining tables.

ANALYZE and EXPLAIN Statements

Articles on the ANALYZE and EXPLAIN statements

BACKUP Commands

Commands used by backup tools.

FLUSH Commands

Commands to flush or reset various caches in MariaDB.

Replication Commands

List of replication-related commands.

Plugin SQL Statements

List of SQL statements related to plugins.

SET Commands

The SET commands

SHOW

Articles on the various SHOW commands.

System Tables

BINLOG

Internal statement generated by mariadb-binlog.

PURGE BINARY LOGS

PURGE BINARY LOGS removes all binary logs from the server, prior to the provided date or log file.

CACHE INDEX

Caches MyISAM or Aria indexes

DESCRIBE

Information about columns in a table.

EXECUTE Statement

Executes a previously PREPAREd statement

HELP Command

The HELP command will retrieve syntax and help within the mariadb client.

KILL [CONNECTION | QUERY]

Kill connection by query or thread id.

LOAD INDEX

Loads one or more indexes from one or more MyISAM/Aria tables into a key buffer.

RESET

Overall description of the different RESET commands

SHUTDOWN

Shuts down the server.

USE [DATABASE]

Set the current default database.

There are 2 related questions.

1.1.1.2.1 Table Statements

Articles about creating, modifying, and maintaining tables in MariaDB.

ALTER

The various ALTER statements in MariaDB.

ANALYZE TABLE

Store key distributions for a table.

CHECK TABLE

Check table for errors.

CHECK VIEW

Check whether the view algorithm is correct.

CHECKSUM TABLE

Report a table checksum.

CREATE TABLE

Creates a new table.

DELETE

Delete rows from one or more tables.

DROP TABLE

Removes definition and data from one or more tables.

Installing System Tables (mariadb-install-db)

Using mariadb-install-db to create the system tables in the 'mysql' database directory.

mysqlcheck

Symlink or old name for mariadb-check.

mysql_upgrade

Symlink or old name for mariadb-upgrade.

OPTIMIZE TABLE

Reclaim unused space and defragment data.

RENAME TABLE

Change a table's name.

REPAIR TABLE

Repairs a table, if the storage engine supports this statement.

REPAIR VIEW

Fix view if the algorithms are swapped.

REPLACE

Equivalent to DELETE + INSERT, or just an INSERT if no rows are returned.

SHOW COLUMNS

Column information.

SHOW CREATE TABLE

Shows the CREATE TABLE statement that created the table.

SHOW INDEX

Information about table indexes.

TRUNCATE TABLE

DROP and re-CREATE a table.

UPDATE

Modify rows in one or more tables.

Obsolete Table Commands

Table commands that have been removed from MariaDB

IGNORE

Suppress errors while trying to violate a UNIQUE constraint.

System-Versioned Tables

System-versioned tables record the history of all changes to table data.

There are 2 related questions.

1.1.1.2.1.1 ALTER

This category is for documentation on the various ALTER statements.

ALTER TABLE

Modify a table's definition.

ALTER DATABASE

Change the overall characteristics of a database.

ALTER EVENT

Change an existing event.

ALTER FUNCTION

Change the characteristics of a stored function.

ALTER LOGFILE GROUP

Only useful with MySQL Cluster, and has no effect in MariaDB.

ALTER PROCEDURE

Change stored procedure characteristics.

ALTER SEQUENCE

Change options for a SEQUENCE.

ALTER SERVER

Updates mysql.servers table.

ALTER TABLESPACE

ALTER TABLESPACE is not available in MariaDB.

ALTER USER

Modify an existing MariaDB account.

ALTER VIEW

Change a view definition.

There are 1 related questions.

1.1.1.2.1.1.1 ALTER TABLE

Syntax

```
ALTER [ONLINE] [IGNORE] TABLE [IF EXISTS] tbl_name
    [WAIT n | NOWAIT]
    alter_specification [, alter_specification] ...

alter_specification:
    table_option ...
  | ADD [COLUMN] [IF NOT EXISTS] col_name column_definition
        [FIRST | AFTER col_name ]
  | ADD [COLUMN] [IF NOT EXISTS] (col_name column_definition,...)
  | ADD {INDEX|KEY} [IF NOT EXISTS] [index_name]
        [index_type] (index_col_name,...) [index_option] ...
  | ADD [CONSTRAINT [symbol]] PRIMARY KEY
        [index_type] (index_col_name,...) [index_option] ...
  | ADD [CONSTRAINT [symbol]]
        UNIQUE [INDEX|KEY] [index_name]
        [index_type] (index_col_name,...) [index_option] ...
  | ADD FULLTEXT [INDEX|KEY] [index_name]
        (index_col_name,...) [index_option] ...
  | ADD SPATIAL [INDEX|KEY] [index_name]
        (index_col_name,...) [index_option] ...
  | ADD [CONSTRAINT [symbol]]
        FOREIGN KEY [IF NOT EXISTS] [index_name] (index_col_name,...)
        reference_definition
  | ADD PERIOD FOR SYSTEM_TIME (start_column_name, end_column_name)
  | ALTER [COLUMN] col_name SET DEFAULT literal | (expression)
  | ALTER [COLUMN] col_name DROP DEFAULT
  | ALTER {INDEX|KEY} index_name [NOT] INVISIBLE
  | CHANGE [COLUMN] [IF EXISTS] old_col_name new_col_name column_definition
        [FIRST|AFTER col_name]
  | MODIFY [COLUMN] [IF EXISTS] col_name column_definition
        [FIRST | AFTER col_name]
  | DROP [COLUMN] [IF EXISTS] col_name [RESTRICT|CASCADE]
  | DROP PRIMARY KEY
  | DROP {INDEX|KEY} [IF EXISTS] index_name
  | DROP FOREIGN KEY [IF EXISTS] fk_symbol
```

```
  | DROP CONSTRAINT [IF EXISTS] constraint_name
  | DISABLE KEYS
  | ENABLE KEYS
  | RENAME [TO] new_tbl_name
  | ORDER BY col_name [, col_name] ...
  | RENAME COLUMN old_col_name TO new_col_name
  | RENAME {INDEX|KEY} old_index_name TO new_index_name
  | CONVERT TO CHARACTER SET charset_name [COLLATE collation_name]
  | [DEFAULT] CHARACTER SET [=] charset_name
  | [DEFAULT] COLLATE [=] collation_name
  | DISCARD TABLESPACE
  | IMPORT TABLESPACE
  | ALGORITHM [=] {DEFAULT|INPLACE|COPY|NOCOPY|INSTANT}
  | LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}
  | FORCE
  | partition_options
  | CONVERT TABLE normal_table TO partition_definition
  | CONVERT PARTITION partition_name TO TABLE tbl_name
  | ADD PARTITION [IF NOT EXISTS] (partition_definition)
  | DROP PARTITION [IF EXISTS] partition_names
  | TRUNCATE PARTITION partition_names
  | COALESCE PARTITION number
  | REORGANIZE PARTITION [partition_names INTO (partition_definitions)]
  | ANALYZE PARTITION partition_names
  | CHECK PARTITION partition_names
  | OPTIMIZE PARTITION partition_names
  | REBUILD PARTITION partition_names
  | REPAIR PARTITION partition_names
  | EXCHANGE PARTITION partition_name WITH TABLE tbl_name
  | REMOVE PARTITIONING
  | ADD SYSTEM VERSIONING
  | DROP SYSTEM VERSIONING

index_col_name:
    col_name [(length)] [ASC | DESC]

index_type:
    USING {BTREE | HASH | RTREE}

index_option:
    [ KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'
  | CLUSTERING={YES| NO} ]
  [ IGNORED | NOT IGNORED ]

table_options:
    table_option [[,] table_option] ...
```

Contents

Description

`ALTER TABLE` enables you to change the structure of an existing table. For example, you can add or delete columns, create or destroy indexes, change the type of existing columns, or rename columns or the table itself. You can also change the comment for the table and the storage engine of the table.

If another connection is using the table, a metadata lock is active, and this statement will wait until the lock is released. This is also true for non-transactional tables.

When adding a `UNIQUE` index on a column (or a set of columns) which have duplicated values, an error will be produced and the statement will be stopped. To suppress the error and force the creation of `UNIQUE` indexes, discarding duplicates, the IGNORE option can be specified. This can be useful if a column (or a set of columns) should be UNIQUE but it contains duplicate values; however, this technique provides no control on which rows are preserved and which are deleted. Also, note that `IGNORE` is accepted but ignored in `ALTER TABLE ... EXCHANGE PARTITION` statements.

This statement can also be used to rename a table. For details see RENAME TABLE.

When an index is created, the storage engine may use a configurable buffer in the process. Incrementing the buffer speeds up the index creation. Aria and MyISAM allocate a buffer whose size is defined by aria_sort_buffer_size or myisam_sort_buffer_size, also used for REPAIR TABLE. InnoDB allocates three buffers whose size is defined by innodb_sort_buffer_size.

Privileges

Executing the `ALTER TABLE` statement generally requires at least the ALTER privilege for the table or the database..

If you are renaming a table, then it also requires the DROP, CREATE and INSERT privileges for the table or the database as well.

Online DDL

Online DDL is supported with the ALGORITHM and LOCK clauses.

See InnoDB Online DDL Overview for more information on online DDL with InnoDB.

ALTER ONLINE TABLE

ALTER ONLINE TABLE also works for partitioned tables.

Online `ALTER TABLE` is available by executing the following:

```
ALTER ONLINE TABLE ...;
```

This statement has the following semantics:

This statement is equivalent to the following:

```
ALTER TABLE ... LOCK=NONE;
```

See the LOCK alter specification for more information.

WAIT/NOWAIT

Set the lock wait timeout. See WAIT and NOWAIT.

IF EXISTS

The `IF EXISTS` and `IF NOT EXISTS` clauses are available for the following:

```
ADD COLUMN       [IF NOT EXISTS]
ADD INDEX        [IF NOT EXISTS]
ADD FOREIGN KEY  [IF NOT EXISTS]
ADD PARTITION    [IF NOT EXISTS]
CREATE INDEX     [IF NOT EXISTS]

DROP COLUMN      [IF EXISTS]
DROP INDEX       [IF EXISTS]
DROP FOREIGN KEY [IF EXISTS]
DROP PARTITION   [IF EXISTS]
CHANGE COLUMN    [IF EXISTS]
MODIFY COLUMN    [IF EXISTS]
DROP INDEX       [IF EXISTS]
```

When `IF EXISTS` and `IF NOT EXISTS` are used in clauses, queries will not report errors when the condition is triggered for that clause. A warning with the same message text will be issued and the ALTER will move on to the next clause in the statement (or end if finished).

MariaDB starting with 10.5.2

If this is directive is used after `ALTER ... TABLE`, one will not get an error if the table doesn't exist.

Column Definitions

See CREATE TABLE: Column Definitions for information about column definitions.

Index Definitions

See CREATE TABLE: Index Definitions for information about index definitions.

The CREATE INDEX and DROP INDEX statements can also be used to add or remove an index.

Character Sets and Collations

```
CONVERT TO CHARACTER SET charset_name [COLLATE collation_name]
[DEFAULT] CHARACTER SET [=] charset_name
[DEFAULT] COLLATE [=] collation_name
```

See Setting Character Sets and Collations for details on setting the character sets and collations.

Alter Specifications

Table Options

See CREATE TABLE: Table Options for information about table options.

ADD COLUMN

```
... ADD COLUMN [IF NOT EXISTS]  (col_name column_definition,...)
```

Adds a column to the table. The syntax is the same as in CREATE TABLE. If you are using `IF NOT_EXISTS` the column will not be added if it was not there already. This is very useful when doing scripts to modify tables.

The `FIRST` and `AFTER` clauses affect the physical order of columns in the datafile. Use `FIRST` to add a column in the first (leftmost) position, or `AFTER` followed by a column name to add the new column in any other position. Note that, nowadays, the physical position of a column is usually irrelevant.

See also Instant ADD COLUMN for InnoDB.

DROP COLUMN

```
... DROP COLUMN [IF EXISTS] col_name [CASCADE|RESTRICT]
```

Drops the column from the table. If you are using `IF EXISTS` you will not get an error if the column didn't exist. If the column is part of any index, the column will be dropped from them, except if you add a new column with identical name at the same time. The index will be dropped if all columns from the index were dropped. If the column was used in a view or trigger, you will get an error next time the view or trigger is accessed. Dropping a column that is part of a multi-column `UNIQUE` constraint is not permitted. For example:

```
CREATE TABLE a (
  a int,
  b int,
  primary key (a,b)
);

ALTER TABLE x DROP COLUMN a;
[42000][1072] Key column 'A' doesn't exist in table
```

The reason is that dropping column `a` would result in the new constraint that all values in column `b` be unique. In order to drop the column, an explicit `DROP PRIMARY KEY` and `ADD PRIMARY KEY` would be required. Up until MariaDB 10.2.7, the column was dropped and the additional constraint applied, resulting in the following structure:

```
ALTER TABLE x DROP COLUMN a;
Query OK, 0 rows affected (0.46 sec)

DESC x;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| b     | int(11) | NO   | PRI | NULL    |       |
+-------+---------+------+-----+---------+-------+
```

MariaDB starting with 10.4.0

MariaDB 10.4.0 supports instant DROP COLUMN. DROP COLUMN of an indexed column would imply DROP INDEX (and in the case of a non-UNIQUE multi-column index, possibly ADD INDEX). These will not be allowed with ALGORITHM=INSTANT, but unlike before, they can be allowed with ALGORITHM=NOCOPY

`RESTRICT` and `CASCADE` are allowed to make porting from other database systems easier. In MariaDB, they do nothing.

MODIFY COLUMN

Allows you to modify the type of a column. The column will be at the same place as the original column and all indexes on the column will be kept. Note that when modifying column, you should specify all attributes for the new column.

```
CREATE TABLE t1 (a INT UNSIGNED AUTO_INCREMENT, PRIMARY KEY((a));
ALTER TABLE t1 MODIFY a BIGINT UNSIGNED AUTO_INCREMENT;
```

CHANGE COLUMN

Works like `MODIFY COLUMN` except that you can also change the name of the column. The column will be at the same place as the original column and all index on the column will be kept.

```
CREATE TABLE t1 (a INT UNSIGNED AUTO_INCREMENT, PRIMARY KEY(a));
ALTER TABLE t1 CHANGE a b BIGINT UNSIGNED AUTO_INCREMENT;
```

ALTER COLUMN

This lets you change column options.

```
CREATE TABLE t1 (a INT UNSIGNED AUTO_INCREMENT, b varchar(50), PRIMARY KEY(a));
ALTER TABLE t1 ALTER b SET DEFAULT 'hello';
```

RENAME INDEX/KEY

MariaDB starting with 10.5.2

MariaDB starting with 10.5.2

From MariaDB 10.5.2, it is possible to rename an index using the `RENAME INDEX` (or `RENAME KEY`) syntax, for example:

```
ALTER TABLE t1 RENAME INDEX i_old TO i_new;
```

RENAME COLUMN

MariaDB starting with 10.5.2

From MariaDB 10.5.2, it is possible to rename a column using the `RENAME COLUMN` syntax, for example:

```
ALTER TABLE t1 RENAME COLUMN c_old TO c_new;
```

ADD PRIMARY KEY

Add a primary key.

For `PRIMARY KEY` indexes, you can specify a name for the index, but it is silently ignored, and the name of the index is always `PRIMARY`.

See Getting Started with Indexes: Primary Key for more information.

DROP PRIMARY KEY

Drop a primary key.

For `PRIMARY KEY` indexes, you can specify a name for the index, but it is silently ignored, and the name of the index is always `PRIMARY`.

See Getting Started with Indexes: Primary Key for more information.

ADD FOREIGN KEY

Add a foreign key.

For `FOREIGN KEY` indexes, a reference definition must be provided.

For `FOREIGN KEY` indexes, you can specify a name for the constraint, using the `CONSTRAINT` keyword. That name will be used in error messages.

First, you have to specify the name of the target (parent) table and a column or a column list which must be indexed and whose values must match to the foreign key's values. The `MATCH` clause is accepted to improve the compatibility with other DBMS's, but has no meaning in MariaDB. The `ON DELETE` and `ON UPDATE` clauses specify what must be done when a `DELETE` (or a `REPLACE`) statements attempts to delete a referenced row from the parent table, and when an `UPDATE` statement attempts to modify the referenced foreign key columns in a parent table row, respectively. The following options are allowed:

- `RESTRICT`: The delete/update operation is not performed. The statement terminates with a 1451 error (SQLSTATE '2300').
- `NO ACTION`: Synonym for `RESTRICT`.
- `CASCADE`: The delete/update operation is performed in both tables.
- `SET NULL`: The update or delete goes ahead in the parent table, and the corresponding foreign key fields in the child table are set to `NULL`. (They must not be defined as `NOT NULL` for this to succeed).
- `SET DEFAULT`: This option is implemented only for the legacy PBXT storage engine, which is disabled by default and no longer maintained. It sets the child table's foreign key fields to their `DEFAULT` values when the referenced parent table key entries are updated or deleted.

If either clause is omitted, the default behavior for the omitted clause is `RESTRICT`.

See Foreign Keys for more information.

DROP FOREIGN KEY

Drop a foreign key.

See Foreign Keys for more information.

ADD INDEX

Add a plain index.

Plain indexes are regular indexes that are not unique, and are not acting as a primary key or a foreign key. They are also not the "specialized" `FULLTEXT` or `SPATIAL` indexes.

See Getting Started with Indexes: Plain Indexes for more information.

DROP INDEX

Drop a plain index.

Plain indexes are regular indexes that are not unique, and are not acting as a primary key or a foreign key. They are also not the "specialized" `FULLTEXT` or `SPATIAL` indexes.

See Getting Started with Indexes: Plain Indexes for more information.

ADD UNIQUE INDEX

Add a unique index.

The `UNIQUE` keyword means that the index will not accept duplicated values, except for NULLs. An error will raise if you try to insert duplicate values in a UNIQUE index.

For `UNIQUE` indexes, you can specify a name for the constraint, using the `CONSTRAINT` keyword. That name will be used in error messages.

See Getting Started with Indexes: Unique Index for more information.

DROP UNIQUE INDEX

Drop a unique index.

The `UNIQUE` keyword means that the index will not accept duplicated values, except for NULLs. An error will raise if you try to insert duplicate values in a UNIQUE index.

For `UNIQUE` indexes, you can specify a name for the constraint, using the `CONSTRAINT` keyword. That name will be used in error messages.

See Getting Started with Indexes: Unique Index for more information.

ADD FULLTEXT INDEX

Add a `FULLTEXT` index.

See Full-Text Indexes for more information.

DROP FULLTEXT INDEX

Drop a `FULLTEXT` index.

See Full-Text Indexes for more information.

ADD SPATIAL INDEX

Add a SPATIAL index.

See SPATIAL INDEX for more information.

DROP SPATIAL INDEX

Drop a SPATIAL index.

See SPATIAL INDEX for more information.

ENABLE/ DISABLE KEYS

`DISABLE KEYS` will disable all non unique keys for the table for storage engines that support this (at least MyISAM and Aria). This can be used to speed up inserts into empty tables.

`ENABLE KEYS` will enable all disabled keys.

RENAME TO

Renames the table. See also RENAME TABLE.

ADD CONSTRAINT

Modifies the table adding a constraint on a particular column or columns.

```
ALTER TABLE table_name
ADD CONSTRAINT [constraint_name] CHECK(expression);
```

Before a row is inserted or updated, all constraints are evaluated in the order they are defined. If any constraint fails, then the row will not be updated. One can use most deterministic functions in a constraint, including UDF's.

```
CREATE TABLE account_ledger (
 id INT PRIMARY KEY AUTO_INCREMENT,
 transaction_name VARCHAR(100),
 credit_account VARCHAR(100),
 credit_amount INT,
 debit_account VARCHAR(100),
 debit_amount INT);

ALTER TABLE account_ledger
ADD CONSTRAINT is_balanced
    CHECK((debit_amount + credit_amount) = 0);
```

The `constraint_name` is optional. If you don't provide one in the `ALTER TABLE` statement, MariaDB auto-generates a name for you. This is done so that you can remove it later using DROP CONSTRAINT clause.

You can disable all constraint expression checks by setting the variable check_constraint_checks to `OFF`. You may find this useful when loading a table that violates some constraints that you want to later find and fix in SQL.

To view constraints on a table, query information_schema.TABLE_CONSTRAINTS:

```
SELECT CONSTRAINT_NAME, TABLE_NAME, CONSTRAINT_TYPE
FROM information_schema.TABLE_CONSTRAINTS
WHERE TABLE_NAME = 'account_ledger';

+-----------------+----------------+-----------------+
| CONSTRAINT_NAME | TABLE_NAME     | CONSTRAINT_TYPE |
+-----------------+----------------+-----------------+
| is_balanced     | account_ledger | CHECK           |
+-----------------+----------------+-----------------+
```

DROP CONSTRAINT

`DROP CONSTRAINT` for `UNIQUE` and `FOREIGN KEY` constraints was introduced in MariaDB 10.2.22 and MariaDB 10.3.13.

`DROP CONSTRAINT` for `CHECK` constraints was introduced in MariaDB 10.2.1

Modifies the table, removing the given constraint.

```
ALTER TABLE table_name
DROP CONSTRAINT constraint_name;
```

When you add a constraint to a table, whether through a CREATE TABLE or ALTER TABLE...ADD CONSTRAINT statement, you can either set a `constraint_name` yourself, or allow MariaDB to auto-generate one for you. To view constraints on a table, query information_schema.TABLE_CONSTRAINTS. For instance,

```
CREATE TABLE t (
   a INT,
   b INT,
   c INT,
   CONSTRAINT CHECK(a > b),
   CONSTRAINT check_equals CHECK(a = c));

SELECT CONSTRAINT_NAME, TABLE_NAME, CONSTRAINT_TYPE
FROM information_schema.TABLE_CONSTRAINTS
WHERE TABLE_NAME = 't';

+-----------------+---------------+-----------------+
| CONSTRAINT_NAME | TABLE_NAME    | CONSTRAINT_TYPE |
+-----------------+---------------+-----------------+
| check_equals    | t             | CHECK           |
| CONSTRAINT_1    | t             | CHECK           |
+-----------------+---------------+-----------------+
```

To remove a constraint from the table, issue an `ALTER TABLE...DROP CONSTRAINT` statement. For example,

```
ALTER TABLE t DROP CONSTRAINT is_unique;
```

ADD SYSTEM VERSIONING

Add system versioning. See System-versioned tables.

DROP SYSTEM VERSIONING

Drop system versioning. See System-versioned tables.

ADD PERIOD FOR SYSTEM_TIME

See System-versioned tables.

FORCE

`ALTER TABLE ... FORCE` can force MariaDB to re-build the table.

In MariaDB 5.5 and before, this could only be done by setting the ENGINE table option to its old value. For example, for an InnoDB table, one could execute the following:

```
ALTER TABLE tab_name ENGINE = InnoDB;
```

The `FORCE` option can be used instead. For example, :

```
ALTER TABLE tab_name FORCE;
```

With InnoDB, the table rebuild will only reclaim unused space (i.e. the space previously used for deleted rows) if the innodb_file_per_table system variable is set to `ON` (the default). If the system variable is `OFF`, then the space will not be reclaimed, but it will be-re-used for new data that's later added.

ADD PARTITION

See Partitioning Overview: Adding Partitions.

CONVERT PARTITION

`CONVERT PARTITION` was introduced in MariaDB 10.7.1.

`CONVERT PARTITION` can be used to remove a partition from a table and make this an ordinary table. For example:

```
ALTER TABLE partitioned_table CONVERT PARTITION part1 TO TABLE normal_table;
```

See Partitioning Overview: Converting Partitions to Tables for more details.

CONVERT TABLE /

`CONVERT TABLE` was introduced in MariaDB 10.7.1.

`CONVERT PARTITION` will take an existing table and move this to another table as its own partition with a specified partition definition. For example the following moves `normal_table` to a partition of `partitioned_table` with a definition that its values, based on the `PARTITION BY` of the `partitioned_table`, are less than 12345.

```
ALTER TABLE partitioned_table CONVERT TABLE normal_table
  TO PARTITION part1 VALUES LESS THAN (12345);
```

See also 10.7 preview feature: CONVERT PARTITION (mariadb.org blog post)

DROP PARTITION

See Partitioning Overview: Dropping Partitions.

EXCHANGE PARTITION

This is used to exchange the contents of a partition with another table.

This is performed by swapping the tablespaces of the partition with the other table.

See copying InnoDB's transportable tablespaces for more information.

REMOVE PARTITIONING

See Partitioning Overview: Removing Partitioning.

TRUNCATE PARTITION

See Partitioning Overview: Truncating Partitions.

DISCARD TABLESPACE

This is used to discard an InnoDB table's tablespace.

See copying InnoDB's transportable tablespaces for more information.

IMPORT TABLESPACE

This is used to import an InnoDB table's tablespace. The tablespace should have been copied from its original server after executing FLUSH TABLES FOR EXPORT.

See copying InnoDB's transportable tablespaces for more information.

`ALTER TABLE ... IMPORT` only applies to InnoDB tables. Most other popular storage engines, such as Aria and MyISAM, will recognize their data files as soon as they've been placed in the proper directory under the datadir, and no special DDL is required to import them.

ALGORITHM

The `ALTER TABLE` statement supports the `ALGORITHM` clause. This clause is one of the clauses that is used to implement online DDL. `ALTER TABLE` supports several different algorithms. An algorithm can be explicitly chosen for an `ALTER TABLE` operation by setting the `ALGORITHM` clause. The supported values are:

- `ALGORITHM=DEFAULT` - This implies the default behavior for the specific statement, such as if no `ALGORITHM` clause is specified.
- `ALGORITHM=COPY`
- `ALGORITHM=INPLACE`
- `ALGORITHM=NOCOPY` - This was added in MariaDB 10.3.7.
- `ALGORITHM=INSTANT` - This was added in MariaDB 10.3.7.

See InnoDB Online DDL Overview: ALGORITHM for information on how the `ALGORITHM` clause affects InnoDB.

ALGORITHM=DEFAULT

The default behavior, which occurs if `ALGORITHM=DEFAULT` is specified, or if `ALGORITHM` is not specified at all, usually only makes a copy if the operation doesn't support being done in-place at all. In this case, the most efficient available algorithm will usually be used.

However, in MariaDB 10.3.6 and before, if the value of the old_alter_table system variable is set to `ON`, then the default behavior is to perform `ALTER TABLE` operations by making a copy of the table using the old algorithm.

In MariaDB 10.3.7 and later, the old_alter_table system variable is deprecated. Instead, the alter_algorithm system variable defines the default algorithm for `ALTER TABLE` operations.

ALGORITHM=COPY

`ALGORITHM=COPY` is the name for the original ALTER TABLE algorithm from early MariaDB versions.

When `ALGORITHM=COPY` is set, MariaDB essentially does the following operations:

```
-- Create a temporary table with the new definition
CREATE TEMPORARY TABLE tmp_tab (
...
);

-- Copy the data from the original table
INSERT INTO tmp_tab
   SELECT * FROM original_tab;

-- Drop the original table
DROP TABLE original_tab;

-- Rename the temporary table, so that it replaces the original one
RENAME TABLE tmp_tab TO original_tab;
```

This algorithm is very inefficient, but it is generic, so it works for all storage engines.

If `ALGORITHM=COPY` is specified, then the copy algorithm will be used even if it is not necessary. This can result in a lengthy table copy. If multiple ALTER TABLE operations are required that each require the table to be rebuilt, then it is best to specify all operations in a single ALTER TABLE statement, so that the table is only rebuilt once.

From MariaDB 11.2, ALTER TABLE can now do most operations with ALGORITHM=COPY, LOCK=NONE. See LOCK=NONE.

ALGORITHM=INPLACE

`ALGORITHM=COPY` can be incredibly slow, because the whole table has to be copied and rebuilt. `ALGORITHM=INPLACE` was introduced as a way to avoid this by performing operations in-place and avoiding the table copy and rebuild, when possible.

When `ALGORITHM=INPLACE` is set, the underlying storage engine uses optimizations to perform the operation while avoiding the table copy and rebuild. However, `INPLACE` is a bit of a misnomer, since some operations may still require the table to be rebuilt for some storage engines. Regardless, several operations can be performed without a full copy of the table for some storage engines.

A more accurate name would have been `ALGORITHM=ENGINE`, where `ENGINE` refers to an "engine-specific" algorithm.

If an ALTER TABLE operation supports `ALGORITHM=INPLACE`, then it can be performed using optimizations by the underlying storage engine, but it may rebuilt.

See InnoDB Online DDL Operations with ALGORITHM=INPLACE for more.

ALGORITHM=NOCOPY

`ALGORITHM=NOCOPY` was introduced in MariaDB 10.3.7.

`ALGORITHM=INPLACE` can sometimes be surprisingly slow in instances where it has to rebuild the clustered index, because when the clustered index has to be rebuilt, the whole table has to be rebuilt. `ALGORITHM=NOCOPY` was introduced as a way to avoid this.

If an `ALTER TABLE` operation supports `ALGORITHM=NOCOPY`, then it can be performed without rebuilding the clustered index.

If `ALGORITHM=NOCOPY` is specified for an `ALTER TABLE` operation that does not support `ALGORITHM=NOCOPY`, then an error will be raised. In this case, raising an error is preferable, if the alternative is for the operation to rebuild the clustered index, and perform unexpectedly slowly.

See InnoDB Online DDL Operations with ALGORITHM=NOCOPY for more.

ALGORITHM=INSTANT

`ALGORITHM=INSTANT` was introduced in MariaDB 10.3.7.

`ALGORITHM=INPLACE` can sometimes be surprisingly slow in instances where it has to modify data files. `ALGORITHM=INSTANT` was introduced as a way to avoid this.

If an `ALTER TABLE` operation supports `ALGORITHM=INSTANT`, then it can be performed without modifying any data files.

If `ALGORITHM=INSTANT` is specified for an `ALTER TABLE` operation that does not support `ALGORITHM=INSTANT`, then an error will be raised. In this case, raising an error is preferable, if the alternative is for the operation to modify data files, and perform unexpectedly slowly.

See InnoDB Online DDL Operations with ALGORITHM=INSTANT for more.

LOCK

The `ALTER TABLE` statement supports the `LOCK` clause. This clause is one of the clauses that is used to implement online DDL. `ALTER TABLE` supports several different locking strategies. A locking strategy can be explicitly chosen for an `ALTER TABLE` operation by setting the `LOCK` clause. The supported values are:

DEFAULT

Acquire the least restrictive lock on the table that is supported for the specific operation. Permit the maximum amount of concurrency that is supported for the specific operation.

NONE

Acquire no lock on the table. Permit **all** concurrent DML. If this locking strategy is not permitted for an operation, then an error is raised. From MariaDB 11.2, `ALTER TABLE` can do most operations with `ALGORITHM=COPY, LOCK=NONE`, that is, in most cases, unless the algorithm and lock level are explicitly specified, `ALTER TABLE` will be performed using the `COPY` algorithm while simultaneously allowing concurrent DML statements on the altered table. If this is not desired, one can explicitly specify a different lock level or set old_mode to LOCK_ALTER_TABLE_COPY that will make `ALGORITHM=COPY` use `LOCK=SHARED` by default (but still allowing `LOCK=NONE` to be specified explicitly).

SHARED

Acquire a read lock on the table. Permit **read-only** concurrent DML. If this locking strategy is not permitted for an operation, then an error is raised.

EXCLUSIVE

Acquire a write lock on the table. Do **not** permit concurrent DML.

Different storage engines support different locking strategies for different operations. If a specific locking strategy is chosen for an `ALTER TABLE` operation, and that table's storage engine does not support that locking strategy for that specific operation, then an error will be raised.

If the `LOCK` clause is not explicitly set, then the operation uses `LOCK=DEFAULT`.

ALTER ONLINE TABLE is equivalent to `LOCK=NONE`. Therefore, the ALTER ONLINE TABLE statement can be used to ensure that your `ALTER TABLE` operation allows all concurrent DML.

See InnoDB Online DDL Overview: LOCK for information on how the `LOCK` clause affects InnoDB.

Progress Reporting

MariaDB provides progress reporting for `ALTER TABLE` statement for clients that support the new progress reporting protocol. For example, if you were using the mariadb client, then the progress report might look like this::

```
ALTER TABLE test ENGINE=Aria;
Stage: 1 of 2 'copy to tmp table'    46% of stage
```

The progress report is also shown in the output of the SHOW PROCESSLIST statement and in the contents of the information_schema.PROCESSLIST table.

See Progress Reporting for more information.

Aborting ALTER TABLE Operations

If an `ALTER TABLE` operation is being performed and the connection is killed, the changes will be rolled back in a controlled manner. The rollback can be a slow operation as the time it takes is relative to how far the operation has progressed.

Aborting `ALTER TABLE ... ALGORITHM=COPY` was made faster in MariaDB 10.2.13 by removing excessive undo logging (MDEV-11415). This significantly shortened the time it takes to abort a running ALTER TABLE operation, compared with earlier releases.

Atomic ALTER TABLE

MariaDB starting with 10.6.1

From MariaDB 10.6, `ALTER TABLE` is atomic for most engines, including InnoDB, MyRocks, MyISAM and Aria (MDEV-25180). This means that if there is a crash (server down or power outage) during an `ALTER TABLE` operation, after recovery, either the old table and associated triggers and status will be intact, or the new table will be active.

In older MariaDB versions one could get leftover #sql-alter..', '#sql-backup..' or 'table_name.frm~' files if the system crashed during the `ALTER TABLE` operation.

See Atomic DDL for more information.

Replication

MariaDB starting with 10.8.1

Before MariaDB 10.8.1, ALTER TABLE got fully executed on the primary first, and only then was it replicated and started executing on replicas. From MariaDB 10.8.1, ALTER TABLE gains an option to replicate sooner and begin executing on replicas when it merely *starts* executing on the primary, not when it *finishes*. This way the replication lag caused by a heavy ALTER TABLE can be completely eliminated (MDEV-11675).

Examples

Adding a new column:

```
ALTER TABLE t1 ADD x INT;
```

Dropping a column:

```
ALTER TABLE t1 DROP x;
```

Modifying the type of a column:

```
ALTER TABLE t1 MODIFY x bigint unsigned;
```

Changing the name and type of a column:

```
ALTER TABLE t1 CHANGE a b bigint unsigned auto_increment;
```

Combining multiple clauses in a single ALTER TABLE statement, separated by commas:

```
ALTER TABLE t1 DROP x, ADD x2 INT,  CHANGE y y2 INT;
```

Changing the storage engine and adding a comment:

```
ALTER TABLE t1
  ENGINE = InnoDB
  COMMENT = 'First of three tables containing usage info';
```

Rebuilding the table (the previous example will also rebuild the table if it was already InnoDB):

```
ALTER TABLE t1 FORCE;
```

Dropping an index:

```
ALTER TABLE rooms DROP INDEX u;
```

Adding a unique index:

```
ALTER TABLE rooms ADD UNIQUE INDEX u(room_number);
```

From MariaDB 10.5.3, adding a primary key for an application-time period table with a WITHOUT OVERLAPS constraint:

```
ALTER TABLE rooms ADD PRIMARY KEY(room_number, p WITHOUT OVERLAPS);
```

From MariaDB 10.8.1, ALTER query can be replicated faster with the setting of

```
SET @@SESSION.binlog_alter_two_phase = true;
```

prior the ALTER query. Binlog would contain two event groups

```
| master-bin.000001 | 495 | Gtid              |         1 |         537 | GTID 0-1-2 START ALTER
| master-bin.000001 | 537 | Query             |         1 |         655 | use `test`; alter table
| master-bin.000001 | 655 | Gtid              |         1 |         700 | GTID 0-1-3 COMMIT ALTER
| master-bin.000001 | 700 | Query             |         1 |         835 | use `test`; alter table
```

of which the first one gets delivered to replicas before ALTER is taken to actual execution on the primary.

1.1.1.2.1.1.2 ALTER DATABASE

Modifies a database, changing its overall characteristics.

Syntax

```
ALTER {DATABASE | SCHEMA} [db_name]
    alter_specification ...
ALTER {DATABASE | SCHEMA} db_name
    UPGRADE DATA DIRECTORY NAME

alter_specification:
    [DEFAULT] CHARACTER SET [=] charset_name
  | [DEFAULT] COLLATE [=] collation_name
  | COMMENT [=] 'comment'
```

Contents

Description

`ALTER DATABASE` enables you to change the overall characteristics of a database. These characteristics are stored in the `db.opt` file in the database directory. To use `ALTER DATABASE`, you need the `ALTER` privilege on the database. `ALTER SCHEMA` is a synonym for ALTER DATABASE.

The `CHARACTER SET` clause changes the default database character set. The `COLLATE` clause changes the default database collation. See Character Sets and Collations for more.

You can see what character sets and collations are available using, respectively, the SHOW CHARACTER SET and SHOW COLLATION statements.

Changing the default character set/collation of a database does not change the character set/collation of any stored procedures or stored functions that were previously created, and relied on the defaults. These need to be dropped and recreated in order to apply the character set/collation changes.

The database name can be omitted from the first syntax, in which case the statement applies to the default database.

The syntax that includes the `UPGRADE DATA DIRECTORY NAME` clause was added in MySQL 5.1.23. It updates the name of the directory associated with the database to use the encoding implemented in MySQL 5.1 for mapping database names to database directory names (see Identifier to File Name Mapping). This clause is for use under these conditions:

- It is intended when upgrading MySQL to 5.1 or later from older versions.
- It is intended to update a database directory name to the current encoding format if the name contains special characters that need encoding.
- The statement is used by mariadb-check (as invoked by mariadb-upgrade).

For example,if a database in MySQL 5.0 has a name of a-b-c, the name contains instance of the `-' character. In 5.0, the database directory is also named a-b-c, which is not necessarily safe for all file systems. In MySQL 5.1 and up, the same database name is encoded as a@002db@002dc to produce a file system-neutral directory name.

When a MySQL installation is upgraded to MySQL 5.1 or later from an older version,the server displays a name such as a-b-c (which is in the old format) as #mysql50#a-b-c, and you must refer to the name using the #mysql50# prefix. Use `UPGRADE DATA DIRECTORY NAME` in this case to explicitly tell the server to re-encode the database directory name to the current encoding format:

```
ALTER DATABASE `#mysql50#a-b-c` UPGRADE DATA DIRECTORY NAME;
```

After executing this statement, you can refer to the database as a-b-c without the special #mysql50# prefix.

COMMENT

MariaDB starting with 10.5.0

From MariaDB 10.5.0, it is possible to add a comment of a maximum of 1024 bytes. If the comment length exceeds this length, a error/warning code 4144 is thrown. The database comment is also added to the db.opt file, as well as to the information_schema.schemata table.

Examples

```
ALTER DATABASE test CHARACTER SET='utf8'  COLLATE='utf8_bin';
```

From MariaDB 10.5.0:

```
ALTER DATABASE p COMMENT='Presentations';
```

1.1.1.2.1.1.3 ALTER EVENT

Modifies one or more characteristics of an existing event.

Syntax

```
ALTER
    [DEFINER = { user | CURRENT_USER }]
    EVENT event_name
    [ON SCHEDULE schedule]
    [ON COMPLETION [NOT] PRESERVE]
    [RENAME TO new_event_name]
    [ENABLE | DISABLE | DISABLE ON SLAVE]
    [COMMENT 'comment']
    [DO sql_statement]
```

Contents

Description

The `ALTER EVENT` statement is used to change one or more of the characteristics of an existing event without the need to drop and recreate it. The syntax for each of the `DEFINER`, `ON SCHEDULE`, `ON COMPLETION`, `COMMENT`, `ENABLE` /

`DISABLE` , and `DO` clauses is exactly the same as when used with CREATE EVENT.

This statement requires the EVENT privilege. When a user executes a successful `ALTER EVENT` statement, that user becomes the definer for the affected event.

(In MySQL 5.1.11 and earlier, an event could be altered only by its definer, or by a user having the SUPER privilege.)

`ALTER EVENT` works only with an existing event:

```
ALTER EVENT no_such_event ON SCHEDULE EVERY '2:3' DAY_HOUR;
ERROR 1539 (HY000): Unknown event 'no_such_event'
```

Examples

```
ALTER EVENT myevent
  ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 2 HOUR
  DO
    UPDATE myschema.mytable SET mycol = mycol + 1;
```

1.1.1.2.1.1.4 ALTER FUNCTION

Syntax

```
ALTER FUNCTION func_name [characteristic ...]

characteristic:
    { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
  | COMMENT 'string'
```

Contents

Description

This statement can be used to change the characteristics of a stored function. More than one change may be specified in an `ALTER FUNCTION` statement. However, you cannot change the parameters or body of a stored function using this statement; to make such changes, you must drop and re-create the function using DROP FUNCTION and CREATE FUNCTION.

You must have the `ALTER ROUTINE` privilege for the function. (That privilege is granted automatically to the function creator.) If binary logging is enabled, the `ALTER FUNCTION` statement might also require the `SUPER` privilege, as described in Binary Logging of Stored Routines.

Example

```
ALTER FUNCTION hello SQL SECURITY INVOKER;
```

1.1.1.2.1.1.5 ALTER LOGFILE GROUP

Syntax

```
ALTER LOGFILE GROUP logfile_group
    ADD UNDOFILE 'file_name'
    [INITIAL_SIZE [=] size]
    [WAIT]
    ENGINE [=] engine_name
```

The `ALTER LOGFILE GROUP` statement is not supported by MariaDB. It was originally inherited from MySQL NDB Cluster. See MDEV-19295 for more information.

1.1.1.2.1.1.6 ALTER PROCEDURE

Syntax

```
ALTER PROCEDURE proc_name [characteristic ...]

characteristic:
    { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
  | COMMENT 'string'
```

Description

This statement can be used to change the characteristics of a stored procedure. More than one change may be specified in an `ALTER PROCEDURE` statement. However, you cannot change the parameters or body of a stored procedure using this statement. To make such changes, you must drop and re-create the procedure using either CREATE OR REPLACE PROCEDURE (since MariaDB 10.1.3) or DROP PROCEDURE and CREATE PROCEDURE (MariaDB 10.1.2 and before).

You must have the `ALTER ROUTINE` privilege for the procedure. By default, that privilege is granted automatically to the procedure creator. See Stored Routine Privileges.

Example

```
ALTER PROCEDURE simpleproc SQL SECURITY INVOKER;
```

1.1.6.4 ALTER SEQUENCE

1.1.1.2.1.1.8 ALTER SERVER

Syntax

```
ALTER SERVER server_name
    OPTIONS (option [, option] ...)
```

Contents

Description

Alters the server information for *server_name*, adjusting the specified options as per the CREATE SERVER command. The corresponding fields in the mysql.servers table are updated accordingly. This statement requires the SUPER privilege or, from MariaDB 10.5.2, the FEDERATED ADMIN privilege.

ALTER SERVER is not written to the binary log, irrespective of the binary log format being used. From MariaDB 10.1.13, Galera replicates the CREATE SERVER, ALTER SERVER and DROP SERVER statements.

Examples

```
ALTER SERVER s OPTIONS (USER 'sally');
```

1.1.1.2.1.1.9 ALTER TABLESPACE

The `ALTER TABLESPACE` statement is not supported by MariaDB. It was originally inherited from MySQL NDB Cluster. In MySQL 5.7 and later, the statement is also supported for InnoDB. However, MariaDB has chosen not to include that specific feature. See MDEV-19294 for more information.

1.1.1.1.2 ALTER USER

1.1.1.2.1.1.11 ALTER VIEW

Syntax

```
ALTER
    [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
    [DEFINER = { user | CURRENT_USER }]
    [SQL SECURITY { DEFINER | INVOKER }]
    VIEW view_name [(column_list)]
    AS select_statement
    [WITH [CASCADED | LOCAL] CHECK OPTION]
```

Contents

Description

This statement changes the definition of a view, which must exist. The syntax is similar to that for CREATE VIEW and the effect is the same as for `CREATE OR REPLACE VIEW` if the view exists. This statement requires the `CREATE VIEW` and `DROP` privileges for the view, and some privilege for each column referred to in the `SELECT` statement. `ALTER VIEW` is allowed only to the definer or users with the SUPER privilege.

Example

```
ALTER VIEW v AS SELECT a, a*3 AS a2 FROM t;
```

1.1.1.2.1.2 ANALYZE TABLE

Syntax

```
ANALYZE [NO_WRITE_TO_BINLOG | LOCAL] TABLE tbl_name [,tbl_name ...]
  [PERSISTENT FOR
    { ALL
      | COLUMNS ([col_name [,col_name ...]]) INDEXES ([index_name [,index_name ...]])
    }
  ]
```

Contents

Description

`ANALYZE TABLE` analyzes and stores the key distribution for a table (index statistics). This statement works with MyISAM, Aria and InnoDB tables. During the analysis, InnoDB will allow reads/writes, and MyISAM/Aria reads/inserts. For MyISAM tables, this statement is equivalent to using myisamchk --analyze.

For more information on how the analysis works within InnoDB, see InnoDB Limitations.

MariaDB uses the stored key distribution to decide the order in which tables should be joined when you perform a join on something other than a constant. In addition, key distributions can be used when deciding which indexes to use for a specific table within a query.

This statement requires SELECT and INSERT privileges for the table.

By default, ANALYZE TABLE statements are written to the binary log and will be replicated. The `NO_WRITE_TO_BINLOG` keyword (`LOCAL` is an alias) will ensure the statement is not written to the binary log.

From MariaDB 10.3.19, `ANALYZE TABLE` statements are not logged to the binary log if read_only is set. See also Read-Only Replicas.

`ANALYZE TABLE` is also supported for partitioned tables. You can use `ALTER TABLE ... ANALYZE PARTITION` to analyze one or more partitions.

The Aria storage engine supports progress reporting for the `ANALYZE TABLE` statement.

Engine-Independent Statistics / PERSISTENT FOR

`ANALYZE TABLE` supports engine-independent statistics. See Engine-Independent Table Statistics: Collecting Statistics with the ANALYZE TABLE Statement for more information.

Useful Variables

For calculating the number of duplicates, `ANALYZE TABLE` uses a buffer of sort_buffer_size bytes per column. You can slightly increase the speed of `ANALYZE TABLE` by increasing this variable.

Examples

```
-- update all engine-independent statistics for all columns and indexes
ANALYZE TABLE tbl PERSISTENT FOR ALL;

-- update specific columns and indexes:
ANALYZE TABLE tbl PERSISTENT FOR COLUMNS (col1,col2,...) INDEXES (idx1,idx2,...);

-- empty lists are allowed:
ANALYZE TABLE tbl PERSISTENT FOR COLUMNS (col1,col2,...) INDEXES ();
ANALYZE TABLE tbl PERSISTENT FOR COLUMNS () INDEXES (idx1,idx2,...);

-- the following will only update mysql.table_stats fields:
ANALYZE TABLE tbl PERSISTENT FOR COLUMNS () INDEXES ();

-- when use_stat_tables is set to 'COMPLEMENTARY' or 'PREFERABLY',
-- a simple ANALYZE TABLE  collects engine-independent statistics for all columns and indexes.
SET SESSION use_stat_tables='COMPLEMENTARY';
ANALYZE TABLE tbl;
```

1.1.1.2.1.3 CHECK TABLE

Syntax

```
CHECK TABLE tbl_name [, tbl_name] ... [option] ...

option = {FOR UPGRADE | QUICK | FAST | MEDIUM | EXTENDED | CHANGED}
```

Description

`CHECK TABLE` checks a table or tables for errors. `CHECK TABLE` works for Archive, Aria, CSV, InnoDB and MyISAM tables. For Aria and MyISAM tables, the key statistics are updated as well. For CSV, see also Checking and Repairing CSV Tables.

As an alternative, myisamchk is a commandline tool for checking MyISAM tables when the tables are not being accessed. For Aria tables, there is a similar tool: aria_chk.

For checking dynamic columns integrity, COLUMN_CHECK() can be used.

`CHECK TABLE` can also check views for problems, such as tables that are referenced in the view definition that no longer exist.

`CHECK TABLE` is also supported for partitioned tables. You can use `ALTER TABLE ... CHECK PARTITION` to check one or more partitions.

The meaning of the different options are as follows - note that this can vary a bit between storage engines:

FOR UPGRADE	Do a very quick check if the storage format for the table has changed so that one needs to do a REPAIR. This is only needed when one upgrades between major versions of MariaDB or MySQL. This is usually done by running mariadb-upgrade.
FAST	Only check tables that has not been closed properly or are marked as corrupt. Only supported by the MyISAM and Aria engines. For other engines the table is checked normally
CHANGED	Check only tables that has changed since last REPAIR / CHECK. Only supported by the MyISAM and Aria engines. For other engines the table is checked normally.
QUICK	Do a fast check. For MyISAM and Aria, this means skipping the check of the delete link chain, which may take some time.
MEDIUM	Scan also the data files. Checks integrity between data and index files with checksums. In most cases this should find all possible errors.
EXTENDED	Does a full check to verify every possible error. For InnoDB, Aria, and MyISAM, verify for each row that all its keys exists, and for those index keys, they point back to the primary clustered key. This may take a long time on large tables. This option was previously ignored by InnoDB before MariaDB 10.6.11, MariaDB 10.7.7, MariaDB 10.8.6 and MariaDB 10.9.4.

For most cases running `CHECK TABLE` without options or `MEDIUM` should be good enough.

The Aria storage engine supports progress reporting for this statement.

If you want to know if two tables are identical, take a look at CHECKSUM TABLE.

InnoDB

If `CHECK TABLE` finds an error in an InnoDB table, MariaDB might shutdown to prevent the error propagation. In this case, the problem will be reported in the error log. Otherwise the table or an index might be marked as corrupted, to prevent use. This does not happen with some minor problems, like a wrong number of entries in a secondary index. Those problems are reported in the output of `CHECK TABLE` .

Each tablespace contains a header with metadata. This header is not checked by this statement.

During the execution of `CHECK TABLE` , other threads may be blocked.

Examples

```
check table y extended;
+--------+-------+----------+----------+
| Table  | Op    | Msg_type | Msg_text |
+--------+-------+----------+----------+
| test.y | check | status   | OK       |
+--------+-------+----------+----------+
```

1.1.1.2.1.4 CHECK VIEW

Syntax

```
CHECK VIEW view_name
```

Description

The `CHECK VIEW` statement was introduced in MariaDB 10.0.18 to assist with fixing MDEV-6916, an issue introduced in MariaDB 5.2 where the view algorithms were swapped. It checks whether the view algorithm is correct. It is run as part of mariadb-upgrade, and should not normally be required in regular use.

1.1.1.2.1.5 CHECKSUM TABLE

Syntax

```
CHECKSUM TABLE tbl_name [, tbl_name] ... [ QUICK | EXTENDED ]
```

Contents

Description

`CHECKSUM TABLE` reports a table checksum. This is very useful if you want to know if two tables are the same (for example on a master and slave).

With `QUICK` , the live table checksum is reported if it is available, or `NULL` otherwise. This is very fast. A live checksum is enabled by specifying the `CHECKSUM=1` table option when you create the table; currently, this is supported only for Aria and MyISAM tables.

With `EXTENDED` , the entire table is read row by row and the checksum is calculated. This can be very slow for large tables.

If neither `QUICK` nor `EXTENDED` is specified, MariaDB returns a live checksum if the table storage engine supports it and scans the table otherwise.

`CHECKSUM TABLE` requires the SELECT privilege for the table.

For a nonexistent table, `CHECKSUM TABLE` returns `NULL` and generates a warning.

The table row format affects the checksum value. If the row format changes, the checksum will change. This means that when a table created with a MariaDB/MySQL version is upgraded to another version, the checksum value will probably change.

Two identical tables should always match to the same checksum value; however, also for non-identical tables there is a very slight chance that they will return the same value as the hashing algorithm is not completely collision-free.

Identical Tables

Identical tables mean that the CREATE statement is identical and that the following variable, which affects the storage formats, was the same when the tables were created:

- mysql56-temporal-format

Differences Between MariaDB and MySQL

`CHECKSUM TABLE` may give a different result as MariaDB doesn't ignore `NULL` s in the columns as MySQL 5.1 does (Later MySQL versions should calculate checksums the same way as MariaDB). You can get the 'old style' checksum in MariaDB by starting mysqld with the `--old` option. Note however that that the MyISAM and Aria storage engines in MariaDB are using the new checksum internally, so if you are using `--old` , the `CHECKSUM` command will be slower as it needs to calculate the checksum row by row. Starting from MariaDB Server 10.9, `--old` is deprecated and will be removed in a future release. Set `--old-mode` or `OLD_MODE` to `COMPAT_5_1_CHECKSUM` to get 'old style' checksum.

1.1.1.2.1.6 CREATE TABLE

Syntax

```
CREATE [OR REPLACE] [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    (create_definition,...) [table_options    ]... [partition_options]
CREATE [OR REPLACE] [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    [(create_definition,...)] [table_options   ]... [partition_options]
    select_statement
CREATE [OR REPLACE] [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
   { LIKE old_table_name | (LIKE old_table_name) }

select_statement:
    [IGNORE | REPLACE] [AS] SELECT ...   (Some legal select statement)
```

Contents

Description

Use the `CREATE TABLE` statement to create a table with the given name.

In its most basic form, the `CREATE TABLE` statement provides a table name followed by a list of columns, indexes, and constraints. By default, the table is created in the default database. Specify a database with *`db_name.tbl_name`*. If you quote the table name, you must quote the database name and table name separately as *`` `db_name`.`tbl_name` ``*. This is particularly useful for CREATE TABLE ... SELECT, because it allows to create a table into a database, which contains data

from other databases. See Identifier Qualifiers.

If a table with the same name exists, error 1050 results. Use IF NOT EXISTS to suppress this error and issue a note instead. Use SHOW WARNINGS to see notes.

The `CREATE TABLE` statement automatically commits the current transaction, except when using the TEMPORARY keyword.

For valid identifiers to use as table names, see Identifier Names.

> **Note:** if the default_storage_engine is set to ColumnStore then it needs setting on all UMs. Otherwise when the tables using the default engine are replicated across UMs they will use the wrong engine. You should therefore not use this option as a session variable with ColumnStore.

Microsecond precision can be between 0-6. If no precision is specified it is assumed to be 0, for backward compatibility reasons.

Privileges

Executing the `CREATE TABLE` statement requires the CREATE privilege for the table or the database.

CREATE OR REPLACE

If the `OR REPLACE` clause is used and the table already exists, then instead of returning an error, the server will drop the existing table and replace it with the newly defined table.

This syntax was originally added to make replication more robust if it has to rollback and repeat statements such as `CREATE ... SELECT` on replicas.

```
CREATE OR REPLACE TABLE table_name (a int);
```

is basically the same as:

```
DROP TABLE IF EXISTS table_name;
CREATE TABLE table_name (a int);
```

with the following exceptions:

- If `table_name` was locked with LOCK TABLES it will continue to be locked after the statement.
- Temporary tables are only dropped if the `TEMPORARY` keyword was used. (With DROP TABLE, temporary tables are preferred to be dropped before normal tables).

Things to be Aware of With CREATE OR REPLACE

- The table is dropped first (if it existed), after that the `CREATE` is done. Because of this, if the `CREATE` fails, then the table will not exist anymore after the statement. If the table was used with `LOCK TABLES` it will be unlocked.
- One can't use `OR REPLACE` together with `IF EXISTS`.
- Slaves in replication will by default use `CREATE OR REPLACE` when replicating `CREATE` statements that don"t use `IF EXISTS`. This can be changed by setting the variable slave-ddl-exec-mode to `STRICT`.

CREATE TABLE IF NOT EXISTS

If the `IF NOT EXISTS` clause is used, then the table will only be created if a table with the same name does not already exist. If the table already exists, then a warning will be triggered by default.

CREATE TEMPORARY TABLE

Use the `TEMPORARY` keyword to create a temporary table that is only available to the current session. Temporary tables are dropped when the session ends. Temporary table names are specific to the session. They will not conflict with other temporary tables from other sessions even if they share the same name. They will shadow names of non-temporary tables or views, if they are identical. A temporary table can have the same name as a non-temporary table which is located in the same database. In that case, their name will reference the temporary table when used in SQL statements. You must have the CREATE TEMPORARY TABLES privilege on the database to create temporary tables. If no storage engine is specified,

the default_tmp_storage_engine setting will determine the engine.

ROCKSDB temporary tables cannot be created by setting the default_tmp_storage_engine system variable, or using `CREATE TEMPORARY TABLE LIKE`. Before MariaDB 10.7, they could be specified, but would silently fail, and a MyISAM table would be created instead. From MariaDB 10.7 an error is returned. Explicitly creating a temporary table with `ENGINE=ROCKSDB` has never been permitted.

CREATE TABLE ... LIKE

Use the `LIKE` clause instead of a full table definition to create a table with the same definition as another table, including columns, indexes, and table options. Foreign key definitions, as well as any DATA DIRECTORY or INDEX DIRECTORY table options specified on the original table, will not be created.

CREATE TABLE ... SELECT

You can create a table containing data from other tables using the `CREATE ... SELECT` statement. Columns will be created in the table for each field returned by the `SELECT` query.

You can also define some columns normally and add other columns from a `SELECT`. You can also create columns in the normal way and assign them some values using the query, this is done to force a certain type or other field characteristics. The columns that are not named in the query will be placed before the others. For example:

```
CREATE TABLE test (a INT NOT NULL, b CHAR(10)) ENGINE=MyISAM
    SELECT 5 AS b, c, d FROM another_table;
```

Remember that the query just returns data. If you want to use the same indexes, or the same columns attributes (`[NOT] NULL`, `DEFAULT`, `AUTO_INCREMENT`) in the new table, you need to specify them manually. Types and sizes are not automatically preserved if no data returned by the `SELECT` requires the full size, and `VARCHAR` could be converted into `CHAR`. The CAST() function can be used to forcee the new table to use certain types.

Aliases (`AS`) are taken into account, and they should always be used when you `SELECT` an expression (function, arithmetical operation, etc).

If an error occurs during the query, the table will not be created at all.

If the new table has a primary key or `UNIQUE` indexes, you can use the IGNORE or `REPLACE` keywords to handle duplicate key errors during the query. `IGNORE` means that the newer values must not be inserted an identical value exists in the index. `REPLACE` means that older values must be overwritten.

If the columns in the new table are more than the rows returned by the query, the columns populated by the query will be placed after other columns. Note that if the strict `SQL_MODE` is on, and the columns that are not names in the query do not have a `DEFAULT` value, an error will raise and no rows will be copied.

Concurrent inserts are not used during the execution of a `CREATE ... SELECT`.

If the table already exists, an error similar to the following will be returned:

```
ERROR 1050 (42S01): Table 't' already exists
```

If the `IF NOT EXISTS` clause is used and the table exists, a note will be produced instead of an error.

To insert rows from a query into an existing table, INSERT ... SELECT can be used.

Column Definitions

```
create_definition:
  { col_name column_definition | index_definition | period_definition | CHECK (expr) }

column_definition:
  data_type
    [NOT NULL | NULL] [DEFAULT default_value | (expression)]
    [ON UPDATE [NOW | CURRENT_TIMESTAMP] [(precision)]]
    [AUTO_INCREMENT] [ZEROFILL] [UNIQUE [KEY] | [PRIMARY] KEY]
    [INVISIBLE] [{WITH|WITHOUT} SYSTEM VERSIONING]
    [COMMENT 'string'] [REF_SYSTEM_ID = value]
    [reference_definition]
  | data_type [GENERATED ALWAYS]
  AS { { ROW {START|END} } | { (expression) [VIRTUAL | PERSISTENT | STORED] } }
      [UNIQUE [KEY]] [COMMENT 'string']

constraint_definition:
   CONSTRAINT [constraint_name] CHECK (expression)
```

Note: Until MariaDB 10.4, MariaDB accepts the shortcut format with a REFERENCES clause only in ALTER TABLE and CREATE TABLE statements, but that syntax does nothing. For example:

```
CREATE TABLE b(for_key INT REFERENCES a(not_key));
```

MariaDB simply parses it without returning any error or warning, for compatibility with other DBMS's. Before MariaDB 10.2.1 this was also true for `CHECK` constraints. However, only the syntax described below creates foreign keys.

From MariaDB 10.5, MariaDB will attempt to apply the constraint. See Foreign Keys examples.

Each definition either creates a column in the table or specifies and index or constraint on one or more columns. See Indexes below for details on creating indexes.

Create a column by specifying a column name and a data type, optionally followed by column options. See Data Types for a full list of data types allowed in MariaDB.

NULL and NOT NULL

Use the `NULL` or `NOT NULL` options to specify that values in the column may or may not be `NULL`, respectively. By default, values may be `NULL`. See also NULL Values in MariaDB.

DEFAULT Column Option

Specify a default value using the `DEFAULT` clause. If you don't specify `DEFAULT` then the following rules apply:

- If the column is not defined with `NOT NULL`, `AUTO_INCREMENT` or `TIMESTAMP`, an explicit `DEFAULT NULL` will be added. Note that in MySQL and in MariaDB before 10.1.6, you may get an explicit `DEFAULT` for primary key parts, if not specified with NOT NULL.

The default value will be used if you INSERT a row without specifying a value for that column, or if you specify DEFAULT for that column. Before MariaDB 10.2.1 you couldn't usually provide an expression or function to evaluate at insertion time. You had to provide a constant default value instead. The one exception is that you may use CURRENT_TIMESTAMP as the default value for a TIMESTAMP column to use the current timestamp at insertion time.

CURRENT_TIMESTAMP may also be used as the default value for a DATETIME

You can use most functions in `DEFAULT`. Expressions should have parentheses around them. If you use a non deterministic function in `DEFAULT` then all inserts to the table will be replicated in row mode. You can even refer to earlier columns in the `DEFAULT` expression (excluding `AUTO_INCREMENT` columns):

```
CREATE TABLE t1 (a int DEFAULT (1+1), b int DEFAULT (a+1));
CREATE TABLE t2 (a bigint primary key DEFAULT UUID_SHORT());
```

The `DEFAULT` clause cannot contain any stored functions or subqueries, and a column used in the clause must already have been defined earlier in the statement.

It is possible to assign BLOB or TEXT columns a `DEFAULT` value. In versions prior to MariaDB 10.2.1, assigning a default to these columns was not possible.

You can also use DEFAULT (NEXT VALUE FOR sequence)

AUTO_INCREMENT Column Option

Use AUTO_INCREMENT to create a column whose value can can be set automatically from a simple counter. You can only use `AUTO_INCREMENT` on a column with an integer type. The column must be a key, and there can only be one `AUTO_INCREMENT` column in a table. If you insert a row without specifying a value for that column (or if you specify `0` , `NULL` , or DEFAULT as the value), the actual value will be taken from the counter, with each insertion incrementing the counter by one. You can still insert a value explicitly. If you insert a value that is greater than the current counter value, the counter is set based on the new value. An `AUTO_INCREMENT` column is implicitly `NOT NULL` . Use LAST_INSERT_ID to get the AUTO_INCREMENT value most recently used by an INSERT statement.

ZEROFILL Column Option

If the `ZEROFILL` column option is specified for a column using a numeric data type, then the column will be set to `UNSIGNED` and the spaces used by default to pad the field are replaced with zeros. `ZEROFILL` is ignored in expressions or as part of a UNION. `ZEROFILL` is a non-standard MySQL and MariaDB enhancement.

PRIMARY KEY Column Option

Use `PRIMARY KEY` to make a column a primary key. A primary key is a special type of a unique key. There can be at most one primary key per table, and it is implicitly `NOT NULL` .

Specifying a column as a unique key creates a unique index on that column. See the Index Definitions section below for more information.

UNIQUE KEY Column Option

Use `UNIQUE KEY` (or just `UNIQUE`) to specify that all values in the column must be distinct from each other. Unless the column is `NOT NULL` , there may be multiple rows with `NULL` in the column.

Specifying a column as a unique key creates a unique index on that column.

See the Index Definitions section below for more information.

COMMENT Column Option

You can provide a comment for each column using the `COMMENT` clause. The maximum length is 1024 characters. Use the SHOW FULL COLUMNS statement to see column comments.

REF_SYSTEM_ID

`REF_SYSTEM_ID` can be used to specify Spatial Reference System IDs for spatial data type columns. For example:

```
CREATE TABLE t1(g GEOMETRY(9,4) REF_SYSTEM_ID=101);
```

Generated Columns

A generated column is a column in a table that cannot explicitly be set to a specific value in a DML query. Instead, its value is automatically generated based on an expression. This expression might generate the value based on the values of other columns in the table, or it might generate the value by calling built-in functions or user-defined functions (UDFs).

There are two types of generated columns:

- `PERSISTENT` or `STORED` : This type's value is actually stored in the table.
- `VIRTUAL` : This type's value is not stored at all. Instead, the value is generated dynamically when the table is queried. This type is the default.

Generated columns are also sometimes called computed columns or virtual columns.

For a complete description about generated columns and their limitations, see Generated (Virtual and Persistent/Stored) Columns.

COMPRESSED

Certain columns may be compressed. See Storage-Engine Independent Column Compression.

INVISIBLE

Columns may be made invisible, and hidden in certain contexts. See Invisible Columns.

WITH SYSTEM VERSIONING Column Option

Columns may be explicitly marked as included from system versioning. See System-versioned tables for details.

WITHOUT SYSTEM VERSIONING Column Option

Columns may be explicitly marked as excluded from system versioning. See System-versioned tables for details.

Index Definitions

```
index_definition:
    {INDEX|KEY} [index_name] [index_type] (index_col_name,...) [index_option] ...
  {{{|}}} {FULLTEXT|SPATIAL} [INDEX|KEY] [index_name] (index_col_name,...) [index_option] ...
  {{{|}}} [CONSTRAINT [symbol]] PRIMARY KEY [index_type] (index_col_name,...) [index_option] ...
  {{{|}}} [CONSTRAINT [symbol]] UNIQUE [INDEX|KEY] [index_name] [index_type]
(index_col_name,...) [index_option] ...
  {{{|}}} [CONSTRAINT [symbol]] FOREIGN KEY [index_name] (index_col_name,...)
reference_definition

index_col_name:
    col_name [(length)] [ASC | DESC]

index_type:
    USING {BTREE | HASH | RTREE}

index_option:
    [ KEY_BLOCK_SIZE [=] value
  {{{|}}} index_type
  {{{|}}} WITH PARSER parser_name
  {{{|}}} COMMENT 'string'
  {{{|}}} CLUSTERING={YES| NO} ]
  [ IGNORED | NOT IGNORED ]

reference_definition:
    REFERENCES tbl_name (index_col_name,...)
      [MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]
      [ON DELETE reference_option]
      [ON UPDATE reference_option]

reference_option:
    RESTRICT | CASCADE | SET NULL | NO ACTION
```

`INDEX` and `KEY` are synonyms.

Index names are optional, if not specified an automatic name will be assigned. Index name are needed to drop indexes and appear in error messages when a constraint is violated.

Index Categories

Plain Indexes

Plain indexes are regular indexes that are not unique, and are not acting as a primary key or a foreign key. They are also not the "specialized" `FULLTEXT` or `SPATIAL` indexes.

See Getting Started with Indexes: Plain Indexes for more information.

PRIMARY KEY

For `PRIMARY KEY` indexes, you can specify a name for the index, but it is ignored, and the name of the index is always `PRIMARY`. From MariaDB 10.3.18 and MariaDB 10.4.8, a warning is explicitly issued if a name is specified. Before then,

the name was silently ignored.

See Getting Started with Indexes: Primary Key for more information.

UNIQUE

The `UNIQUE` keyword means that the index will not accept duplicated values, except for NULLs. An error will raise if you try to insert duplicate values in a UNIQUE index.

For `UNIQUE` indexes, you can specify a name for the constraint, using the `CONSTRAINT` keyword. That name will be used in error messages.

> **MariaDB starting with 10.5**
> Unique, if index type is not specified, is normally a BTREE index that can also be used by the optimizer to find rows. If the key is longer than the max key length for the used storage engine, a HASH key will be created. This enables MariaDB to enforce uniqueness for any type or number of columns.

See Getting Started with Indexes: Unique Index for more information.

FOREIGN KEY

For `FOREIGN KEY` indexes, a reference definition must be provided.

For `FOREIGN KEY` indexes, you can specify a name for the constraint, using the `CONSTRAINT` keyword. That name will be used in error messages.

First, you have to specify the name of the target (parent) table and a column or a column list which must be indexed and whose values must match to the foreign key's values. The `MATCH` clause is accepted to improve the compatibility with other DBMS's, but has no meaning in MariaDB. The `ON DELETE` and `ON UPDATE` clauses specify what must be done when a `DELETE` (or a `REPLACE`) statements attempts to delete a referenced row from the parent table, and when an `UPDATE` statement attempts to modify the referenced foreign key columns in a parent table row, respectively. The following options are allowed:

- `RESTRICT`: The delete/update operation is not performed. The statement terminates with a 1451 error (SQLSTATE '2300').
- `NO ACTION`: Synonym for `RESTRICT`.
- `CASCADE`: The delete/update operation is performed in both tables.
- `SET NULL`: The update or delete goes ahead in the parent table, and the corresponding foreign key fields in the child table are set to `NULL`. (They must not be defined as `NOT NULL` for this to succeed).
- `SET DEFAULT`: This option is currently implemented only for the PBXT storage engine, which is disabled by default and no longer maintained. It sets the child table's foreign key fields to their `DEFAULT` values when the referenced parent table key entries are updated or deleted.

If either clause is omitted, the default behavior for the omitted clause is `RESTRICT`.

See Foreign Keys for more information.

FULLTEXT

Use the `FULLTEXT` keyword to create full-text indexes.

See Full-Text Indexes for more information.

SPATIAL

Use the `SPATIAL` keyword to create geometric indexes.

See SPATIAL INDEX for more information.

Index Options

KEY_BLOCK_SIZE Index Option

The `KEY_BLOCK_SIZE` index option is similar to the KEY_BLOCK_SIZE table option.

With the InnoDB storage engine, if you specify a non-zero value for the `KEY_BLOCK_SIZE` table option for the whole table, then the table will implicitly be created with the ROW_FORMAT table option set to `COMPRESSED`. However, this does not happen if you just set the `KEY_BLOCK_SIZE` index option for one or more indexes in the table. The InnoDB storage engine ignores the `KEY_BLOCK_SIZE` index option. However, the SHOW CREATE TABLE statement may still report it for the

index.

For information about the `KEY_BLOCK_SIZE` index option, see the KEY_BLOCK_SIZE table option below.

Index Types

Each storage engine supports some or all index types. See Storage Engine Index Types for details on permitted index types for each storage engine.

Different index types are optimized for different kind of operations:

- `BTREE` is the default type, and normally is the best choice. It is supported by all storage engines. It can be used to compare a column's value with a value using the =, >, >=, <, <=, `BETWEEN` , and `LIKE` operators. `BTREE` can also be used to find `NULL` values. Searches against an index prefix are possible.
- `HASH` is only supported by the MEMORY storage engine. `HASH` indexes can only be used for =, <=, and >= comparisons. It can not be used for the `ORDER BY` clause. Searches against an index prefix are not possible.
- `RTREE` is the default for SPATIAL indexes, but if the storage engine does not support it `BTREE` can be used.

Index columns names are listed between parenthesis. After each column, a prefix length can be specified. If no length is specified, the whole column will be indexed. `ASC` and `DESC` can be specified for compatibility with are DBMS's, but have no meaning in MariaDB.

WITH PARSER Index Option

The `WITH PARSER` index option only applies to FULLTEXT indexes and contains the fulltext parser name. The fulltext parser must be an installed plugin.

COMMENT Index Option

A comment of up to 1024 characters is permitted with the `COMMENT` index option.

The `COMMENT` index option allows you to specify a comment with user-readable text describing what the index is for. This information is not used by the server itself.

CLUSTERING Index Option

The `CLUSTERING` index option is only valid for tables using the TokuDB storage engine.

IGNORED / NOT IGNORED

MariaDB starting with 10.6.0

From MariaDB 10.6.0, indexes can be specified to be ignored by the optimizer. See Ignored Indexes.

Periods

```
period_definition:
    PERIOD FOR SYSTEM_TIME (start_column_name, end_column_name)
```

MariaDB supports a subset of the standard syntax for periods. At the moment it's only used for creating System-versioned tables. Both columns must be created, must be either of a `TIMESTAMP(6)` or `BIGINT UNSIGNED` type, and be generated as `ROW START` and `ROW END` accordingly. See System-versioned tables for details.

The table must also have the `WITH SYSTEM VERSIONING` clause.

Constraint Expressions

Note: Before MariaDB 10.2.1, constraint expressions were accepted in the syntax but ignored.

MariaDB 10.2.1 introduced two ways to define a constraint:

- `CHECK(expression)` given as part of a column definition.
- `CONSTRAINT [constraint_name] CHECK (expression)`

Before a row is inserted or updated, all constraints are evaluated in the order they are defined. If any constraints fails, then the row will not be updated. One can use most deterministic functions in a constraint, including UDFs.

```
create table t1 (a int check(a>0) ,b int check (b> 0), constraint abc check (a>b));
```

If you use the second format and you don't give a name to the constraint, then the constraint will get a auto generated name. This is done so that you can later delete the constraint with ALTER TABLE DROP constraint_name.

One can disable all constraint expression checks by setting the variable `check_constraint_checks` to `OFF`. This is useful for example when loading a table that violates some constraints that you want to later find and fix in SQL.

See CONSTRAINT for more information.

Table Options

For each individual table you create (or alter), you can set some table options. The general syntax for setting options is:

<OPTION_NAME> = <option_value>, [<OPTION_NAME> = <option_value> ...]

The equal sign is optional.

Some options are supported by the server and can be used for all tables, no matter what storage engine they use; other options can be specified for all storage engines, but have a meaning only for some engines. Also, engines can extend `CREATE TABLE` with new options.

If the `IGNORE_BAD_TABLE_OPTIONS` SQL_MODE is enabled, wrong table options generate a warning; otherwise, they generate an error.

```
table_option:
    [STORAGE] ENGINE [=] engine_name
  | AUTO_INCREMENT [=] value
  | AVG_ROW_LENGTH [=] value
  | [DEFAULT] CHARACTER SET [=] charset_name
  | CHECKSUM [=] {0 | 1}
  | [DEFAULT] COLLATE [=] collation_name
  | COMMENT [=] 'string'
  | CONNECTION [=] 'connect_string'
  | DATA DIRECTORY [=] 'absolute path to directory'
  | DELAY_KEY_WRITE [=] {0 | 1}
  | ENCRYPTED [=] {YES | NO}
  | ENCRYPTION_KEY_ID [=] value
  | IETF_QUOTES [=] {YES | NO}
  | INDEX DIRECTORY [=] 'absolute path to directory'
  | INSERT_METHOD [=] { NO | FIRST | LAST }
  | KEY_BLOCK_SIZE [=] value
  | MAX_ROWS [=] value
  | MIN_ROWS [=] value
  | PACK_KEYS [=] {0 | 1 | DEFAULT}
  | PAGE_CHECKSUM [=] {0 | 1}
  | PAGE_COMPRESSED [=] {0 | 1}
  | PAGE_COMPRESSION_LEVEL [=] {0 .. 9}
  | PASSWORD [=] 'string'
  | ROW_FORMAT [=] {DEFAULT|DYNAMIC|FIXED|COMPRESSED|REDUNDANT|COMPACT|PAGE}
  | SEQUENCE [=] {0|1}
  | STATS_AUTO_RECALC [=] {DEFAULT|0|1}
  | STATS_PERSISTENT [=] {DEFAULT|0|1}
  | STATS_SAMPLE_PAGES [=] {DEFAULT|value}
  | TABLESPACE tablespace_name
  | TRANSACTIONAL [=]  {0 | 1}
  | UNION [=] (tbl_name[,tbl_name]...)
  | WITH SYSTEM VERSIONING
```

[STORAGE] ENGINE

`[STORAGE] ENGINE` specifies a storage engine for the table. If this option is not used, the default storage engine is used instead. That is, the default_storage_engine session option value if it is set, or the value specified for the `--default-storage-engine` mariadbd startup option, or the default storage engine, InnoDB. If the specified storage engine is not installed and active, the default value will be used, unless the `NO_ENGINE_SUBSTITUTION` SQL MODE is set (default). This is only true for `CREATE TABLE`, not for `ALTER TABLE`. For a list of storage engines that are present in your server, issue a SHOW ENGINES.

AUTO_INCREMENT

`AUTO_INCREMENT` specifies the initial value for the AUTO_INCREMENT primary key. This works for MyISAM, Aria, InnoDB, MEMORY, and ARCHIVE tables. You can change this option with `ALTER TABLE`, but in that case the new value must be higher than the highest value which is present in the `AUTO_INCREMENT` column. If the storage engine does not support this option, you can insert (and then delete) a row having the wanted value - 1 in the `AUTO_INCREMENT` column.

AVG_ROW_LENGTH

`AVG_ROW_LENGTH` is the average rows size. It only applies to tables using MyISAM and Aria storage engines that have the ROW_FORMAT table option set to `FIXED` format.

MyISAM uses `MAX_ROWS` and `AVG_ROW_LENGTH` to decide the maximum size of a table (default: 256TB, or the maximum file size allowed by the system).

[DEFAULT] CHARACTER SET/CHARSET

`[DEFAULT] CHARACTER SET` (or `[DEFAULT] CHARSET`) is used to set a default character set for the table. This is the character set used for all columns where an explicit character set is not specified. If this option is omitted or `DEFAULT` is specified, database's default character set will be used. See Setting Character Sets and Collations for details on setting the character sets.

CHECKSUM/TABLE_CHECKSUM

`CHECKSUM` (or `TABLE_CHECKSUM`) can be set to 1 to maintain a live checksum for all table's rows. This makes write operations slower, but CHECKSUM TABLE will be very fast. This option is only supported for MyISAM and Aria tables.

[DEFAULT] COLLATE

`[DEFAULT] COLLATE` is used to set a default collation for the table. This is the collation used for all columns where an explicit character set is not specified. If this option is omitted or `DEFAULT` is specified, database's default option will be used. See Setting Character Sets and Collations for details on setting the collations

COMMENT

`COMMENT` is a comment for the table. The maximum length is 2048 characters. Also used to define table parameters when creating a Spider table.

CONNECTION

`CONNECTION` is used to specify a server name or a connection string for a Spider, CONNECT, Federated or FederatedX table.

DATA DIRECTORY/INDEX DIRECTORY

`DATA DIRECTORY` and `INDEX DIRECTORY` are supported for MyISAM and Aria, and DATA DIRECTORY is also supported by InnoDB if the innodb_file_per_table server system variable is enabled, but only in CREATE TABLE, not in ALTER TABLE. So, carefully choose a path for InnoDB tables at creation time, because it cannot be changed without dropping and re-creating the table. These options specify the paths for data files and index files, respectively. If these options are omitted, the database's directory will be used to store data files and index files. Note that these table options do not work for partitioned tables (use the partition options instead), or if the server has been invoked with the --skip-symbolic-links startup option. To avoid the overwriting of old files with the same name that could be present in the directories, you can use the `--keep_files_on_create` option (an error will be issued if files already exist). These options are ignored if the `NO_DIR_IN_CREATE` SQL_MODE is enabled (useful for replication slaves). Also note that symbolic links cannot be used for InnoDB tables.

`DATA DIRECTORY` works by creating symlinks from where the table would normally have been (inside the datadir) to where the option specifies. For security reasons, to avoid bypassing the privilege system, the server does not permit symlinks inside the datadir. Therefore, `DATA DIRECTORY` cannot be used to specify a location inside the datadir. An attempt to do so will result in an error `1210 (HY000) Incorrect arguments to DATA DIRECTORY`.

DELAY_KEY_WRITE

DELAY_KEY_WRITE is supported by MyISAM and Aria, and can be set to 1 to speed up write operations. In that case, when data are modified, the indexes are not updated until the table is closed. Writing the changes to the index file altogether can be much faster. However, note that this option is applied only if the delay_key_write server variable is set to 'ON'. If it is 'OFF' the delayed index writes are always disabled, and if it is 'ALL' the delayed index writes are always used, disregarding the value of DELAY_KEY_WRITE .

ENCRYPTED

The ENCRYPTED table option can be used to manually set the encryption status of an InnoDB table. See InnoDB Encryption for more information.

Aria does not support the ENCRYPTED table option. See MDEV-18049.

See Data-at-Rest Encryption for more information.

ENCRYPTION_KEY_ID

The ENCRYPTION_KEY_ID table option can be used to manually set the encryption key of an InnoDB table. See InnoDB Encryption for more information.

Aria does not support the ENCRYPTION_KEY_ID table option. See MDEV-18049.

See Data-at-Rest Encryption for more information.

IETF_QUOTES

For the CSV storage engine, the IETF_QUOTES option, when set to YES , enables IETF-compatible parsing of embedded quote and comma characters. Enabling this option for a table improves compatibility with other tools that use CSV, but is not compatible with MySQL CSV tables, or MariaDB CSV tables created without this option. Disabled by default.

INSERT_METHOD

INSERT_METHOD is only used with MERGE tables. This option determines in which underlying table the new rows should be inserted. If you set it to 'NO' (which is the default) no new rows can be added to the table (but you will still be able to perform INSERT s directly against the underlying tables). FIRST means that the rows are inserted into the first table, and LAST means that thet are inserted into the last table.

KEY_BLOCK_SIZE

KEY_BLOCK_SIZE is used to determine the size of key blocks, in bytes or kilobytes. However, this value is just a hint, and the storage engine could modify or ignore it. If KEY_BLOCK_SIZE is set to 0, the storage engine's default value will be used.

With the InnoDB storage engine, if you specify a non-zero value for the KEY_BLOCK_SIZE table option for the whole table, then the table will implicitly be created with the ROW_FORMAT table option set to COMPRESSED .

MIN_ROWS/MAX_ROWS

MIN_ROWS and MAX_ROWS let the storage engine know how many rows you are planning to store as a minimum and as a maximum. These values will not be used as real limits, but they help the storage engine to optimize the table. MIN_ROWS is only used by MEMORY storage engine to decide the minimum memory that is always allocated. MAX_ROWS is used to decide the minimum size for indexes.

PACK_KEYS

PACK_KEYS can be used to determine whether the indexes will be compressed. Set it to 1 to compress all keys. With a value of 0, compression will not be used. With the DEFAULT value, only long strings will be compressed. Uncompressed keys are faster.

PAGE_CHECKSUM

PAGE_CHECKSUM is only applicable to Aria tables, and determines whether indexes and data should use page checksums for extra safety.

PAGE_COMPRESSED

`PAGE_COMPRESSED` is used to enable InnoDB page compression for InnoDB tables.

PAGE_COMPRESSION_LEVEL

`PAGE_COMPRESSION_LEVEL` is used to set the compression level for InnoDB page compression for InnoDB tables. The table must also have the PAGE_COMPRESSED table option set to `1`.

Valid values for `PAGE_COMPRESSION_LEVEL` are 1 (the best speed) through 9 (the best compression), .

PASSWORD

`PASSWORD` is unused.

RAID_TYPE

`RAID_TYPE` is an obsolete option, as the raid support has been disabled since MySQL 5.0.

ROW_FORMAT

The `ROW_FORMAT` table option specifies the row format for the data file. Possible values are engine-dependent.

Supported MyISAM Row Formats

For MyISAM, the supported row formats are:

- `FIXED`
- `DYNAMIC`
- `COMPRESSED`

The `COMPRESSED` row format can only be set by the `myisampack` command line tool.

See MyISAM Storage Formats for more information.

Supported Aria Row Formats

For Aria, the supported row formats are:

- `PAGE`
- `FIXED`
- `DYNAMIC`.

See Aria Storage Formats for more information.

Supported InnoDB Row Formats

For InnoDB, the supported row formats are:

- `COMPACT`
- `REDUNDANT`
- `COMPRESSED`
- `DYNAMIC`.

If the `ROW_FORMAT` table option is set to `FIXED` for an InnoDB table, then the server will either return an error or a warning depending on the value of the innodb_strict_mode system variable. If the innodb_strict_mode system variable is set to `OFF`, then a warning is issued, and MariaDB will create the table using the default row format for the specific MariaDB server version. If the innodb_strict_mode system variable is set to `ON`, then an error will be raised.

See InnoDB Storage Formats for more information.

Other Storage Engines and ROW_FORMAT

Other storage engines do not support the `ROW_FORMAT` table option.

SEQUENCE

If the table is a sequence, then it will have the `SEQUENCE` set to `1`.

STATS_AUTO_RECALC

`STATS_AUTO_RECALC` indicates whether to automatically recalculate persistent statistics (see `STATS_PERSISTENT`, below) for an InnoDB table. If set to `1`, statistics will be recalculated when more than 10% of the data has changed. When set to `0`, stats will be recalculated only when an ANALYZE TABLE is run. If set to `DEFAULT`, or left out, the value set by the innodb_stats_auto_recalc system variable applies. See InnoDB Persistent Statistics.

STATS_PERSISTENT

`STATS_PERSISTENT` indicates whether the InnoDB statistics created by ANALYZE TABLE will remain on disk or not. It can be set to `1` (on disk), `0` (not on disk, the pre-MariaDB 10 behavior), or `DEFAULT` (the same as leaving out the option), in which case the value set by the innodb_stats_persistent system variable will apply. Persistent statistics stored on disk allow the statistics to survive server restarts, and provide better query plan stability. See InnoDB Persistent Statistics.

STATS_SAMPLE_PAGES

`STATS_SAMPLE_PAGES` indicates how many pages are used to sample index statistics. If 0 or DEFAULT, the default value, the innodb_stats_sample_pages value is used. See InnoDB Persistent Statistics.

TRANSACTIONAL

`TRANSACTIONAL` is only applicable for Aria tables. In future Aria tables created with this option will be fully transactional, but currently this provides a form of crash protection. See Aria Storage Engine for more details.

UNION

`UNION` must be specified when you create a MERGE table. This option contains a comma-separated list of MyISAM tables which are accessed by the new table. The list is enclosed between parenthesis. Example: `UNION = (t1,t2)`

WITH SYSTEM VERSIONING

`WITH SYSTEM VERSIONING` is used for creating System-versioned tables.

Partitions

```
partition_options:
    PARTITION BY
        { [LINEAR] HASH(expr)
        | [LINEAR] KEY(column_list)
        | RANGE(expr)
        | LIST(expr)
        | SYSTEM_TIME [INTERVAL time_quantity time_unit] [LIMIT num] }
    [PARTITIONS num]
    [SUBPARTITION BY
        { [LINEAR] HASH(expr)
        | [LINEAR] KEY(column_list) }
      [SUBPARTITIONS num]
    ]
    [(partition_definition [, partition_definition] ...)]

partition_definition:
    [PARTITION] partition_name
        [VALUES {LESS THAN {(expr) | MAXVALUE} | IN (value_list)}]
        [[STORAGE] ENGINE [=] engine_name]
        [COMMENT [=] 'comment_text' ]
        [DATA DIRECTORY [=] 'data_dir']
        [INDEX DIRECTORY [=] 'index_dir']
        [MAX_ROWS [=] max_number_of_rows]
        [MIN_ROWS [=] min_number_of_rows]
        [TABLESPACE [=] tablespace_name]
        [NODEGROUP [=] node_group_id]
        [(subpartition_definition [, subpartition_definition] ...)]

subpartition_definition:
    SUBPARTITION logical_name
        [[STORAGE] ENGINE [=] engine_name]
        [COMMENT [=] 'comment_text' ]
        [DATA DIRECTORY [=] 'data_dir']
        [INDEX DIRECTORY [=] 'index_dir']
        [MAX_ROWS [=] max_number_of_rows]
        [MIN_ROWS [=] min_number_of_rows]
        [TABLESPACE [=] tablespace_name]
        [NODEGROUP [=] node_group_id]
```

If the `PARTITION BY` clause is used, the table will be partitioned. A partition method must be explicitly indicated for partitions and subpartitions. Partition methods are:

- `[LINEAR] HASH` creates a hash key which will be used to read and write rows. The partition function can be any valid SQL expression which returns an `INTEGER` number. Thus, it is possible to use the HASH method on an integer column, or on functions which accept integer columns as an argument. However, `VALUES LESS THAN` and `VALUES IN` clauses can not be used with HASH. An example:

```
CREATE TABLE t1 (a INT, b CHAR(5), c DATETIME)
    PARTITION BY HASH ( YEAR(c) );
```

`[LINEAR] HASH` can be used for subpartitions, too.

- `[LINEAR] KEY` is similar to HASH, but the index has an even distribution of data. Also, the expression can only be a column or a list of columns. `VALUES LESS THAN` and `VALUES IN` clauses can not be used with KEY.
- RANGE partitions the rows using on a range of values, using the `VALUES LESS THAN` operator. `VALUES IN` is not allowed with `RANGE`. The partition function can be any valid SQL expression which returns a single value.
- LIST assigns partitions based on a table's column with a restricted set of possible values. It is similar to `RANGE`, but `VALUES IN` must be used for at least 1 columns, and `VALUES LESS THAN` is disallowed.
- `SYSTEM_TIME` partitioning is used for System-versioned tables to store historical data separately from current data.

Only HASH and KEY can be used for subpartitions, and they can be `[LINEAR]`.

It is possible to define up to 8092 partitions and subpartitions.

The number of defined partitions can be optionally specified as `PARTITION count`. This can be done to avoid specifying all partitions individually. But you can also declare each individual partition and, additionally, specify a `PARTITIONS count` clause; in the case, the number of `PARTITION`s must equal count.

Also see Partitioning Types Overview.

MariaDB starting with 10.7.1

From MariaDB 10.7, the PARTITION keyword is now optional as part of the partition definition, for example, instead of:

```
create or replace table t1 (x int)
  partition by range(x) (
    partition p1 values less than (10),
    partition p2 values less than (20),
    partition p3 values less than (30),
    partition p4 values less than (40),
    partition p5 values less than (50),
    partition pn values less than maxvalue);
```

the following can also be used:

```
create or replace table t1 (x int)
  partition by range(x) (
    p1 values less than (10),
    p2 values less than (20),
    p3 values less than (30),
    p4 values less than (40),
    p5 values less than (50),
    pn values less than maxvalue);
```

Sequences

`CREATE TABLE` can also be used to create a SEQUENCE. See CREATE SEQUENCE and Sequence Overview.

Atomic DDL

MariaDB starting with 10.6.1

MariaDB 10.6.1 supports Atomic DDL. `CREATE TABLE` is atomic, except for `CREATE OR REPLACE`, which is only crash safe.

Examples

```
create table if not exists test (
a bigint auto_increment primary key,
name varchar(128) charset utf8,
key name (name(32))
) engine=InnoDB default charset latin1;
```

This example shows a couple of things:

- Usage of `IF NOT EXISTS`; If the table already existed, it will not be created. There will not be any error for the client, just a warning.
- How to create a `PRIMARY KEY` that is automatically generated.
- How to specify a table-specific character set and another for a column.
- How to create an index (`name`) that is only partly indexed (to save space).

The following clauses will work from MariaDB 10.2.1 only.

```
CREATE TABLE t1(
  a int DEFAULT (1+1),
  b int DEFAULT (a+1),
  expires DATETIME DEFAULT(NOW() + INTERVAL 1 YEAR),
  x BLOB DEFAULT USER()
);
```

1.1.1.2.1.7 DELETE

Contents

Syntax

Single-table syntax:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
  FROM tbl_name [PARTITION (partition_list)]
  [FOR PORTION OF period FROM expr1 TO expr2]
  [WHERE where_condition]
  [ORDER BY ...]
  [LIMIT row_count]
  [RETURNING select_expr
    [, select_expr ...]]
```

Multiple-table syntax:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
    tbl_name[.*] [, tbl_name[.*]] ...
    FROM table_references
    [WHERE where_condition]
```

Or:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
    FROM tbl_name[.*] [, tbl_name[.*]] ...
    USING table_references
    [WHERE where_condition]
```

Trimming history:

```
DELETE HISTORY
  FROM tbl_name [PARTITION (partition_list)]
  [BEFORE SYSTEM_TIME [TIMESTAMP|TRANSACTION] expression]
```

Description

Option	Description
LOW_PRIORITY	Wait until all SELECT's are done before starting the statement. Used with storage engines that uses table locking (MyISAM, Aria etc). See HIGH_PRIORITY and LOW_PRIORITY clauses for details.
QUICK	Signal the storage engine that it should expect that a lot of rows are deleted. The storage engine engine can do things to speed up the DELETE like ignoring merging of data blocks until all rows are deleted from the block (instead of when a block is half full). This speeds up things at the expanse of lost space in data blocks. At least MyISAM and Aria support this feature.
IGNORE	Don't stop the query even if a not-critical error occurs (like data overflow). See How IGNORE works for a full description.

For the single-table syntax, the `DELETE` statement deletes rows from tbl_name and returns a count of the number of deleted rows. This count can be obtained by calling the ROW_COUNT() function. The `WHERE` clause, if given, specifies the conditions that identify which rows to delete. With no `WHERE` clause, all rows are deleted. If the ORDER BY clause is specified, the rows are deleted in the order that is specified. The LIMIT clause places a limit on the number of rows that can be deleted.

For the multiple-table syntax, `DELETE` deletes from each `tbl_name` the rows that satisfy the conditions. In this case,

ORDER BY and LIMIT> cannot be used. A `DELETE` can also reference tables which are located in different databases; see Identifier Qualifiers for the syntax.

`where_condition` is an expression that evaluates to true for each row to be deleted. It is specified as described in SELECT.

Currently, you cannot delete from a table and select from the same table in a subquery.

You need the `DELETE` privilege on a table to delete rows from it. You need only the `SELECT` privilege for any columns that are only read, such as those named in the `WHERE` clause. See GRANT.

As stated, a `DELETE` statement with no `WHERE` clause deletes all rows. A faster way to do this, when you do not need to know the number of deleted rows, is to use `TRUNCATE TABLE` . However, within a transaction or if you have a lock on the table, `TRUNCATE TABLE` cannot be used whereas `DELETE` can. See TRUNCATE TABLE, and LOCK.

PARTITION

See Partition Pruning and Selection for details.

FOR PORTION OF

MariaDB starting with 10.4.3

See Application Time Periods - Deletion by Portion.

RETURNING

It is possible to return a resultset of the deleted rows for a single table to the client by using the syntax `DELETE ... RETURNING select_expr [, select_expr2 ...]]`

Any of SQL expression that can be calculated from a single row fields is allowed. Subqueries are allowed. The AS keyword is allowed, so it is possible to use aliases.

The use of aggregate functions is not allowed. RETURNING cannot be used in multi-table DELETEs.

MariaDB starting with 10.3.1

Same Source and Target Table

Until MariaDB 10.3.1, deleting from a table with the same source and target was not possible. From MariaDB 10.3.1, this is now possible. For example:

```
DELETE FROM t1 WHERE c1 IN (SELECT b.c1 FROM t1 b WHERE b.c2=0);
```

MariaDB starting with 10.3.4

DELETE HISTORY

One can use `DELETE HISTORY` to delete historical information from System-versioned tables.

Examples

How to use the ORDER BY and LIMIT clauses:

```
DELETE FROM page_hit ORDER BY timestamp LIMIT 1000000;
```

How to use the RETURNING clause:

```
DELETE FROM t RETURNING f1;
+------+
| f1   |
+------+
|    5 |
|   50 |
|  500 |
+------+
```

The following statement joins two tables: one is only used to satisfy a WHERE condition, but no row is deleted from it; rows from the other table are deleted, instead.

```
DELETE post FROM blog INNER JOIN post WHERE blog.id = post.blog_id;
```

Deleting from the Same Source and Target

```
CREATE TABLE t1 (c1 INT, c2 INT);
DELETE FROM t1 WHERE c1 IN (SELECT b.c1 FROM t1 b WHERE b.c2=0);
```

Until MariaDB 10.3.1, this returned:

```
ERROR 1093 (HY000): Table 't1' is specified twice, both as a target for 'DELETE'
  and as a separate source for
```

From MariaDB 10.3.1:

```
Query OK, 0 rows affected (0.00 sec)
```

1.1.1.2.1.8 DROP TABLE

Syntax

```
DROP [TEMPORARY] TABLE [IF EXISTS] [/*COMMENT TO SAVE*/]
    tbl_name [, tbl_name] ...
    [WAIT n|NOWAIT]
    [RESTRICT | CASCADE]
```

Contents

Description

`DROP TABLE` removes one or more tables. You must have the `DROP` privilege for each table. All table data and the table definition are removed, as well as triggers associated to the table, so be careful with this statement! If any of the tables named in the argument list do not exist, MariaDB returns an error indicating by name which non-existing tables it was unable to drop, but it also drops all of the tables in the list that do exist.

Important: When a table is dropped, user privileges on the table are not automatically dropped. See GRANT.

If another thread is using the table in an explicit transaction or an autocommit transaction, then the thread acquires a metadata lock (MDL) on the table. The `DROP TABLE` statement will wait in the "Waiting for table metadata lock" thread state until the MDL is released. MDLs are released in the following cases:

- If an MDL is acquired in an explicit transaction, then the MDL will be released when the transaction ends.
- If an MDL is acquired in an autocommit transaction, then the MDL will be released when the statement ends.
- Transactional and non-transactional tables are handled the same.

Note that for a partitioned table, `DROP TABLE` permanently removes the table definition, all of its partitions, and all of the data which was stored in those partitions. It also removes the partitioning definition (.par) file associated with the dropped table.

For each referenced table, `DROP TABLE` drops a temporary table with that name, if it exists. If it does not exist, and the `TEMPORARY` keyword is not used, it drops a non-temporary table with the same name, if it exists. The `TEMPORARY` keyword ensures that a non-temporary table will not accidentally be dropped.

Use `IF EXISTS` to prevent an error from occurring for tables that do not exist. A `NOTE` is generated for each non-existent table when using `IF EXISTS`. See SHOW WARNINGS.

If a foreign key references this table, the table cannot be dropped. In this case, it is necessary to drop the foreign key first.

`RESTRICT` and `CASCADE` are allowed to make porting from other database systems easier. In MariaDB, they do nothing.

The comment before the table names (`/*COMMENT TO SAVE*/`) is stored in the binary log. That feature can be used by replication tools to send their internal messages.

It is possible to specify table names as `db_name.tab_name`. This is useful to delete tables from multiple databases with one statement. See Identifier Qualifiers for details.

The DROP privilege is required to use `DROP TABLE` on non-temporary tables. For temporary tables, no privilege is required, because such tables are only visible for the current session.

Note: `DROP TABLE` automatically commits the current active transaction, unless you use the `TEMPORARY` keyword.

MariaDB starting with 10.5.4

From MariaDB 10.5.4, `DROP TABLE` reliably deletes table remnants inside a storage engine even if the `.frm` file is missing. Before then, a missing `.frm` file would result in the statement failing.

MariaDB starting with 10.3.1

WAIT/NOWAIT

Set the lock wait timeout. See WAIT and NOWAIT.

DROP TABLE in replication

`DROP TABLE` has the following characteristics in replication:

- `DROP TABLE IF EXISTS` are always logged.
- `DROP TABLE` without `IF EXISTS` for tables that don't exist are not written to the binary log.
- Dropping of `TEMPORARY` tables are prefixed in the log with `TEMPORARY`. These drops are only logged when running statement or mixed mode replication.
- One `DROP TABLE` statement can be logged with up to 3 different `DROP` statements:
  - `DROP TEMPORARY TABLE list_of_non_transactional_temporary_tables`
  - `DROP TEMPORARY TABLE list_of_transactional_temporary_tables`
  - `DROP TABLE list_of_normal_tables`

`DROP TABLE` on the primary is treated on the replica as `DROP TABLE IF EXISTS`. You can change that by setting slave-ddl-exec-mode to `STRICT`.

Dropping an Internal #sql-... Table

From MariaDB 10.6, DROP TABLE is atomic and the following does not apply. Until MariaDB 10.5, if the mariadbd/mysqld process is killed during an ALTER TABLE you may find a table named #sql-... in your data directory. In MariaDB 10.3, InnoDB tables with this prefix will be deleted automatically during startup. From MariaDB 10.4, these temporary tables will always be deleted automatically.

If you want to delete one of these tables explicitly you can do so by using the following syntax:

```
DROP TABLE `#mysql50##sql-...`;
```

When running an `ALTER TABLE…ALGORITHM=INPLACE` that rebuilds the table, InnoDB will create an internal `#sql-ib` table. Until MariaDB 10.3.2, for these tables, the `.frm` file will be called something else. In order to drop such a table after a server crash, you must rename the `#sql*.frm` file to match the `#sql-ib*.ibd` file.

From MariaDB 10.3.3, the same name as the .frm file is used for the intermediate copy of the table. The #sql-ib names are used by TRUNCATE and delayed DROP.

From MariaDB 10.2.19 and MariaDB 10.3.10, the #sql-ib tables will be deleted automatically.

Dropping All Tables in a Database

The best way to drop all tables in a database is by executing DROP DATABASE, which will drop the database itself, and all

tables in it.

However, if you want to drop all tables in the database, but you also want to keep the database itself and any other non-table objects in it, then you would need to execute `DROP TABLE` to drop each individual table. You can construct these `DROP TABLE` commands by querying the TABLES table in the information_schema database. For example:

```
SELECT CONCAT('DROP TABLE IF EXISTS `', TABLE_SCHEMA, '`.`', TABLE_NAME, '`;')
FROM information_schema.TABLES
WHERE TABLE_SCHEMA = 'mydb';
```

Atomic DROP TABLE

MariaDB starting with 10.6.1

From MariaDB 10.6, `DROP TABLE` for a single table is atomic (MDEV-25180) for most engines, including InnoDB, MyRocks, MyISAM and Aria.

This means that if there is a crash (server down or power outage) during `DROP TABLE`, all tables that have been processed so far will be completely dropped, including related trigger files and status entries, and the binary log will include a `DROP TABLE` statement for the dropped tables. Tables for which the drop had not started will be left intact.

In older MariaDB versions, there was a small chance that, during a server crash happening in the middle of `DROP TABLE`, some storage engines that were using multiple storage files, like MyISAM, could have only a part of its internal files dropped.

In MariaDB 10.5, `DROP TABLE` was extended to be able to delete a table that was only partly dropped (MDEV-11412) as explained above. Atomic `DROP TABLE` is the final piece to make `DROP TABLE` fully reliable.

Dropping multiple tables is crash-safe.

See Atomic DDL for more information.

Examples

```
DROP TABLE Employees, Customers;
```

Notes

Beware that `DROP TABLE` can drop both tables and sequences. This is mainly done to allow old tools like mariadb-dump (previously mysqldump) to work with sequences.

1.1.1.2.1.9 Installing System Tables (mariadb-install-db)

`mariadb-install-db` initializes the MariaDB data directory and creates the system tables in the mysql database, if they do not exist. MariaDB uses these tables to manage privileges, roles, and plugins. It also uses them to provide the data for the help command in the mariadb client.

mariadb-install-db works by starting MariaDB Server's `mysqld` process in --bootstrap mode and sending commands to create the system tables and their content.

There is a version specifically for Windows, mysql_install_db.exe.

To invoke `mariadb-install-db`, use the following syntax:

```
mariadb-install-db --user=mysql
```

For the options supported by mariadb-install-db, see mariadb-install-db: Options.

For the option groups read by mariadb-install-db, see mariadb-install-db: Option Groups.

See mariadb-install-db: Installing System Tables for information on the installation process.

See mariadb-install-db: Troubleshooting Issues for information on how to troubleshoot the installation process.

1.1.1.2.1.10 mysqlcheck

`mariadb-check` is a tool for checking, repairing, analyzing and optimizing tables.

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadb-check` is a symlink to `mysqlcheck, the old name for the tool.`

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mariadb-check` is the name of the tool, with `mysqlcheck` a symlink .

See mariadb-check for details.

1.1.1.2.1.11 OPTIMIZE TABLE

Syntax

```
OPTIMIZE [NO_WRITE_TO_BINLOG | LOCAL] TABLE
    tbl_name [, tbl_name] ...
    [WAIT n | NOWAIT]
```

Contents

Description

`OPTIMIZE TABLE` has two main functions. It can either be used to defragment tables, or to update the InnoDB fulltext index.

WAIT/NOWAIT

Set the lock wait timeout. See WAIT and NOWAIT.

Defragmenting

`OPTIMIZE TABLE` works for InnoDB (before MariaDB 10.1.1, only if the innodb_file_per_table server system variable is set), Aria, MyISAM and ARCHIVE tables, and should be used if you have deleted a large part of a table or if you have made many changes to a table with variable-length rows (tables that have VARCHAR, VARBINARY, BLOB, or TEXT columns). Deleted rows are maintained in a linked list and subsequent `INSERT` operations reuse old row positions.

This statement requires SELECT and INSERT privileges for the table.

By default, `OPTIMIZE TABLE` statements are written to the binary log and will be replicated. The `NO_WRITE_TO_BINLOG` keyword (`LOCAL` is an alias) will ensure the statement is not written to the binary log.

`OPTIMIZE TABLE` statements are not logged to the binary log if read_only is set. See also Read-Only Replicas.

`OPTIMIZE TABLE` is also supported for partitioned tables. You can use `ALTER TABLE ... OPTIMIZE PARTITION` to optimize one or more partitions.

You can use `OPTIMIZE TABLE` to reclaim the unused space and to defragment the data file. With other storage engines, `OPTIMIZE TABLE` does nothing by default, and returns this message: " The storage engine for the table doesn't support optimize". However, if the server has been started with the `--skip-new` option, `OPTIMIZE TABLE` is linked to ALTER TABLE, and recreates the table. This operation frees the unused space and updates index statistics.

The Aria storage engine supports progress reporting for this statement.

If a MyISAM table is fragmented, concurrent inserts will not be performed until an `OPTIMIZE TABLE` statement is executed on that table, unless the concurrent_insert server system variable is set to `ALWAYS` .

Updating an InnoDB fulltext index

When rows are added or deleted to an InnoDB fulltext index, the index is not immediately re-organized, as this can be an expensive operation. Change statistics are stored in a separate location . The fulltext index is only fully re-organized when an `OPTIMIZE TABLE` statement is run.

By default, an OPTIMIZE TABLE will defragment a table. In order to use it to update fulltext index statistics, the innodb_optimize_fulltext_only system variable must be set to `1` . This is intended to be a temporary setting, and should be reset to `0` once the fulltext index has been re-organized.

Since fulltext re-organization can take a long time, the innodb_ft_num_word_optimize variable limits the re-organization to a number of words (2000 by default). You can run multiple OPTIMIZE statements to fully re-organize the index.

Defragmenting InnoDB tablespaces

MariaDB 10.1.1 merged the Facebook/Kakao defragmentation patch, allowing one to use `OPTIMIZE TABLE` to defragment InnoDB tablespaces. For this functionality to be enabled, the innodb_defragment system variable must be enabled. No new tables are created and there is no need to copy data from old tables to new tables. Instead, this feature loads `n` pages (determined by innodb-defragment-n-pages) and tries to move records so that pages would be full of records and then frees pages that are fully empty after the operation. Note that tablespace files (including ibdata1) will not shrink as the result of defragmentation, but one will get better memory utilization in the InnoDB buffer pool as there are fewer data pages in use.

See Defragmenting InnoDB Tablespaces for more details.

1.1.1.2.1.12 RENAME TABLE

Syntax

```
RENAME TABLE[S] [IF EXISTS] tbl_name
  [WAIT n | NOWAIT]
  TO new_tbl_name
    [, tbl_name2 TO new_tbl_name2] ...
```

Contents

Description

This statement renames one or more tables or views, but not the privileges associated with them.

IF EXISTS

MariaDB starting with 10.5.2

If this directive is used, one will not get an error if the table to be renamed doesn't exist.

The rename operation is done atomically, which means that no other session can access any of the tables while the rename is running. For example, if you have an existing table `old_table` , you can create another table `new_table` that has the same structure but is empty, and then replace the existing table with the empty one as follows (assuming that `backup_table` does not already exist):

```
CREATE TABLE new_table (...);
RENAME TABLE old_table TO backup_table, new_table TO old_table;
```

`tbl_name` can optionally be specified as `db_name . tbl_name` . See Identifier Qualifiers. This allows to use `RENAME` to move a table from a database to another (as long as they are on the same filesystem):

```
RENAME TABLE db1.t TO db2.t;
```

Note that moving a table to another database is not possible if it has some triggers. Trying to do so produces the following error:

```
ERROR 1435 (HY000): Trigger in wrong schema
```

Also, views cannot be moved to another database:

```
ERROR 1450 (HY000): Changing schema from 'old_db' to 'new_db' is not allowed.
```

Multiple tables can be renamed in a single statement. The presence or absence of the optional `S` (`RENAME TABLE` or `RENAME TABLES`) has no impact, whether a single or multiple tables are being renamed.

If a `RENAME TABLE` renames more than one table and one renaming fails, all renames executed by the same statement are rolled back.

Renames are always executed in the specified order. Knowing this, it is also possible to swap two tables' names:

```
RENAME TABLE t1 TO tmp_table,
    t2 TO t1,
    tmp_table TO t2;
```

WAIT/NOWAIT

Set the lock wait timeout. See WAIT and NOWAIT.

Privileges

Executing the `RENAME TABLE` statement requires the DROP, CREATE and INSERT privileges for the table or the database.

Atomic RENAME TABLE

MariaDB starting with 10.6.1

From MariaDB 10.6, `RENAME TABLE` is atomic for most engines, including InnoDB, MyRocks, MyISAM and Aria (MDEV-23842). This means that if there is a crash (server down or power outage) during `RENAME TABLE`, all tables will revert to their original names and any changes to trigger files will be reverted.

In older MariaDB version there was a small chance that, during a server crash happening in the middle of `RENAME TABLE`, some tables could have been renamed (in the worst case partly) while others would not be renamed.

See Atomic DDL for more information.

1.1.1.2.1.13 REPAIR TABLE

Syntax

```
REPAIR [NO_WRITE_TO_BINLOG | LOCAL] TABLE
    tbl_name [, tbl_name] ...
    [QUICK] [EXTENDED] [USE_FRM]
```

Description

`REPAIR TABLE` repairs a possibly corrupted table. By default, it has the same effect as

```
myisamchk --recover tbl_name
```

or

```
aria_chk --recover tbl_name
```

See aria_chk and myisamchk for more.

`REPAIR TABLE` works for Archive, Aria, CSV and MyISAM tables. For InnoDB, see recovery modes. For CSV, see also Checking and Repairing CSV Tables. For Archive, this statement also improves compression. If the storage engine does not support this statement, a warning is issued.

This statement requires SELECT and INSERT privileges for the table.

By default, `REPAIR TABLE` statements are written to the binary log and will be replicated. The `NO_WRITE_TO_BINLOG` keyword (`LOCAL` is an alias) will ensure the statement is not written to the binary log.

From MariaDB 10.3.19, `REPAIR TABLE` statements are not logged to the binary log if read_only is set. See also Read-Only Replicas.

When an index is recreated, the storage engine may use a configurable buffer in the process. Incrementing the buffer speeds up the index creation. Aria and MyISAM allocate a buffer whose size is defined by `aria_sort_buffer_size` or `myisam_sort_buffer_size` , also used for `ALTER TABLE` .

`REPAIR TABLE` is also supported for partitioned tables. However, the `USE_FRM` option cannot be used with this statement on a partitioned table.

`ALTER TABLE ... REPAIR PARTITION` can be used to repair one or more partitions.

The Aria storage engine supports progress reporting for this statement.

1.1.1.2.1.14 REPAIR VIEW

Syntax

```
REPAIR [NO_WRITE_TO_BINLOG | LOCAL] VIEW  view_name[, view_name] ... [FROM MYSQL]
```

Contents

Description

The `REPAIR VIEW` statement was introduced to assist with fixing MDEV-6916, an issue introduced in MariaDB 5.2 where the view algorithms were swapped compared to their MySQL on disk representation. It checks whether the view algorithm is correct. It is run as part of mariadb-upgrade, and should not normally be required in regular use.

By default it corrects the checksum and if necessary adds the mariadb-version field. If the optional `FROM MYSQL` clause is used, and no mariadb-version field is present, the MERGE and TEMPTABLE algorithms are toggled.

By default, `REPAIR VIEW` statements are written to the binary log and will be replicated. The `NO_WRITE_TO_BINLOG` keyword (`LOCAL` is an alias) will ensure the statement is not written to the binary log.

1.1.1.2.1.15 REPLACE

Syntax

```
REPLACE [LOW_PRIORITY | DELAYED]
 [INTO] tbl_name [PARTITION (partition_list)] [(col,...)]
 {VALUES | VALUE} ({expr | DEFAULT},...),(...),...
[RETURNING select_expr
      [, select_expr ...]]
```

Or:

```
REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name [PARTITION (partition_list)]
    SET col={expr | DEFAULT}, ...
[RETURNING select_expr
      [, select_expr ...]]
```

Or:

```
REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name [PARTITION (partition_list)] [(col,...)]
    SELECT ...
[RETURNING select_expr
      [, select_expr ...]]
```

Contents

Description

`REPLACE` works exactly like `INSERT`, except that if an old row in the table has the same value as a new row for a `PRIMARY KEY` or a `UNIQUE` index, the old row is deleted before the new row is inserted. If the table has more than one `UNIQUE` keys, it is possible that the new row conflicts with more than one row. In this case, all conflicting rows will be deleted.

The table name can be specified in the form `db_name.tbl_name` or, if a default database is selected, in the form `tbl_name` (see Identifier Qualifiers). This allows to use `REPLACE ... SELECT` to copy rows between different databases.

MariaDB starting with 10.5.0

The RETURNING clause was introduced in MariaDB 10.5.0

Basically it works like this:

```
BEGIN;
SELECT 1 FROM t1 WHERE key=# FOR UPDATE;
IF found-row
  DELETE FROM t1 WHERE key=# ;
ENDIF
INSERT INTO t1 VALUES (...);
END;
```

The above can be replaced with:

```
REPLACE INTO t1 VALUES (...)
```

`REPLACE` is a MariaDB/MySQL extension to the SQL standard. It either inserts, or deletes and inserts. For other MariaDB/MySQL extensions to standard SQL --- that also handle duplicate values --- see IGNORE and INSERT ON DUPLICATE KEY UPDATE.

Note that unless the table has a `PRIMARY KEY` or `UNIQUE` index, using a `REPLACE` statement makes no sense. It becomes equivalent to `INSERT`, because there is no index to be used to determine whether a new row duplicates another.

Values for all columns are taken from the values sSee Partition Pruning and Selection for details.pecified in the `REPLACE` statement. Any missing columns are set to their default values, just as happens for `INSERT`. You cannot refer to values from the current row and use them in the new row. If you use an assignment such as `'SET col = col + 1'`, the reference to the column name on the right hand side is treated as `DEFAULT(col)`, so the assignment is equivalent to `'SET col = DEFAULT(col) + 1'`.

To use `REPLACE`, you must have both the `INSERT` and `DELETE` privileges for the table.

There are some gotchas you should be aware of, before using `REPLACE`:

- If there is an `AUTO_INCREMENT` field, a new value will be generated.
- If there are foreign keys, `ON DELETE` action will be activated by `REPLACE`.
- Triggers on `DELETE` and `INSERT` will be activated by `REPLACE`.

To avoid some of these behaviors, you can use `INSERT ... ON DUPLICATE KEY UPDATE`.

This statement activates INSERT and DELETE triggers. See Trigger Overview for details.

PARTITION

See Partition Pruning and Selection for details.

REPLACE RETURNING

`REPLACE ... RETURNING` returns a resultset of the replaced rows.

This returns the listed columns for all the rows that are replaced, or alternatively, the specified SELECT expression. Any SQL expressions which can be calculated can be used in the select expression for the RETURNING clause, including virtual columns and aliases, expressions which use various operators such as bitwise, logical and arithmetic operators, string functions, date-time functions, numeric functions, control flow functions, secondary functions and stored functions. Along with this, statements which have subqueries and prepared statements can also be used.

Examples

Simple REPLACE statement

```
REPLACE INTO t2 VALUES (1,'Leopard'),(2,'Dog') RETURNING id2, id2+id2
as Total ,id2|id2, id2&&id2;
+-----+-------+---------+----------+
| id2 | Total | id2|id2 | id2&&id2 |
+-----+-------+---------+----------+
|   1 |     2 |       1 |        1 |
|   2 |     4 |       2 |        1 |
+-----+-------+---------+----------+
```

Using stored functions in RETURNING

```
DELIMITER |
CREATE FUNCTION f(arg INT) RETURNS INT
    BEGIN
      RETURN (SELECT arg+arg);
    END|

DELIMITER ;
PREPARE stmt FROM "REPLACE INTO t2 SET id2=3, animal2='Fox' RETURNING f2(id2),
UPPER(animal2)";

EXECUTE stmt;
+---------+-----------------+
| f2(id2) | UPPER(animal2) |
+---------+-----------------+
|       6 | FOX             |
+---------+-----------------+
```

Subqueries in the statement

```
REPLACE INTO t1 SELECT * FROM t2 RETURNING (SELECT id2 FROM t2 WHERE
id2 IN (SELECT id2 FROM t2 WHERE id2=1)) AS new_id;
+--------+
| new_id |
+--------+
|      1 |
|      1 |
|      1 |
|      1 |
+--------+
```

Subqueries in the RETURNING clause that return more than one row or column cannot be used..

Aggregate functions cannot be used in the RETURNING clause. Since aggregate functions work on a set of values and if the purpose is to get the row count, ROW_COUNT() with SELECT can be used, or it can be used in REPLACE...SEL== Description

`REPLACE ... RETURNING` returns a resultset of the replaced rows.

This returns the listed columns for all the rows that are replaced, or alternatively, the specified SELECT expression. Any SQL expressions which can be calculated can be used in the select expression for the RETURNING clause, including virtual columns and aliases, expressions which use various operators such as bitwise, logical and arithmetic operators, string functions, date-time functions, numeric functions, control flow functions, secondary functions and stored functions. Along with this, statements which have subqueries and prepared statements can also be used.

Examples

Simple REPLACE statement

```
REPLACE INTO t2 VALUES (1,'Leopard'),(2,'Dog') RETURNING id2, id2+id2
as Total ,id2|id2, id2&&id2;
+-----+-------+---------+----------+
| id2 | Total | id2|id2 | id2&&id2 |
+-----+-------+---------+----------+
|   1 |     2 |       1 |        1 |
|   2 |     4 |       2 |        1 |
+-----+-------+---------+----------+
```

Using stored functions in RETURNING

```
DELIMITER |
CREATE FUNCTION f(arg INT) RETURNS INT
    BEGIN
      RETURN (SELECT arg+arg);
    END|

DELIMITER ;
PREPARE stmt FROM "REPLACE INTO t2 SET id2=3, animal2='Fox' RETURNING f2(id2),
UPPER(animal2)";

EXECUTE stmt;
+---------+----------------+
| f2(id2) | UPPER(animal2) |
+---------+----------------+
|       6 | FOX            |
+---------+----------------+
```

Subqueries in the statement

```
REPLACE INTO t1 SELECT * FROM t2 RETURNING (SELECT id2 FROM t2 WHERE
id2 IN (SELECT id2 FROM t2 WHERE id2=1)) AS new_id;
+--------+
| new_id |
+--------+
|      1 |
|      1 |
|      1 |
|      1 |
+--------+
```

Subqueries in the RETURNING clause that return more than one row or column cannot be used..

Aggregate functions cannot be used in the RETURNING clause. Since aggregate functions work on a set of values and if the purpose is to get the row count, ROW_COUNT() with SELECT can be used, or it can be used in REPLACE...SELECT...RETURNING if the table in the RETURNING clause is not the same as the REPLACE table. ECT...RETURNING if the table in the RETURNING clause is not the same as the REPLACE table.

1.1.1.2.1.16 SHOW COLUMNS

Syntax

```
SHOW [FULL] {COLUMNS | FIELDS} FROM tbl_name [FROM db_name]
    [LIKE 'pattern' | WHERE expr]
```

Contents

Description

`SHOW COLUMNS` displays information about the columns in a given table. It also works for views. The `LIKE` clause, if present on its own, indicates which column names to match. The `WHERE` and `LIKE` clauses can be given to select rows using more general conditions, as discussed in Extended SHOW.

If the data types differ from what you expect them to be based on a `CREATE TABLE` statement, note that MariaDB sometimes changes data types when you create or alter a table. The conditions under which this occurs are described in the Silent Column Changes article.

The `FULL` keyword causes the output to include the column collation and comments, as well as the privileges you have for each column.

You can use `db_name.tbl_name` as an alternative to the `tbl_name FROM db_name` syntax. In other words, these two statements are equivalent:

```
SHOW COLUMNS FROM mytable FROM mydb;
SHOW COLUMNS FROM mydb.mytable;
```

`SHOW COLUMNS` displays the following values for each table column:

Field indicates the column name.

Type indicates the column data type.

Collation indicates the collation for non-binary string columns, or NULL for other columns. This value is displayed only if you use the FULL keyword.

The **Null** field contains YES if NULL values can be stored in the column, NO if not.

The **Key** field indicates whether the column is indexed:

- If **Key** is empty, the column either is not indexed or is indexed only as a secondary column in a multiple-column, non-unique index.
- If **Key** is ***PRI***, the column is a `PRIMARY KEY` or is one of the columns in a multiple-column `PRIMARY KEY`.
- If **Key** is ***UNI***, the column is the first column of a unique-valued index that cannot contain `NULL` values.
- If **Key** is ***MUL***, multiple occurrences of a given value are allowed within the column. The column is the first column of a non-unique index or a unique-valued index that can contain `NULL` values.

If more than one of the **Key** values applies to a given column of a table, **Key** displays the one with the highest priority, in the order PRI, UNI, MUL.

A `UNIQUE` index may be displayed as `PRI` if it cannot contain `NULL` values and there is no `PRIMARY KEY` in the table. A `UNIQUE` index may display as `MUL` if several columns form a composite `UNIQUE` index; although the combination of the columns is unique, each column can still hold multiple occurrences of a given value.

The **Default** field indicates the default value that is assigned to the column.

The **Extra** field contains any additional information that is available about a given column.

Value	Description
`AUTO_INCREMENT`	The column was created with the `AUTO_INCREMENT` keyword.
`PERSISTENT`	The column was created with the `PERSISTENT` keyword. (New in 5.3)
`VIRTUAL`	The column was created with the `VIRTUAL` keyword. (New in 5.3)
on update `CURRENT_TIMESTAMP`	The column is a `TIMESTAMP` column that is automatically updated on `INSERT` and `UPDATE`.

Privileges indicates the privileges you have for the column. This value is displayed only if you use the `FULL` keyword.

Comment indicates any comment the column has. This value is displayed only if you use the `FULL` keyword.

`SHOW FIELDS` is a synonym for `SHOW COLUMNS`. Also DESCRIBE and EXPLAIN can be used as shortcuts.

You can also list a table's columns with:

```
mariadb-show db_name tbl_name
```

See the mariadb-show command for more details.

The DESCRIBE statement provides information similar to `SHOW COLUMNS`. The information_schema.COLUMNS table provides similar, but more complete, information.

The SHOW CREATE TABLE, SHOW TABLE STATUS, and SHOW INDEX statements also provide information about tables.

Examples

```
SHOW COLUMNS FROM city;
+------------+----------+------+-----+---------+----------------+
| Field      | Type     | Null | Key | Default | Extra          |
+------------+----------+------+-----+---------+----------------+
| Id         | int(11)  | NO   | PRI | NULL    | auto_increment |
| Name       | char(35) | NO   |     |         |                |
| Country    | char(3)  | NO   | UNI |         |                |
| District   | char(20) | YES  | MUL |         |                |
| Population | int(11)  | NO   |     | 0       |                |
+------------+----------+------+-----+---------+----------------+
```

```
SHOW COLUMNS FROM employees WHERE Type LIKE 'Varchar%';
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| first_name    | varchar(30) | NO   | MUL | NULL    |       |
| last_name     | varchar(40) | NO   |     | NULL    |       |
| position      | varchar(25) | NO   |     | NULL    |       |
| home_address  | varchar(50) | NO   |     | NULL    |       |
| home_phone    | varchar(12) | NO   |     | NULL    |       |
| employee_code | varchar(25) | NO   | UNI | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
```

1.1.1.2.1.17 SHOW CREATE TABLE

Syntax

```
SHOW CREATE TABLE tbl_name
```

Contents

Description

Shows the CREATE TABLE statement that created the given table. The statement requires the SELECT privilege for the table. This statement also works with views and SEQUENCE.

`SHOW CREATE TABLE` quotes table and column names according to the value of the sql_quote_show_create server system variable.

Certain SQL_MODE values can result in parts of the original CREATE statement not being included in the output. MariaDB-specific table options, column options, and index options are not included in the output of this statement if the NO_TABLE_OPTIONS, NO_FIELD_OPTIONS and NO_KEY_OPTIONS SQL_MODE flags are used. All MariaDB-specific table attributes are also not shown when a non-MariaDB/MySQL emulation mode is used, which includes ANSI, DB2, POSTGRESQL, MSSQL, MAXDB or ORACLE.

Invalid table options, column options and index options are normally commented out (note, that it is possible to create a table with invalid options, by altering a table of a different engine, where these options were valid). To have them uncommented, enable the IGNORE_BAD_TABLE_OPTIONS SQL_MODE. Remember that replaying a CREATE TABLE statement with uncommented invalid options will fail with an error, unless the IGNORE_BAD_TABLE_OPTIONS SQL_MODE is in effect.

Note that `SHOW CREATE TABLE` is not meant to provide metadata about a table. It provides information about how the table was declared, but the real table structure could differ a bit. For example, if an index has been declared as `HASH`, the `CREATE TABLE` statement returned by `SHOW CREATE TABLE` will declare that index as `HASH`; however, it is possible that the index is in fact a `BTREE`, because the storage engine does not support `HASH`.

MariaDB permits TEXT and BLOB data types to be assigned a DEFAULT value. As a result, `SHOW CREATE TABLE` will append a `DEFAULT NULL` to nullable TEXT or BLOB fields if no specific default is provided.

Numbers are no longer quoted in the `DEFAULT` clause in `SHOW CREATE` statement. Prior to MariaDB 10.2.2, MariaDB quoted numbers.

Index Order

Indexes are sorted and displayed in the following order, which may differ from the order of the CREATE TABLE statement.

- PRIMARY KEY
- UNIQUE keys where all column are NOT NULL
- UNIQUE keys that don't contain partial segments
- Other UNIQUE keys
- LONG UNIQUE keys
- Normal keys
- Fulltext keys

See sql/sql_table.cc for details.

Examples

```
SHOW CREATE TABLE t\G
*************************** 1. row ***************************
       Table: t
Create Table: CREATE TABLE `t` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `s` char(60) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

With sql_quote_show_create off:

```
SHOW CREATE TABLE t\G
*************************** 1. row ***************************
       Table: t
Create Table: CREATE TABLE t (
  id int(11) NOT NULL AUTO_INCREMENT,
  s char(60) DEFAULT NULL,
  PRIMARY KEY (id)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

Unquoted numeric DEFAULTs, from MariaDB 10.2.2:

```
CREATE TABLE td (link TINYINT DEFAULT 1);

SHOW CREATE TABLE td\G
*************************** 1. row ***************************
       Table: td
Create Table: CREATE TABLE `td` (
  `link` tinyint(4) DEFAULT 1
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

Quoted numeric DEFAULTs, until MariaDB 10.2.1:

```
CREATE TABLE td (link TINYINT DEFAULT 1);

SHOW CREATE TABLE td\G
*************************** 1. row ***************************
       Table: td
Create Table: CREATE TABLE `td` (
  `link` tinyint(4) DEFAULT '1'
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

SQL_MODE impacting the output:

```
SELECT @@sql_mode;
+--------------------------------------------------------------------------------------------+
| @@sql_mode                                                                                 |
+--------------------------------------------------------------------------------------------+
| STRICT_TRANS_TABLES,ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION |
+--------------------------------------------------------------------------------------------+

CREATE TABLE `t1` (
       `id` int(11) NOT NULL AUTO_INCREMENT,
       `msg` varchar(100) DEFAULT NULL,
       PRIMARY KEY (`id`)
     ) ENGINE=InnoDB DEFAULT CHARSET=latin1
;

SHOW CREATE TABLE t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE `t1` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `msg` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1

SET SQL_MODE=ORACLE;

SHOW CREATE TABLE t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE "t1" (
  "id" int(11) NOT NULL,
  "msg" varchar(100) DEFAULT NULL,
  PRIMARY KEY ("id")
```

1.1.1.2.1.18 SHOW INDEX

Syntax

```
SHOW {INDEX | INDEXES | KEYS}
 FROM tbl_name [FROM db_name]
 [WHERE expr]
```

Contents

Description

`SHOW INDEX` returns table index information. The format resembles that of the SQLStatistics call in ODBC.

You can use `db_name.tbl_name` as an alternative to the `tbl_name FROM db_name` syntax. These two statements are equivalent:

```
SHOW INDEX FROM mytable FROM mydb;
SHOW INDEX FROM mydb.mytable;
```

`SHOW KEYS` and `SHOW INDEXES` are synonyms for `SHOW INDEX`.

You can also list a table's indexes with the mariadb-show command:

```
mariadb-show -k db_name tbl_name
```

The information_schema.STATISTICS table stores similar information.

The following fields are returned by `SHOW INDEX`.

Field	Description
`Table`	Table name
`Non_unique`	`1` if the index permits duplicate values, `0` if values must be unique.
`Key_name`	Index name. The primary key is always named `PRIMARY`.
`Seq_in_index`	The column's sequence in the index, beginning with `1`.
`Column_name`	Column name.
`Collation`	Either `A`, if the column is sorted in ascending order in the index, or `NULL` if it's not sorted.
`Cardinality`	Estimated number of unique values in the index. The cardinality statistics are calculated at various times, and can help the optimizer make improved decisions.
`Sub_part`	`NULL` if the entire column is included in the index, or the number of included characters if not.
`Packed`	`NULL` if the index is not packed, otherwise how the index is packed.
`Null`	`NULL` if `NULL` values are permitted in the column, an empty string if `NULL`s are not permitted.
`Index_type`	The index type, which can be `BTREE`, `FULLTEXT`, `HASH` or `RTREE`. See Storage Engine Index Types.
`Comment`	Other information, such as whether the index is disabled.
`Index_comment`	Contents of the `COMMENT` attribute when the index was created.
`Ignored`	Whether or not an index will be ignored by the optimizer. See Ignored Indexes. From MariaDB 10.6.0.

The `WHERE` and `LIKE` clauses can be given to select rows using more general conditions, as discussed in Extended SHOW.

Examples

```
CREATE TABLE IF NOT EXISTS `employees_example` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `first_name` varchar(30) NOT NULL,
  `last_name` varchar(40) NOT NULL,
  `position` varchar(25) NOT NULL,
  `home_address` varchar(50) NOT NULL,
  `home_phone` varchar(12) NOT NULL,
  `employee_code` varchar(25) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `employee_code` (`employee_code`),
  KEY `first_name` (`first_name`,`last_name`)
) ENGINE=Aria;

INSERT INTO `employees_example` (`first_name`, `last_name`, `position`, `home_address`,
`home_phone`, `employee_code`)
  VALUES
  ('Mustapha', 'Mond', 'Chief Executive Officer', '692 Promiscuous Plaza', '326-555-3492',
'MM1'),
  ('Henry', 'Foster', 'Store Manager', '314 Savage Circle', '326-555-3847', 'HF1'),
  ('Bernard', 'Marx', 'Cashier', '1240 Ambient Avenue', '326-555-8456', 'BM1'),
  ('Lenina', 'Crowne', 'Cashier', '281 Bumblepuppy Boulevard', '328-555-2349', 'LC1'),
  ('Fanny', 'Crowne', 'Restocker', '1023 Bokanovsky Lane', '326-555-6329', 'FC1'),
  ('Helmholtz', 'Watson', 'Janitor', '944 Soma Court', '329-555-2478', 'HW1');
```

```
SHOW INDEXES FROM employees_example\G
*************************** 1. row ***************************
        Table: employees_example
   Non_unique: 0
     Key_name: PRIMARY
 Seq_in_index: 1
  Column_name: id
    Collation: A
  Cardinality: 6
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
      Ignored: NO
*************************** 2. row ***************************
        Table: employees_example
   Non_unique: 0
     Key_name: employee_code
 Seq_in_index: 1
  Column_name: employee_code
    Collation: A
  Cardinality: 6
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
      Ignored: NO
*************************** 3. row ***************************
        Table: employees_example
   Non_unique: 1
     Key_name: first_name
 Seq_in_index: 1
  Column_name: first_name
    Collation: A
  Cardinality: NULL
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
      Ignored: NO
*************************** 4. row ***************************
        Table: employees_example
   Non_unique: 1
     Key_name: first_name
 Seq_in_index: 2
  Column_name: last_name
    Collation: A
  Cardinality: NULL
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
      Ignored: NO
```

1.1.1.2.1.19 TRUNCATE TABLE

Syntax

```
TRUNCATE [TABLE] tbl_name
  [WAIT n | NOWAIT]
```

Contents

Description

`TRUNCATE TABLE` empties a table completely. It requires the `DROP` privilege. See GRANT.

`tbl_name` can also be specified in the form `db_name . tbl_name` (see Identifier Qualifiers).

Logically, `TRUNCATE TABLE` is equivalent to a DELETE statement that deletes all rows, but there are practical differences under some circumstances.

`TRUNCATE TABLE` will fail for an InnoDB table if any FOREIGN KEY constraints from other tables reference the table, returning the error:

```
ERROR 1701 (42000): Cannot truncate a table referenced in a foreign key constraint
```

Foreign Key constraints between columns in the same table are permitted.

For an InnoDB table, if there are no `FOREIGN KEY` constraints, InnoDB performs fast truncation by dropping the original table and creating an empty one with the same definition, which is much faster than deleting rows one by one. The AUTO_INCREMENT counter is reset by `TRUNCATE TABLE`, regardless of whether there is a `FOREIGN KEY` constraint.

The count of rows affected by `TRUNCATE TABLE` is accurate only when it is mapped to a `DELETE` statement.

For other storage engines, `TRUNCATE TABLE` differs from `DELETE` in the following ways:

- Truncate operations drop and re-create the table, which is much faster than deleting rows one by one, particularly for large tables.
- Truncate operations cause an implicit commit.
- Truncation operations cannot be performed if the session holds an active table lock.
- Truncation operations do not return a meaningful value for the number of deleted rows. The usual result is "0 rows affected," which should be interpreted as "no information."
- As long as the table format file `tbl_name.frm` is valid, the table can be re-created as an empty table with `TRUNCATE TABLE`, even if the data or index files have become corrupted.
- The table handler does not remember the last used AUTO_INCREMENT value, but starts counting from the beginning. This is true even for MyISAM and InnoDB, which normally do not reuse sequence values.
- When used with partitioned tables, `TRUNCATE TABLE` preserves the partitioning; that is, the data and index files are dropped and re-created, while the partition definitions (.par) file is unaffected.
- Since truncation of a table does not make any use of `DELETE`, the `TRUNCATE` statement does not invoke `ON DELETE` triggers.
- `TRUNCATE TABLE` will only reset the values in the Performance Schema summary tables to zero or null, and will not remove the rows.

For the purposes of binary logging and replication, `TRUNCATE TABLE` is treated as DROP TABLE followed by CREATE TABLE (DDL rather than DML).

`TRUNCATE TABLE` does not work on views. Currently, `TRUNCATE TABLE` drops all historical records from a system-versioned table.

WAIT/NOWAIT

Set the lock wait timeout. See WAIT and NOWAIT.

Oracle-mode

Oracle-mode from MariaDB 10.3 permits the optional keywords REUSE STORAGE or DROP STORAGE to be used.

```
TRUNCATE [TABLE] tbl_name [{DROP | REUSE} STORAGE] [WAIT n | NOWAIT]
```

These have no effect on the operation.

Performance

`TRUNCATE TABLE` is faster than DELETE, because it drops and re-creates a table.

With InnoDB, `TRUNCATE TABLE` is slower if innodb_file_per_table=ON is set (the default). This is because `TRUNCATE TABLE` unlinks the underlying tablespace file, which can be an expensive operation. See MDEV-8069 for more details.

The performance issues with innodb_file_per_table=ON can be exacerbated in cases where the InnoDB buffer pool is very large and innodb_adaptive_hash_index=ON is set. In that case, using DROP TABLE followed by CREATE TABLE instead of `TRUNCATE TABLE` may perform better. Setting innodb_adaptive_hash_index=OFF (it defaults to ON before MariaDB 10.5) can also help. In MariaDB 10.2 only, from MariaDB 10.2.19, this performance can also be improved by setting innodb_safe_truncate=OFF. See MDEV-9459 for more details.

Setting innodb_adaptive_hash_index=OFF can also improve `TRUNCATE TABLE` performance in general. See MDEV-16796 for more details.

1.1.1.2.1.20 UPDATE

Syntax

Single-table syntax:

```
UPDATE [LOW_PRIORITY] [IGNORE] table_reference
  [PARTITION (partition_list)]
  [FOR PORTION OF period FROM expr1 TO expr2]
  SET col1={expr1|DEFAULT} [,col2={expr2|DEFAULT}] ...
  [WHERE where_condition]
  [ORDER BY ...]
  [LIMIT row_count]
```

Multiple-table syntax:

```
UPDATE [LOW_PRIORITY] [IGNORE] table_references
    SET col1={expr1|DEFAULT} [, col2={expr2|DEFAULT}] ...
    [WHERE where_condition]
```

Contents

Description

For the single-table syntax, the `UPDATE` statement updates columns of existing rows in the named table with new values. The `SET` clause indicates which columns to modify and the values they should be given. Each value can be given as an expression, or the keyword `DEFAULT` to set a column explicitly to its default value. The `WHERE` clause, if given, specifies the conditions that identify which rows to update. With no `WHERE` clause, all rows are updated. If the ORDER BY clause is specified, the rows are updated in the order that is specified. The LIMIT clause places a limit on the number of rows that can be updated.

Until MariaDB 10.3.2, for the multiple-table syntax, `UPDATE` updates rows in each table named in table_references that satisfy the conditions. In this case, ORDER BY and LIMIT cannot be used. This restriction was lifted in MariaDB 10.3.2 and both clauses can be used with multiple-table updates. An `UPDATE` can also reference tables which are located in different databases; see Identifier Qualifiers for the syntax.

`where_condition` is an expression that evaluates to true for each row to be updated.

`table_references` and `where_condition` are as specified as described in SELECT.

For single-table updates, assignments are evaluated in left-to-right order, while for multi-table updates, there is no guarantee of a particular order. If the `SIMULTANEOUS_ASSIGNMENT` sql_mode (available from MariaDB 10.3.5) is set, UPDATE statements evaluate all assignments simultaneously.

You need the `UPDATE` privilege only for columns referenced in an `UPDATE` that are actually updated. You need only the SELECT privilege for any columns that are read but not modified. See GRANT.

The `UPDATE` statement supports the following modifiers:

- If you use the `LOW_PRIORITY` keyword, execution of the `UPDATE` is delayed until no other clients are reading from the table. This affects only storage engines that use only table-level locking (MyISAM, MEMORY, MERGE). See HIGH_PRIORITY and LOW_PRIORITY clauses for details.
- If you use the `IGNORE` keyword, the update statement does not abort even if errors occur during the update. Rows for which duplicate-key conflicts occur are not updated. Rows for which columns are updated to values that would cause data conversion errors are updated to the closest valid values instead.

PARTITION

See Partition Pruning and Selection for details.

FOR PORTION OF

MariaDB starting with 10.4.3

See Application Time Periods - Updating by Portion.

UPDATE Statements With the Same Source and Target

MariaDB starting with 10.3.2

From MariaDB 10.3.2, UPDATE statements may have the same source and target.

For example, given the following table:

```
DROP TABLE t1;
CREATE TABLE t1 (c1 INT, c2 INT);
INSERT INTO t1 VALUES (10,10), (20,20);
```

Until MariaDB 10.3.1, the following UPDATE statement would not work:

```
UPDATE t1 SET c1=c1+1 WHERE c2=(SELECT MAX(c2) FROM t1);
ERROR 1093 (HY000): Table 't1' is specified twice,
  both as a target for 'UPDATE' and as a separate source for data
```

From MariaDB 10.3.2, the statement executes successfully:

```
UPDATE t1 SET c1=c1+1 WHERE c2=(SELECT MAX(c2) FROM t1);

SELECT * FROM t1;
+------+------+
| c1   | c2   |
+------+------+
|   10 |   10 |
|   21 |   20 |
+------+------+
```

Example

Single-table syntax:

```
UPDATE table_name SET column1 = value1, column2 = value2 WHERE id=100;
```

Multiple-table syntax:

```
UPDATE tab1, tab2 SET tab1.column1 = value1, tab1.column2 = value2 WHERE tab1.id = tab2.id;
```

1.1.1.2.1.21 IGNORE

The `IGNORE` option tells the server to ignore some common errors.

`IGNORE` can be used with the following statements:

- DELETE
- INSERT (see also INSERT IGNORE)
- LOAD DATA INFILE
- UPDATE
- ALTER TABLE
- CREATE TABLE ... SELECT
- INSERT ... SELECT

The logic used:

- Variables out of ranges are replaced with the maximum/minimum value.
- SQL_MODEs `STRICT_TRANS_TABLES`, `STRICT_ALL_TABLES`, `NO_ZERO_IN_DATE`, `NO_ZERO_DATE` are ignored.
- Inserting `NULL` in a `NOT NULL` field will insert 0 (in a numerical field), 0000-00-00 (in a date field) or an empty string (in a character field).
- Rows that cause a duplicate key error or break a foreign key constraint are not inserted, updated, or deleted.

The following errors are ignored:

Error number	Symbolic error name	Description
1022	ER_DUP_KEY	Can't write; duplicate key in table '%s'
1048	ER_BAD_NULL_ERROR	Column '%s' cannot be null
1062	ER_DUP_ENTRY	Duplicate entry '%s' for key %d
1242	ER_SUBQUERY_NO_1_ROW	Subquery returns more than 1 row
1264	ER_WARN_DATA_OUT_OF_RANGE	Out of range value for column '%s' at row %ld
1265	WARN_DATA_TRUNCATED	Data truncated for column '%s' at row %ld
1292	ER_TRUNCATED_WRONG_VALUE	Truncated incorrect %s value: '%s'
1366	ER_TRUNCATED_WRONG_VALUE_FOR_FIELD	Incorrect integer value
1369	ER_VIEW_CHECK_FAILED	CHECK OPTION failed '%s.%s'
1451	ER_ROW_IS_REFERENCED_2	Cannot delete or update a parent row
1452	ER_NO_REFERENCED_ROW_2	Cannot add or update a child row: a foreign key constraint fails (%s)
1526	ER_NO_PARTITION_FOR_GIVEN_VALUE	Table has no partition for value %s
1586	ER_DUP_ENTRY_WITH_KEY_NAME	Duplicate entry '%s' for key '%s'
1591	ER_NO_PARTITION_FOR_GIVEN_VALUE_SILENT	Table has no partition for some existing values
1748	ER_ROW_DOES_NOT_MATCH_GIVEN_PARTITION_SET	Found a row not matching the given partition set

Ignored errors normally generate a warning.

A property of the `IGNORE` clause consists in causing transactional engines and non-transactional engines (like InnoDB and Aria) to behave the same way. For example, normally a multi-row insert which tries to violate a `UNIQUE` contraint is completely rolled back on InnoDB, but might be partially executed on Aria. With the `IGNORE` clause, the statement will be partially executed in both engines.

Duplicate key errors also generate warnings. The OLD_MODE server variable can be used to prevent this.

1.1.7.1 System-Versioned Tables

1.1.1.2.2 ANALYZE and EXPLAIN Statements

ANALYZE FORMAT=JSON

Mix of the EXPLAIN FORMAT=JSON and ANALYZE statement features.

ANALYZE FORMAT=JSON Examples

Examples with ANALYZE FORMAT=JSON.

ANALYZE Statement

Invokes the optimizer, executes the statement, and then produces EXPLAIN output.

EXPLAIN

EXPLAIN returns information about index usage, as well as being a synonym for DESCRIBE.

EXPLAIN ANALYZE

Old implementation, now ANALYZE statement

EXPLAIN FORMAT=JSON

Variant of EXPLAIN that produces output in JSON form

SHOW EXPLAIN

Shows an execution plan for a running query.

Using Buffer UPDATE Algorithm

Explanation of UPDATE's "Using Buffer" algorithm.

1.1.1.2.2.1 ANALYZE FORMAT=JSON

Contents

`ANALYZE FORMAT=JSON` is a mix of the EXPLAIN FORMAT=JSON and ANALYZE statement features. The `ANALYZE FORMAT=JSON $statement` will execute `$statement`, and then print the output of `EXPLAIN FORMAT=JSON`, amended with data from the query execution.

Basic Execution Data

You can get the following also from tabular `ANALYZE` statement form:

- **`r_rows`** is provided for any node that reads rows. It shows how many rows were read, on average
- **`r_filtered`** is provided whenever there is a condition that is checked. It shows the percentage of rows left after checking the condition.

Advanced Execution Data

The most important data not available in the regular tabular `ANALYZE` statement are:

- **`r_loops`** field. This shows how many times the node was executed. Most query plan elements have this field.
- **`r_total_time_ms`** field. It shows how much time in total, in milliseconds, was spent executing this node. If the node has subnodes, their execution time is included.
- **`r_buffer_size`** field. Query plan nodes that make use of buffers report the size of buffer that was was used.

InnoDB engine statistics

Starting from MariaDB 10.6.15, MariaDB 10.8.8, MariaDB 10.9.8, MariaDB 10.10.6, MariaDB 10.11.5, MariaDB 11.0.3, MariaDB 11.1.2 and MariaDB 11.2.1 (MDEV-31577), the following statistics are reported for InnoDB tables:

```
      "r_engine_stats": {
        "pages_accessed":  integer,
        "pages_read_count": integer,
        "pages_read_time_ms": double,
        "old_rows_read": integer
      }
```

Only non-zero members are printed.

- `pages_accessed` is the total number of buffer pool pages accessed when reading this table

- `pages_read_count` is the number of pages that InnoDB had to read from disk for this table. (If the query touches "hot" data in the InnoDB buffer pool, this value will be 0 and not present)
- `pages_read_time_ms` is the total time spent reading the table.
- `old_rows_read` is the number of old row versions that InnoDB had to read. Old row version is the version of the row that is not visible to this transaction.

SHOW ANALYZE FORMAT=JSON

MariaDB starting with 10.9

`SHOW ANALYZE FORMAT=JSON for <connection_id>` extends `ANALYZE [FORMAT=JSON] <select>` to allow one to analyze a query currently running in another connection.

Data About Individual Query Plan Nodes

- **`filesort`** node reports whether sorting was done with `LIMIT n` parameter, and how many rows were in the sort result.
- **`block-nl-join`** node has `r_loops` field, which allows to tell whether `Using join buffer` was efficient
- **`range-checked-for-each-record`** reports counters that show the result of the check.
- **`expression-cache`** is used for subqueries, and it reports how many times the cache was used, and what cache hit ratio was.
- **`union_result`** node has `r_rows` so one can see how many rows were produced after UNION operation
- and so forth

Use Cases

See Examples of ANALYZE FORMAT=JSON.

1.1.1.2.2.2 ANALYZE FORMAT=JSON Examples

Example #1

Customers who have ordered more than 1M goods.

```
ANALYZE FORMAT=JSON
SELECT COUNT(*)
FROM customer
WHERE
  (SELECT SUM(o_totalprice) FROM orders WHERE o_custkey=c_custkey) > 1000*1000;
```

The query takes 40 seconds over cold cache

```
EXPLAIN: {
  "query_block": {
    "select_id": 1,
    "r_loops": 1,
    "r_total_time_ms": 39872,
    "table": {
      "table_name": "customer",
      "access_type": "index",
      "key": "i_c_nationkey",
      "key_length": "5",
      "used_key_parts": ["c_nationkey"],
      "r_loops": 1,
      "rows": 150303,
      "r_rows": 150000,
      "r_total_time_ms": 270.3,
      "filtered": 100,
      "r_filtered": 60.691,
      "attached_condition": "((subquery#2) > <cache>((1000 * 1000)))",
      "using_index": true
    },
    "subqueries": [
      {
        "query_block": {
          "select_id": 2,
          "r_loops": 150000,
          "r_total_time_ms": 39531,
          "table": {
            "table_name": "orders",
            "access_type": "ref",
            "possible_keys": ["i_o_custkey"],
            "key": "i_o_custkey",
            "key_length": "5",
            "used_key_parts": ["o_custkey"],
            "ref": ["dbt3sf1.customer.c_custkey"],
            "r_loops": 150000,
            "rows": 7,
            "r_rows": 10,
            "r_total_time_ms": 39208,
            "filtered": 100,
            "r_filtered": 100
          }
        }
      }
    ]
  }
}
```

`ANALYZE` shows that 39.2 seconds were spent in the subquery, which was executed 150K times (for every row of outer table).

1.1.1.2.2.3 ANALYZE Statement

Contents

Description

The `ANALYZE statement` is similar to the `EXPLAIN statement`. `ANALYZE statement` will invoke the optimizer, execute

the statement, and then produce `EXPLAIN` output instead of the result set. The `EXPLAIN` output will be annotated with statistics from statement execution.

This lets one check how close the optimizer's estimates about the query plan are to the reality. `ANALYZE` produces an overview, while the ANALYZE FORMAT=JSON command provides a more detailed view of the query plan and the query execution.

The syntax is

```
ANALYZE explainable_statement;
```

where the statement is any statement for which one can run EXPLAIN.

Command Output

Consider an example:

```
ANALYZE SELECT * FROM tbl1
WHERE key1
  BETWEEN 10 AND 200 AND
  col1 LIKE 'foo%'\G
```

```
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: tbl1
         type: range
possible_keys: key1
          key: key1
      key_len: 5
          ref: NULL
         rows: 181
       r_rows: 181
     filtered: 100.00
   r_filtered: 10.50
        Extra: Using index condition; Using where
```

Compared to `EXPLAIN`, `ANALYZE` produces two extra columns:

- **`r_rows`** is an observation-based counterpart of the **rows** column. It shows how many rows were actually read from the table.
- **`r_filtered`** is an observation-based counterpart of the **filtered** column. It shows which fraction of rows was left after applying the WHERE condition.

Interpreting the Output

Joins

Let's consider a more complicated example.

```
ANALYZE SELECT *
FROM orders, customer
WHERE
  customer.c_custkey=orders.o_custkey AND
  customer.c_acctbal < 0 AND
  orders.o_totalprice > 200*1000
```

```
+----+-------------+----------+------+---------------+-------------+---------+---------------
-----+--------+--------+----------+------------+-------------+
| id | select_type | table    | type | possible_keys | key         | key_len | ref
| rows   | r_rows | filtered | r_filtered | Extra       |
+----+-------------+----------+------+---------------+-------------+---------+---------------
-----+--------+--------+----------+------------+-------------+
|  1 | SIMPLE      | customer | ALL  | PRIMARY,...   | NULL        | NULL    | NULL
| 149095 | 150000 |    18.08 |       9.13 | Using where |
|  1 | SIMPLE      | orders   | ref  | i_o_custkey   | i_o_custkey | 5       |
customer.c_custkey |      7 |     10 |   100.00 |      30.03 | Using where |
+----+-------------+----------+------+---------------+-------------+---------+---------------
-----+--------+--------+----------+------------+-------------+
```

Here, one can see that

- For table customer, **customer.rows=149095, customer.r_rows=150000**. The estimate for number of rows we will read was fairly precise
- **customer.filtered=18.08, customer.r_filtered=9.13**. The optimizer somewhat overestimated the number of records that will match selectivity of condition attached to `customer` table (in general, when you have a full scan and r_filtered is less than 15%, it's time to consider adding an appropriate index).
- For table orders, **orders.rows=7, orders.r_rows=10**. This means that on average, there are 7 orders for a given c_custkey, but in our case there were 10, which is close to the expectation (when this number is consistently far from the expectation, it may be time to run ANALYZE TABLE, or even edit the table statistics manually to get better query plans).
- **orders.filtered=100, orders.r_filtered=30.03**. The optimizer didn't have any way to estimate which fraction of records will be left after it checks the condition that is attached to table orders (it's orders.o_totalprice > 200*1000). So, it used 100%. In reality, it is 30%. 30% is typically not selective enough to warrant adding new indexes. For joins with many tables, it might be worth to collect and use column statistics for columns in question, this may help the optimizer to pick a better query plan.

Meaning of NULL in r_rows and r_filtered

Let's modify the previous example slightly

```
ANALYZE SELECT *
FROM orders, customer
WHERE
  customer.c_custkey=orders.o_custkey AND
  customer.c_acctbal < -0 AND
  customer.c_comment LIKE '%foo%' AND
  orders.o_totalprice > 200*1000;
```

```
+----+-------------+----------+------+---------------+-------------+---------+---------------
-----+--------+--------+----------+------------+-------------+
| id | select_type | table    | type | possible_keys | key         | key_len | ref
| rows   | r_rows | filtered | r_filtered | Extra       |
+----+-------------+----------+------+---------------+-------------+---------+---------------
-----+--------+--------+----------+------------+-------------+
|  1 | SIMPLE      | customer | ALL  | PRIMARY,...   | NULL        | NULL    | NULL
| 149095 | 150000 |    18.08 |       0.00 | Using where |
|  1 | SIMPLE      | orders   | ref  | i_o_custkey   | i_o_custkey | 5       |
customer.c_custkey |      7 |   NULL |   100.00 |       NULL | Using where |
+----+-------------+----------+------+---------------+-------------+---------+---------------
-----+--------+--------+----------+------------+-------------+
```

Here, one can see that **orders.r_rows=NULL** and **orders.r_filtered=NULL**. This means that table orders was not scanned even once. Indeed, we can also see customer.r_filtered=0.00. This shows that a part of WHERE attached to table `customer` was never satisfied (or, satisfied in less than 0.01% of cases).

ANALYZE FORMAT=JSON

ANALYZE FORMAT=JSON produces JSON output. It produces much more information than tabular `ANALYZE`.

Notes

- `ANALYZE UPDATE` or `ANALYZE DELETE` will actually make updates/deletes (`ANALYZE SELECT` will perform the

select operation and then discard the resultset).
- PostgreSQL has a similar command, `EXPLAIN ANALYZE`.
- The EXPLAIN in the slow query log feature allows MariaDB to have `ANALYZE` output of slow queries printed into the slow query log (see MDEV-6388).

1.1.1.2.2.4 EXPLAIN

Syntax

```
EXPLAIN tbl_name [col_name | wild]
```

Or

```
EXPLAIN [EXTENDED | PARTITIONS | FORMAT=JSON]
  {SELECT select_options | UPDATE update_options | DELETE delete_options}
```

Contents

Description

The `EXPLAIN` statement can be used either as a synonym for DESCRIBE or as a way to obtain information about how MariaDB executes a `SELECT`, `UPDATE` or `DELETE` statement:

- `'EXPLAIN tbl_name'` is synonymous with `'DESCRIBE tbl_name'` or `'SHOW COLUMNS FROM tbl_name'`.
- When you precede a `SELECT`, `UPDATE` or a `DELETE` statement with the keyword `EXPLAIN`, MariaDB displays information from the optimizer about the query execution plan. That is, MariaDB explains how it would process the `SELECT`, `UPDATE` or `DELETE`, including information about how tables are joined and in which order. `EXPLAIN EXTENDED` can be used to provide additional information.
- `EXPLAIN PARTITIONS` is useful only when examining queries involving partitioned tables. For details, see Partition pruning and selection.
- ANALYZE statement performs the query as well as producing EXPLAIN output, and provides actual as well as estimated statistics.
- `EXPLAIN` output can be printed in the slow query log. See EXPLAIN in the Slow Query Log for details.

SHOW EXPLAIN shows the output of a running statement. In some cases, its output can be closer to reality than `EXPLAIN`.

The ANALYZE statement runs a statement and returns information about its execution plan. It also shows additional columns, to check how much the optimizer's estimation about filtering and found rows are close to reality.

There is an online EXPLAIN Analyzer that you can use to share `EXPLAIN` and `EXPLAIN EXTENDED` output with others.

> `EXPLAIN` can acquire metadata locks in the same way that `SELECT` does, as it needs to know table metadata and, sometimes, data as well.

Columns in EXPLAIN ... SELECT

Column name	Description
`id`	Sequence number that shows in which order tables are joined.
`select_type`	What kind of `SELECT` the table comes from.
`table`	Alias name of table. Materialized temporary tables for sub queries are named <subquery#>

type	How rows are found from the table (join type).
possible_keys	keys in table that could be used to find rows in the table
key	The name of the key that is used to retrieve rows. NULL is no key was used.
key_len	How many bytes of the key that was used (shows if we are using only parts of the multi-column key).
ref	The reference that is used as the key value.
rows	An estimate of how many rows we will find in the table for each key lookup.
Extra	Extra information about this join.

Here are descriptions of the values for some of the more complex columns in EXPLAIN ... SELECT:

"Select_type" Column

The select_type column can have the following values:

Value	Description	Comment
DEPENDENT SUBQUERY	The SUBQUERY is DEPENDENT.	
DEPENDENT UNION	The UNION is DEPENDENT.	
DERIVED	The SELECT is DERIVED from the PRIMARY.	
MATERIALIZED	The SUBQUERY is MATERIALIZED.	Materialized tables will be populated at first access and will be accessed by the primary key (= one key lookup). Number of rows in EXPLAIN shows the cost of populating the table
PRIMARY	The SELECT is in the outermost query, but there is also a SUBQUERY within it.	
SIMPLE	It is a simple SELECT query without any SUBQUERY or UNION.	
SUBQUERY	The SELECT is a SUBQUERY of the PRIMARY.	
UNCACHEABLE SUBQUERY	The SUBQUERY is UNCACHEABLE.	
UNCACHEABLE UNION	The UNION is UNCACHEABLE.	
UNION	The SELECT is a UNION of the PRIMARY.	
UNION RESULT	The result of the UNION.	
LATERAL DERIVED	The SELECT uses a Lateral Derived optimization	

"Type" Column

This column contains information on how the table is accessed.

Value	Description
ALL	A full table scan is done for the table (all rows are read). This is bad if the table is large and the table is joined against a previous table! This happens when the optimizer could not find any usable index to access rows.
const	There is only one possibly matching row in the table. The row is read before the optimization phase and all columns in the table are treated as constants.
eq_ref	A unique index is used to find the rows. This is the best possible plan to find the row.

`fulltext`	A fulltext index is used to access the rows.
`index_merge`	A 'range' access is done for for several index and the found rows are merged. The key column shows which keys are used.
`index_subquery`	This is similar as ref, but used for sub queries that are transformed to key lookups.
`index`	A full scan over the used index. Better than ALL but still bad if index is large and the table is joined against a previous table.
`range`	The table will be accessed with a key over one or more value ranges.
`ref_or_null`	Like 'ref' but in addition another search for the 'null' value is done if the first value was not found. This happens usually with sub queries.
`ref`	A non unique index or prefix of an unique index is used to find the rows. Good if the prefix doesn't match many rows.
`system`	The table has 0 or 1 rows.
`unique_subquery`	This is similar as eq_ref, but used for sub queries that are transformed to key lookups

"Extra" Column

This column consists of one or more of the following values, separated by ';'

Note that some of these values are detected after the optimization phase.

The optimization phase can do the following changes to the `WHERE` clause:

- Add the expressions from the `ON` and `USING` clauses to the `WHERE` clause.
- Constant propagation: If there is `column=constant` , replace all column instances with this constant.
- Replace all columns from ' `const` ' tables with their values.
- Remove the used key columns from the `WHERE` (as this will be tested as part of the key lookup).
- Remove impossible constant sub expressions. For example `WHERE '(a=1 and a=2) OR b=1'` becomes `'b=1'` .
- Replace columns with other columns that has identical values: Example: `WHERE a=b` and `a=c` may be treated as `'WHERE a=b and a=c and b=c'` .
- Add extra conditions to detect impossible row conditions earlier. This happens mainly with `OUTER JOIN` where we in some cases add detection of `NULL` values in the `WHERE` (Part of ' `Not exists` ' optimization). This can cause an unexpected ' `Using where` ' in the Extra column.
- For each table level we remove expressions that have already been tested when we read the previous row. Example: When joining tables `t1` with `t2` using the following `WHERE 't1.a=1 and t1.a=t2.b'` , we don't have to test `'t1.a=1'` when checking rows in `t2` as we already know that this expression is true.

Value	Description
`const row not found`	The table was a system table (a table with should exactly one row), but no row was found.
`Distinct`	If distinct optimization (remove duplicates) was used. This is marked only for the last table in the `SELECT` .
`Full scan on NULL key`	The table is a part of the sub query and if the value that is used to match the sub query will be `NULL` , we will do a full table scan.
`Impossible HAVING`	The used `HAVING` clause is always false so the `SELECT` will return no rows.
`Impossible WHERE noticed after reading const tables.`	The used `WHERE` clause is always false so the `SELECT` will return no rows. This case was detected after we had read all 'const' tables and used the column values as constant in the `WHERE` clause. For example: `WHERE const_column=5` and `const_column` had a value of 4.
`Impossible WHERE`	The used `WHERE` clause is always false so the `SELECT` will return no rows. For example: `WHERE 1=2`
`No matching min/max row`	During early optimization of `MIN()` / `MAX()` values it was detected that no row could match the `WHERE` clause. The `MIN()` / `MAX()` function will return `NULL` .
`no matching row in const table`	The table was a const table (a table with only one possible matching row), but no row was found.
`No tables used`	The `SELECT` was a sub query that did not use any tables. For example a there was no `FROM` clause or a `FROM DUAL` clause.

`Not exists`	Stop searching after more row if we find one single matching row. This optimization is used with `LEFT JOIN` where one is explicitly searching for rows that doesn't exists in the `LEFT JOIN TABLE`. Example: `SELECT * FROM t1 LEFT JOIN t2 on (...) WHERE t2.not_null_column IS NULL`. As `t2.not_null_column` can only be `NULL` if there was no matching row for on condition, we can stop searching if we find a single matching row.
`Open_frm_only`	For `information_schema` tables. Only the `frm` (table definition file was opened) was opened for each matching row.
`Open_full_table`	For `information_schema` tables. A full table open for each matching row is done to retrieve the requested information. (Slow)
`Open_trigger_only`	For `information_schema` tables. Only the trigger file definition was opened for each matching row.
`Range checked for each record (index map: ...)`	This only happens when there was no good default index to use but there may some index that could be used when we can treat all columns from previous table as constants. For each row combination the optimizer will decide which index to use (if any) to fetch a row from this table. This is not fast, but faster than a full table scan that is the only other choice. The index map is a bitmask that shows which index are considered for each row condition.
`Scanned 0/1/all databases`	For `information_schema` tables. Shows how many times we had to do a directory scan.
`Select tables optimized away`	All tables in the join was optimized away. This happens when we are only using `COUNT(*)`, `MIN()` and `MAX()` functions in the `SELECT` and we where able to replace all of these with constants.
`Skip_open_table`	For `information_schema` tables. The queried table didn't need to be opened.
`unique row not found`	The table was detected to be a const table (a table with only one possible matching row) during the early optimization phase, but no row was found.
`Using filesort`	Filesort is needed to resolve the query. This means an extra phase where we first collect all columns to sort, sort them with a disk based merge sort and then use the sorted set to retrieve the rows in sorted order. If the column set is small, we store all the columns in the sort file to not have to go to the database to retrieve them again.
`Using index`	Only the index is used to retrieve the needed information from the table. There is no need to perform an extra seek to retrieve the actual record.
`Using index condition`	Like '`Using where`' but the where condition is pushed down to the table engine for internal optimization at the index level.
`Using index condition(BKA)`	Like '`Using index condition`' but in addition we use batch key access to retrieve rows.
`Using index for group-by`	The index is being used to resolve a `GROUP BY` or `DISTINCT` query. The rows are not read. This is very efficient if the table has a lot of identical index entries as duplicates are quickly jumped over.
`Using intersect(...)`	For `index_merge` joins. Shows which index are part of the intersect.
`Using join buffer`	We store previous row combinations in a row buffer to be able to match each row against all of the rows combinations in the join buffer at one go.
`Using sort_union(...)`	For `index_merge` joins. Shows which index are part of the union.
`Using temporary`	A temporary table is created to hold the result. This typically happens if you are using `GROUP BY`, `DISTINCT` or `ORDER BY`.
`Using where`	A `WHERE` expression (in additional to the possible key lookup) is used to check if the row should be accepted. If you don't have 'Using where' together with a join type of `ALL`, you are probably doing something wrong!
`Using where with pushed condition`	Like '`Using where`' but the where condition is pushed down to the table engine for internal optimization at the row level.
`Using buffer`	The `UPDATE` statement will first buffer the rows, and then run the updates, rather than do updates on the fly. See Using Buffer UPDATE Algorithm for a detailed explanation.

EXPLAIN EXTENDED

The `EXTENDED` keyword adds another column, *filtered*, to the output. This is a percentage estimate of the table rows that will be filtered by the condition.

An `EXPLAIN EXTENDED` will always throw a warning, as it adds extra *Message* information to a subsequent `SHOW WARNINGS` statement. This includes what the `SELECT` query would look like after optimizing and rewriting rules are applied and how the optimizer qualifies columns and tables.

Examples

As synonym for `DESCRIBE` or `SHOW COLUMNS FROM`:

```
DESCRIBE city;
+------------+----------+------+-----+---------+----------------+
| Field      | Type     | Null | Key | Default | Extra          |
+------------+----------+------+-----+---------+----------------+
| Id         | int(11)  | NO   | PRI | NULL    | auto_increment |
| Name       | char(35) | YES  |     | NULL    |                |
| Country    | char(3)  | NO   | UNI |         |                |
| District   | char(20) | YES  | MUL |         |                |
| Population | int(11)  | YES  |     | NULL    |                |
+------------+----------+------+-----+---------+----------------+
```

A simple set of examples to see how `EXPLAIN` can identify poor index usage:

```
CREATE TABLE IF NOT EXISTS `employees_example` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `first_name` varchar(30) NOT NULL,
  `last_name` varchar(40) NOT NULL,
  `position` varchar(25) NOT NULL,
  `home_address` varchar(50) NOT NULL,
  `home_phone` varchar(12) NOT NULL,
  `employee_code` varchar(25) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `employee_code` (`employee_code`),
  KEY `first_name` (`first_name`,`last_name`)
) ENGINE=Aria;

INSERT INTO `employees_example` (`first_name`, `last_name`, `position`, `home_address`,
`home_phone`, `employee_code`)
  VALUES
  ('Mustapha', 'Mond', 'Chief Executive Officer', '692 Promiscuous Plaza', '326-555-3492',
'MM1'),
  ('Henry', 'Foster', 'Store Manager', '314 Savage Circle', '326-555-3847', 'HF1'),
  ('Bernard', 'Marx', 'Cashier', '1240 Ambient Avenue', '326-555-8456', 'BM1'),
  ('Lenina', 'Crowne', 'Cashier', '281 Bumblepuppy Boulevard', '328-555-2349', 'LC1'),
  ('Fanny', 'Crowne', 'Restocker', '1023 Bokanovsky Lane', '326-555-6329', 'FC1'),
  ('Helmholtz', 'Watson', 'Janitor', '944 Soma Court', '329-555-2478', 'HW1');

SHOW INDEXES FROM employees_example;
+------------------+------------+---------------+--------------+---------------+-----------
+-------------+----------+--------+------+------------+---------+---------------+
| Table            | Non_unique | Key_name      | Seq_in_index | Column_name   | Collation |
Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment |
+------------------+------------+---------------+--------------+---------------+-----------
+-------------+----------+--------+------+------------+---------+---------------+
| employees_example |          0 | PRIMARY       |            1 | id            | A         |
7 |     NULL | NULL   |      | BTREE      |         |               |
| employees_example |          0 | employee_code |            1 | employee_code | A         |
7 |     NULL | NULL   |      | BTREE      |         |               |
| employees_example |          1 | first_name    |            1 | first_name    | A         |
NULL |     NULL | NULL   |      | BTREE      |         |               |
| employees_example |          1 | first_name    |            2 | last_name     | A         |
NULL |     NULL | NULL   |      | BTREE      |         |               |
+------------------+------------+---------------+--------------+---------------+-----------
+-------------+----------+--------+------+------------+---------+---------------+
```

`SELECT` on a primary key:

```
EXPLAIN SELECT * FROM employees_example WHERE id=1;
+------+-------------+------------------+-------+---------------+---------+---------+-------
+------+-------+
| id   | select_type | table            | type  | possible_keys | key     | key_len | ref   |
rows | Extra |
+------+-------------+------------------+-------+---------------+---------+---------+-------
+------+-------+
|    1 | SIMPLE      | employees_example | const | PRIMARY       | PRIMARY | 4       | const |
1 |       |
+------+-------------+------------------+-------+---------------+---------+---------+-------
+------+-------+
```

The type is *const*, which means that only one possible result could be returned. Now, returning the same record but searching by their phone number:

```
EXPLAIN SELECT * FROM employees_example WHERE home_phone='326-555-3492';
+------+-------------+------------------+------+---------------+------+---------+------+----
--+-------------+
| id   | select_type | table            | type | possible_keys | key  | key_len | ref  | rows
| Extra       |
+------+-------------+------------------+------+---------------+------+---------+------+----
--+-------------+
|    1 | SIMPLE      | employees_example | ALL  | NULL          | NULL | NULL    | NULL |    6
| Using where |
+------+-------------+------------------+------+---------------+------+---------+------+----
--+-------------+
```

Here, the type is *All*, which means no index could be used. Looking at the rows count, a full table scan (all six rows) had to be performed in order to retrieve the record. If it's a requirement to search by phone number, an index will have to be created.

SHOW EXPLAIN example:

```
SHOW EXPLAIN FOR 1;
+------+-------------+-------+-------+---------------+------+---------+------+---------+---------
| id   | select_type | table | type  | possible_keys | key  | key_len | ref  | rows    | Extra
+------+-------------+-------+-------+---------------+------+---------+------+---------+---------
|    1 | SIMPLE      | tbl   | index | NULL          | a    | 5       | NULL | 1000107 | Using in
+------+-------------+-------+-------+---------------+------+---------+------+---------+---------
1 row in set, 1 warning (0.00 sec)
```

Example of `ref_or_null` Optimization

```
SELECT * FROM table_name
  WHERE key_column=expr OR key_column IS NULL;
```

`ref_or_null` is something that often happens when you use subqueries with `NOT IN` as then one has to do an extra check for `NULL` values if the first value didn't have a matching row.

1.1.1.2.2.5 EXPLAIN ANALYZE

The syntax for the `EXPLAIN ANALYZE` feature was changed to `ANALYZE statement` , available since MariaDB 10.1.0. See ANALYZE statement.

1.1.1.2.2.6 EXPLAIN FORMAT=JSON

Contents

Synopsis

`EXPLAIN FORMAT=JSON` is a variant of EXPLAIN command that produces output in JSON form. The output always has one row which has only one column titled " `JSON` ". The contents are a JSON representation of the query plan, formatted for readability:

```
EXPLAIN FORMAT=JSON SELECT * FROM t1 WHERE col1=1\G
```

```
*************************** 1. row ***************************
EXPLAIN: {
  "query_block": {
    "select_id": 1,
    "table": {
      "table_name": "t1",
      "access_type": "ALL",
      "rows": 1000,
      "filtered": 100,
      "attached_condition": "(t1.col1 = 1)"
    }
  }
}
```

Output is different from MySQL

The output of MariaDB's `EXPLAIN FORMAT=JSON` is different from `EXPLAIN FORMAT=JSON` in MySQL.The reasons for that are:

- MySQL's output has deficiencies. Some are listed here: EXPLAIN FORMAT=JSON in MySQL
- The output of MySQL's `EXPLAIN FORMAT=JSON` is not defined. Even MySQL Workbench has trouble parsing it (see this blog post).
- MariaDB has query optimizations that MySQL does not have. Ergo, MariaDB generates query plans that MySQL does not generate.

A (as yet incomplete) list of how MariaDB's output is different from MySQL can be found here: EXPLAIN FORMAT=JSON differences from MySQL.

Output Format

TODO: MariaDB's output format description.

1.1.1.2.2.7 SHOW EXPLAIN

Contents

Syntax

```
SHOW EXPLAIN [FORMAT=JSON] FOR <connection_id>;
EXPLAIN [FORMAT=JSON] FOR CONNECTION <connection_id>;
```

Description

The `SHOW EXPLAIN` command allows one to get an EXPLAIN (that is, a description of a query plan) of a query running in a certain connection.

```
SHOW EXPLAIN FOR <connection_id>;
```

will produce an `EXPLAIN` output for the query that connection number `connection_id` is running. The connection id can be obtained with SHOW PROCESSLIST.

```
SHOW EXPLAIN FOR 1;
+------+-------------+-------+-------+---------------+------+---------+------+---------+---------
| id   | select_type | table | type  | possible_keys | key  | key_len | ref  | rows    | Extra
+------+-------------+-------+-------+---------------+------+---------+------+---------+---------
|    1 | SIMPLE      | tbl   | index | NULL          | a    | 5       | NULL | 1000107 | Using in
+------+-------------+-------+-------+---------------+------+---------+------+---------+---------
1 row in set, 1 warning (0.00 sec)
```

The output is always accompanied with a warning which shows the query the target connection is running (this shows what the `EXPLAIN` is for):

```
SHOW WARNINGS;
+-------+------+-----------------------+
| Level | Code | Message               |
+-------+------+-----------------------+
| Note  | 1003 | select sum(a) from tbl |
+-------+------+-----------------------+
1 row in set (0.00 sec)
```

EXPLAIN FOR CONNECTION

MariaDB starting with 10.9

The `EXPLAIN FOR CONNECTION` syntax was added for MySQL compatibility.

FORMAT=JSON

MariaDB starting with 10.9

`SHOW EXPLAIN [FORMAT=JSON] FOR <connection_id>` extends `SHOW EXPLAIN` to return more detailed JSON output.

Possible Errors

The output can be only produced if the target connection is *currently* running a query, which has a ready query plan. If this is not the case, the output will be:

```
SHOW EXPLAIN FOR 2;
ERROR 1932 (HY000): Target is not running an EXPLAINable command
```

You will get this error when:

- the target connection is not running a command for which one can run `EXPLAIN`
- the target connection is running a command for which one can run `EXPLAIN` , but
  - there is no query plan yet (for example, tables are open and locks are acquired before the query plan is produced)

Differences Between SHOW EXPLAIN and EXPLAIN Outputs

Background

In MySQL, `EXPLAIN` execution takes a slightly different route from the way the real query (typically the `SELECT`) is optimized. This is unfortunate, and has caused a number of bugs in `EXPLAIN` . (For example, see MDEV-326, MDEV-410, and lp:1013343. lp:992942 is not directly about `EXPLAIN` , but it also would not have existed if MySQL didn't try to delete parts of a query plan in the middle of the query)

`SHOW EXPLAIN` examines a running `SELECT` , and hence its output may be slightly different from what `EXPLAIN SELECT` would produce. We did our best to make sure that either the difference is negligible, or `SHOW EXPLAIN` 's output is closer to reality than `EXPLAIN` 's output.

List of Recorded Differences

- `SHOW EXPLAIN` may have Extra='`no matching row in const table`', where `EXPLAIN` would produce Extra='`Impossible WHERE ...`'
- For queries with subqueries, `SHOW EXPLAIN` may print `select_type==PRIMARY` where regular `EXPLAIN` used to print `select_type==SIMPLE`, or vice versa.

Required Permissions

Running `SHOW EXPLAIN` requires the same permissions as running `SHOW PROCESSLIST` would.

1.1.1.2.2.8 Using Buffer UPDATE Algorithm

This article explains the UPDATE statement's *Using Buffer* algorithm.

Take the following table and query:

Name	Salary
Babatunde	1000
Jolana	1050
Pankaja	1300

```
UPDATE employees SET salary = salary+100 WHERE salary < 2000;
```

Suppose the *employees* table has an index on the *salary* column, and the optimizer decides to use a range scan on that index.

The optimizer starts a range scan on the *salary* index. We find the first record *Babatunde, 1000*. If we do an on-the-fly update, we immediately instruct the storage engine to change this record to be *Babatunde, 1000+100=1100*.

Then we proceed to search for the next record, and find *Jolana, 1050*. We instruct the storage engine to update it to be *Jolana, 1050+100=1150*.

Then we proceed to search for the next record ... and what happens next depends on the storage engine. In some storage engines, data changes are visible immediately, so we will find find the *Babatunde, 1100* record that we wrote at the first step, modifying it again, giving Babatunde an undeserved raise. Then we will see Babatunde again and again, looping continually.

In order to prevent such situations, the optimizer checks whether the UPDATE statement is going to change key values for the keys it is using. In that case, it will use a different algorithm:

1. Scan everyone with "salary<2000", remembering the rowids of the rows in a buffer.
2. Read the buffer and apply the updates.

This way, each row will be updated only once.

The `Using buffer` EXPLAIN output indicates that the buffer as described above will be used.

1.1.1.2.3 BACKUP Commands

Commands used by backup tools

BACKUP STAGE

Commands to be used by a MariaDB backup tool.

BACKUP LOCK

Blocks a table from DDL statements.

Mariabackup and BACKUP STAGE Commands

How Mariabackup could use BACKUP STAGE commands.

Storage Snapshots and BACKUP STAGE Commands

How storage snapshots could use BACKUP STAGE commands.

1.1.1.2.3.1 BACKUP STAGE

MariaDB starting with 10.4.1

The `BACKUP STAGE` commands were introduced in MariaDB 10.4.1.

Contents

The `BACKUP STAGE` commands are a set of commands to make it possible to make an efficient external backup tool.

Syntax

```
BACKUP STAGE [START | FLUSH | BLOCK_DDL | BLOCK_COMMIT | END ]
```

In the following text, a transactional table means InnoDB or "InnoDB-like engine with redo log that can lock redo purges and can be copied without locks by an outside process".

Goals with BACKUP STAGE Commands

- To be able to do a majority of the backup with the minimum possible server locks. Especially for transactional tables (InnoDB, MyRocks etc) there is only need for a very short block of new commits while copying statistics and log tables.
- DDL are only needed to be blocked for a very short duration of the backup while mariabackup is copying the tables affected by DDL during the initial part of the backup.
- Most non transactional tables (those that are not in use) will be copied during `BACKUP STAGE START`. The exceptions are system statistic and log tables that are not blocked during the backup until `BLOCK_COMMIT`.
- Should work efficiently with backup tools that use disk snapshots.
- Should work as efficiently as possible for all table types that store data on the local disks.
- As little copying as possible under higher level stages/locks. For example, .frm (dictionary) and .trn (trigger) files should be copying while copying the table data.

`BACKUP STAGE` Commands

`BACKUP STAGE START`

The `START` stage is designed for the following tasks:

- Blocks purge of redo files for storage engines that needs this (Aria)
- Start logging of DDL commands into 'datadir'/ddl.log. This may take a short time as the command has to wait until there are no active DDL commands.

`BACKUP STAGE FLUSH`

The `FLUSH` stage is designed for the following tasks:

- FLUSH all changes for inactive non-transactional tables, except for statistics and log tables.
- Close all tables that are not in use, to ensure they are marked as closed for the backup.
- BLOCK all new write locks for all non transactional tables (except statistics and log tables). The command will not wait for tables that are in use by read-only transactions.

DDLs don't have to be blocked at this stage as they can't cause the table to be in an inconsistent state. This is true also for non-transactional tables.

BACKUP STAGE BLOCK_DDL

The BLOCK_DDL stage is designed for the following tasks:

- Wait for all statements using write locked non-transactional tables to end.
- Blocks CREATE TABLE, DROP TABLE, TRUNCATE TABLE, and RENAME TABLE.
- Blocks also start off a **new** ALTER TABLE and the **final rename phase** of ALTER TABLE. Running ALTER TABLES are not blocked.

BACKUP STAGE BLOCK_COMMIT

The BLOCK_COMMIT stage is designed for the following tasks:

- Lock the binary log and commit/rollback to ensure that no changes are committed to any tables. If there are active commits or data to be copied to the binary log this will be allowed to finish. Active transactions will not affect BLOCK_COMMIT .
- This doesn't lock temporary tables that are not used by replication. However these will be blocked when it's time to write to the binary log.
- Lock system log tables and statistics tables, flush them and mark them closed.

When the BLOCK_COMMIT 's stages return, this is the 'backup time'. Everything committed will be in the backup and everything not committed will roll back.

Transactional engines will continue to do changes to the redo log during the BLOCK COMMIT stage, but this is not important as all of these will roll back later as the changes will not be committed.

BACKUP STAGE END

The END stage is designed for the following tasks:

- End DDL logging
- Free resources

Using BACKUP STAGE Commands with Backup Tools

Using BACKUP STAGE Commands with Mariabackup

The BACKUP STAGE commands are a set of commands to make it possible to make an efficient external backup tool. How Mariabackup uses these commands depends on whether you are using the version that is bundled with MariaDB Community Server or the version that is bundled with MariaDB Enterprise Server. See Mariabackup and BACKUP STAGE Commands for some examples on how Mariabackup uses these commands.

> If you would like to use a version of Mariabackup that uses the BACKUP STAGE commands in an efficient way, then one option is to use MariaDB Enterprise Backup that is bundled with MariaDB Enterprise Server.

Using BACKUP STAGE Commands with Storage Snapshots

The BACKUP STAGE commands are a set of commands to make it possible to make an efficient external backup tool. These commands could even be used by tools that perform backups by taking a snapshot of a file system, SAN, or some other kind of storage device. See Storage Snapshots and BACKUP STAGE Commands for some examples on how to use each BACKUP STAGE command in an efficient way.

Privileges

BACKUP STAGE requires the RELOAD privilege.

Notes

- Only one connection can run BACKUP STAGE START . If a second connection tries, it will wait until the first one has executed BACKUP STAGE END .
- If the user skips a BACKUP STAGE , then all intermediate backup stages will automatically be run. This will allow us to add new stages within the BACKUP STAGE hierarchy in the future with even more precise locks without causing

problems for tools using an earlier version of the `BACKUP STAGE` implementation.
- One can use the max_statement_time or lock_wait_timeout system variables to ensure that a `BACKUP STAGE` command doesn't block the server too long.
- DDL logging will only be available in MariaDB Enterprise Server 10.2 and later.
- A disconnect will automatically release backup stages.
- There is no easy way to see which is the current stage.

1.1.1.2.3.2 BACKUP LOCK

MariaDB starting with 10.4.2
The BACKUP LOCK command was introduced in MariaDB 10.4.2.

Contents

BACKUP LOCK blocks a table from DDL statements. This is mainly intended to be used by tools like mariabackup that need to ensure there are no DDLs on a table while the table files are opened. For example, for an Aria table that stores data in 3 files with extensions .frm, .MAI and .MAD. Normal read/write operations can continue as normal.

Syntax

To lock a table:

```
BACKUP LOCK table_name
```

To unlock a table:

```
BACKUP UNLOCK
```

Usage in a Backup Tool

```
BACKUP LOCK [database.]table_name;
 - Open all files related to a table (for example, t.frm, t.MAI and t.MYD)
BACKUP UNLOCK;
- Copy data
- Close files
```

This ensures that all files are from the same generation, that is created at the same time by the MariaDB server. This works, because the open files will point to the original table files which will not be affected if there is any ALTER TABLE while copying the files.

Privileges

BACKUP LOCK requires the RELOAD privilege.

Notes

- The idea is that the `BACKUP LOCK` should be held for as short a time as possible by the backup tool. The time to take an uncontested lock is very short! One can easily do 50,000 locks/unlocks per second on low end hardware.
- One should use different connections for BACKUP STAGE commands and `BACKUP LOCK` .

Implementation

- Internally, BACKUP LOCK is implemented by taking an `MDLSHARED_HIGH_PRIO` MDL lock on the table object, which protects the table from any DDL operations.

2.3.4.12 Mariabackup and BACKUP STAGE Commands

1.1.1.2.3.4 Storage Snapshots and BACKUP STAGE Commands

MariaDB starting with 10.4.1

The `BACKUP STAGE` commands were introduced in MariaDB 10.4.1.

Contents

The BACKUP STAGE commands are a set of commands to make it possible to make an efficient external backup tool. These commands could even be used by tools that perform backups by taking a snapshot of a file system, SAN, or some other kind of storage device.

Generic Backup Process with Storage Snapshots

A tool that backs up MariaDB by taking a snapshot of a file system, SAN, or some other kind of storage device could use each `BACKUP STAGE` command in the following way:

- First, execute the following:

```
BACKUP STAGE START
BACKUP STAGE BLOCK_COMMIT
```

- Then, take the snapshot.
- Then, execute the following:

```
BACKUP STAGE END
```

The above ensures that all non-transactional tables are properly flushed to disk before the snapshot is done. Using `BACKUP STAGE` commands is also more efficient than using the FLUSH TABLES WITH READ LOCK command as the above set of commands will not block or be blocked by write operations to transactional tables.

Note that when the backup is completed, one should delete all files with the "#sql" prefix, as these are files used by concurrent running `ALTER TABLE` . Note that InnoDB will on server restart automatically delete any tables with the "#sql" prefix.

1.1.1.2.4 FLUSH Commands

Commands to reset (flush) various caches in MariaDB.

FLUSH

Clear or reload various internal caches.

FLUSH QUERY CACHE

Defragmenting the query cache

FLUSH TABLES FOR EXPORT

Flushes changes to disk for specific tables.

There are 2 related questions.

1.1.1.2.4.1 FLUSH

Syntax

```
FLUSH [NO_WRITE_TO_BINLOG | LOCAL]
    flush_option [, flush_option] ...
```

or when flushing tables:

```
FLUSH [NO_WRITE_TO_BINLOG | LOCAL] TABLES [table_list]  [table_flush_option]
```

Contents

where `table_list` is a list of tables separated by `,` (comma).

Description

The `FLUSH` statement clears or reloads various internal caches used by MariaDB. To execute `FLUSH`, you must have the `RELOAD` privilege. See GRANT.

The `RESET` statement is similar to `FLUSH`. See RESET.

You cannot issue a FLUSH statement from within a stored function or a trigger. Doing so within a stored procedure is permitted, as long as it is not called by a stored function or trigger. See Stored Routine Limitations, Stored Function Limitations and Trigger Limitations.

If a listed table is a view, an error like the following will be produced:

```
ERROR 1347 (HY000): 'test.v' is not BASE TABLE
```

By default, `FLUSH` statements are written to the binary log and will be replicated. The `NO_WRITE_TO_BINLOG` keyword (`LOCAL` is an alias) will ensure the statement is not written to the binary log.

The different flush options are:

Option	Description
`CHANGED_PAGE_BITMAPS`	XtraDB only. Internal command used for backup purposes. See the Information Schema CHANGED_PAGE_BITMAPS Table.
`CLIENT_STATISTICS`	Reset client statistics (see SHOW CLIENT_STATISTICS).
`DES_KEY_FILE`	Reloads the DES key file (Specified with the --des-key-file startup option).
`HOSTS`	Flush the hostname cache (used for converting ip to host names and for unblocking blocked hosts. See max_connect_errors and performance_schema.host_cache

INDEX_STATISTICS	Reset index statistics (see SHOW INDEX_STATISTICS).
[ERROR | ENGINE | GENERAL | SLOW | BINARY | RELAY] LOGS	Close and reopen the specified log type, or all log types if none are specified. FLUSH RELAY LOGS [connection-name] can be used to flush the relay logs for a specific connection. Only one connection can be specified per FLUSH command. See Multi-source replication. FLUSH ENGINE LOGS will delete all unneeded Aria redo logs. FLUSH BINARY LOGS DELETE_DOMAIN_ID=(list-of-domains) can be used to discard obsolete GTID domains from the server's binary log state. In order for this to be successful, no event group from the listed GTID domains can be present in existing binary log files. If some still exist, then they must be purged prior to executing this command. If the command completes successfully, then it also rotates the binary log.
MASTER	Deprecated option, use RESET MASTER instead.
PRIVILEGES	Reload all privileges from the privilege tables in the mysql database. If the server is started with --skip-grant-table option, this will activate the privilege tables again.
QUERY CACHE	Defragment the query cache to better utilize its memory. If you want to reset the query cache, you can do it with RESET QUERY CACHE.
QUERY_RESPONSE_TIME	See the QUERY_RESPONSE_TIME plugin.
SLAVE	Deprecated option, use RESET REPLICA or RESET SLAVE instead.
SSL	Used to dynamically reinitialize the server's TLS context by reloading the files defined by several TLS system variables. See FLUSH SSL for more information.
STATUS	Resets all server status variables that can be reset to 0. Not all global status variables support this, so not all global values are reset. See FLUSH STATUS for more information.
TABLE	Close tables given as options or all open tables if no table list was used. From MariaDB 10.4.1, using without any table list will only close tables not in use, and tables not locked by the FLUSH TABLES connection. If there are no locked tables, FLUSH TABLES will be instant and will not cause any waits, as it no longer waits for tables in use. When a table list is provided, from MariaDB 10.4.1, the server will wait for the end of any transactions that are using the tables. Previously, FLUSH TABLES only waited for the statements to complete.
TABLES	Same as FLUSH TABLE .
TABLES ... FOR EXPORT	For InnoDB tables, flushes table changes to disk to permit binary table copies while the server is running. See FLUSH TABLES ... FOR EXPORT for more.
TABLES WITH READ LOCK	Closes all open tables. New tables are only allowed to be opened with read locks until an UNLOCK TABLES is given.
TABLES WITH READ LOCK AND DISABLE CHECKPOINT	As TABLES WITH READ LOCK but also disable all checkpoint writes by transactional table engines. This is useful when doing a disk snapshot of all tables.
TABLE_STATISTICS	Reset table statistics (see SHOW TABLE_STATISTICS).
USER_RESOURCES	Resets all per hour user resources. This enables clients that have exhausted their resources to connect again.
USER_STATISTICS	Reset user statistics (see SHOW USER_STATISTICS).
USER_VARIABLES	Reset user variables (see User-defined variables).

You can also use the mariadb-admin client to flush things. Use mariadb-admin --help to examine what flush commands it supports.

FLUSH RELAY LOGS

```
FLUSH RELAY LOGS 'connection_name';
```

Compatibility with MySQL

MariaDB starting with 10.7.0

The FOR CHANNEL keyword was added for MySQL compatibility. This is identical to using the channel_name directly after the FLUSH command .

For example, one can now use:

```
FLUSH RELAY LOGS FOR CHANNEL 'connection_name';
```

FLUSH STATUS

Server status variables can be reset by executing the following:

```
FLUSH STATUS;
```

Global Status Variables that Support `FLUSH STATUS`

Not all global status variables support being reset by `FLUSH STATUS`. Currently, the following status variables are reset by `FLUSH STATUS`:

- Aborted_clients
- Aborted_connects
- Binlog_cache_disk_use
- Binlog_cache_use
- Binlog_stmt_cache_disk_use
- Binlog_stmt_cache_use
- Connection_errors_accept
- Connection_errors_internal
- Connection_errors_max_connections
- Connection_errors_peer_address
- Connection_errors_select
- Connection_errors_tcpwrap
- Created_tmp_files
- Delayed_errors
- Delayed_writes
- Feature_check_constraint
- Feature_delay_key_write
- Max_used_connection_time
- Max_used_connections
- Opened_plugin_libraries
- Performance_schema_accounts_lost
- Performance_schema_cond_instances_lost
- Performance_schema_digest_lost
- Performance_schema_file_handles_lost
- Performance_schema_file_instances_lost
- Performance_schema_hosts_lost
- Performance_schema_locker_lost
- Performance_schema_mutex_instances_lost
- Performance_schema_rwlock_instances_lost
- Performance_schema_session_connect_attrs_lost
- Performance_schema_socket_instances_lost
- Performance_schema_stage_classes_lost
- Performance_schema_statement_classes_lost
- Performance_schema_table_handles_lost
- Performance_schema_table_instances_lost
- Performance_schema_thread_instances_lost
- Performance_schema_users_lost
- Qcache_hits
- Qcache_inserts
- Qcache_lowmem_prunes
- Qcache_not_cached
- Rpl_semi_sync_master_no_times
- Rpl_semi_sync_master_no_tx
- Rpl_semi_sync_master_timefunc_failures
- Rpl_semi_sync_master_wait_pos_backtraverse
- Rpl_semi_sync_master_yes_tx
- Rpl_transactions_multi_engine
- Server_audit_writes_failed
- Slave_retried_transactions
- Slow_launch_threads

- Ssl_accept_renegotiates
- Ssl_accepts
- Ssl_callback_cache_hits
- Ssl_client_connects
- Ssl_connect_renegotiates
- Ssl_ctx_verify_depth
- Ssl_ctx_verify_mode
- Ssl_finished_accepts
- Ssl_finished_connects
- Ssl_session_cache_hits
- Ssl_session_cache_misses
- Ssl_session_cache_overflows
- Ssl_session_cache_size
- Ssl_session_cache_timeouts
- Ssl_sessions_reused
- Ssl_used_session_cache_entries
- Subquery_cache_hit
- Subquery_cache_miss
- Table_locks_immediate
- Table_locks_waited
- Tc_log_max_pages_used
- Tc_log_page_waits
- Transactions_gtid_foreign_engine
- Transactions_multi_engine

The different usage of FLUSH TABLES

Purpose of FLUSH TABLES

The purpose of `FLUSH TABLES` is to clean up the open table cache and table definition cache from not in use tables. This frees up memory and file descriptors. Normally this is not needed as the caches works on a FIFO bases, but can be useful if the server seams to use up to much memory for some reason.

Purpose of FLUSH TABLES WITH READ LOCK

`FLUSH TABLES WITH READ LOCK` is useful if you want to take a backup of some tables. When `FLUSH TABLES WITH READ LOCK` returns, all write access to tables are blocked and all tables are marked as 'properly closed' on disk. The tables can still be used for read operations.

Purpose of FLUSH TABLES table_list

`FLUSH TABLES` table_list is useful if you want to copy a table object/files to or from the server. This command puts a lock that stops new users of the table and will wait until everyone has stopped using the table. The table is then removed from the table definition and table cache.

Note that it's up to the user to ensure that no one is accessing the table between `FLUSH TABLES` and the table is copied to or from the server. This can be secured by using LOCK TABLES.

If there are any tables locked by the connection that is using `FLUSH TABLES` all the locked tables will be closed as part of the flush and reopened and relocked before `FLUSH TABLES` returns. This allows one to copy the table after `FLUSH TABLES` returns without having any writes on the table. For now this works works with most tables, except InnoDB as InnoDB may do background purges on the table even while it's write locked.

Purpose of FLUSH TABLES table_list WITH READ LOCK

`FLUSH TABLES table_list WITH READ LOCK` should work as `FLUSH TABLES WITH READ LOCK`, but only those tables that are listed will be properly closed. However in practice this works exactly like `FLUSH TABLES WITH READ LOCK` as the `FLUSH` command has anyway to wait for all WRITE operations to end because we are depending on a global read lock for this code. In the future we should consider fixing this to instead use meta data locks.

Implementation of FLUSH TABLES commands from MariaDB 10.4.8

Implementation of FLUSH TABLES

- Free memory and file descriptors not in use

Implementation of FLUSH TABLES WITH READ LOCK

- Lock all tables read only for simple old style backup.
- All background writes are suspended and tables are marked as closed.
- No statement requiring table changes are allowed for any user until `UNLOCK TABLES`.

Instead of using `FLUSH TABLE WITH READ LOCK` one should in most cases instead use BACKUP STAGE BLOCK_COMMIT.

Implementation of FLUSH TABLES table_list

- Free memory and file descriptors for tables not in use from table list.
- Lock given tables as read only.
- Wait until all translations has ended that uses any of the given tables.
- Wait until all background writes are suspended and tables are marked as closed.

Implementation of FLUSH TABLES table_list FOR EXPORT

- Free memory and file descriptors for tables not in use from table list
- Lock given tables as read.
- Wait until all background writes are suspended and tables are marked as closed.
- Check that all tables supports `FOR EXPORT`
- No changes to these tables allowed until `UNLOCK TABLES`

This is basically the same behavior as in old MariaDB version if one first lock the tables, then do `FLUSH TABLES`. The tables will be copyable until `UNLOCK TABLES`.

FLUSH SSL

In MariaDB 10.4 and later, the `FLUSH SSL` command can be used to dynamically reinitialize the server's TLS context. This is most useful if you need to replace a certificate that is about to expire without restarting the server.

This operation is performed by reloading the files defined by the following TLS system variables:

- ssl_cert
- ssl_key
- ssl_ca
- ssl_capath
- ssl_crl
- ssl_crlpath

These TLS system variables are not dynamic, so their values can **not** be changed without restarting the server.

If you want to dynamically reinitialize the server's TLS context, then you need to change the certificate and key files at the relevant paths defined by these TLS system variables, without actually changing the values of the variables. See MDEV-19341 for more information.

Reducing Memory Usage

To flush some of the global caches that take up memory, you could execute the following command:

```
FLUSH LOCAL HOSTS,
   QUERY CACHE,
   TABLE_STATISTICS,
   INDEX_STATISTICS,
   USER_STATISTICS;
```

1.1.1.2.4.2 FLUSH QUERY CACHE

Description

You can defragment the query cache to better utilize its memory with the `FLUSH QUERY CACHE` statement. The statement does not remove any queries from the cache.

The RESET QUERY CACHE statement removes all query results from the query cache. The FLUSH TABLES statement also does this.

1.1.1.2.4.3 FLUSH TABLES FOR EXPORT

Syntax

```
FLUSH TABLES table_name [, table_name] FOR EXPORT
```

Contents

Description

`FLUSH TABLES ... FOR EXPORT` flushes changes to the specified tables to disk so that binary copies can be made while the server is still running. This works for Archive, Aria, CSV, InnoDB, MyISAM, MERGE, and XtraDB tables.

The table is read locked until one has issued UNLOCK TABLES.

If a storage engine does not support `FLUSH TABLES FOR EXPORT`, a 1031 error (SQLSTATE 'HY000') is produced.

If `FLUSH TABLES ... FOR EXPORT` is in effect in the session, the following statements will produce an error if attempted:

- `FLUSH TABLES WITH READ LOCK`
- `FLUSH TABLES ... WITH READ LOCK`
- `FLUSH TABLES ... FOR EXPORT`
- Any statement trying to update any table

If any of the following statements is in effect in the session, attempting `FLUSH TABLES ... FOR EXPORT` will produce an error.

- `FLUSH TABLES ... WITH READ LOCK`
- `FLUSH TABLES ... FOR EXPORT`
- `LOCK TABLES ... READ`
- `LOCK TABLES ... WRITE`

`FLUSH FOR EXPORT` is not written to the binary log.

This statement requires the RELOAD and the LOCK TABLES privileges.

If one of the specified tables cannot be locked, none of the tables will be locked.

If a table does not exist, an error like the following will be produced:

```
ERROR 1146 (42S02): Table 'test.xxx' doesn't exist
```

If a table is a view, an error like the following will be produced:

```
ERROR 1347 (HY000): 'test.v' is not BASE TABLE
```

Example

```
FLUSH TABLES test.t1 FOR EXPORT;
#  Copy files related to the table (see below)
UNLOCK TABLES;
```

For a full description, please see copying MariaDB tables.

1.1.1.2.5 Replication Commands

The terms *master* and *slave* have historically been used in replication, but the terms terms *primary* and *replica* are now preferred. The old terms are used still used in parts of the documentation, and in MariaDB commands, although MariaDB 10.5 has begun the process of renaming. The documentation process is ongoing. See MDEV-18777 to follow progress on this effort.

A list of replication-related commands. See replication for more replication-related information.

CHANGE MASTER TO
Set or change replica parameters for connecting to the primary.

START SLAVE
Start replica threads.

STOP SLAVE
Stop replica threads.

RESET REPLICA/SLAVE
Forget replica connection information and start a new relay log file.

SET GLOBAL SQL_SLAVE_SKIP_COUNTER
Skips a number of events from the primary.

SHOW RELAYLOG EVENTS
Show events in the relay log.

SHOW SLAVE STATUS
Show status for one or all primaries.

SHOW MASTER STATUS
Status information about the binary log.

SHOW SLAVE HOSTS
Display replicas currently registered with the primary.

RESET MASTER
Delete binary log files.

1.1.1.2.5.1 CHANGE MASTER TO

The terms *master* and *slave* have historically been used in replication, but the terms terms *primary* and *replica* are now preferred. The old terms are used still used in parts of the documentation, and in MariaDB commands, although MariaDB 10.5 has begun the process of renaming. The documentation process is ongoing. See MDEV-18777 to follow progress on this effort.

Syntax

```
CHANGE MASTER ['connection_name'] TO master_def  [, master_def] ...
  [FOR CHANNEL 'channel_name']

master_def:
    MASTER_BIND = 'interface_name'
  | MASTER_HOST = 'host_name'
  | MASTER_USER = 'user_name'
  | MASTER_PASSWORD = 'password'
  | MASTER_PORT = port_num
  | MASTER_CONNECT_RETRY = interval
  | MASTER_HEARTBEAT_PERIOD = interval
  | MASTER_LOG_FILE = 'master_log_name'
  | MASTER_LOG_POS = master_log_pos
  | RELAY_LOG_FILE = 'relay_log_name'
  | RELAY_LOG_POS = relay_log_pos
  | MASTER_DELAY = interval
  | MASTER_SSL = {0|1}
  | MASTER_SSL_CA = 'ca_file_name'
  | MASTER_SSL_CAPATH = 'ca_directory_name'
  | MASTER_SSL_CERT = 'cert_file_name'
  | MASTER_SSL_CRL = 'crl_file_name'
  | MASTER_SSL_CRLPATH = 'crl_directory_name'
  | MASTER_SSL_KEY = 'key_file_name'
  | MASTER_SSL_CIPHER = 'cipher_list'
  | MASTER_SSL_VERIFY_SERVER_CERT = {0|1}
  | MASTER_USE_GTID = {current_pos|slave_pos|no}
  | MASTER_DEMOTE_TO_SLAVE = bool
  | IGNORE_SERVER_IDS = (server_id_list)
  | DO_DOMAIN_IDS = ([N,..])
  | IGNORE_DOMAIN_IDS = ([N,..])
```

Contents

Description

The `CHANGE MASTER` statement sets the options that a replica uses to connect to and replicate from a primary.

MariaDB starting with 10.7.0

The `FOR CHANNEL` keyword was added for MySQL compatibility. This is identical to using the channel_name directly after `CHANGE MASTER`.

Multi-Source Replication

If you are using multi-source replication, then you need to specify a connection name when you execute `CHANGE MASTER`. There are two ways to do this:

- Setting the default_master_connection system variable prior to executing `CHANGE MASTER`.
- Setting the `connection_name` parameter when executing `CHANGE MASTER`.

default_master_connection

```
SET default_master_connection = 'gandalf';
STOP SLAVE;
CHANGE MASTER TO
   MASTER_PASSWORD='new3cret';
START SLAVE;
```

connection_name

```
STOP SLAVE 'gandalf';
CHANGE MASTER 'gandalf' TO
   MASTER_PASSWORD='new3cret';
START SLAVE 'gandalf';
```

Options

Connection Options

MASTER_USER

The `MASTER_USER` option for `CHANGE MASTER` defines the user account that the replica will use to connect to the primary.

This user account will need the REPLICATION SLAVE privilege (or, from MariaDB 10.5.1, the REPLICATION REPLICA on the primary.

For example:

```
STOP SLAVE;
CHANGE MASTER TO
   MASTER_USER='repl',
   MASTER_PASSWORD='new3cret';
START SLAVE;
```

The maximum length of the `MASTER_USER` string is 96 characters until MariaDB 10.5, and 128 characters from MariaDB 10.6.

MASTER_PASSWORD

The `MASTER_PASSWORD` option for `CHANGE MASTER` defines the password that the replica will use to connect to the primary as the user account defined by the MASTER_USER option.

For example:

```
STOP SLAVE;
CHANGE MASTER TO
   MASTER_PASSWORD='new3cret';
START SLAVE;
```

The maximum length of the `MASTER_PASSWORD` string is 32 characters. The effective maximum length of the string depends on how many bytes are used per character and can be up to 96 characters.

Due to MDEV-29994, the password can be silently truncated to 41 characters when MariaDB is restarted. For this reason it is recommended to use a password that is shorter than this.

MASTER_HOST

The `MASTER_HOST` option for `CHANGE MASTER` defines the hostname or IP address of the primary.

If you set the value of the `MASTER_HOST` option to the empty string, then that is not the same as not setting the option's value at all. If you set the value of the `MASTER_HOST` option to the empty string, then the `CHANGE MASTER` command will fail with an error. In MariaDB 5.3 and before, if you set the value of the `MASTER_HOST` option to the empty string, then the `CHANGE MASTER` command would succeed, but the subsequent START SLAVE command would fail.

For example:

```
STOP SLAVE;
CHANGE MASTER TO
   MASTER_HOST='dbserver1.example.com',
   MASTER_USER='repl',
   MASTER_PASSWORD='new3cret',
   MASTER_USE_GTID=slave_pos;
START SLAVE;
```

If you set the value of the `MASTER_HOST` option in a `CHANGE MASTER` command, then the replica assumes that the primary is different from before, even if you set the value of this option to the same value it had previously. In this scenario, the replica will consider the old values for the primary's binary log file name and position to be invalid for the new primary. As a side effect, if you do not explicitly set the values of the MASTER_LOG_FILE and MASTER_LOG_POS options in the statement, then the statement will be implicitly appended with `MASTER_LOG_FILE=''` and `MASTER_LOG_POS=4`. However, if you enable GTID mode for replication by setting the MASTER_USE_GTID option to some value other than `no` in the statement, then these values will effectively be ignored anyway.

Replicas cannot connect to primaries using Unix socket files or Windows named pipes. The replica must connect to the primary using TCP/IP.

The maximum length of the `MASTER_HOST` string is 60 characters until MariaDB 10.5, and 255 characters from MariaDB 10.6.

MASTER_PORT

The `MASTER_PORT` option for `CHANGE MASTER` defines the TCP/IP port of the primary.

For example:

```
STOP SLAVE;
CHANGE MASTER TO
   MASTER_HOST='dbserver1.example.com',
   MASTER_PORT=3307,
   MASTER_USER='repl',
   MASTER_PASSWORD='new3cret',
   MASTER_USE_GTID=slave_pos;
START SLAVE;
```

If you set the value of the `MASTER_PORT` option in a `CHANGE MASTER` command, then the replica assumes that the primary is different from before, even if you set the value of this option to the same value it had previously. In this scenario, the replica will consider the old values for the primary's binary log file name and position to be invalid for the new primary. As a side effect, if you do not explicitly set the values of the MASTER_LOG_FILE and MASTER_LOG_POS options in the statement, then the statement will be implicitly appended with `MASTER_LOG_FILE=''` and `MASTER_LOG_POS=4`. However, if you enable GTID mode for replication by setting the MASTER_USE_GTID option to some value other than `no` in the statement, then these values will effectively be ignored anyway.

Replicas cannot connect to primaries using Unix socket files or Windows named pipes. The replica must connect to the primary using TCP/IP.

MASTER_CONNECT_RETRY

The `MASTER_CONNECT_RETRY` option for `CHANGE MASTER` defines how many seconds that the replica will wait between connection retries. The default is `60`.

```
STOP SLAVE;
CHANGE MASTER TO
   MASTER_CONNECT_RETRY=20;
START SLAVE;
```

The number of connection attempts is limited by the master_retry_count option. It can be set either on the command-line or in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
master_retry_count=4294967295
```

MASTER_BIND

The `MASTER_BIND` option for `CHANGE MASTER` is only supported by MySQL 5.6.2 and later and by MySQL NDB Cluster 7.3.1 and later. This option is not supported by MariaDB. See MDEV-19248 for more information.

MASTER_HEARTBEAT_PERIOD

The `MASTER_HEARTBEAT_PERIOD` option for `CHANGE MASTER` can be used to set the interval in seconds between replication heartbeats. Whenever the primary's binary log is updated with an event, the waiting period for the next heartbeat is reset.

This option's *interval* argument has the following characteristics:

- It is a decimal value with a range of `0` to `4294967` seconds.
- It has a resolution of hundredths of a second.
- Its smallest valid non-zero value is `0.001`.
- Its default value is the value of the slave_net_timeout system variable divided by 2.
- If it's set to `0`, then heartbeats are disabled.

Heartbeats are sent by the primary only if there are no unsent events in the binary log file for a period longer than the interval.

If the RESET SLAVE statement is executed, then the heartbeat interval is reset to the default.

If the slave_net_timeout system variable is set to a value that is lower than the current heartbeat interval, then a warning will be issued.

TLS Options

The TLS options are used for providing information about TLS. The options can be set even on replicas that are compiled without TLS support. The TLS options are saved to either the default `master.info` file or the file that is configured by the master_info_file option, but these TLS options are ignored unless the replica supports TLS.

See Replication with Secure Connections for more information.

MASTER_SSL

The `MASTER_SSL` option for `CHANGE MASTER` tells the replica whether to force TLS for the connection. The valid values are `0` or `1`. Required to be set to `1` for the other `MASTER_SSL*` options to have any effect.

For example:

```
STOP SLAVE;
CHANGE MASTER TO
   MASTER_SSL=1;
START SLAVE;
```

MASTER_SSL_CA

The `MASTER_SSL_CA` option for `CHANGE MASTER` defines a path to a PEM file that should contain one or more X509 certificates for trusted Certificate Authorities (CAs) to use for TLS. This option requires that you use the absolute path, not a relative path.

For example:

```
STOP SLAVE;
CHANGE MASTER TO
   MASTER_SSL_CERT='/etc/my.cnf.d/certificates/server-cert.pem',
   MASTER_SSL_KEY='/etc/my.cnf.d/certificates/server-key.pem',
   MASTER_SSL_CA='/etc/my.cnf.d/certificates/ca.pem',
   MASTER_SSL_VERIFY_SERVER_CERT=1;
START SLAVE;
```

See Secure Connections Overview: Certificate Authorities (CAs) for more information.

The maximum length of `MASTER_SSL_CA` string is 511 characters.

MASTER_SSL_CAPATH

The `MASTER_SSL_CAPATH` option for `CHANGE MASTER` defines a path to a directory that contains one or more PEM files that should each contain one X509 certificate for a trusted Certificate Authority (CA) to use for TLS. This option requires that you use the absolute path, not a relative path. The directory specified by this option needs to be run through the openssl rehash command.

For example:

```
STOP SLAVE;
CHANGE MASTER TO
   MASTER_SSL_CERT='/etc/my.cnf.d/certificates/server-cert.pem',
   MASTER_SSL_KEY='/etc/my.cnf.d/certificates/server-key.pem',
   MASTER_SSL_CAPATH='/etc/my.cnf.d/certificates/ca/',
   MASTER_SSL_VERIFY_SERVER_CERT=1;
START SLAVE;
```

See Secure Connections Overview: Certificate Authorities (CAs) for more information.

The maximum length of `MASTER_SSL_CA_PATH` string is 511 characters.

MASTER_SSL_CERT

The `MASTER_SSL_CERT` option for `CHANGE MASTER` defines a path to the X509 certificate file to use for TLS. This option requires that you use the absolute path, not a relative path.

For example:

```
STOP SLAVE;
CHANGE MASTER TO
   MASTER_SSL_CERT='/etc/my.cnf.d/certificates/server-cert.pem',
   MASTER_SSL_KEY='/etc/my.cnf.d/certificates/server-key.pem',
   MASTER_SSL_CA='/etc/my.cnf.d/certificates/ca.pem',
   MASTER_SSL_VERIFY_SERVER_CERT=1;
START SLAVE;
```

The maximum length of `MASTER_SSL_CERT` string is 511 characters.

MASTER_SSL_CRL

The `MASTER_SSL_CRL` option for `CHANGE MASTER` defines a path to a PEM file that should contain one or more revoked X509 certificates to use for TLS. This option requires that you use the absolute path, not a relative path.

This option is only supported if the server was built with OpenSSL. If the server was built with yaSSL, then this option is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.

For example:

```
STOP SLAVE;
CHANGE MASTER TO
   MASTER_SSL_CERT='/etc/my.cnf.d/certificates/server-cert.pem',
   MASTER_SSL_KEY='/etc/my.cnf.d/certificates/server-key.pem',
   MASTER_SSL_CA='/etc/my.cnf.d/certificates/ca.pem',
   MASTER_SSL_VERIFY_SERVER_CERT=1,
   MASTER_SSL_CRL='/etc/my.cnf.d/certificates/crl.pem';
START SLAVE;
```

See Secure Connections Overview: Certificate Revocation Lists (CRLs) for more information.

The maximum length of `MASTER_SSL_CRL` string is 511 characters.

MASTER_SSL_CRLPATH

The `MASTER_SSL_CRLPATH` option for `CHANGE MASTER` defines a path to a directory that contains one or more PEM files that should each contain one revoked X509 certificate to use for TLS. This option requires that you use the absolute path, not a relative path. The directory specified by this variable needs to be run through the openssl rehash command.

This option is only supported if the server was built with OpenSSL. If the server was built with yaSSL, then this option is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.

For example:

```
STOP SLAVE;
CHANGE MASTER TO
   MASTER_SSL_CERT='/etc/my.cnf.d/certificates/server-cert.pem',
   MASTER_SSL_KEY='/etc/my.cnf.d/certificates/server-key.pem',
   MASTER_SSL_CA='/etc/my.cnf.d/certificates/ca.pem',
   MASTER_SSL_VERIFY_SERVER_CERT=1,
   MASTER_SSL_CRLPATH='/etc/my.cnf.d/certificates/crl/';
START SLAVE;
```

See Secure Connections Overview: Certificate Revocation Lists (CRLs) for more information.

The maximum length of `MASTER_SSL_CRL_PATH` string is 511 characters.

MASTER_SSL_KEY

The `MASTER_SSL_KEY` option for `CHANGE MASTER` defines a path to a private key file to use for TLS. This option requires that you use the absolute path, not a relative path.

For example:

```
STOP SLAVE;
CHANGE MASTER TO
   MASTER_SSL_CERT='/etc/my.cnf.d/certificates/server-cert.pem',
   MASTER_SSL_KEY='/etc/my.cnf.d/certificates/server-key.pem',
   MASTER_SSL_CA='/etc/my.cnf.d/certificates/ca.pem',
   MASTER_SSL_VERIFY_SERVER_CERT=1;
START SLAVE;
```

The maximum length of `MASTER_SSL_KEY` string is 511 characters.

MASTER_SSL_CIPHER

The `MASTER_SSL_CIPHER` option for `CHANGE MASTER` defines the list of permitted ciphers or cipher suites to use for TLS. Besides cipher names, if MariaDB was compiled with OpenSSL, this option could be set to "SSLv3" or "TLSv1.2" to allow all SSLv3 or all TLSv1.2 ciphers. Note that the TLSv1.3 ciphers cannot be excluded when using OpenSSL, even by using this option. See Using TLSv1.3 for details.

For example:

```
STOP SLAVE;
CHANGE MASTER TO
   MASTER_SSL_CERT='/etc/my.cnf.d/certificates/server-cert.pem',
   MASTER_SSL_KEY='/etc/my.cnf.d/certificates/server-key.pem',
   MASTER_SSL_CA='/etc/my.cnf.d/certificates/ca.pem',
   MASTER_SSL_VERIFY_SERVER_CERT=1,
   MASTER_SSL_CIPHER='TLSv1.2';
START SLAVE;
```

The maximum length of `MASTER_SSL_CIPHER` string is 511 characters.

MASTER_SSL_VERIFY_SERVER_CERT

The `MASTER_SSL_VERIFY_SERVER_CERT` option for `CHANGE MASTER` enables server certificate verification. This option is disabled by default prior to MariaDB 11.3.0, and enabled by default from MariaDB 11.3.0.

For example:

```
STOP SLAVE;
CHANGE MASTER TO
   MASTER_SSL_CERT='/etc/my.cnf.d/certificates/server-cert.pem',
   MASTER_SSL_KEY='/etc/my.cnf.d/certificates/server-key.pem',
   MASTER_SSL_CA='/etc/my.cnf.d/certificates/ca.pem',
   MASTER_SSL_VERIFY_SERVER_CERT=1;
START SLAVE;
```

See Secure Connections Overview: Server Certificate Verification for more information.

Binary Log Options

These options are related to the binary log position on the primary.

MASTER_LOG_FILE

The `MASTER_LOG_FILE` option for `CHANGE MASTER` can be used along with `MASTER_LOG_POS` to specify the coordinates at which the replica's I/O thread should begin reading from the primary's binary logs the next time the thread starts.

For example:

```
STOP SLAVE;
CHANGE MASTER TO
   MASTER_LOG_FILE='master2-bin.001',
   MASTER_LOG_POS=4;
START SLAVE;
```

The MASTER_LOG_FILE and MASTER_LOG_POS options cannot be specified if the RELAY_LOG_FILE and RELAY_LOG_POS options were also specified.

The MASTER_LOG_FILE and MASTER_LOG_POS options are effectively ignored if you enable GTID mode for replication by setting the MASTER_USE_GTID option to some value other than `no` in the statement.

MASTER_LOG_POS

The `MASTER_LOG_POS` option for `CHANGE MASTER` can be used along with `MASTER_LOG_FILE` to specify the coordinates at which the replica's I/O thread should begin reading from the primary's binary logs the next time the thread starts.

For example:

```
STOP SLAVE;
CHANGE MASTER TO
   MASTER_LOG_FILE='master2-bin.001',
   MASTER_LOG_POS=4;
START SLAVE;
```

The MASTER_LOG_FILE and MASTER_LOG_POS options cannot be specified if the RELAY_LOG_FILE and RELAY_LOG_POS options were also specified.

The MASTER_LOG_FILE and MASTER_LOG_POS options are effectively ignored if you enable GTID mode for replication by setting the MASTER_USE_GTID option to some value other than `no` in the statement.

Relay Log Options

These options are related to the relay log position on the replica.

RELAY_LOG_FILE

The `RELAY_LOG_FILE` option for `CHANGE MASTER` can be used along with the RELAY_LOG_POS option to specify the coordinates at which the replica's SQL thread should begin reading from the relay log the next time the thread starts.

The `CHANGE MASTER` statement usually deletes all relay log files. However, if the `RELAY_LOG_FILE` and/or `RELAY_LOG_POS` options are specified, then existing relay log files are kept.

When you want to change the relay log position, you only need to stop the replica's SQL thread. The replica's I/O thread can continue running. The STOP SLAVE and START SLAVE statements support the `SQL_THREAD` option for this scenario. For example:

```
STOP SLAVE SQL_THREAD;
CHANGE MASTER TO
   RELAY_LOG_FILE='slave-relay-bin.006',
   RELAY_LOG_POS=4025;
START SLAVE SQL_THREAD;
```

When the value of this option is changed, the metadata about the replica's SQL thread's position in the relay logs will also be changed in the `relay-log.info` file or the file that is configured by the relay_log_info_file system variable.

> The RELAY_LOG_FILE and RELAY_LOG_POS options cannot be specified if the MASTER_LOG_FILE and MASTER_LOG_POS options were also specified.

RELAY_LOG_POS

The `RELAY_LOG_POS` option for `CHANGE MASTER` can be used along with the RELAY_LOG_FILE option to specify the coordinates at which the replica's SQL thread should begin reading from the relay log the next time the thread starts.

The `CHANGE MASTER` statement usually deletes all relay log files. However, if the `RELAY_LOG_FILE` and/or `RELAY_LOG_POS` options are specified, then existing relay log files are kept.

When you want to change the relay log position, you only need to stop the replica's SQL thread. The replica's I/O thread can continue running. The STOP SLAVE and START SLAVE statements support the `SQL_THREAD` option for this scenario. For example:

```
STOP SLAVE SQL_THREAD;
CHANGE MASTER TO
   RELAY_LOG_FILE='slave-relay-bin.006',
   RELAY_LOG_POS=4025;
START SLAVE SQL_THREAD;
```

When the value of this option is changed, the metadata about the replica's SQL thread's position in the relay logs will also be changed in the `relay-log.info` file or the file that is configured by the relay_log_info_file system variable.

> The RELAY_LOG_FILE and RELAY_LOG_POS options cannot be specified if the MASTER_LOG_FILE and MASTER_LOG_POS options were also specified.

GTID Options

MASTER_USE_GTID

The `MASTER_USE_GTID` option for `CHANGE MASTER` can be used to configure the replica to use the global transaction ID (GTID) when connecting to a primary. The possible values are:

- `current_pos` - Replicate in GTID mode and use gtid_current_pos as the position to start downloading transactions from the primary. Deprecated from MariaDB 10.10. Using to transition to primary can break the replication state if the replica executes local transactions due to actively updating gtid_current_pos with gtid_binlog_pos and gtid_slave_pos. Use the new, safe, MASTER_DEMOTE_TO_SLAVE=<bool> option instead.
- `slave_pos` - Replicate in GTID mode and use gtid_slave_pos as the position to start downloading transactions from the primary. From MariaDB 10.5.1, `replica_pos` is an alias for `slave_pos`.
- `no` - Don't replicate in GTID mode.

For example:

```
STOP SLAVE;
CHANGE MASTER TO
   MASTER_USE_GTID = current_pos;
START SLAVE;
```

Or:

```
STOP SLAVE;
SET GLOBAL gtid_slave_pos='0-1-153';
CHANGE MASTER TO
   MASTER_USE_GTID = slave_pos;
START SLAVE;
```

MASTER_DEMOTE_TO_SLAVE

MariaDB starting with 10.10

Used to transition a primary to become a replica. Replaces the old MASTER_USE_GTID=current_pos with a safe alternative by forcing users to set `Using_Gtid=Slave_Pos` and merging `gtid_binlog_pos` into `gtid_slave_pos` once at `CHANGE MASTER TO` time. If `gtid_slave_pos` is more recent than `gtid_binlog_pos` (as in the case of chain replication), the replication state should be preserved.

For example:

```
STOP SLAVE;
CHANGE MASTER TO
   MASTER_DEMOTE_TO_SLAVE = 1;
START SLAVE;
```

Replication Filter Options

Also see Replication filters.

IGNORE_SERVER_IDS

The `IGNORE_SERVER_IDS` option for `CHANGE MASTER` can be used to configure a replica to ignore binary log events that originated from certain servers. Filtered binary log events will not get logged to the replica's relay log, and they will not be applied by the replica.

The option's value can be specified by providing a comma-separated list of server_id values. For example:

```
STOP SLAVE;
CHANGE MASTER TO
   IGNORE_SERVER_IDS = (3,5);
START SLAVE;
```

If you would like to clear a previously set list, then you can set the value to an empty list. For example:

```
STOP SLAVE;
CHANGE MASTER TO
   IGNORE_SERVER_IDS = ();
START SLAVE;
```

DO_DOMAIN_IDS

The `DO_DOMAIN_IDS` option for `CHANGE MASTER` can be used to configure a replica to only apply binary log events if the transaction's GTID is in a specific gtid_domain_id value. Filtered binary log events will not get logged to the replica's relay log, and they will not be applied by the replica.

The option's value can be specified by providing a comma-separated list of gtid_domain_id values. Duplicate values are automatically ignored. For example:

```
STOP SLAVE;
CHANGE MASTER TO
   DO_DOMAIN_IDS = (1,2);
START SLAVE;
```

If you would like to clear a previously set list, then you can set the value to an empty list. For example:

```
STOP SLAVE;
CHANGE MASTER TO
   DO_DOMAIN_IDS = ();
START SLAVE;
```

The DO_DOMAIN_IDS option and the IGNORE_DOMAIN_IDS option cannot both be set to non-empty values at the same time. If you want to set the DO_DOMAIN_IDS option, and the IGNORE_DOMAIN_IDS option was previously set, then you need to clear the value of the IGNORE_DOMAIN_IDS option. For example:

```
STOP SLAVE;
CHANGE MASTER TO
   IGNORE_DOMAIN_IDS = (),
   DO_DOMAIN_IDS = (1,2);
START SLAVE;
```

The `DO_DOMAIN_IDS` option can only be specified if the replica is replicating in GTID mode. Therefore, the MASTER_USE_GTID option must also be set to some value other than `no` in order to use this option.

IGNORE_DOMAIN_IDS

The `IGNORE_DOMAIN_IDS` option for `CHANGE MASTER` can be used to configure a replica to ignore binary log events if the transaction's GTID is in a specific gtid_domain_id value. Filtered binary log events will not get logged to the replica's relay log, and they will not be applied by the replica.

The option's value can be specified by providing a comma-separated list of gtid_domain_id values. Duplicate values are automatically ignored. For example:

```
STOP SLAVE;
CHANGE MASTER TO
   IGNORE_DOMAIN_IDS = (1,2);
START SLAVE;
```

If you would like to clear a previously set list, then you can set the value to an empty list. For example:

```
STOP SLAVE;
CHANGE MASTER TO
   IGNORE_DOMAIN_IDS = ();
START SLAVE;
```

The DO_DOMAIN_IDS option and the IGNORE_DOMAIN_IDS option cannot both be set to non-empty values at the same time. If you want to set the IGNORE_DOMAIN_IDS option, and the DO_DOMAIN_IDS option was previously set, then you need to clear the value of the DO_DOMAIN_IDS option. For example:

```
STOP SLAVE;
CHANGE MASTER TO
   DO_DOMAIN_IDS = (),
   IGNORE_DOMAIN_IDS = (1,2);
START SLAVE;
```

The `IGNORE_DOMAIN_IDS` option can only be specified if the replica is replicating in GTID mode. Therefore, the MASTER_USE_GTID option must also be set to some value other than `no` in order to use this option.

Delayed Replication Options

MASTER_DELAY

The `MASTER_DELAY` option for `CHANGE MASTER` can be used to enable delayed replication. This option specifies the time in seconds (at least) that a replica should lag behind the primary up to a maximum value of 2147483647, or about 68 years. Before executing an event, the replica will first wait, if necessary, until the given time has passed since the event was created on the primary. The result is that the replica will reflect the state of the primary some time back in the past. The default is zero, no delay.

```
STOP SLAVE;
CHANGE MASTER TO
   MASTER_DELAY=3600;
START SLAVE;
```

Changing Option Values

If you don't specify a given option when executing the `CHANGE MASTER` statement, then the option keeps its old value in most cases. Most of the time, there is no need to specify the options that do not need to change. For example, if the password for the user account that the replica uses to connect to its primary has changed, but no other options need to change, then you can just change the MASTER_PASSWORD option by executing the following commands:

```
STOP SLAVE;
CHANGE MASTER TO
   MASTER_PASSWORD='new3cret';
START SLAVE;
```

There are some cases where options are implicitly reset, such as when the MASTER_HOST and MASTER_PORT options are changed.

Option Persistence

The values of the MASTER_LOG_FILE and MASTER_LOG_POS options (i.e. the binary log position on the primary) and most other options are written to either the default `master.info` file or the file that is configured by the master_info_file option. The replica's I/O thread keeps this binary log position updated as it downloads events only when MASTER_USE_GTID option is set to `NO`. Otherwise the file is not updated on a per event basis.

The master_info_file option can be set either on the command-line or in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
master_info_file=/mariadb/myserver1-master.info
```

The values of the RELAY_LOG_FILE and RELAY_LOG_POS options (i.e. the relay log position) are written to either the default `relay-log.info` file or the file that is configured by the relay_log_info_file system variable. The replica's SQL thread keeps this relay log position updated as it applies events.

The relay_log_info_file system variable can be set either on the command-line or in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
relay_log_info_file=/mariadb/myserver1-relay-log.info
```

GTID Persistence

If the replica is replicating binary log events that contain GTIDs, then the replica's SQL thread will write every GTID that it applies to the mysql.gtid_slave_pos table. This GTID can be inspected and modified through the gtid_slave_pos system variable.

If the replica has the log_slave_updates system variable enabled and if the replica has the binary log enabled, then every write by the replica's SQL thread will also go into the replica's binary log. This means that GTIDs of replicated transactions would be reflected in the value of the gtid_binlog_pos system variable.

Creating a Replica from a Backup

The `CHANGE MASTER` statement is useful for setting up a replica when you have a backup of the primary and you also have

the binary log position or GTID position corresponding to the backup.

After restoring the backup on the replica, you could execute something like this to use the binary log position:

```
CHANGE MASTER TO
   MASTER_LOG_FILE='master2-bin.001',
   MASTER_LOG_POS=4;
START SLAVE;
```

Or you could execute something like this to use the GTID position:

```
SET GLOBAL gtid_slave_pos='0-1-153';
CHANGE MASTER TO
   MASTER_USE_GTID=slave_pos;
START SLAVE;
```

See Setting up a Replication Slave with Mariabackup for more information on how to do this with Mariabackup.

Example

The following example changes the primary and primary's binary log coordinates. This is used when you want to set up the replica to replicate the primary:

```
CHANGE MASTER TO
   MASTER_HOST='master2.mycompany.com',
   MASTER_USER='replication',
   MASTER_PASSWORD='bigs3cret',
   MASTER_PORT=3306,
   MASTER_LOG_FILE='master2-bin.001',
   MASTER_LOG_POS=4,
   MASTER_CONNECT_RETRY=10;
START SLAVE;
```

1.1.1.2.5.2 START SLAVE

The terms *master* and *slave* have historically been used in replication, but the terms terms *primary* and *replica* are now preferred. The old terms are used still used in parts of the documentation, and in MariaDB commands, although MariaDB 10.5 has begun the process of renaming. The documentation process is ongoing. See MDEV-18777 to follow progress on this effort.

Syntax

```
START SLAVE ["connection_name"] [thread_type [, thread_type] ... ] [FOR CHANNEL
"connection_name"]
START SLAVE ["connection_name"] [SQL_THREAD] UNTIL
    MASTER_LOG_FILE = 'log_name', MASTER_LOG_POS = log_pos [FOR CHANNEL "connection_name"]
START SLAVE ["connection_name"] [SQL_THREAD] UNTIL
    RELAY_LOG_FILE = 'log_name', RELAY_LOG_POS = log_pos [FOR CHANNEL "connection_name"]
START SLAVE ["connection_name"] [SQL_THREAD] UNTIL
    MASTER_GTID_POS = <GTID position> [FOR CHANNEL "connection_name"]
START ALL SLAVES [thread_type [, thread_type]]

START REPLICA ["connection_name"] [thread_type [, thread_type] ... ] -- from 10.5.1
START REPLICA ["connection_name"] [SQL_THREAD] UNTIL
    MASTER_LOG_FILE = 'log_name', MASTER_LOG_POS = log_pos -- from 10.5.1
START REPLICA ["connection_name"] [SQL_THREAD] UNTIL
    RELAY_LOG_FILE = 'log_name', RELAY_LOG_POS = log_pos -- from 10.5.1
START REPLICA ["connection_name"] [SQL_THREAD] UNTIL
    MASTER_GTID_POS = <GTID position> -- from 10.5.1
START ALL REPLICAS [thread_type [, thread_type]] -- from 10.5.1

thread_type: IO_THREAD | SQL_THREAD
```

Contents

Description

`START SLAVE` (`START REPLICA` from MariaDB 10.5.1) with no thread_type options starts both of the replica threads (see replication). The I/O thread reads events from the primary server and stores them in the relay log. The SQL thread reads events from the relay log and executes them. `START SLAVE` requires the SUPER privilege, or, from MariaDB 10.5.2, the REPLICATION SLAVE ADMIN privilege.

If `START SLAVE` succeeds in starting the replica threads, it returns without any error. However, even in that case, it might be that the replica threads start and then later stop (for example, because they do not manage to connect to the primary or read its binary log, or some other problem). `START SLAVE` does not warn you about this. You must check the replica's error log for error messages generated by the replica threads, or check that they are running satisfactorily with SHOW SLAVE STATUS (SHOW REPLICA STATUS from MariaDB 10.5.1).

START SLAVE UNTIL

`START SLAVE UNTIL` refers to the `SQL_THREAD` replica position at which the `SQL_THREAD` replication will halt. If `SQL_THREAD` isn't specified both threads are started.

`START SLAVE UNTIL master_gtid_pos=xxx` is also supported. See Global Transaction ID/START SLAVE UNTIL master_gtid_pos=xxx for more details.

MariaDB starting with 11.3.0

SQL_BEFORE_GTIDS|SQL_AFTER_GTIDS

MariaDB 11.3 extended the START SLAVE UNTIL command with the options `SQL_BEFORE_GTIDS` and `SQL_AFTER_GTIDS` to allow control of whether the replica stops before or after a provided GTID state. Its syntax is:

```
START SLAVE UNTIL (SQL_BEFORE_GTIDS|SQL_AFTER_GTIDS)="<gtid_list>"
```

See Global Transaction ID#SQL_BEFORE_GTIDS/SQL_AFTER_GTIDS for details.

connection_name

If there is only one nameless primary, or the default primary (as specified by the default_master_connection system variable) is intended, `connection_name` can be omitted. If provided, the `START SLAVE` statement will apply to the specified primary. `connection_name` is case-insensitive.

MariaDB starting with 10.7.0

The `FOR CHANNEL` keyword was added for MySQL compatibility. This is identical as using the channel_name directly after `START SLAVE`.

START ALL SLAVES

`START ALL SLAVES` starts all configured replicas (replicas with master_host not empty) that were not started before. It will give a `note` for all started connections. You can check the notes with SHOW WARNINGS.

START REPLICA

MariaDB starting with 10.5.1

`START REPLICA` is an alias for `START SLAVE` from MariaDB 10.5.1.

1.1.1.2.5.3 STOP SLAVE

The terms *master* and *slave* have historically been used in replication, but the terms terms *primary* and *replica* are now preferred. The old terms are used still used in parts of the documentation, and in MariaDB commands, although MariaDB 10.5 has begun the process of renaming. The documentation process is ongoing. See MDEV-18777 to follow progress on this effort.

Syntax

```
STOP SLAVE ["connection_name"] [thread_type [, thread_type] ... ] [FOR CHANNEL "connection_name"]

STOP ALL SLAVES [thread_type [, thread_type]]

STOP REPLICA ["connection_name"] [thread_type [, thread_type] ... ] -- from 10.5.1

STOP ALL REPLICAS [thread_type [, thread_type]] -- from 10.5.1

thread_type: IO_THREAD | SQL_THREAD
```

Contents

Description

Stops the replica threads. `STOP SLAVE` requires the SUPER privilege, or, from MariaDB 10.5.2, the REPLICATION SLAVE ADMIN privilege.

Like START SLAVE, this statement may be used with the `IO_THREAD` and `SQL_THREAD` options to name the thread or threads to be stopped. In almost all cases, one never need to use the `thread_type` options.

`STOP SLAVE` waits until any current replication event group affecting one or more non-transactional tables has finished executing (if there is any such replication group), or until the user issues a KILL QUERY or KILL CONNECTION statement.

Note that `STOP SLAVE` doesn't delete the connection permanently. Next time you execute START SLAVE or the MariaDB server restarts, the replica connection is restored with it's original arguments. If you want to delete a connection, you should execute RESET SLAVE.

STOP ALL SLAVES

`STOP ALL SLAVES` stops all your running replicas. It will give you a `note` for every stopped connection. You can check the notes with SHOW WARNINGS.

connection_name

The `connection_name` option is used for multi-source replication.

If there is only one nameless master, or the default master (as specified by the default_master_connection system variable) is intended, `connection_name` can be omitted. If provided, the `STOP SLAVE` statement will apply to the specified master. `connection_name` is case-insensitive.

MariaDB starting with 10.7.0

The `FOR CHANNEL` keyword was added for MySQL compatibility. This is identical as using the channel_name directly after `STOP SLAVE`.

STOP REPLICA

MariaDB starting with 10.5.1

`STOP REPLICA` is an alias for `STOP SLAVE` from MariaDB 10.5.1.

1.1.1.2.5.4 RESET REPLICA/SLAVE

The terms *master* and *slave* have historically been used in replication, but the terms terms *primary* and *replica* are now preferred. The old terms are used still used in parts of the documentation, and in MariaDB commands, although MariaDB 10.5 has begun the process of renaming. The documentation process is ongoing. See MDEV-18777 to follow progress on this effort.

Syntax

```
RESET REPLICA ["connection_name"] [ALL]  [FOR CHANNEL "connection_name"] -- from MariaDB 10.5.1
RESET SLAVE ["connection_name"] [ALL]  [FOR CHANNEL "connection_name"]
```

Contents

Description

RESET REPLICA/SLAVE makes the replica forget its replication position in the master's binary log. This statement is meant to be used for a clean start. It deletes the master.info and relay-log.info files, all the relay log files, and starts a new relay log file. To use RESET REPLICA/SLAVE, the replica threads must be stopped (use STOP REPLICA/SLAVE if necessary).

> Note: All relay log files are deleted, even if they have not been completely executed by the slave SQL thread. (This is a condition likely to exist on a replication slave if you have issued a STOP REPLICA/SLAVE statement or if the slave is highly loaded.)

> Note: `RESET REPLICA` does not reset the global `gtid_slave_pos` variable. This means that a replica server configured with `CHANGE MASTER TO MASTER_USE_GTID=slave_pos` will not receive events with GTIDs occurring before the state saved in `gtid_slave_pos`. If the intent is to reprocess these events, `gtid_slave_pos` must be manually reset, e.g. by executing `set global gtid_slave_pos=""`.

Connection information stored in the master.info file is immediately reset using any values specified in the corresponding startup options. This information includes values such as master host, master port, master user, and master password. If the replica SQL thread was in the middle of replicating temporary tables when it was stopped, and RESET REPLICA/SLAVE is issued, these replicated temporary tables are deleted on the slave.

The `ALL` also resets the `PORT`, `HOST`, `USER` and `PASSWORD` parameters for the slave. If you are using a connection name, it will permanently delete it and it will not show up anymore in SHOW ALL REPLICAS/SLAVE STATUS.

connection_name

The `connection_name` option is used for multi-source replication.

If there is only one nameless primary, or the default primary (as specified by the default_master_connection system variable) is intended, `connection_name` can be omitted. If provided, the `RESET REPLICA/SLAVE` statement will apply to the specified primary. `connection_name` is case-insensitive.

MariaDB starting with 10.7.0

The `FOR CHANNEL` keyword was added for MySQL compatibility. This is identical as using the channel_name directly after `RESET REPLICA`.

RESET REPLICA

MariaDB starting with 10.5.1

`RESET REPLICA` is an alias for `RESET SLAVE` from MariaDB 10.5.1.

1.1.1.2.5.5 SET GLOBAL SQL_SLAVE_SKIP_COUNTER

Syntax

```
SET GLOBAL sql_slave_skip_counter = N
```

Contents

Description

This statement skips the next `N` events from the primary. This is useful for recovering from replication stops caused by a statement.

If multi-source replication is used, this statement applies to the default connection. It could be necessary to change the value of the default_master_connection system variable.

Note that, if the event is a transaction, the whole transaction will be skipped. With non-transactional engines, an event is always a single statement.

This statement is valid only when the replica threads are not running. Otherwise, it produces an error.

The statement does not automatically restart the replica threads.

Example

```
SHOW SLAVE STATUS \G
...
SET GLOBAL sql_slave_skip_counter = 1;
START SLAVE;
```

Multi-source replication:

```
SET @@default_master_connection = 'master_01';
SET GLOBAL SQL_SLAVE_SKIP_COUNTER = 1;
START SLAVE;
```

Multiple Replication Domains

`sql_slave_skip_counter` can't be used to skip transactions on a replica if GTID replication is in use and if gtid_slave_pos contains multiple gtid_domain_id values. In that case, you'll get an error like the following:

```
ERROR 1966 (HY000): When using parallel replication and GTID with multiple
 replication domains, @@sql_slave_skip_counter can not be used. Instead,
 setting @@gtid_slave_pos explicitly can be  used to skip to after a given GTID
 position.
```

In order to skip transactions in cases like this, you will have to manually change gtid_slave_pos.

1.1.1.2.5.6 SHOW RELAYLOG EVENTS

> The terms *master* and *slave* have historically been used in replication, but the terms terms *primary* and *replica* are now preferred. The old terms are used still used in parts of the documentation, and in MariaDB commands, although MariaDB 10.5 has begun the process of renaming. The documentation process is ongoing. See MDEV-18777 to follow progress on this effort.

Syntax

```
SHOW RELAYLOG ['connection_name'] EVENTS
    [IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]
    [ FOR CHANNEL 'channel_name']
```

Description

On replicas, this command shows the events in the relay log. If `'log_name'` is not specified, the first relay log is shown.

Syntax for the `LIMIT` clause is the same as for SELECT ... LIMIT.

> Using the `LIMIT` clause is highly recommended because the `SHOW RELAYLOG EVENTS` command returns the complete contents of the relay log, which can be quite large.

This command does not return events related to setting user and system variables. If you need those, use mariadb-binlog.

On the primary, this command does nothing.

Requires the REPLICA MONITOR privilege (>= MariaDB 10.5.9), the REPLICATION SLAVE ADMIN privilege (>= MariaDB 10.5.2) or the REPLICATION SLAVE privilege (<= MariaDB 10.5.1).

connection_name

If there is only one nameless primary, or the default primary (as specified by the default_master_connection system variable) is intended, `connection_name` can be omitted. If provided, the `SHOW RELAYLOG` statement will apply to the specified primary. `connection_name` is case-insensitive.

MariaDB starting with 10.7.0

The `FOR CHANNEL` keyword was added for MySQL compatibility. This is identical as using the channel_name directly after `SHOW RELAYLOG`.

1.1.1.2.5.7 SHOW SLAVE STATUS

Syntax

```
SHOW SLAVE ["connection_name"] STATUS [FOR CHANNEL "connection_name"]
SHOW REPLICA ["connection_name"] STATUS -- From MariaDB 10.5.1
```

or

```
SHOW ALL SLAVES STATUS
SHOW ALL REPLICAS STATUS -- From MariaDB 10.5.1
```

Contents

Description

This statement is to be run on a replica and provides status information on essential parameters of the replica threads.

This statement requires the SUPER privilege, the REPLICATION_CLIENT privilege, or, from MariaDB 10.5.2, the REPLICATION SLAVE ADMIN privilege, or, from MariaDB 10.5.9, the REPLICA MONITOR privilege.

Multi-Source

The `ALL` and `"connection_name"` options allow you to connect to many primaries at the same time.

`ALL SLAVES` (or `ALL REPLICAS` from MariaDB 10.5.1) gives you a list of all connections to the primary nodes.

The rows will be sorted according to `Connection_name`.

If you specify a `connection_name`, you only get the information about that connection. If `connection_name` is not used, then the name set by `default_master_connection` is used. If the connection name doesn't exist you will get an error: `There is no master connection for 'xxx'`.

MariaDB starting with 10.7.0

The `FOR CHANNEL` keyword was added for MySQL compatibility. This is identical to using the channel_name directly after `SHOW SLAVE`.

Column Descriptions

The order in which the columns appear depends on the MariaDB version. This means that extracting a column value is best done by comparing the field name instead of using a fixed offset into the row.

Name	Description
Connection_name	Name of the primary connection. Returned with `SHOW ALL SLAVES STATUS` (or `SHOW ALL REPLICAS STATUS` from MariaDB 10.5.1) only.
Slave_SQL_State	State of SQL thread. Returned with `SHOW ALL SLAVES STATUS` (or `SHOW ALL REPLICAS STATUS` from MariaDB 10.5.1) only. See Slave SQL Thread States.
Slave_IO_State	State of I/O thread. See Slave I/O Thread States.
Master_host	Master host that the replica is connected to.
Master_user	Account user name being used to connect to the primary.
Master_port	The port being used to connect to the primary.
Connect_Retry	Time in seconds between retries to connect. The default is 60. The CHANGE MASTER TO statement can set this. The master-retry-count option determines the maximum number of reconnection attempts.
Master_Log_File	Name of the primary binary log file that the I/O thread is currently reading from.
Read_Master_Log_Pos	Position up to which the I/O thread has read in the current primary binary log file.
Relay_Log_File	Name of the relay log file that the SQL thread is currently processing.
Relay_Log_Pos	Position up to which the SQL thread has finished processing in the current relay log file.
Relay_Master_Log_File	Name of the primary binary log file that contains the most recent event executed by the SQL thread.
Slave_IO_Running	Whether the replica I/O thread is running and connected (`Yes`), running but not connected to a primary (`Connecting`) or not running (`No`).
Slave_SQL_Running	Whether or not the SQL thread is running.
Replicate_Rewrite_DB	Databases specified for replicating and rewriting with the `replicate_rewrite_db` option. Added in MariaDB 10.11
Replicate_Do_DB	Databases specified for replicating with the `replicate_do_db` option.
Replicate_Ignore_DB	Databases specified for ignoring with the `replicate_ignore_db` option.
Replicate_Do_Table	Tables specified for replicating with the `replicate_do_table` option.
Replicate_Ignore_Table	Tables specified for ignoring with the `replicate_ignore_table` option.
Replicate_Wild_Do_Table	Tables specified for replicating with the `replicate_wild_do_table` option.
Replicate_Wild_Ignore_Table	Tables specified for ignoring with the `replicate_wild_ignore_table` option.
Last_Errno	Alias for `Last_SQL_Errno` (see below)
Last Error	Alias for `Last_SQL_Error` (see below)
Skip_Counter	Number of events that a replica skips from the master, as recorded in the sql_slave_skip_counter system variable.

Exec_Master_Log_Pos	Position up to which the SQL thread has processed in the current master binary log file. Can be used to start a new replica from a current replica with the CHANGE MASTER TO ... MASTER_LOG_POS option.
Relay_Log_Space	Total size of all relay log files combined.
Until_Condition	
Until_Log_File	The `MASTER_LOG_FILE` value of the START SLAVE UNTIL condition.
Until_Log_Pos	The `MASTER_LOG_POS` value of the START SLAVE UNTIL condition.
Master_SSL_Allowed	Whether an SSL connection is permitted (`Yes`), not permitted (`No`) or permitted but without the replica having SSL support enabled (`Ignored`)
Master_SSL_CA_File	The `MASTER_SSL_CA` option of the `CHANGE MASTER TO` statement.
Master_SSL_CA_Path	The `MASTER_SSL_CAPATH` option of the `CHANGE MASTER TO` statement.
Master_SSL_Cert	The `MASTER_SSL_CERT` option of the `CHANGE MASTER TO` statement.
Master_SSL_Cipher	The `MASTER_SSL_CIPHER` option of the `CHANGE MASTER TO` statement.
Master_SSL_Key	The `MASTER_SSL_KEY` option of the `CHANGE MASTER TO` statement.
Seconds_Behind_Master	Difference between the timestamp logged on the master for the event that the replica is currently processing, and the current timestamp on the replica. Zero if the replica is not currently processing an event. With serial replication, `seconds_behind_master` is updated when the SQL thread begins executing a transaction. With parallel replication, `seconds_behind_master` is updated only after transactions commit. Starting in MariaDB 10.3.38, 10.4.28, 10.5.19, 10.6.12, 10.8.7, 10.9.5, 10.10.3, and 10.11.2, an exception is drawn on the parallel replica to additionally update `seconds_behind_master` when the first transaction received after idling is queued to a worker for execution, to provide a reliable initial value for the duration until a transaction commits. Additional behaviors to be aware of are as follows: 1) `Seconds_Behind_Master` will update for ignored events, e.g. those skipped due to sql_slave_skip_counter. 2) On the serial replica, transactions with prior timestamps can update `Seconds_Behind_Master` such that it can go backwards, though this is not true for the parallel replica. 3) When configured with MASTER_DELAY, as a replicated transaction begins executing (i.e. on a serial or post-idle parallel replica), `Seconds_Behind_Master` will update before delaying, and while delaying occurs will grow to encompass the configured value. 4) There is a known issue, tracked by MDEV-17516, such that `Seconds_Behind_Master` will initially present as `0` on replica restart until a replicated transaction begins executing, even if the last replica session was lagging behind when stopped.
Master_SSL_Verify_Server_Cert	The `MASTER_SSL_VERIFY_SERVER_CERT` option of the `CHANGE MASTER TO` statement.
Last_IO_Errno	Error code of the most recent error that caused the I/O thread to stop (also recorded in the replica's error log). `0` means no error. RESET SLAVE or RESET MASTER will reset this value.
Last_IO_Error	Error message of the most recent error that caused the I/O thread to stop (also recorded in the replica's error log). An empty string means no error. RESET SLAVE or RESET MASTER will reset this value.
Last_SQL_Errno	Error code of the most recent error that caused the SQL thread to stop (also recorded in the replica's error log). `0` means no error. RESET SLAVE or RESET MASTER will reset this value.
Last_SQL_Error	Error message of the most recent error that caused the SQL thread to stop (also recorded in the replica's error log). An empty string means no error. RESET SLAVE or RESET MASTER will reset this value.
Replicate_Ignore_Server_Ids	List of server_ids that are currently being ignored for replication purposes, or an empty string for none, as specified in the `IGNORE_SERVER_IDS` option of the `CHANGE MASTER TO` statement.
Master_Server_Id	The master's server_id value.
Master_SSL_Crl	The `MASTER_SSL_CRL` option of the `CHANGE MASTER TO` statement.

Master_SSL_Crlpath	The `MASTER_SSL_CRLPATH` option of the `CHANGE MASTER TO` statement.
Using_Gtid	Whether or not global transaction ID's are being used for replication (can be `No`, `Slave_Pos`, or `Current_Pos`).
Gtid_IO_Pos	Current global transaction ID value.
Retried_transactions	Number of retried transactions for this connection. Returned with `SHOW ALL SLAVES STATUS` only.
Max_relay_log_size	Max relay log size for this connection. Returned with `SHOW ALL SLAVES STATUS` only.
Executed_log_entries	How many log entries the replica has executed. Returned with `SHOW ALL SLAVES STATUS` only.
Slave_received_heartbeats	How many heartbeats we have got from the master. Returned with `SHOW ALL SLAVES STATUS` only.
Slave_heartbeat_period	How often to request a heartbeat packet from the master (in seconds). Returned with `SHOW ALL SLAVES STATUS` only.
Gtid_Slave_Pos	GTID of the last event group replicated on a replica server, for each replication domain, as stored in the gtid_slave_pos system variable. Returned with `SHOW ALL SLAVES STATUS` only.
SQL_Delay	Value specified by `MASTER_DELAY` in `CHANGE MASTER` (or 0 if none).
SQL_Remaining_Delay	When the replica is delaying the execution of an event due to `MASTER_DELAY`, this is the number of seconds of delay remaining before the event will be applied. Otherwise, the value is `NULL`.
Slave_SQL_Running_State	The state of the SQL driver threads, same as in SHOW PROCESSLIST. When the replica is delaying the execution of an event due to `MASTER_DELAY`, this field displays: "`Waiting until MASTER_DELAY seconds after master executed event`".
Slave_DDL_Groups	This status variable counts the occurrence of DDL statements. This is a replica-side counter for optimistic parallel replication.
Slave_Non_Transactional_Groups	This status variable counts the occurrence of non-transactional event groups. This is a replica-side counter for optimistic parallel replication.
Slave_Transactional_Groups	This status variable counts the occurrence of transactional event groups. This is a replica-side counter for optimistic parallel replication.

SHOW REPLICA STATUS

MariaDB starting with 10.5.1

`SHOW REPLICA STATUS` is an alias for `SHOW SLAVE STATUS` from MariaDB 10.5.1.

Examples

If you issue this statement using the mariadb client, you can use a `\G` statement terminator rather than a semicolon to obtain a more readable vertical layout.

```
SHOW SLAVE STATUS\G
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: db01.example.com
                  Master_User: replicant
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: mariadb-bin.000010
          Read_Master_Log_Pos: 548
               Relay_Log_File: relay-bin.000004
                Relay_Log_Pos: 837
        Relay_Master_Log_File: mariadb-bin.000010
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 548
              Relay_Log_Space: 1497
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Master_SSL_Allowed: No
           Master_SSL_CA_File:
           Master_SSL_CA_Path:
              Master_SSL_Cert:
            Master_SSL_Cipher:
               Master_SSL_Key:
        Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 0
                Last_IO_Error:
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Master_Server_Id: 101
               Master_SSL_Crl:
           Master_SSL_Crlpath:
                   Using_Gtid: No
                  Gtid_IO_Pos:
```

```
SHOW ALL SLAVES STATUS\G
*************************** 1. row ***************************
               Connection_name:
               Slave_SQL_State: Slave has read all relay log; waiting for the slave I/O thread
to update it
                Slave_IO_State: Waiting for master to send event
                   Master_Host: db01.example.com
                   Master_User: replicant
                   Master_Port: 3306
                  Connect_Retry: 60
                Master_Log_File: mariadb-bin.000010
            Read_Master_Log_Pos: 3608
                 Relay_Log_File: relay-bin.000004
                  Relay_Log_Pos: 3897
          Relay_Master_Log_File: mariadb-bin.000010
               Slave_IO_Running: Yes
              Slave_SQL_Running: Yes
                Replicate_Do_DB:
            Replicate_Ignore_DB:
             Replicate_Do_Table:
         Replicate_Ignore_Table:
        Replicate_Wild_Do_Table:
    Replicate_Wild_Ignore_Table:
                     Last_Errno: 0
                     Last_Error:
                   Skip_Counter: 0
            Exec_Master_Log_Pos: 3608
                Relay_Log_Space: 4557
                Until_Condition: None
                 Until_Log_File:
                  Until_Log_Pos: 0
             Master_SSL_Allowed: No
             Master_SSL_CA_File:
             Master_SSL_CA_Path:
                Master_SSL_Cert:
              Master_SSL_Cipher:
                 Master_SSL_Key:
          Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
                 Last_IO_Errno: 0
                 Last_IO_Error:
                Last_SQL_Errno: 0
                Last_SQL_Error:
    Replicate_Ignore_Server_Ids:
               Master_Server_Id: 101
                 Master_SSL_Crl:
             Master_SSL_Crlpath:
                     Using_Gtid: No
                    Gtid_IO_Pos:
           Retried_transactions: 0
             Max_relay_log_size: 104857600
           Executed_log_entries: 40
      Slave_received_heartbeats: 11
         Slave_heartbeat_period: 1800.000
                 Gtid_Slave_Pos: 0-101-2320
```

You can also access some of the variables directly from status variables:

```
SET @@default_master_connection="test" ;
show status like "%slave%"

Variable_name   Value
Com_show_slave_hosts    0
Com_show_slave_status   0
Com_start_all_slaves    0
Com_start_slave 0
Com_stop_all_slaves     0
Com_stop_slave  0
Rpl_semi_sync_slave_status      OFF
Slave_connections       0
Slave_heartbeat_period  1800.000
Slave_open_temp_tables  0
Slave_received_heartbeats       0
Slave_retried_transactions      0
Slave_running   OFF
Slaves_connected        0
Slaves_running  1
```

1.1.1.2.5.8 SHOW MASTER STATUS

Syntax

```
SHOW MASTER STATUS
SHOW BINLOG STATUS -- From MariaDB 10.5.2
```

Description

Provides status information about the binary log files of the primary.

This statement requires the SUPER privilege, the REPLICATION_CLIENT privilege, or, from MariaDB 10.5.2, the BINLOG MONITOR privilege.

To see information about the current GTIDs in the binary log, use the gtid_binlog_pos variable.

`SHOW MASTER STATUS` was renamed to `SHOW BINLOG STATUS` in MariaDB 10.5.2, but the old name remains an alias for compatibility purposes.

Example

```
SHOW MASTER STATUS;
+--------------------+----------+--------------+------------------+
| File               | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+--------------------+----------+--------------+------------------+
| mariadb-bin.000016 |      475 |              |                  |
+--------------------+----------+--------------+------------------+
SELECT @@global.gtid_binlog_pos;
+---------------------------+
| @@global.gtid_binlog_pos |
+---------------------------+
| 0-1-2                     |
+---------------------------+
```

1.1.1.2.5.9 SHOW SLAVE HOSTS

Contents

Syntax

```
SHOW SLAVE HOSTS
SHOW REPLICA HOSTS -- from MariaDB 10.5.1
```

Description

This command is run on the primary and displays a list of replicas that are currently registered with it. Only replicas started with the `--report-host=host_name` option are visible in this list.

The output looks like this:

```
SHOW SLAVE HOSTS;
+------------+-----------+------+-----------+
| Server_id  | Host      | Port | Master_id |
+------------+-----------+------+-----------+
|  192168010 | iconnect2 | 3306 | 192168011 |
| 1921680101 | athena    | 3306 | 192168011 |
+------------+-----------+------+-----------+
```

- **`Server_id`**: The unique server ID of the replica server, as configured in the server's option file, or on the command line with --server-id=value.
- **`Host`**: The host name of the replica server, as configured in the server's option file, or on the command line with `--report-host=host_name`. Note that this can differ from the machine name as configured in the operating system.
- **`Port`**: The port the replica server is listening on.
- **`Master_id`**: The unique server ID of the primary server that the replica server is replicating from.

Some MariaDB and MySQL versions report another variable, rpl_recovery_rank. This variable was never used, and was eventually removed in MariaDB 10.1.2.

Requires the REPLICATION MASTER ADMIN privilege (>= MariaDB 10.5.2) or the REPLICATION SLAVE privilege (<= MariaDB 10.5.1).

SHOW REPLICA HOSTS

MariaDB starting with 10.5.1

`SHOW REPLICA HOSTS` is an alias for `SHOW SLAVE HOSTS` from MariaDB 10.5.1.

1.1.1.2.5.10 RESET MASTER

```
RESET MASTER [TO #]
```

Deletes all binary log files listed in the index file, resets the binary log index file to be empty, and creates a new binary log file with a suffix of .000001.

If `TO #` is given, then the first new binary log file will start from number #.

> This statement is for use only when the master is started for the first time, and should never be used if any slaves are actively replicating from the binary log.

1.1.1.2.6 Plugin SQL Statements

Plugin commands.

SHOW PLUGINS

Display information about installed plugins.

SHOW PLUGINS SONAME

Information about all available plugins, installed or not.

INSTALL PLUGIN

Install a plugin.

UNINSTALL PLUGIN

Remove a single installed plugin.

INSTALL SONAME

Installs all plugins from a given library.

UNINSTALL SONAME

Remove all plugins belonging to a specified library.

mysql_plugin

Symlink or old name for mariadb-plugin.

1.1.1.2.6.1 SHOW PLUGINS

Syntax

```
SHOW PLUGINS;
```

Contents

Description

`SHOW PLUGINS` displays information about installed plugins. The `Library` column indicates the plugin library - if it is `NULL`, the plugin is built-in and cannot be uninstalled.

The `PLUGINS` table in the `information_schema` database contains more detailed information.

For specific information about storage engines (a particular type of plugin), see the `information_schema.ENGINES` table and the `SHOW ENGINES` statement.

Examples

```
SHOW PLUGINS;
+----------------------------+----------+--------------------+-------------+---------+
| Name                       | Status   | Type               | Library     | License |
+----------------------------+----------+--------------------+-------------+---------+
| binlog                     | ACTIVE   | STORAGE ENGINE     | NULL        | GPL     |
| mysql_native_password      | ACTIVE   | AUTHENTICATION     | NULL        | GPL     |
| mysql_old_password         | ACTIVE   | AUTHENTICATION     | NULL        | GPL     |
| MRG_MyISAM                 | ACTIVE   | STORAGE ENGINE     | NULL        | GPL     |
| MyISAM                     | ACTIVE   | STORAGE ENGINE     | NULL        | GPL     |
| CSV                        | ACTIVE   | STORAGE ENGINE     | NULL        | GPL     |
| MEMORY                     | ACTIVE   | STORAGE ENGINE     | NULL        | GPL     |
| FEDERATED                  | ACTIVE   | STORAGE ENGINE     | NULL        | GPL     |
| PERFORMANCE_SCHEMA         | ACTIVE   | STORAGE ENGINE     | NULL        | GPL     |
| Aria                       | ACTIVE   | STORAGE ENGINE     | NULL        | GPL     |
| InnoDB                     | ACTIVE   | STORAGE ENGINE     | NULL        | GPL     |
| INNODB_TRX                 | ACTIVE   | INFORMATION SCHEMA | NULL        | GPL     |
...
| INNODB_SYS_FOREIGN         | ACTIVE   | INFORMATION SCHEMA | NULL        | GPL     |
| INNODB_SYS_FOREIGN_COLS    | ACTIVE   | INFORMATION SCHEMA | NULL        | GPL     |
| SPHINX                     | ACTIVE   | STORAGE ENGINE     | NULL        | GPL     |
| ARCHIVE                    | ACTIVE   | STORAGE ENGINE     | NULL        | GPL     |
| BLACKHOLE                  | ACTIVE   | STORAGE ENGINE     | NULL        | GPL     |
| FEEDBACK                   | DISABLED | INFORMATION SCHEMA | NULL        | GPL     |
| partition                  | ACTIVE   | STORAGE ENGINE     | NULL        | GPL     |
| pam                        | ACTIVE   | AUTHENTICATION     | auth_pam.so | GPL     |
+----------------------------+----------+--------------------+-------------+---------+
```

1.1.1.2.6.2 SHOW PLUGINS SONAME

Syntax

```
SHOW PLUGINS SONAME { library | LIKE 'pattern' | WHERE expr };
```

Description

`SHOW PLUGINS SONAME` displays information about compiled-in and all server plugins in the `plugin_dir` directory, including plugins that haven't been installed.

Examples

```
SHOW PLUGINS SONAME 'ha_example.so';
+----------+---------------+----------------+---------------+---------+
| Name     | Status        | Type           | Library       | License |
+----------+---------------+----------------+---------------+---------+
| EXAMPLE  | NOT INSTALLED | STORAGE ENGINE | ha_example.so | GPL     |
| UNUSABLE | NOT INSTALLED | DAEMON         | ha_example.so | GPL     |
+----------+---------------+----------------+---------------+---------+
```

There is also a corresponding `information_schema` table, called `ALL_PLUGINS`, which contains more complete information.

1.1.1.2.6.3 INSTALL PLUGIN

Syntax

```
INSTALL PLUGIN [IF NOT EXISTS] plugin_name SONAME 'plugin_library'
```

Contents

Description

This statement installs an individual plugin from the specified library. To install the whole library (which could be required), use INSTALL SONAME. See also Installing a Plugin.

`plugin_name` is the name of the plugin as defined in the plugin declaration structure contained in the library file. Plugin names are not case sensitive. For maximal compatibility, plugin names should be limited to ASCII letters, digits, and underscore, because they are used in C source files, shell command lines, M4 and Bourne shell scripts, and SQL environments.

`plugin_library` is the name of the shared library that contains the plugin code. The file name extension can be omitted (which makes the statement look the same on all architectures).

The shared library must be located in the plugin directory (that is, the directory named by the plugin_dir system variable). The library must be in the plugin directory itself, not in a subdirectory. By default, `plugin_dir` is plugin directory under the directory named by the `pkglibdir` configuration variable, but it can be changed by setting the value of `plugin_dir` at server startup. For example, set its value in a `my.cnf` file:

```
[mysqld]
plugin_dir=/path/to/plugin/directory
```

If the value of plugin_dir is a relative path name, it is taken to be relative to the MySQL base directory (the value of the basedir system variable).

`INSTALL PLUGIN` adds a line to the `mysql.plugin` table that describes the plugin. This table contains the plugin name and library file name.

`INSTALL PLUGIN` causes the server to read option (`my.cnf`) files just as during server startup. This enables the plugin to pick up any relevant options from those files. It is possible to add plugin options to an option file even before loading a plugin (if the loose prefix is used). It is also possible to uninstall a plugin, edit `my.cnf` , and install the plugin again. Restarting the plugin this way enables it to the new option values without a server restart.

`INSTALL PLUGIN` also loads and initializes the plugin code to make the plugin available for use. A plugin is initialized by executing its initialization function, which handles any setup that the plugin must perform before it can be used.

To use `INSTALL PLUGIN` , you must have the INSERT privilege for the `mysql.plugin` table.

At server startup, the server loads and initializes any plugin that is listed in the `mysql.plugin` table. This means that a plugin is installed with `INSTALL PLUGIN` only once, not every time the server starts. Plugin loading at startup does not occur if the server is started with the `--skip-grant-tables` option.

When the server shuts down, it executes the de-initialization function for each plugin that is loaded so that the plugin has a chance to perform any final cleanup.

If you need to load plugins for a single server startup when the `--skip-grant-tables` option is given (which tells the server not to read system tables), use the `--plugin-load` mysqld option.

MariaDB starting with 10.4.0

IF NOT EXISTS

When the `IF NOT EXISTS` clause is used, MariaDB will return a note instead of an error if the specified plugin already exists. See SHOW WARNINGS.

Examples

```
INSTALL PLUGIN sphinx SONAME 'ha_sphinx.so';
```

The extension can also be omitted:

```
INSTALL PLUGIN innodb SONAME 'ha_xtradb';
```

From MariaDB 10.4.0:

```
INSTALL PLUGIN IF NOT EXISTS example SONAME 'ha_example';
Query OK, 0 rows affected (0.104 sec)

INSTALL PLUGIN IF NOT EXISTS example SONAME 'ha_example';
Query OK, 0 rows affected, 1 warning (0.000 sec)

SHOW WARNINGS;
+-------+------+--------------------------------------+
| Level | Code | Message                              |
+-------+------+--------------------------------------+
| Note  | 1968 | Plugin 'example' already installed |
+-------+------+--------------------------------------+
```

1.1.1.2.6.4 UNINSTALL PLUGIN

Syntax

```
UNINSTALL PLUGIN [IF EXISTS] plugin_name
```

Contents
1. Syntax
2. Description
   1. IF EXISTS
3. Examples

Description

This statement removes a single installed plugin. To uninstall the whole library which contains the plugin, use UNINSTALL SONAME. You cannot uninstall a plugin if any table that uses it is open.

`plugin_name` must be the name of some plugin that is listed in the mysql.plugin table. The server executes the plugin's deinitialization function and removes the row for the plugin from the `mysql.plugin` table, so that subsequent server restarts will not load and initialize the plugin. `UNINSTALL PLUGIN` does not remove the plugin's shared library file.

To use `UNINSTALL PLUGIN`, you must have the DELETE privilege for the mysql.plugin table.

MariaDB starting with 10.4.0

IF EXISTS

If the `IF EXISTS` clause is used, MariaDB will return a note instead of an error if the plugin does not exist. See SHOW WARNINGS.

Examples

```
UNINSTALL PLUGIN example;
```

From MariaDB 10.4.0:

```
UNINSTALL PLUGIN IF EXISTS example;
Query OK, 0 rows affected (0.099 sec)

UNINSTALL PLUGIN IF EXISTS example;
Query OK, 0 rows affected, 1 warning (0.000 sec)

SHOW WARNINGS;
+-------+------+--------------------------------+
| Level | Code | Message                        |
+-------+------+--------------------------------+
| Note  | 1305 | PLUGIN example does not exist |
+-------+------+--------------------------------+
```

1.1.1.2.6.5 INSTALL SONAME

Syntax

```
INSTALL SONAME 'plugin_library'
```

Contents
1. Syntax
2. Description
3. Examples

Description

This statement is a variant of INSTALL PLUGIN. It installs **all** plugins from a given `plugin_library`. See INSTALL PLUGIN for details.

`plugin_library` is the name of the shared library that contains the plugin code. The file name extension (for example, `libmyplugin.so` or `libmyplugin.dll`) can be omitted (which makes the statement look the same on all architectures).

The shared library must be located in the plugin directory (that is, the directory named by the `plugin_dir` system

variable). The library must be in the plugin directory itself, not in a subdirectory. By default, `plugin_dir` is plugin directory under the directory named by the `pkglibdir` configuration variable, but it can be changed by setting the value of `plugin_dir` at server startup. For example, set its value in a `my.cnf` file:

```
[mysqld]
plugin_dir=/path/to/plugin/directory
```

If the value of plugin_dir is a relative path name, it is taken to be relative to the MySQL base directory (the value of the `basedir` system variable).

`INSTALL SONAME` adds one or more lines to the `mysql.plugin` table that describes the plugin. This table contains the plugin name and library file name.

`INSTALL SONAME` causes the server to read option (`my.cnf`) files just as during server startup. This enables the plugin to pick up any relevant options from those files. It is possible to add plugin options to an option file even before loading a plugin (if the loose prefix is used). It is also possible to uninstall a plugin, edit `my.cnf` , and install the plugin again. Restarting the plugin this way enables it to the new option values without a server restart.

`INSTALL SONAME` also loads and initializes the plugin code to make the plugin available for use. A plugin is initialized by executing its initialization function, which handles any setup that the plugin must perform before it can be used.

To use `INSTALL SONAME` , you must have the INSERT privilege for the `mysql.plugin` table.

At server startup, the server loads and initializes any plugin that is listed in the `mysql.plugin` table. This means that a plugin is installed with `INSTALL SONAME` only once, not every time the server starts. Plugin loading at startup does not occur if the server is started with the `--skip-grant-tables` option.

When the server shuts down, it executes the de-initialization function for each plugin that is loaded so that the plugin has a chance to perform any final cleanup.

If you need to load plugins for a single server startup when the `--skip-grant-tables` option is given (which tells the server not to read system tables), use the `--plugin-load` mysqld option.

If you need to install only one plugin from a library, use the INSTALL PLUGIN statement.

Examples

To load the XtraDB storage engine and all of its `information_schema` tables with one statement, use

```
INSTALL SONAME 'ha_xtradb';
```

This statement can be used instead of `INSTALL PLUGIN` even when the library contains only one plugin:

```
INSTALL SONAME 'ha_sequence';
```

1.1.1.2.6.6 UNINSTALL SONAME

Syntax

```
UNINSTALL SONAME  [IF EXISTS] 'plugin_library'
```

Contents

Description

This statement is a variant of UNINSTALL PLUGIN statement, that removes all plugins belonging to a specified `plugin_library` . See UNINSTALL PLUGIN for details.

`plugin_library` is the name of the shared library that contains the plugin code. The file name extension (for example,

`libmyplugin.so` or `libmyplugin.dll`) can be omitted (which makes the statement look the same on all architectures).

To use `UNINSTALL SONAME` , you must have the DELETE privilege for the `mysql.plugin` table.

MariaDB starting with 10.4.0

IF EXISTS

If the `IF EXISTS` clause is used, MariaDB will return a note instead of an error if the plugin library does not exist. See SHOW WARNINGS.

Examples

To uninstall the XtraDB plugin and all of its `information_schema` tables with one statement, use

```
UNINSTALL SONAME 'ha_xtradb';
```

From MariaDB 10.4.0:

```
UNINSTALL SONAME IF EXISTS 'ha_example';
Query OK, 0 rows affected (0.099 sec)

UNINSTALL SONAME IF EXISTS 'ha_example';
Query OK, 0 rows affected, 1 warning (0.000 sec)

SHOW WARNINGS;
+-------+------+--------------------------------------+
| Level | Code | Message                              |
+-------+------+--------------------------------------+
| Note  | 1305 | SONAME ha_example.so does not exist |
+-------+------+--------------------------------------+
```

1.3.43.12 mysql_plugin

1.1.1.2.7 SET Commands

SET

Set a variable value.

SET CHARACTER SET

Maps all strings sent between the current client and the server with the given mapping.

SET GLOBAL SQL_SLAVE_SKIP_COUNTER

Skips a number of events from the primary.

SET NAMES

The character set used to send statements to the server, and results back to the client.

SET PASSWORD

Assign password to an existing MariaDB user.

SET ROLE

Enable a role.

SET SQL_LOG_BIN

Set binary logging for the current connection.

SET STATEMENT

Set variable values on a per-query basis.

SET TRANSACTION

Sets the transaction isolation level.

SET Variable

Used to insert a value into a variable with a code block.

There are 1 related questions.

1.1.1.2.7.1 SET

Syntax

```
SET variable_assignment [, variable_assignment] ...

variable_assignment:
      user_var_name = expr
    | [GLOBAL | SESSION] system_var_name = expr
    | [@@global. | @@session. | @@]system_var_name = expr
```

Contents

One can also set a user variable in any expression with this syntax:

```
user_var_name:= expr
```

Description

The `SET` statement assigns values to different types of variables that affect the operation of the server or your client. Older versions of MySQL employed `SET OPTION`, but this syntax was deprecated in favor of `SET` without `OPTION`, and was removed in MariaDB 10.0.

Changing a system variable by using the SET statement does not make the change permanently. To do so, the change must be made in a configuration file.

For setting variables on a per-query basis, see SET STATEMENT.

See SHOW VARIABLES for documentation on viewing server system variables.

See Server System Variables for a list of all the system variables.

GLOBAL / SESSION

When setting a system variable, the scope can be specified as either GLOBAL or SESSION.

A global variable change affects all new sessions. It does not affect any currently open sessions, including the one that made the change.

A session variable change affects the current session only.

If the variable has a session value, not specifying either GLOBAL or SESSION will be the same as specifying SESSION. If the variable only has a global value, not specifying GLOBAL or SESSION will apply to the change to the global value.

DEFAULT

Setting a global variable to DEFAULT will restore it to the server default, and setting a session variable to DEFAULT will restore it to the current global value.

Examples

- innodb_sync_spin_loops is a global variable.

- skip_parallel_replication is a session variable.
- max_error_count is both global and session.

```
SELECT VARIABLE_NAME, SESSION_VALUE, GLOBAL_VALUE FROM
 INFORMATION_SCHEMA.SYSTEM_VARIABLES WHERE
  VARIABLE_NAME IN ('max_error_count', 'skip_parallel_replication', 'innodb_sync_spin_loops');
+---------------------------+---------------+--------------+
| VARIABLE_NAME             | SESSION_VALUE | GLOBAL_VALUE |
+---------------------------+---------------+--------------+
| MAX_ERROR_COUNT           | 64            | 64           |
| SKIP_PARALLEL_REPLICATION | OFF           | NULL         |
| INNODB_SYNC_SPIN_LOOPS    | NULL          | 30           |
+---------------------------+---------------+--------------+
```

Setting the session values:

```
SET max_error_count=128;Query OK, 0 rows affected (0.000 sec)

SET skip_parallel_replication=ON;Query OK, 0 rows affected (0.000 sec)

SET innodb_sync_spin_loops=60;
ERROR 1229 (HY000): Variable 'innodb_sync_spin_loops' is a GLOBAL variable
  and should be set with SET GLOBAL

SELECT VARIABLE_NAME, SESSION_VALUE, GLOBAL_VALUE FROM
 INFORMATION_SCHEMA.SYSTEM_VARIABLES WHERE
  VARIABLE_NAME IN ('max_error_count', 'skip_parallel_replication', 'innodb_sync_spin_loops');
+---------------------------+---------------+--------------+
| VARIABLE_NAME             | SESSION_VALUE | GLOBAL_VALUE |
+---------------------------+---------------+--------------+
| MAX_ERROR_COUNT           | 128           | 64           |
| SKIP_PARALLEL_REPLICATION | ON            | NULL         |
| INNODB_SYNC_SPIN_LOOPS    | NULL          | 30           |
+---------------------------+---------------+--------------+
```

Setting the global values:

```
SET GLOBAL max_error_count=256;

SET GLOBAL skip_parallel_replication=ON;
ERROR 1228 (HY000): Variable 'skip_parallel_replication' is a SESSION variable
  and can't be used with SET GLOBAL

SET GLOBAL innodb_sync_spin_loops=120;

SELECT VARIABLE_NAME, SESSION_VALUE, GLOBAL_VALUE FROM
 INFORMATION_SCHEMA.SYSTEM_VARIABLES WHERE
  VARIABLE_NAME IN ('max_error_count', 'skip_parallel_replication', 'innodb_sync_spin_loops');
+---------------------------+---------------+--------------+
| VARIABLE_NAME             | SESSION_VALUE | GLOBAL_VALUE |
+---------------------------+---------------+--------------+
| MAX_ERROR_COUNT           | 128           | 256          |
| SKIP_PARALLEL_REPLICATION | ON            | NULL         |
| INNODB_SYNC_SPIN_LOOPS    | NULL          | 120          |
+---------------------------+---------------+--------------+
```

SHOW VARIABLES will by default return the session value unless the variable is global only.

```
SHOW VARIABLES LIKE 'max_error_count';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| max_error_count | 128   |
+-----------------+-------+

SHOW VARIABLES LIKE 'skip_parallel_replication';
+---------------------------+-------+
| Variable_name             | Value |
+---------------------------+-------+
| skip_parallel_replication | ON    |
+---------------------------+-------+

SHOW VARIABLES LIKE 'innodb_sync_spin_loops';
+------------------------+-------+
| Variable_name          | Value |
+------------------------+-------+
| innodb_sync_spin_loops | 120   |
+------------------------+-------+
```

Using the inplace syntax:

```
SELECT (@a:=1);
+---------+
| (@a:=1) |
+---------+
|       1 |
+---------+

SELECT @a;
+------+
| @a   |
+------+
|    1 |
+------+
```

1.1.1.2.7.2 SET CHARACTER SET

Syntax

```
SET {CHARACTER SET | CHARSET}
    {charset_name | DEFAULT}
```

Description

Sets the character_set_client and character_set_results session system variables to the specified character set and collation_connection to the value of collation_database, which implicitly sets character_set_connection to the value of character_set_database.

This maps all strings sent between the current client and the server with the given mapping.

Example

```
SHOW VARIABLES LIKE 'character_set\_%';
+--------------------------+--------+
| Variable_name            | Value  |
+--------------------------+--------+
| character_set_client     | utf8   |
| character_set_connection | utf8   |
| character_set_database   | latin1 |
| character_set_filesystem | binary |
| character_set_results    | utf8   |
| character_set_server     | latin1 |
| character_set_system     | utf8   |
+--------------------------+--------+

SHOW VARIABLES LIKE 'collation%';
+----------------------+-------------------+
| Variable_name        | Value             |
+----------------------+-------------------+
| collation_connection | utf8_general_ci   |
| collation_database   | latin1_swedish_ci |
| collation_server     | latin1_swedish_ci |
+----------------------+-------------------+

SET CHARACTER SET utf8mb4;

SHOW VARIABLES LIKE 'character_set\_%';
+--------------------------+---------+
| Variable_name            | Value   |
+--------------------------+---------+
| character_set_client     | utf8mb4 |
| character_set_connection | latin1  |
| character_set_database   | latin1  |
| character_set_filesystem | binary  |
| character_set_results    | utf8mb4 |
| character_set_server     | latin1  |
| character_set_system     | utf8    |
+--------------------------+---------+

SHOW VARIABLES LIKE 'collation%';
+----------------------+-------------------+
| Variable_name        | Value             |
+----------------------+-------------------+
| collation_connection | latin1_swedish_ci |
| collation_database   | latin1_swedish_ci |
| collation_server     | latin1_swedish_ci |
+----------------------+-------------------+
```

1.1.1.2.5.5 SET GLOBAL SQL_SLAVE_SKIP_COUNTER

1.1.1.2.7.4 SET NAMES

Syntax

```
SET NAMES {'charset_name'
    [COLLATE 'collation_name'] | DEFAULT}
```

Contents

Description

Sets the character_set_client, character_set_connection, character_set_results and, implicitly, the collation_connection session system variables to the specified character set and collation.

This determines which character set the client will use to send statements to the server, and the server will use for sending

results back to the client.

`ucs2`, `utf16`, `utf16le` and `utf32` are not valid character sets for `SET NAMES`, as they cannot be used as client character sets.

The collation clause is optional. If not defined (or if `DEFAULT` is specified), the default collation for the character set will be used.

Quotes are optional for the character set or collation clauses.

Examples

```
SELECT VARIABLE_NAME, SESSION_VALUE
  FROM INFORMATION_SCHEMA.SYSTEM_VARIABLES WHERE
  VARIABLE_NAME LIKE 'character_set_c%' OR
  VARIABLE_NAME LIKE 'character_set_re%' OR
  VARIABLE_NAME LIKE 'collation_c%';
+--------------------------+-----------------+
| VARIABLE_NAME            | SESSION_VALUE   |
+--------------------------+-----------------+
| CHARACTER_SET_RESULTS    | utf8            |
| CHARACTER_SET_CONNECTION | utf8            |
| CHARACTER_SET_CLIENT     | utf8            |
| COLLATION_CONNECTION     | utf8_general_ci |
+--------------------------+-----------------+

SET NAMES big5;

SELECT VARIABLE_NAME, SESSION_VALUE
  FROM INFORMATION_SCHEMA.SYSTEM_VARIABLES WHERE
  VARIABLE_NAME LIKE 'character_set_c%' OR
  VARIABLE_NAME LIKE 'character_set_re%' OR
  VARIABLE_NAME LIKE 'collation_c%';
+--------------------------+-----------------+
| VARIABLE_NAME            | SESSION_VALUE   |
+--------------------------+-----------------+
| CHARACTER_SET_RESULTS    | big5            |
| CHARACTER_SET_CONNECTION | big5            |
| CHARACTER_SET_CLIENT     | big5            |
| COLLATION_CONNECTION     | big5_chinese_ci |
+--------------------------+-----------------+

SET NAMES 'latin1' COLLATE 'latin1_bin';

SELECT VARIABLE_NAME, SESSION_VALUE
  FROM INFORMATION_SCHEMA.SYSTEM_VARIABLES WHERE
  VARIABLE_NAME LIKE 'character_set_c%' OR
  VARIABLE_NAME LIKE 'character_set_re%' OR
  VARIABLE_NAME LIKE 'collation_c%';
+--------------------------+---------------+
| VARIABLE_NAME            | SESSION_VALUE |
+--------------------------+---------------+
| CHARACTER_SET_RESULTS    | latin1        |
| CHARACTER_SET_CONNECTION | latin1        |
| CHARACTER_SET_CLIENT     | latin1        |
| COLLATION_CONNECTION     | latin1_bin    |
+--------------------------+---------------+

SET NAMES DEFAULT;

SELECT VARIABLE_NAME, SESSION_VALUE
  FROM INFORMATION_SCHEMA.SYSTEM_VARIABLES WHERE
  VARIABLE_NAME LIKE 'character_set_c%' OR
  VARIABLE_NAME LIKE 'character_set_re%' OR
  VARIABLE_NAME LIKE 'collation_c%';
+--------------------------+-------------------+
| VARIABLE_NAME            | SESSION_VALUE     |
+--------------------------+-------------------+
| CHARACTER_SET_RESULTS    | latin1            |
| CHARACTER_SET_CONNECTION | latin1            |
| CHARACTER_SET_CLIENT     | latin1            |
| COLLATION_CONNECTION     | latin1_swedish_ci |
+--------------------------+-------------------+
```

1.1.1.1.7 SET PASSWORD

1.1.1.1.10 SET ROLE

1.1.1.2.7.7 SET SQL_LOG_BIN

Syntax

```
SET [SESSION] sql_log_bin = {0|1}
```

Description

Sets the sql_log_bin system variable, which disables or enables binary logging for the current connection, if the client has the `SUPER` privilege. The statement is refused with an error if the client does not have that privilege.

Before MariaDB 5.5 and before MySQL 5.6 one could also set `sql_log_bin` as a global variable. This was disabled as this was too dangerous as it could damage replication.

1.1.1.2.7.8 SET STATEMENT

Contents

`SET STATEMENT` can be used to set the value of a system variable for the duration of the statement. It is also possible to set multiple variables.

Syntax

```
SET STATEMENT var1=value1 [, var2=value2, ...]
  FOR <statement>
```

where `varN` is a system variable (list of allowed variables is provided below), and `valueN` is a constant literal.

Description

`SET STATEMENT var1=value1 FOR stmt`

is roughly equivalent to

```
SET @save_value=@@var1;
SET SESSION var1=value1;
stmt;
SET SESSION var1=@save_value;
```

The server parses the whole statement before executing it, so any variables set in this fashion that affect the parser may not have the expected effect. Examples include the charset variables, sql_mode=ansi_quotes, etc.

Examples

One can limit statement execution time `max_statement_time`:

```
SET STATEMENT max_statement_time=1000 FOR SELECT ... ;
```

One can switch on/off individual optimizations:

```
SET STATEMENT optimizer_switch='materialization=off' FOR SELECT ....;
```

It is possible to enable MRR/BKA for a query:

```
SET STATEMENT  join_cache_level=6, optimizer_switch='mrr=on'  FOR SELECT ...
```

Note that it makes no sense to try to set a session variable inside a `SET STATEMENT`:

```
#USELESS STATEMENT
SET STATEMENT sort_buffer_size = 100000 for SET SESSION sort_buffer_size = 200000;
```

For the above, after setting sort_buffer_size to 200000 it will be reset to its original state (the state before the `SET STATEMENT` started) after the statement execution.

Limitations

There are a number of variables that cannot be set on per-query basis. These include:

- `autocommit`
- `character_set_client`
- `character_set_connection`
- `character_set_filesystem`
- `collation_connection`
- `default_master_connection`
- `debug_sync`
- `interactive_timeout`
- `gtid_domain_id`
- `last_insert_id`
- `log_slow_filter`
- `log_slow_rate_limit`
- `log_slow_verbosity`
- `long_query_time`
- `min_examined_row_limit`
- `profiling`
- `profiling_history_size`
- `query_cache_type`
- `rand_seed1`
- `rand_seed2`
- `skip_replication`
- `slow_query_log`
- `sql_log_off`
- `tx_isolation`
- `wait_timeout`

Source

- The feature was originally implemented as a Google Summer of Code 2009 project by Joseph Lukas.
- Percona Server 5.6 included it as Per-query variable statement
- MariaDB ported the patch and fixed *many* bugs. The task in MariaDB Jira is MDEV-5231.

1.1.1.9.4 SET TRANSACTION

1.1.1.2.7.10 SET Variable

Syntax

```
SET var_name = expr [, var_name = expr] ...
```

Contents

Description

The `SET` statement in stored programs is an extended version of the general `SET` statement. Referenced variables may be ones declared inside a stored program, global system variables, or user-defined variables.

The `SET` statement in stored programs is implemented as part of the pre-existing SET syntax. This allows an extended syntax of `SET a=x, b=y, ...` where different variable types (locally declared variables, global and session server variables, user-defined variables) can be mixed. This also allows combinations of local variables and some options that make sense only for system variables; in that case, the options are recognized but ignored.

`SET` can be used with both local variables and user-defined variables.

When setting several variables using the columns returned by a query, `SELECT INTO` should be preferred.

To set many variables to the same value, the `LAST_VALUE( )` function can be used.

Below is an example of how a user-defined variable may be set:

```
SET @x = 1;
```

1.1.1.2.8 SHOW

Articles on the various SHOW commands.

About SHOW

General information about the SHOW statement.

Extended Show

Extended SHOW with WHERE and LIKE.

SHOW AUTHORS

Information about the people who work on MariaDB.

SHOW BINARY LOGS

SHOW BINARY LOGS lists all binary logs on the server.

SHOW BINLOG EVENTS

Show events in the binary log.

SHOW CHARACTER SET

Available character sets.

SHOW CLIENT_STATISTICS

Statistics about client connections.

SHOW COLLATION

Supported collations.

SHOW COLUMNS

Column information.

SHOW CONTRIBUTORS

Companies and people who financially contribute to MariaDB.

SHOW CREATE DATABASE

Shows the CREATE DATABASE statement that created the database.

SHOW CREATE EVENT

Displays the CREATE EVENT statement needed to re-create a given event

SHOW CREATE FUNCTION

Statement that created the function.

SHOW CREATE PACKAGE

Show the CREATE statement that creates the given package specification.

SHOW CREATE PACKAGE BODY

Show the CREATE statement that creates the given package body (i.e. implementation).

SHOW CREATE PROCEDURE

Returns the string used for creating a stored procedure.

SHOW CREATE SEQUENCE

Shows the CREATE SEQUENCE statement that created the sequence.

SHOW CREATE TABLE

Shows the CREATE TABLE statement that created the table.

SHOW CREATE TRIGGER

Shows the CREATE TRIGGER statement used to create the trigger

SHOW CREATE USER

Show the CREATE USER statement for a specified user.

SHOW CREATE VIEW

Show the CREATE VIEW statement that created a view.

SHOW DATABASES

Lists the databases on the server.

SHOW ENGINE

Show storage engine information.

SHOW ENGINE INNODB STATUS

Display extensive InnoDB information.

SHOW ENGINES

Server storage engine info

SHOW ERRORS

Displays errors.

SHOW EVENTS

Shows information about events

SHOW EXPLAIN

Shows an execution plan for a running query.

SHOW FUNCTION CODE

Representation of the internal implementation of the stored function

SHOW FUNCTION STATUS

Stored function characteristics

SHOW GRANTS

View GRANT statements.

SHOW INDEX

Information about table indexes.

SHOW INDEX_STATISTICS

Index usage statistics.

SHOW INNODB STATUS (removed)

Removed synonym for SHOW ENGINE INNODB STATUS

SHOW LOCALES

View locales information.

SHOW MASTER STATUS

Status information about the binary log.

SHOW OPEN TABLES

List non-temporary open tables.

SHOW PACKAGE BODY STATUS

Returns characteristics of stored package bodies (implementations).

SHOW PACKAGE STATUS

Returns characteristics of stored package specifications.

SHOW PLUGINS

Display information about installed plugins.

SHOW PLUGINS SONAME

Information about all available plugins, installed or not.

SHOW PRIVILEGES

Shows the list of supported system privileges.

SHOW PROCEDURE CODE

Display internal implementation of a stored procedure.

SHOW PROCEDURE STATUS

Stored procedure characteristics.

SHOW PROCESSLIST

Running threads and information about them.

SHOW PROFILE

Display statement resource usage

SHOW PROFILES

Show statement resource usage

SHOW QUERY_RESPONSE_TIME

Retrieving information from the QUERY_RESPONSE_TIME plugin.

SHOW RELAYLOG EVENTS

Show events in the relay log.

SHOW SLAVE HOSTS

Display replicas currently registered with the primary.

SHOW SLAVE STATUS

Show status for one or all primaries.

SHOW STATUS

Server status information.

SHOW TABLE STATUS

SHOW TABLES with information about non-temporary tables.

SHOW TABLES

List of non-temporary tables, views or sequences.

SHOW TABLE_STATISTICS

Table usage statistics.

SHOW TRIGGERS

Shows currently-defined triggers

SHOW USER_STATISTICS

User activity statistics.

SHOW VARIABLES

Displays the values of system variables.

SHOW WARNINGS

Displays errors, warnings and notes.

SHOW WSREP_MEMBERSHIP

Galera node cluster membership information.

SHOW WSREP_STATUS

Galera node cluster status information.

There are 1 related questions.

1.1.1.2.8.1 About SHOW

`SHOW` has many forms that provide information about databases, tables, columns, or status information about the server. These include:

- SHOW AUTHORS
- SHOW CHARACTER SET [like_or_where]
- SHOW COLLATION [like_or_where]
- SHOW [FULL] COLUMNS FROM tbl_name [FROM db_name] [like_or_where]
- SHOW CONTRIBUTORS
- SHOW CREATE DATABASE db_name
- SHOW CREATE EVENT event_name
- SHOW CREATE PACKAGE package_name
- SHOW CREATE PACKAGE BODY package_name
- SHOW CREATE PROCEDURE proc_name
- SHOW CREATE TABLE tbl_name
- SHOW CREATE TRIGGER trigger_name
- SHOW CREATE VIEW view_name
- SHOW DATABASES [like_or_where]
- SHOW ENGINE engine_name {STATUS | MUTEX}
- SHOW [STORAGE] ENGINES
- SHOW ERRORS [LIMIT [offset,] row_count]
- SHOW [FULL] EVENTS
- SHOW FUNCTION CODE func_name
- SHOW FUNCTION STATUS [like_or_where]
- SHOW GRANTS FOR user
- SHOW INDEX FROM tbl_name [FROM db_name]
- SHOW INNODB STATUS
- SHOW OPEN TABLES [FROM db_name] [like_or_where]
- SHOW PLUGINS
- SHOW PROCEDURE CODE proc_name
- SHOW PROCEDURE STATUS [like_or_where]
- SHOW PRIVILEGES
- SHOW [FULL] PROCESSLIST
- SHOW PROFILE [types] [FOR QUERY n] [OFFSET n] [LIMIT n]
- SHOW PROFILES
- SHOW [GLOBAL | SESSION] STATUS [like_or_where]
- SHOW TABLE STATUS [FROM db_name] [like_or_where]
- SHOW TABLES [FROM db_name] [like_or_where]
- SHOW TRIGGERS [FROM db_name] [like_or_where]

- SHOW [GLOBAL | SESSION] VARIABLES [like_or_where]
- SHOW WARNINGS [LIMIT [offset,] row_count]

```
like_or_where:
    LIKE 'pattern'
  | WHERE expr
```

If the syntax for a given `SHOW` statement includes a `LIKE 'pattern'` part, `'pattern'` is a string that can contain the SQL " `%` " and " _ " wildcard characters. The pattern is useful for restricting statement output to matching values.

Several `SHOW` statements also accept a `WHERE` clause that provides more flexibility in specifying which rows to display. See Extended Show.

1.1.1.2.8.2 Extended Show

Contents

The following `SHOW` statements can be extended by using a `WHERE` clause and a `LIKE` clause to refine the results:

- SHOW CHARACTER SET
- SHOW COLLATION
- SHOW COLUMNS
- SHOW DATABASES
- SHOW FUNCTION STATUS
- SHOW INDEX
- SHOW OPEN TABLES
- SHOW PACKAGE STATUS
- SHOW PACKAGE BODY STATUS
- SHOW INDEX
- SHOW PROCEDURE STATUS
- SHOW STATUS
- SHOW TABLE STATUS
- SHOW TABLES
- SHOW TRIGGERS
- SHOW VARIABLES

As with a regular `SELECT`, the `WHERE` clause can be used for the specific columns returned, and the `LIKE` clause with the regular wildcards.

Examples

```
SHOW TABLES;
+----------------------+
| Tables_in_test       |
+----------------------+
| animal_count         |
| animals              |
| are_the_mooses_loose |
| aria_test2           |
| t1                   |
| view1                |
+----------------------+
```

Showing the tables beginning with *a* only.

```
SHOW TABLES WHERE Tables_in_test LIKE 'a%';
+----------------------+
| Tables_in_test       |
+----------------------+
| animal_count         |
| animals              |
| are_the_mooses_loose |
| aria_test2           |
+----------------------+
```

Variables whose name starts with *aria* and with a valued of greater than 8192:

```
SHOW VARIABLES WHERE Variable_name LIKE 'aria%' AND Value >8192;
+-----------------------------+---------------------+
| Variable_name               | Value               |
+-----------------------------+---------------------+
| aria_checkpoint_log_activity | 1048576             |
| aria_log_file_size          | 1073741824          |
| aria_max_sort_file_size     | 9223372036853727232 |
| aria_pagecache_buffer_size  | 134217728           |
| aria_sort_buffer_size       | 134217728           |
+-----------------------------+---------------------+
```

Shortcut, just returning variables whose name begins with *aria*.

```
SHOW VARIABLES LIKE 'aria%';
+-----------------------------------------+---------------------+
| Variable_name                           | Value               |
+-----------------------------------------+---------------------+
| aria_block_size                         | 8192                |
| aria_checkpoint_interval                | 30                  |
| aria_checkpoint_log_activity            | 1048576             |
| aria_force_start_after_recovery_failures | 0                   |
| aria_group_commit                       | none                |
| aria_group_commit_interval              | 0                   |
| aria_log_file_size                      | 1073741824          |
| aria_log_purge_type                     | immediate           |
| aria_max_sort_file_size                 | 9223372036853727232 |
| aria_page_checksum                      | ON                  |
| aria_pagecache_age_threshold            | 300                 |
| aria_pagecache_buffer_size              | 134217728           |
| aria_pagecache_division_limit           | 100                 |
| aria_recover                            | NORMAL              |
| aria_repair_threads                     | 1                   |
| aria_sort_buffer_size                   | 134217728           |
| aria_stats_method                       | nulls_unequal       |
| aria_sync_log_dir                       | NEWFILE             |
| aria_used_for_temp_tables               | ON                  |
+-----------------------------------------+---------------------+
```

1.1.1.2.8.3 SHOW AUTHORS

Syntax

```
SHOW AUTHORS
```

Description

The `SHOW AUTHORS` statement displays information about the people who work on MariaDB. For each author, it displays Name, Location, and Comment values. All columns are encoded as latin1.

These include:

- First the active people in MariaDB are listed.
- Then the active people in MySQL.
- Last the people that have contributed to MariaDB/MySQL in the past.

The order is somewhat related to importance of the contribution given to the MariaDB project, but this is not 100% accurate. There is still room for improvement and debate...

Example

```
SHOW AUTHORS\G
*************************** 1. row ***************************
    Name: Michael (Monty) Widenius
Location: Tusby, Finland
```

```
 Comment: Lead developer and main author
*************************** 2. row ***************************
    Name: Sergei Golubchik
Location: Kerpen, Germany
 Comment: Architect, Full-text search, precision math, plugin framework, merges etc
*************************** 3. row ***************************
    Name: Igor Babaev
Location: Bellevue, USA
 Comment: Optimizer, keycache, core work
*************************** 4. row ***************************
    Name: Sergey Petrunia
Location: St. Petersburg, Russia
 Comment: Optimizer
*************************** 5. row ***************************
    Name: Oleksandr Byelkin
Location: Lugansk, Ukraine
 Comment: Query Cache (4.0), Subqueries (4.1), Views (5.0)
*************************** 6. row ***************************
    Name: Timour Katchaounov
Location: Sofia , Bulgaria
 Comment: Optimizer
*************************** 7. row ***************************
    Name: Kristian Nielsen
Location: Copenhagen, Denmark
 Comment: Replication, Async client prototocol, General buildbot stuff
*************************** 8. row ***************************
    Name: Alexander (Bar) Barkov
Location: Izhevsk, Russia
 Comment: Unicode and character sets
*************************** 9. row ***************************
    Name: Alexey Botchkov (Holyfoot)
Location: Izhevsk, Russia
 Comment: GIS extensions, embedded server, precision math
*************************** 10. row ***************************
    Name: Daniel Bartholomew
Location: Raleigh, USA
 Comment: MariaDB documentation, Buildbot, releases
*************************** 11. row ***************************
    Name: Colin Charles
Location: Selangor, Malesia
 Comment: MariaDB documentation, talks at a LOT of conferences
*************************** 12. row ***************************
    Name: Sergey Vojtovich
Location: Izhevsk, Russia
 Comment: initial implementation of plugin architecture, maintained native storage engines
(MyISAM, MEMORY, ARCHIVE, etc), rewrite of table cache
*************************** 13. row ***************************
    Name: Vladislav Vaintroub
Location: Mannheim, Germany
 Comment: MariaDB Java connector, new thread pool, Windows optimizations
*************************** 14. row ***************************
    Name: Elena Stepanova
Location: Sankt Petersburg, Russia
 Comment: QA, test cases
*************************** 15. row ***************************
    Name: Georg Richter
Location: Heidelberg, Germany
 Comment: New LGPL C connector, PHP connector
*************************** 16. row ***************************
    Name: Jan Lindström
Location: Ylämylly, Finland
 Comment: Working on InnoDB
*************************** 17. row ***************************
    Name: Lixun Peng
Location: Hangzhou, China
 Comment: Multi Source replication
*************************** 18. row ***************************
    Name: Olivier Bertrand
Location: Paris, France
 Comment: CONNECT storage engine
*************************** 19. row ***************************
    Name: Kentoku Shiba
Location: Tokyo, Japan
 Comment: Spider storage engine, metadata_lock_info Information schema
*************************** 20. row ***************************
```

```
    Name: Percona
Location: CA, USA
 Comment: XtraDB, microslow patches, extensions to slow log
*************************** 21. row ***************************
    Name: Vicentiu Ciorbaru
Location: Bucharest, Romania
 Comment: Roles
*************************** 22. row ***************************
    Name: Sudheera Palihakkara
Location:
 Comment: PCRE Regular Expressions
*************************** 23. row ***************************
    Name: Pavel Ivanov
Location: USA
 Comment: Some patches and bug fixes
*************************** 24. row ***************************
    Name: Konstantin Osipov
Location: Moscow, Russia
 Comment: Prepared statements (4.1), Cursors (5.0), GET_LOCK (10.0)
*************************** 25. row ***************************
    Name: Ian Gilfillan
Location: South Africa
 Comment: MariaDB documentation
*************************** 26. row ***************************
    Name: Federico Razolli
Location: Italy
 Comment: MariaDB documentation Italian translation
*************************** 27. row ***************************
    Name: Guilhem Bichot
Location: Bordeaux, France
 Comment: Replication (since 4.0)
*************************** 28. row ***************************
    Name: Andrei Elkin
Location: Espoo, Finland
 Comment: Replication
*************************** 29. row ***************************
    Name: Dmitri Lenev
Location: Moscow, Russia
 Comment: Time zones support (4.1), Triggers (5.0)
*************************** 30. row ***************************
    Name: Marc Alff
Location: Denver, CO, USA
 Comment: Signal, Resignal, Performance schema
*************************** 31. row ***************************
    Name: Mikael Ronström
Location: Stockholm, Sweden
 Comment: NDB Cluster, Partitioning, online alter table
*************************** 32. row ***************************
    Name: Ingo Strüwing
Location: Berlin, Germany
 Comment: Bug fixing in MyISAM, Merge tables etc
*************************** 33. row ***************************
    Name: Marko Mäkelä
Location: Helsinki, Finland
 Comment: InnoDB core developer
...
```

1.1.1.2.8.4 SHOW BINARY LOGS

Syntax

```
SHOW BINARY LOGS
SHOW MASTER LOGS
```

Description

Lists the binary log files on the server. This statement is used as part of the procedure described in `PURGE BINARY LOGS`

, that shows how to determine which logs can be purged.

This statement requires the SUPER privilege, the REPLICATION_CLIENT privilege, or, from MariaDB 10.5.2, the BINLOG MONITOR privilege.

Examples

```
SHOW BINARY LOGS;
+--------------------+-----------+
| Log_name           | File_size |
+--------------------+-----------+
| mariadb-bin.000001 |     19039 |
| mariadb-bin.000002 |    717389 |
| mariadb-bin.000003 |       300 |
| mariadb-bin.000004 |       333 |
| mariadb-bin.000005 |       899 |
| mariadb-bin.000006 |       125 |
| mariadb-bin.000007 |     18907 |
| mariadb-bin.000008 |     19530 |
| mariadb-bin.000009 |       151 |
| mariadb-bin.000010 |       151 |
| mariadb-bin.000011 |       125 |
| mariadb-bin.000012 |       151 |
| mariadb-bin.000013 |       151 |
| mariadb-bin.000014 |       125 |
| mariadb-bin.000015 |       151 |
| mariadb-bin.000016 |       314 |
+--------------------+-----------+
```

1.1.1.2.8.5 SHOW BINLOG EVENTS

Syntax

```
SHOW BINLOG EVENTS
   [IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]
```

Description

Shows the events in the binary log. If you do not specify ' `log_name` ', the first binary log is displayed.

Requires the BINLOG MONITOR privilege (>= MariaDB 10.5.2) or the REPLICATION SLAVE privilege (<= MariaDB 10.5.1).

Example

```
SHOW BINLOG EVENTS IN 'mysql_sandbox10019-bin.000002';
+------------------------------+-----+------------------+-----------+-------------+--------
---------------------------------------+
| Log_name                     | Pos | Event_type       | Server_id | End_log_pos | Info
|
+------------------------------+-----+------------------+-----------+-------------+--------
---------------------------------------+
| mysql_sandbox10019-bin.000002 |   4 | Format_desc      |         1 |         248 | Server
ver: 10.0.19-MariaDB-log, Binlog ver: 4 |
| mysql_sandbox10019-bin.000002 | 248 | Gtid_list        |         1 |         273 | []
|
| mysql_sandbox10019-bin.000002 | 273 | Binlog_checkpoint |         1 |         325 |
mysql_sandbox10019-bin.000002                 |
| mysql_sandbox10019-bin.000002 | 325 | Gtid             |         1 |         363 | GTID 0-
1-1                                |
| mysql_sandbox10019-bin.000002 | 363 | Query            |         1 |         446 | CREATE
DATABASE blog                      |
| mysql_sandbox10019-bin.000002 | 446 | Gtid             |         1 |         484 | GTID 0-
1-2                                |
| mysql_sandbox10019-bin.000002 | 484 | Query            |         1 |         571 | use
`blog`; CREATE TABLE bb (id INT)          |
+------------------------------+-----+------------------+-----------+-------------+--------
---------------------------------------+
```

1.1.1.2.8.6 SHOW CHARACTER SET

Syntax

```
SHOW CHARACTER SET
    [LIKE 'pattern' | WHERE expr]
```

Contents

Description

The `SHOW CHARACTER SET` statement shows all available character sets. The `LIKE` clause, if present on its own, indicates which character set names to match. The `WHERE` and `LIKE` clauses can be given to select rows using more general conditions, as discussed in Extended SHOW.

The same information can be queried from the Information Schema CHARACTER_SETS table.

See Setting Character Sets and Collations for details on specifying the character set at the server, database, table and column levels.

Examples

```
SHOW CHARACTER SET LIKE 'latin%';
+---------+-----------------------------+-------------------+--------+
| Charset | Description                 | Default collation | Maxlen |
+---------+-----------------------------+-------------------+--------+
| latin1  | cp1252 West European        | latin1_swedish_ci |      1 |
| latin2  | ISO 8859-2 Central European | latin2_general_ci |      1 |
| latin5  | ISO 8859-9 Turkish          | latin5_turkish_ci |      1 |
| latin7  | ISO 8859-13 Baltic          | latin7_general_ci |      1 |
+---------+-----------------------------+-------------------+--------+
```

```
SHOW CHARACTER SET WHERE Maxlen LIKE '2';
+---------+----------------------------+-------------------+--------+
| Charset | Description                | Default collation | Maxlen |
+---------+----------------------------+-------------------+--------+
| big5    | Big5 Traditional Chinese   | big5_chinese_ci   |      2 |
| sjis    | Shift-JIS Japanese         | sjis_japanese_ci  |      2 |
| euckr   | EUC-KR Korean              | euckr_korean_ci   |      2 |
| gb2312  | GB2312 Simplified Chinese  | gb2312_chinese_ci |      2 |
| gbk     | GBK Simplified Chinese     | gbk_chinese_ci    |      2 |
| ucs2    | UCS-2 Unicode              | ucs2_general_ci   |      2 |
| cp932   | SJIS for Windows Japanese  | cp932_japanese_ci |      2 |
+---------+----------------------------+-------------------+--------+
```

1.1.1.2.8.7 SHOW CLIENT_STATISTICS

Syntax

```
SHOW CLIENT_STATISTICS
```

Description

The `SHOW CLIENT_STATISTICS` statement is part of the User Statistics feature. It was removed as a separate statement in MariaDB 10.1.1, but effectively replaced by the generic SHOW information_schema_table statement. The information_schema.CLIENT_STATISTICS table holds statistics about client connections.

The userstat system variable must be set to 1 to activate this feature. See the User Statistics and information_schema.CLIENT_STATISTICS articles for more information.

Example

```
SHOW CLIENT_STATISTICS\G
*************************** 1. row ***************************
               Client: localhost
     Total_connections: 35
Concurrent_connections: 0
        Connected_time: 708
             Busy_time: 2.5557979999999985
              Cpu_time: 0.04123740000000002
        Bytes_received: 3883
            Bytes_sent: 21595
  Binlog_bytes_written: 0
             Rows_read: 18
             Rows_sent: 115
          Rows_deleted: 0
         Rows_inserted: 0
          Rows_updated: 0
       Select_commands: 70
       Update_commands: 0
        Other_commands: 0
   Commit_transactions: 1
 Rollback_transactions: 0
    Denied_connections: 0
      Lost_connections: 0
         Access_denied: 0
         Empty_queries: 35
```

5.2.6 SHOW COLLATION

1.1.1.2.1.16 SHOW COLUMNS

1.1.1.2.8.10 SHOW CONTRIBUTORS

Syntax

```
SHOW CONTRIBUTORS
```

Description

The `SHOW CONTRIBUTORS` statement displays information about the companies and people who financially contribute to MariaDB. For each contributor, it displays `Name`, `Location`, and `Comment` values. All columns are encoded as `latin1`.

It displays all members and sponsors of the MariaDB Foundation as well as other financial contributors.

Example

```
SHOW CONTRIBUTORS;
+---------------------+------------------------------+--------------------------------------
-----------------------+
| Name                | Location                     | Comment
|
+---------------------+------------------------------+--------------------------------------
-----------------------+
| Alibaba Cloud       | https://www.alibabacloud.com/ | Platinum Sponsor of the MariaDB
Foundation                  |
| Tencent Cloud       | https://cloud.tencent.com     | Platinum Sponsor of the MariaDB
Foundation                  |
| Microsoft           | https://microsoft.com/        | Platinum Sponsor of the MariaDB
Foundation                  |
| MariaDB Corporation | https://mariadb.com           | Founding member, Platinum Sponsor of
the MariaDB Foundation |
| ServiceNow          | https://servicenow.com        | Platinum Sponsor of the MariaDB
Foundation                  |
| Intel               | https://www.intel.com         | Platinum Sponsor of the MariaDB
Foundation                  |
| SIT                 | https://sit.org               | Platinum Sponsor of the MariaDB
Foundation                  |
| Visma               | https://visma.com             | Gold Sponsor of the MariaDB Foundation
|
| DBS                 | https://dbs.com               | Gold Sponsor of the MariaDB Foundation
|
| IBM                 | https://www.ibm.com           | Gold Sponsor of the MariaDB Foundation
|
| Automattic          | https://automattic.com        | Silver Sponsor of the MariaDB
Foundation                   |
| Percona             | https://www.percona.com/      | Sponsor of the MariaDB Foundation
|
| Galera Cluster      | https://galeracluster.com     | Sponsor of the MariaDB Foundation
|
| Google              | USA                           | Sponsoring encryption, parallel
replication and GTID        |
| Facebook            | USA                           | Sponsoring non-blocking API, LIMIT
ROWS EXAMINED etc         |
| Ronald Bradford     | Brisbane, Australia           | EFF contribution for UC2006 Auction
|
| Sheeri Kritzer      | Boston, Mass. USA             | EFF contribution for UC2006 Auction
|
| Mark Shuttleworth   | London, UK.                   | EFF contribution for UC2006 Auction
|
+---------------------+------------------------------+--------------------------------------
-----------------------+
```

1.1.1.2.8.11 SHOW CREATE DATABASE

Syntax

```
SHOW CREATE {DATABASE | SCHEMA} db_name
```

Contents

Description

Shows the CREATE DATABASE statement that creates the given database. `SHOW CREATE SCHEMA` is a synonym for `SHOW CREATE DATABASE`. `SHOW CREATE DATABASE` quotes database names according to the value of the sql_quote_show_create server system variable.

Examples

```
SHOW CREATE DATABASE test;
+----------+-----------------------------------------------------------------+
| Database | Create Database                                                 |
+----------+-----------------------------------------------------------------+
| test     | CREATE DATABASE `test` /*!40100 DEFAULT CHARACTER SET latin1 */ |
+----------+-----------------------------------------------------------------+

SHOW CREATE SCHEMA test;
+----------+-----------------------------------------------------------------+
| Database | Create Database                                                 |
+----------+-----------------------------------------------------------------+
| test     | CREATE DATABASE `test` /*!40100 DEFAULT CHARACTER SET latin1 */ |
+----------+-----------------------------------------------------------------+
```

With `sql_quote_show_create` off:

```
SHOW CREATE DATABASE test;
+----------+-----------------------------------------------------------------+
| Database | Create Database                                                 |
+----------+-----------------------------------------------------------------+
| test     | CREATE DATABASE test /*!40100 DEFAULT CHARACTER SET latin1 */ |
+----------+-----------------------------------------------------------------+
```

With a comment, from MariaDB 10.5:

```
SHOW CREATE DATABASE p;
+----------+-----------------------------------------------------------------------------------------------
| Database | Create Database
+----------+-----------------------------------------------------------------------------------------------
| p        | CREATE DATABASE `p` /*!40100 DEFAULT CHARACTER SET latin1 */ COMMENT 'presentations'
+----------+-----------------------------------------------------------------------------------------------
```

1.1.1.2.8.12 SHOW CREATE EVENT

Syntax

```
SHOW CREATE EVENT event_name
```

Description

This statement displays the `CREATE EVENT` statement needed to re-create a given event, as well as the `SQL_MODE` that was used when the trigger has been created and the character set used by the connection. To find out which events are present, use `SHOW EVENTS`.

The output of this statement is unreliably affected by the `sql_quote_show_create` server system variable - see

http://bugs.mysql.com/bug.php?id=12719

The `information_schema.EVENTS` table provides similar, but more complete, information.

Examples

```
SHOW CREATE EVENT test.e_daily\G
*************************** 1. row ***************************
               Event: e_daily
            sql_mode:
           time_zone: SYSTEM
       Create Event: CREATE EVENT `e_daily`
                       ON SCHEDULE EVERY 1 DAY
                       STARTS CURRENT_TIMESTAMP + INTERVAL 6 HOUR
                       ON COMPLETION NOT PRESERVE
                       ENABLE
                       COMMENT 'Saves total number of sessions then
                               clears the table each day'
                       DO BEGIN
                         INSERT INTO site_activity.totals (time, total)
                           SELECT CURRENT_TIMESTAMP, COUNT(*)
                           FROM site_activity.sessions;
                         DELETE FROM site_activity.sessions;
                       END
character_set_client: latin1
collation_connection: latin1_swedish_ci
  Database Collation: latin1_swedish_ci
```

1.1.1.2.8.13 SHOW CREATE FUNCTION

Syntax

```
SHOW CREATE FUNCTION func_name
```

Description

This statement is similar to SHOW CREATE PROCEDURE but for stored functions.

The output of this statement is unreliably affected by the sql_quote_show_create server system variable - see http://bugs.mysql.com/bug.php?id=12719

Example

```
SHOW CREATE FUNCTION VatCents\G
*************************** 1. row ***************************
            Function: VatCents
            sql_mode:
     Create Function: CREATE DEFINER=`root`@`localhost` FUNCTION `VatCents`(price
DECIMAL(10,2)) RETURNS int(11)
    DETERMINISTIC
BEGIN
 DECLARE x INT;
 SET x = price * 114;
 RETURN x;
END
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
```

1.1.1.2.8.14 SHOW CREATE PACKAGE

MariaDB starting with 10.3.5
Oracle-style packages were introduced in MariaDB 10.3.5.

Syntax

```
SHOW CREATE PACKAGE  [ db_name . ] package_name
```

Contents

Description

The `SHOW CREATE PACKAGE` statement can be used when Oracle SQL_MODE is set.

Shows the `CREATE` statement that creates the given package specification.

Examples

```
SHOW CREATE PACKAGE employee_tools\G
*************************** 1. row ***************************
             Package: employee_tools
            sql_mode:
PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE,ORACLE,NO_KEY_OPTIONS,NO_TABLE_OPTIONS,NO_FIELD_OPTION
S,NO_AUTO_CREATE_USER
      Create Package: CREATE DEFINER="root"@"localhost" PACKAGE "employee_tools" AS
  FUNCTION getSalary(eid INT) RETURN DECIMAL(10,2);
  PROCEDURE raiseSalary(eid INT, amount DECIMAL(10,2));
  PROCEDURE raiseSalaryStd(eid INT);
  PROCEDURE hire(ename TEXT, esalary DECIMAL(10,2));
END
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
```

1.1.1.2.8.15 SHOW CREATE PACKAGE BODY

MariaDB starting with 10.3.5
Oracle-style packages were introduced in MariaDB 10.3.5.

Syntax

```
SHOW CREATE PACKAGE BODY  [ db_name . ] package_name
```

Contents

Description

The `SHOW CREATE PACKAGE BODY` statement can be used when Oracle SQL_MODE is set.

Shows the `CREATE` statement that creates the given package body (i.e. the implementation).

Examples

```
SHOW CREATE PACKAGE BODY employee_tools\G
*************************** 1. row ***************************
        Package body: employee_tools
            sql_mode:
PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE,ORACLE,NO_KEY_OPTIONS,NO_TABLE_OPTIONS,NO_FIELD_OPTION
S,NO_AUTO_CREATE_USER
 Create Package Body: CREATE DEFINER="root"@"localhost" PACKAGE BODY "employee_tools" AS

  stdRaiseAmount DECIMAL(10,2):=500;

  PROCEDURE log (eid INT, ecmnt TEXT) AS
  BEGIN
    INSERT INTO employee_log (id, cmnt) VALUES (eid, ecmnt);
  END;

  PROCEDURE hire(ename TEXT, esalary DECIMAL(10,2)) AS
    eid INT;
  BEGIN
    INSERT INTO employee (name, salary) VALUES (ename, esalary);
    eid:= last_insert_id();
    log(eid, 'hire ' || ename);
  END;

  FUNCTION getSalary(eid INT) RETURN DECIMAL(10,2) AS
    nSalary DECIMAL(10,2);
  BEGIN
    SELECT salary INTO nSalary FROM employee WHERE id=eid;
    log(eid, 'getSalary id=' || eid || ' salary=' || nSalary);
    RETURN nSalary;
  END;

  PROCEDURE raiseSalary(eid INT, amount DECIMAL(10,2)) AS
  BEGIN
    UPDATE employee SET salary=salary+amount WHERE id=eid;
    log(eid, 'raiseSalary id=' || eid || ' amount=' || amount);
  END;

  PROCEDURE raiseSalaryStd(eid INT) AS
  BEGIN
    raiseSalary(eid, stdRaiseAmount);
    log(eid, 'raiseSalaryStd id=' || eid);
  END;

BEGIN
  log(0, 'Session ' || connection_id() || ' ' || current_user || ' started');
END
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
```

1.1.1.2.8.16 SHOW CREATE PROCEDURE

Syntax

```
SHOW CREATE PROCEDURE proc_name
```

Contents

Description

This statement is a MariaDB extension. It returns the exact string that can be used to re-create the named stored procedure, as well as the `SQL_MODE` that was used when the trigger has been created and the character set used by the connection.. A similar statement, `SHOW CREATE FUNCTION`, displays information about stored functions.

Both statements require that you are the owner of the routine or have the `SELECT` privilege on the `mysql.proc` table. When neither is true, the statements display `NULL` for the `Create Procedure` or `Create Function` field.

Warning Users with `SELECT` privileges on `mysql.proc` or `USAGE` privileges on `*.*` can view the text of routines, even when they do not have privileges for the function or procedure itself.

The output of these statements is unreliably affected by the `sql_quote_show_create` server system variable - see http://bugs.mysql.com/bug.php?id=12719

Examples

Here's a comparison of the `SHOW CREATE PROCEDURE` and `SHOW CREATE FUNCTION` statements.

```
SHOW CREATE PROCEDURE test.simpleproc\G
*************************** 1. row ***************************
           Procedure: simpleproc
            sql_mode:
    Create Procedure: CREATE PROCEDURE `simpleproc`(OUT param1 INT)
                      BEGIN
                      SELECT COUNT(*) INTO param1 FROM t;
                      END
character_set_client: latin1
collation_connection: latin1_swedish_ci
  Database Collation: latin1_swedish_ci

SHOW CREATE FUNCTION test.hello\G
*************************** 1. row ***************************
            Function: hello
            sql_mode:
     Create Function: CREATE FUNCTION `hello`(s CHAR(20))
                      RETURNS CHAR(50)
                      RETURN CONCAT('Hello, ',s,'!')
character_set_client: latin1
collation_connection: latin1_swedish_ci
  Database Collation: latin1_swedish_ci
```

When the user issuing the statement does not have privileges on the routine, attempting to `CALL` the procedure raises Error 1370.

```
CALL test.prc1();
Error 1370 (42000): execute command denied to
  user 'test_user'@'localhost' for routine 'test'.'prc1'
```

If the user neither has privilege to the routine nor the SELECT privilege on mysql.proc table, it raises Error 1305, informing them that the procedure does not exist.

```
SHOW CREATE TABLES test.prc1\G
Error 1305 (42000): PROCEDURE prc1 does not exist
```

1.1.1.2.8.17 SHOW CREATE SEQUENCE

Syntax

```
SHOW CREATE SEQUENCE sequence_name;
```

Contents

Description

Shows the CREATE SEQUENCE statement that created the given sequence. The statement requires the `SELECT` privilege for the table.

Example

```
CREATE SEQUENCE s1 START WITH 50;
SHOW CREATE SEQUENCE s1\G;
*************************** 1. row ***************************
       Table: s1
Create Table: CREATE SEQUENCE `s1` start with 50 minvalue 1 maxvalue 9223372036854775806
  increment by 1 cache 1000 nocycle ENGINE=InnoDB
```

Notes

If you want to see the underlying table structure used for the `SEQUENCE` you can use SHOW CREATE TABLE on the `SEQUENCE`. You can also use `SELECT` to read the current recorded state of the `SEQUENCE`:

```
SHOW CREATE TABLE s1\G
*************************** 1. row ***************************
       Table: s1
Create Table: CREATE TABLE `s1` (
  `next_not_cached_value` bigint(21) NOT NULL,
  `minimum_value` bigint(21) NOT NULL,
  `maximum_value` bigint(21) NOT NULL,
  `start_value` bigint(21) NOT NULL COMMENT 'start value when sequences is created
     or value if RESTART is used',
  `increment` bigint(21) NOT NULL COMMENT 'increment value',
  `cache_size` bigint(21) unsigned NOT NULL,
  `cycle_option` tinyint(1) unsigned NOT NULL COMMENT '0 if no cycles are allowed,
     1 if the sequence should begin a new cycle when maximum_value is passed',
  `cycle_count` bigint(21) NOT NULL COMMENT 'How many cycles have been done'
) ENGINE=InnoDB SEQUENCE=1

SELECT * FROM s1\G
*************************** 1. row ***************************
next_not_cached_value: 50
        minimum_value: 1
        maximum_value: 9223372036854775806
          start_value: 50
            increment: 1
           cache_size: 1000
         cycle_option: 0
          cycle_count: 0
```

1.1.1.2.1.17 SHOW CREATE TABLE

1.1.1.2.8.19 SHOW CREATE TRIGGER

Syntax

```
SHOW CREATE TRIGGER trigger_name
```

Contents

Description

This statement shows a `CREATE TRIGGER` statement that creates the given trigger, as well as the `SQL_MODE` that was used when the trigger has been created and the character set used by the connection.

The `TRIGGER` privilege is required on the table the trigger is defined for to execute this statement.

The output of this statement is unreliably affected by the `sql_quote_show_create` server system variable - see http://bugs.mysql.com/bug.php?id=12719

Examples

```
SHOW CREATE TRIGGER example\G
*************************** 1. row ***************************
               Trigger: example
              sql_mode: ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,STRICT_ALL_TABLES
,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,NO_
ENGINE_SUBSTITUTION
SQL Original Statement: CREATE DEFINER=`root`@`localhost` TRIGGER example BEFORE
 INSERT ON t FOR EACH ROW
BEGIN
        SET NEW.c = NEW.c * 2;
END
  character_set_client: cp850
  collation_connection: cp850_general_ci
  Database Collation: utf8_general_ci
  Created: 2016-09-29 13:53:34.35
```

MariaDB starting with 10.2.3

The `Created` column was added in MySQL 5.7 and MariaDB 10.2.3 as part of introducing multiple trigger events per action.

1.1.1.1.13 SHOW CREATE USER

1.1.1.2.8.21 SHOW CREATE VIEW

Syntax

```
SHOW CREATE VIEW view_name
```

Description

This statement shows a CREATE VIEW statement that creates the given view, as well as the character set used by the connection when the view was created. This statement also works with views.

`SHOW CREATE VIEW` quotes table, column and stored function names according to the value of the sql_quote_show_create server system variable.

Examples

```
SHOW CREATE VIEW example\G
*************************** 1. row ***************************
                View: example
         Create View: CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL
SECURITY DEFINER VIEW `example` AS (select `t`.`id` AS `id`,`t`.`s` AS `s` from
`t`)
character_set_client: cp850
collation_connection: cp850_general_ci
```

With `sql_quote_show_create` off:

```
SHOW CREATE VIEW example\G
*************************** 1. row ***************************
                View: example
         Create View: CREATE ALGORITHM=UNDEFINED DEFINER=root@localhost SQL SECU
RITY DEFINER VIEW example AS (select t.id AS id,t.s AS s from t)
character_set_client: cp850
collation_connection: cp850_general_ci
```

Grants

To be able to see a view, you need to have the SHOW VIEW and the SELECT privilege on the view:

```
GRANT SHOW VIEW,SELECT ON test_database.test_view TO 'test'@'localhost';
```

1.1.1.2.8.22 SHOW DATABASES

Syntax

```
SHOW {DATABASES | SCHEMAS}
    [LIKE 'pattern' | WHERE expr]
```

Contents

Description

`SHOW DATABASES` lists the databases on the MariaDB server host. `SHOW SCHEMAS` is a synonym for `SHOW DATABASES`. The `LIKE` clause, if present on its own, indicates which database names to match. The `WHERE` and `LIKE` clauses can be given to select rows using more general conditions, as discussed in Extended SHOW.

You see only those databases for which you have some kind of privilege, unless you have the global SHOW DATABASES privilege. You can also get this list using the mariadb-show command.

If the server was started with the `--skip-show-database` option, you cannot use this statement at all unless you have the SHOW DATABASES privilege.

The list of results returned by `SHOW DATABASES` is based on directories in the data directory, which is how MariaDB implements databases. It's possible that output includes directories that do not correspond to actual databases.

The Information Schema SCHEMATA table also contains database information.

Examples

```
SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| test               |
+--------------------+
```

```
SHOW DATABASES LIKE 'm%';
+---------------+
| Database (m%) |
+---------------+
| mysql         |
+---------------+
```

1.1.1.2.8.23 SHOW ENGINE

Contents

Syntax

```
SHOW ENGINE engine_name {STATUS | MUTEX}
```

Description

`SHOW ENGINE` displays operational information about a storage engine. The following statements currently are supported:

```
SHOW ENGINE INNODB STATUS
SHOW ENGINE INNODB MUTEX
SHOW ENGINE PERFORMANCE_SCHEMA STATUS
SHOW ENGINE ROCKSDB STATUS
```

If the Sphinx Storage Engine is installed, the following is also supported:

```
SHOW ENGINE SPHINX STATUS
```

See `SHOW ENGINE SPHINX STATUS`.

Older (and now removed) synonyms were `SHOW INNODB STATUS` for `SHOW ENGINE INNODB STATUS` and `SHOW MUTEX STATUS` for `SHOW ENGINE INNODB MUTEX`.

SHOW ENGINE INNODB STATUS

`SHOW ENGINE INNODB STATUS` displays extensive information from the standard InnoDB Monitor about the state of the InnoDB storage engine. See `SHOW ENGINE INNODB STATUS` for more.

SHOW ENGINE INNODB MUTEX

`SHOW ENGINE INNODB MUTEX` displays InnoDB mutex statistics.

The statement displays the following output fields:

- **Type:** Always InnoDB.
- **Name:** The source file where the mutex is implemented, and the line number in the file where the mutex is created. The line number is dependent on the MariaDB version.
- **Status:** This field displays the following values if `UNIV_DEBUG` was defined at compilation time (for example, in include/univ.h in the InnoDB part of the source tree). Only the `os_waits` value is displayed if `UNIV_DEBUG` was not defined. Without `UNIV_DEBUG`, the information on which the output is based is insufficient to distinguish regular mutexes and mutexes that protect rw-locks (which allow multiple readers or a single writer). Consequently, the output may appear to contain multiple rows for the same mutex.
  - **count** indicates how many times the mutex was requested.
  - **spin_waits** indicates how many times the spinlock had to run.
  - **spin_rounds** indicates the number of spinlock rounds. (spin_rounds divided by spin_waits provides the average round count.)
  - **os_waits** indicates the number of operating system waits. This occurs when the spinlock did not work (the mutex was not locked during the spinlock and it was necessary to yield to the operating system and wait).
  - **os_yields** indicates the number of times a the thread trying to lock a mutex gave up its timeslice and yielded to the operating system (on the presumption that allowing other threads to run will free the mutex so that it can be locked).
  - **os_wait_times** indicates the amount of time (in ms) spent in operating system waits, if the timed_mutexes system variable is 1 (ON). If timed_mutexes is 0 (OFF), timing is disabled, so os_wait_times is 0. timed_mutexes is off by default.

Information from this statement can be used to diagnose system problems. For example, large values of spin_waits and spin_rounds may indicate scalability problems.

The `information_schema.INNODB_MUTEXES` table provides similar information.

SHOW ENGINE PERFORMANCE_SCHEMA STATUS

This statement shows how much memory is used for `performance_schema` tables and internal buffers.

The output contains the following fields:

- **Type:** Always `performance_schema` .
- **Name:** The name of a table, the name of an internal buffer, or the `performance_schema` word, followed by a dot and an attribute. Internal buffers names are enclosed by parenthesis. `performance_schema` means that the attribute refers to the whole database (it is a total).
- **Status:** The value for the attribute.

The following attributes are shown, in this order, for all tables:

- **row_size:** The memory used for an individual record. This value will never change.
- **row_count:** The number of rows in the table or buffer. For some tables, this value depends on a server system variable.
- **memory:** For tables and `performance_schema` , this is the result of `row_size` * `row_count` .

For internal buffers, the attributes are:

- **count**
- **size**

SHOW ENGINE ROCKSDB STATUS

See also MyRocks Performance Troubleshooting

1.1.1.2.8.24 SHOW ENGINE INNODB STATUS

`SHOW ENGINE INNODB STATUS` is a specific form of the SHOW ENGINE statement that displays the InnoDB Monitor output, which is extensive InnoDB information which can be useful in diagnosing problems.

The following sections are displayed

- **Status:** Shows the timestamp, monitor name and the number of seconds, or the elapsed time between the current time and the time the InnoDB Monitor output was last displayed. The per-second averages are based upon this time.
- **BACKGROUND THREAD:** srv_master_thread lines show work performed by the main background thread.
- **SEMAPHORES:** Threads waiting for a semaphore and stats on how the number of times threads have needed a spin or a wait on a mutex or rw-lock semaphore. If this number of threads is large, there may be I/O or contention issues. Reducing the size of the innodb_thread_concurrency system variable may help if contention is related to thread scheduling. `Spin rounds per wait` shows the number of spinlock rounds per OS wait for a mutex.
- **LATEST FOREIGN KEY ERROR:** Only shown if there has been a foreign key constraint error, it displays the failed statement and information about the constraint and the related tables.
- **LATEST DETECTED DEADLOCK:** Only shown if there has been a deadlock, it displays the transactions involved in the deadlock and the statements being executed, held and required locked and the transaction rolled back to.
- **TRANSACTIONS:** The output of this section can help identify lock contention, as well as reasons for the deadlocks.
- **FILE I/O:** InnoDB thread information as well as pending I/O operations and I/O performance statistics.
- **INSERT BUFFER AND ADAPTIVE HASH INDEX:** InnoDB insert buffer (old name for the change buffer) and adaptive hash index status information, including the number of each type of operation performed, and adaptive hash index performance.
- **LOG:** InnoDB log information, including current log sequence number, how far the log has been flushed to disk, the position at which InnoDB last took a checkpoint, pending writes and write performance statistics.
- **BUFFER POOL AND MEMORY:** Information on buffer pool pages read and written, which allows you to see the number of data file I/O operations performed by your queries. See InnoDB Buffer Pool for more. Similar information is also available from the INFORMATION_SCHEMA.INNODB_BUFFER_POOL_STATS table.
- **ROW OPERATIONS:**Information about the main thread, including the number and performance rate for each type of row operation.

If the innodb_status_output_locks system variable is set to `1` , extended lock information will be displayed.

Example output:

```
=====================================
2019-09-06 12:44:13 0x7f93cc236700 INNODB MONITOR OUTPUT
=====================================
```

```
Per second averages calculated from the last 4 seconds
-----------------
BACKGROUND THREAD
-----------------
srv_master_thread loops: 2 srv_active, 0 srv_shutdown, 83698 srv_idle
srv_master_thread log flush and writes: 83682
----------
SEMAPHORES
----------
OS WAIT ARRAY INFO: reservation count 15
OS WAIT ARRAY INFO: signal count 8
RW-shared spins 0, rounds 20, OS waits 7
RW-excl spins 0, rounds 0, OS waits 0
RW-sx spins 0, rounds 0, OS waits 0
Spin rounds per wait: 20.00 RW-shared, 0.00 RW-excl, 0.00 RW-sx
------------
TRANSACTIONS
------------
Trx id counter 236
Purge done for trx's n:o < 236 undo n:o < 0 state: running
History list length 22
LIST OF TRANSACTIONS FOR EACH SESSION:
---TRANSACTION 421747401994584, not started
0 lock struct(s), heap size 1136, 0 row lock(s)
---TRANSACTION 421747401990328, not started
0 lock struct(s), heap size 1136, 0 row lock(s)
--------
FILE I/O
--------
I/O thread 0 state: waiting for completed aio requests (insert buffer thread)
I/O thread 1 state: waiting for completed aio requests (log thread)
I/O thread 2 state: waiting for completed aio requests (read thread)
I/O thread 3 state: waiting for completed aio requests (read thread)
I/O thread 4 state: waiting for completed aio requests (read thread)
I/O thread 5 state: waiting for completed aio requests (read thread)
I/O thread 6 state: waiting for completed aio requests (write thread)
I/O thread 7 state: waiting for completed aio requests (write thread)
I/O thread 8 state: waiting for completed aio requests (write thread)
I/O thread 9 state: waiting for completed aio requests (write thread)
Pending normal aio reads: [0, 0, 0, 0] , aio writes: [0, 0, 0, 0] ,
 ibuf aio reads:, log i/o's:, sync i/o's:
Pending flushes (fsync) log: 0; buffer pool: 0
286 OS file reads, 171 OS file writes, 22 OS fsyncs
0.00 reads/s, 0 avg bytes/read, 0.00 writes/s, 0.00 fsyncs/s
-------------------------------------
INSERT BUFFER AND ADAPTIVE HASH INDEX
-------------------------------------
Ibuf: size 1, free list len 0, seg size 2, 0 merges
merged operations:
 insert 0, delete mark 0, delete 0
discarded operations:
 insert 0, delete mark 0, delete 0
Hash table size 34679, node heap has 0 buffer(s)
Hash table size 34679, node heap has 0 buffer(s)
Hash table size 34679, node heap has 0 buffer(s)
Hash table size 34679, node heap has 0 buffer(s)
Hash table size 34679, node heap has 0 buffer(s)
Hash table size 34679, node heap has 0 buffer(s)
Hash table size 34679, node heap has 0 buffer(s)
Hash table size 34679, node heap has 0 buffer(s)
0.00 hash searches/s, 0.00 non-hash searches/s
---
LOG
---
Log sequence number 445926
Log flushed up to   445926
Pages flushed up to 445926
Last checkpoint at  445917
0 pending log flushes, 0 pending chkp writes
18 log i/o's done, 0.00 log i/o's/second
----------------------
BUFFER POOL AND MEMORY
----------------------
Total large memory allocated 167772160
Dictionary memory allocated 50768
```

```
Dictionary memory allocated 50700
Buffer pool size   8012
Free buffers       7611
Database pages     401
Old database pages 0
Modified db pages  0
Percent of dirty pages(LRU & free pages): 0.000
Max dirty pages percent: 75.000
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages made young 0, not young 0
0.00 youngs/s, 0.00 non-youngs/s
Pages read 264, created 137, written 156
0.00 reads/s, 0.00 creates/s, 0.00 writes/s
No buffer pool page gets since the last printout
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead 0.00/s
LRU len: 401, unzip_LRU len: 0
I/O sum[0]:cur[0], unzip sum[0]:cur[0]
--------------
ROW OPERATIONS
--------------
0 queries inside InnoDB, 0 queries in queue
0 read views open inside InnoDB
Process ID=4267, Main thread ID=140272021272320, state: sleeping
Number of rows inserted 1, updated 0, deleted 0, read 1
0.00 inserts/s, 0.00 updates/s, 0.00 deletes/s, 0.00 reads/s
Number of system rows inserted 0, updated 0, deleted 0, read 0
0.00 inserts/s, 0.00 updates/s, 0.00 deletes/s, 0.00 reads/s
----------------------------
END OF INNODB MONITOR OUTPUT
============================
```

1.1.1.2.8.25 SHOW ENGINES

Syntax

```
SHOW [STORAGE] ENGINES
```

Description

`SHOW ENGINES` displays status information about the server's storage engines. This is particularly useful for checking whether a storage engine is supported, or to see what the default engine is. `SHOW TABLE TYPES` is a deprecated synonym.

The `information_schema.ENGINES` table provides the same information.

Since storage engines are plugins, different information about them is also shown in the `information_schema.PLUGINS` table and by the `SHOW PLUGINS` statement.

Note that both MySQL's InnoDB and Percona's XtraDB replacement are labeled as `InnoDB`. However, if XtraDB is in use, it will be specified in the `COMMENT` field. See XtraDB and InnoDB. The same applies to FederatedX.

The output consists of the following columns:

- `Engine` indicates the engine's name.
- `Support` indicates whether the engine is installed, and whether it is the default engine for the current session.
- `Comment` is a brief description.
- `Transactions`, `XA` and `Savepoints` indicate whether transactions, XA transactions and transaction savepoints are supported by the engine.

Examples

```
SHOW ENGINES\G
*************************** 1. row ***************************
      Engine: InnoDB
     Support: DEFAULT
     Comment: Supports transactions, row-level locking, and foreign keys
Transactions: YES
          XA: YES
  Savepoints: YES
*************************** 2. row ***************************
      Engine: CSV
     Support: YES
     Comment: CSV storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 3. row ***************************
      Engine: MyISAM
     Support: YES
     Comment: MyISAM storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 4. row ***************************
      Engine: BLACKHOLE
     Support: YES
     Comment: /dev/null storage engine (anything you write to it disappears)
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 5. row ***************************
      Engine: FEDERATED
     Support: YES
     Comment: FederatedX pluggable storage engine
Transactions: YES
          XA: NO
  Savepoints: YES
*************************** 6. row ***************************
      Engine: MRG_MyISAM
     Support: YES
     Comment: Collection of identical MyISAM tables
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 7. row ***************************
      Engine: ARCHIVE
     Support: YES
     Comment: Archive storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 8. row ***************************
      Engine: MEMORY
     Support: YES
     Comment: Hash based, stored in memory, useful for temporary tables
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 9. row ***************************
      Engine: PERFORMANCE_SCHEMA
     Support: YES
     Comment: Performance Schema
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 10. row ***************************
      Engine: Aria
     Support: YES
     Comment: Crash-safe tables with MyISAM heritage
Transactions: NO
          XA: NO
  Savepoints: NO
10 rows in set (0.00 sec)
```

1.1.1.2.8.26 SHOW ERRORS

Syntax

```
SHOW ERRORS [LIMIT [offset,] row_count]
SHOW ERRORS [LIMIT row_count OFFSET offset]
SHOW COUNT(*) ERRORS
```

Contents

Description

This statement is similar to SHOW WARNINGS, except that instead of displaying errors, warnings, and notes, it displays only errors.

The `LIMIT` clause has the same syntax as for the SELECT statement.

The `SHOW COUNT(*) ERRORS` statement displays the number of errors. You can also retrieve this number from the error_count variable.

```
SHOW COUNT(*) ERRORS;
SELECT @@error_count;
```

The value of error_count might be greater than the number of messages displayed by SHOW WARNINGS if the max_error_count system variable is set so low that not all messages are stored.

For a list of MariaDB error codes, see MariaDB Error Codes.

Examples

```
SELECT f();
ERROR 1305 (42000): FUNCTION f does not exist

SHOW COUNT(*) ERRORS;
+-----------------------+
| @@session.error_count |
+-----------------------+
|                     1 |
+-----------------------+

SHOW ERRORS;
+-------+------+---------------------------+
| Level | Code | Message                   |
+-------+------+---------------------------+
| Error | 1305 | FUNCTION f does not exist |
+-------+------+---------------------------+
```

1.1.1.2.8.27 SHOW EVENTS

Syntax

```
SHOW EVENTS [{FROM | IN} schema_name]
    [LIKE 'pattern' | WHERE expr]
```

Description

Shows information about Event Manager events (created with `CREATE EVENT`). Requires the `EVENT` privilege. Without any

arguments, `SHOW EVENTS` lists all of the events in the current schema:

```
SELECT CURRENT_USER(), SCHEMA();
+----------------+----------+
| CURRENT_USER() | SCHEMA() |
+----------------+----------+
| jon@ghidora    | myschema |
+----------------+----------+

SHOW EVENTS\G
*************************** 1. row ***************************
                  Db: myschema
                Name: e_daily
             Definer: jon@ghidora
           Time zone: SYSTEM
                Type: RECURRING
          Execute at: NULL
      Interval value: 10
      Interval field: SECOND
             Starts: 2006-02-09 10:41:23
               Ends: NULL
              Status: ENABLED
          Originator: 0
character_set_client: latin1
collation_connection: latin1_swedish_ci
  Database Collation: latin1_swedish_ci
```

To see the event action, use `SHOW CREATE EVENT` instead, or look at the `information_schema.EVENTS` table.

To see events for a specific schema, use the `FROM` clause. For example, to see events for the test schema, use the following statement:

```
SHOW EVENTS FROM test;
```

The `LIKE` clause, if present, indicates which event names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in Extended Show.

4.2.2.8 SHOW FUNCTION CODE

1.1.1.2.8.29 SHOW FUNCTION STATUS

Syntax

```
SHOW FUNCTION STATUS
    [LIKE 'pattern' | WHERE expr]
```

Description

This statement is similar to `SHOW PROCEDURE STATUS` but for stored functions.

The LIKE clause, if present on its own, indicates which function names to match.

The `WHERE` and `LIKE` clauses can be given to select rows using more general conditions, as discussed in Extended SHOW.

The `information_schema.ROUTINES` table contains more detailed information.

Examples

Showing all stored functions:

```
SHOW FUNCTION STATUS\G
*************************** 1. row ***************************
                  Db: test
                Name: VatCents
                Type: FUNCTION
             Definer: root@localhost
            Modified: 2013-06-01 12:40:31
             Created: 2013-06-01 12:40:31
       Security_type: DEFINER
             Comment:
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
```

Stored functions whose name starts with 'V':

```
SHOW FUNCTION STATUS LIKE 'V%' \G
*************************** 1. row ***************************
                  Db: test
                Name: VatCents
                Type: FUNCTION
             Definer: root@localhost
            Modified: 2013-06-01 12:40:31
             Created: 2013-06-01 12:40:31
       Security_type: DEFINER
             Comment:
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
```

Stored functions with a security type of 'DEFINER':

```
SHOW FUNCTION STATUS WHERE Security_type LIKE 'DEFINER' \G
*************************** 1. row ***************************
                  Db: test
                Name: VatCents
                Type: FUNCTION
             Definer: root@localhost
            Modified: 2013-06-01 12:40:31
             Created: 2013-06-01 12:40:31
       Security_type: DEFINER
             Comment:
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
```

1.1.1.1.12 SHOW GRANTS

1.1.1.2.1.18 SHOW INDEX

1.1.1.2.8.32 SHOW INDEX_STATISTICS

Syntax

```
SHOW INDEX_STATISTICS
```

Description

The `SHOW INDEX_STATISTICS` statement was introduced in MariaDB 5.2 as part of the User Statistics feature. It was removed as a separate statement in MariaDB 10.1.1, but effectively replaced by the generic SHOW information_schema_table statement. The information_schmea.INDEX_STATISTICS table shows statistics on index usage and makes it possible to do such things as locating unused indexes and generating the commands to remove them.

The userstat system variable must be set to 1 to activate this feature. See the User Statistics and information_schema.INDEX_STATISTICS table for more information.

Example

```
SHOW INDEX_STATISTICS;
+--------------+------------------+------------+-----------+
| Table_schema | Table_name       | Index_name | Rows_read |
+--------------+------------------+------------+-----------+
| test         | employees_example | PRIMARY    |         1 |
+--------------+------------------+------------+-----------+
```

1.1.1.2.8.33 SHOW LOCALES

`SHOW LOCALES` was introduced as part of the Information Schema plugin extension.

`SHOW LOCALES` is used to return `locales` information as part of the Locales plugin. While the `information_schema.LOCALES` table has 8 columns, the `SHOW LOCALES` statement will only display 4 of them:

Example

```
SHOW LOCALES;
+-----+-------+-----------------------------------+-------------------------+
| Id  | Name  | Description                       | Error_Message_Language |
+-----+-------+-----------------------------------+-------------------------+
|   0 | en_US | English - United States           | english                 |
|   1 | en_GB | English - United Kingdom          | english                 |
|   2 | ja_JP | Japanese - Japan                  | japanese                |
|   3 | sv_SE | Swedish - Sweden                  | swedish                 |
...
```

1.1.1.2.5.8 SHOW BINLOG STATUS

1.1.1.2.8.35 SHOW OPEN TABLES

Syntax

```
SHOW OPEN TABLES [FROM db_name]
    [LIKE 'pattern' | WHERE expr]
```

Description

`SHOW OPEN TABLES` lists the non- `TEMPORARY` tables that are currently open in the table cache. See http://dev.mysql.com/doc/refman/5.1/en/table-cache.html.

The `FROM` and `LIKE` clauses may be used.

The `FROM` clause, if present, restricts the tables shown to those present in the `db_name` database.

The `LIKE` clause, if present on its own, indicates which table names to match. The `WHERE` and `LIKE` clauses can be given to select rows using more general conditions, as discussed in Extended SHOW.

The following information is returned:

Column	Description
Database	Database name.
Name	Table name.
In_use	Number of table instances being used.

Name_locked	`1` if the table is name-locked, e.g. if it is being dropped or renamed, otherwise `0`.

Before MariaDB 5.5, each use of, for example, LOCK TABLE ... WRITE would increment `In_use` for that table. With the implementation of the metadata locking improvements in MariaDB 5.5, `LOCK TABLE... WRITE` acquires a strong MDL lock, and concurrent connections will wait on this MDL lock, so any subsequent `LOCK TABLE... WRITE` will not increment `In_use`.

Example

```
SHOW OPEN TABLES;
+----------+----------------------------+--------+-------------+
| Database | Table                      | In_use | Name_locked |
+----------+----------------------------+--------+-------------+
...
| test     | xjson                      |      0 |           0 |
| test     | jauthor                    |      0 |           0 |
| test     | locks                      |      1 |           0 |
...
+----------+----------------------------+--------+-------------+
```

1.1.1.2.8.36 SHOW PACKAGE BODY STATUS

MariaDB starting with 10.3.5
Oracle-style packages were introduced in MariaDB 10.3.5.

Syntax

```
SHOW PACKAGE BODY STATUS
    [LIKE 'pattern' | WHERE expr]
```

Contents

Description

The `SHOW PACKAGE BODY STATUS` statement returns characteristics of stored package bodies (implementations), such as the database, name, type, creator, creation and modification dates, and character set information. A similar statement, `SHOW PACKAGE STATUS`, displays information about stored package specifications.

The `LIKE` clause, if present, indicates which package names to match. The `WHERE` and `LIKE` clauses can be given to select rows using more general conditions, as discussed in Extended SHOW.

The ROUTINES table in the INFORMATION_SCHEMA database contains more detailed information.

Examples

```
SHOW PACKAGE BODY STATUS LIKE 'pkg1'\G
*************************** 1. row ***************************
                  Db: test
                Name: pkg1
                Type: PACKAGE BODY
             Definer: root@localhost
            Modified: 2018-02-27 14:44:14
             Created: 2018-02-27 14:44:14
       Security_type: DEFINER
             Comment: This is my first package body
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
```

1.1.1.2.8.37 SHOW PACKAGE STATUS

MariaDB starting with 10.3.5
Oracle-style packages were introduced in MariaDB 10.3.5.

Syntax

```
SHOW PACKAGE STATUS
    [LIKE 'pattern' | WHERE expr]
```

Contents

Description

The `SHOW PACKAGE STATUS` statement returns characteristics of stored package specifications, such as the database, name, type, creator, creation and modification dates, and character set information. A similar statement, `SHOW PACKAGE BODY STATUS`, displays information about stored package bodies (i.e. implementations).

The `LIKE` clause, if present, indicates which package names to match. The `WHERE` and `LIKE` clauses can be given to select rows using more general conditions, as discussed in Extended SHOW.

The ROUTINES table in the INFORMATION_SCHEMA database contains more detailed information.

Examples

```
SHOW PACKAGE STATUS LIKE 'pkg1'\G
*************************** 1. row ***************************
                  Db: test
                Name: pkg1
                Type: PACKAGE
             Definer: root@localhost
            Modified: 2018-02-27 14:38:15
             Created: 2018-02-27 14:38:15
       Security_type: DEFINER
             Comment: This is my first package
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
```

1.1.1.2.6.1 SHOW PLUGINS

1.1.1.2.6.2 SHOW PLUGINS SONAME

1.1.1.2.8.40 SHOW PRIVILEGES

Syntax

```
SHOW PRIVILEGES
```

Description

`SHOW PRIVILEGES` shows the list of system privileges that the MariaDB server supports. The exact list of privileges depends on the version of your server.

Note that before MariaDB 10.3.23, MariaDB 10.4.13 and MariaDB 10.5.2, the Delete history privilege displays as

`Delete versioning rows` (MDEV-20382).

Example

From MariaDB 10.5.9

```
SHOW PRIVILEGES;
+--------------------------+---------------------------------------+-------------------------
-----------------------------------------+
| Privilege                | Context                               | Comment
|
+--------------------------+---------------------------------------+-------------------------
-----------------------------------------+
| Alter                    | Tables                                | To alter the table
|
| Alter routine            | Functions,Procedures                  | To alter or drop stored
functions/procedures                      |
| Create                   | Databases,Tables,Indexes              | To create new databases
and tables                                |
| Create routine           | Databases                             | To use CREATE
FUNCTION/PROCEDURE                                 |
| Create temporary tables  | Databases                             | To use CREATE TEMPORARY
TABLE                                     |
| Create view              | Tables                                | To create new views
|
| Create user              | Server Admin                          | To create new users
|
| Delete                   | Tables                                | To delete existing rows
|
| Delete history           | Tables                                | To delete versioning
table historical rows                          |
| Drop                     | Databases,Tables                      | To drop databases,
tables, and views                               |
| Event                    | Server Admin                          | To create, alter, drop
and execute events                             |
| Execute                  | Functions,Procedures                  | To execute stored routines
|
| File                     | File access on server                 | To read and write files
on the server                             |
| Grant option             | Databases,Tables,Functions,Procedures | To give to other users
those privileges you possess                   |
| Index                    | Tables                                | To create or drop indexes
|
| Insert                   | Tables                                | To insert data into
tables                                      |
| Lock tables              | Databases                             | To use LOCK TABLES
(together with SELECT privilege)                |
| Process                  | Server Admin                          | To view the plain text of
currently executing queries                |
| Proxy                    | Server Admin                          | To make proxy user
possible                                        |
| References               | Databases,Tables                      | To have references on
tables                                        |
| Reload                   | Server Admin                          | To reload or refresh
tables, logs and privileges                     |
| Binlog admin             | Server                                | To purge binary logs
|
| Binlog monitor           | Server                                | To use SHOW BINLOG STATUS
and SHOW BINARY LOG                           |
| Binlog replay            | Server                                | To use BINLOG (generated
by mariadb-binlog)                             |
| Replication master admin | Server                                | To monitor connected
slaves                                          |
| Replication slave admin  | Server                                | To start/stop slave and
apply binlog events                            |
| Slave monitor            | Server                                | To use SHOW SLAVE STATUS
and SHOW RELAYLOG EVENTS                       |
| Replication slave        | Server Admin                          | To read binary log events
from the master                               |
| Select                   | Tables                                | To retrieve rows from
table                                         |
| Show databases           | Server Admin                          | To see all databases with
SHOW DATABASES                                |
```

```
| Show view                | Tables                                  | To see views with SHOW
CREATE VIEW                                 |
| Shutdown                 | Server Admin                            | To shut down the server
|
| Super                    | Server Admin                            | To use KILL thread, SET
GLOBAL, CHANGE MASTER, etc.                 |
| Trigger                  | Tables                                  | To use triggers
|
| Create tablespace        | Server Admin                            | To create/alter/drop
tablespaces                                   |
| Update                   | Tables                                  | To update existing rows
|
| Set user                 | Server                                  | To create views and
stored routines with a different definer       |
| Federated admin          | Server                                  | To execute the CREATE
SERVER, ALTER SERVER, DROP SERVER statements |
| Connection admin         | Server                                  | To bypass connection
limits and kill other users' connections      |
| Read_only admin          | Server                                  | To perform write
operations even if @@read_only=ON                |
| Usage                    | Server Admin                            | No privileges - allow
connect only                                 |
+--------------------------+-----------------------------------------+-------------------------
-------------------------------------------+
41 rows in set (0.000 sec)
```

1.1.1.2.8.41 SHOW PROCEDURE CODE

Syntax

```
SHOW PROCEDURE CODE proc_name
```

Contents

Description

This statement is a MariaDB extension that is available only for servers that have been built with debugging support. It displays a representation of the internal implementation of the named stored procedure. A similar statement, `SHOW FUNCTION CODE`, displays information about stored functions.

Both statements require that you be the owner of the routine or have `SELECT` access to the `mysql.proc` table.

If the named routine is available, each statement produces a result set. Each row in the result set corresponds to one "instruction" in the routine. The first column is Pos, which is an ordinal number beginning with 0. The second column is Instruction, which contains an SQL statement (usually changed from the original source), or a directive which has meaning only to the stored-routine handler.

Examples

```
DELIMITER //

CREATE PROCEDURE p1 ()
  BEGIN
    DECLARE fanta INT DEFAULT 55;
    DROP TABLE t2;
    LOOP
      INSERT INTO t3 VALUES (fanta);
      END LOOP;
  END//
Query OK, 0 rows affected (0.00 sec)

SHOW PROCEDURE CODE p1//
+-----+----------------------------------------+
| Pos | Instruction                            |
+-----+----------------------------------------+
|   0 | set fanta@0 55                         |
|   1 | stmt 9 "DROP TABLE t2"                 |
|   2 | stmt 5 "INSERT INTO t3 VALUES (fanta)" |
|   3 | jump 2                                 |
+-----+----------------------------------------+
```

1.1.1.2.8.42 SHOW PROCEDURE STATUS

Syntax

```
SHOW PROCEDURE STATUS
    [LIKE 'pattern' | WHERE expr]
```

Description

This statement is a MariaDB extension. It returns characteristics of a stored procedure, such as the database, name, type, creator, creation and modification dates, and character set information. A similar statement, `SHOW FUNCTION STATUS`, displays information about stored functions.

The `LIKE` clause, if present, indicates which procedure or function names to match. The `WHERE` and `LIKE` clauses can be given to select rows using more general conditions, as discussed in Extended SHOW.

The ROUTINES table in the INFORMATION_SCHEMA database contains more detailed information.

Examples

```
SHOW PROCEDURE STATUS LIKE 'p1'\G
*************************** 1. row ***************************
                  Db: test
                Name: p1
                Type: PROCEDURE
             Definer: root@localhost
            Modified: 2010-08-23 13:23:03
             Created: 2010-08-23 13:23:03
       Security_type: DEFINER
             Comment:
character_set_client: latin1
collation_connection: latin1_swedish_ci
  Database Collation: latin1_swedish_ci
```

1.1.1.2.8.43 SHOW PROCESSLIST

Syntax

```
SHOW [FULL] PROCESSLIST
```

Description

`SHOW PROCESSLIST` shows you which threads are running. You can also get this information from the information_schema.PROCESSLIST table or the mariadb-admin processlist command. If you have the `PROCESS` privilege, you can see all threads. Otherwise, you can see only your own threads (that is, threads associated with the MariaDB account that you are using). If you do not use the `FULL` keyword, only the first 100 characters of each statement are shown in the Info field.

The columns shown in `SHOW PROCESSLIST` are:

Name	Description
`ID`	The client's process ID.
`USER`	The username associated with the process.
`HOST`	The host the client is connected to.
`DB`	The default database of the process (NULL if no default).
`COMMAND`	The command type. See Thread Command Values.
`TIME`	The amount of time, in seconds, the process has been in its current state. For a replica SQL thread before MariaDB 10.1, this is the time in seconds between the last replicated event's timestamp and the replica machine's real time.
`STATE`	See Thread States.
`INFO`	The statement being executed.
`PROGRESS`	The total progress of the process (0-100%) (see Progress Reporting).

See `TIME_MS` column in information_schema.PROCESSLIST for differences in the `TIME` column between MariaDB and MySQL.

The information_schema.PROCESSLIST table contains the following additional columns:

Name	Description
`TIME_MS`	The amount of time, in milliseconds, the process has been in its current state.
`STAGE`	The stage the process is currently in.
`MAX_STAGE`	The maximum number of stages.
`PROGRESS`	The progress of the process within the current stage (0-100%).
`MEMORY_USED`	The amount of memory used by the process.
`EXAMINED_ROWS`	The number of rows the process has examined.
`QUERY_ID`	Query ID.

Note that the `PROGRESS` field from the information schema, and the `PROGRESS` field from `SHOW PROCESSLIST` display different results. `SHOW PROCESSLIST` shows the total progress, while the information schema shows the progress for the current stage only.

Threads can be killed using their thread_id or their query_id, with the KILL statement.

Since queries on this table are locking, if the performance_schema is enabled, you may want to query the `THREADS` table instead.

Examples

```
SHOW PROCESSLIST;
+----+-----------------+-----------+------+---------+------+-------------------------+--------
----------+----------+
| Id | User            | Host      | db   | Command | Time | State                   | Info
| Progress |
+----+-----------------+-----------+------+---------+------+-------------------------+--------
----------+----------+
|  2 | event_scheduler | localhost | NULL | Daemon  | 2693 | Waiting on empty queue | NULL
|    0.000 |
|  4 | root            | localhost | NULL | Query   |    0 | Table lock             | SHOW
PROCESSLIST |    0.000 |
+----+-----------------+-----------+------+---------+------+-------------------------+--------
----------+----------+
```

1.1.1.2.8.44 SHOW PROFILE

Syntax

```
SHOW PROFILE [type [, type] ... ]
    [FOR QUERY n]
    [LIMIT row_count [OFFSET offset]]

type:
    ALL
  | BLOCK IO
  | CONTEXT SWITCHES
  | CPU
  | IPC
  | MEMORY
  | PAGE FAULTS
  | SOURCE
  | SWAPS
```

Description

The `SHOW PROFILE` and `SHOW PROFILES` statements display profiling information that indicates resource usage for statements executed during the course of the current session.

Profiling is controlled by the profiling session variable, which has a default value of `0` (`OFF`). Profiling is enabled by setting profiling to `1` or `ON` :

```
SET profiling = 1;
```

`SHOW PROFILES` displays a list of the most recent statements sent to the master. The size of the list is controlled by the `profiling_history_size` session variable, which has a default value of `15` . The maximum value is `100` . Setting the value to `0` has the practical effect of disabling profiling.

All statements are profiled except `SHOW PROFILES` and `SHOW PROFILE` , so you will find neither of those statements in the profile list. Malformed statements are profiled. For example, `SHOW PROFILING` is an illegal statement, and a syntax error occurs if you try to execute it, but it will show up in the profiling list.

`SHOW PROFILE` displays detailed information about a single statement. Without the `FOR QUERY n` clause, the output pertains to the most recently executed statement. If `FOR QUERY n` is included, `SHOW PROFILE` displays information for statement *n*. The values of *n* correspond to the `Query_ID` values displayed by `SHOW PROFILES` .

The `LIMIT row_count` clause may be given to limit the output to *row_count* rows. If `LIMIT` is given, `OFFSET offset` may be added to begin the output offset rows into the full set of rows.

By default, `SHOW PROFILE` displays Status and Duration columns. The Status values are like the State values displayed by `SHOW PROCESSLIST` , although there might be some minor differences in interpretation for the two statements for some status values (see http://dev.mysql.com/doc/refman/5.6/en/thread-information.html).

Optional type values may be specified to display specific additional types of information:

- **`ALL`** displays all information
- **`BLOCK IO`** displays counts for block input and output operations

- `CONTEXT SWITCHES` displays counts for voluntary and involuntary context switches
- `CPU` displays user and system CPU usage times
- `IPC` displays counts for messages sent and received
- `MEMORY` is not currently implemented
- `PAGE FAULTS` displays counts for major and minor page faults
- `SOURCE` displays the names of functions from the source code, together with the name and line number of the file in which the function occurs
- `SWAPS` displays swap counts

Profiling is enabled per session. When a session ends, its profiling information is lost.

The `information_schema.PROFILING` table contains similar information.

Examples

```
SELECT @@profiling;
+-------------+
| @@profiling |
+-------------+
|           0 |
+-------------+

SET profiling = 1;

USE test;

DROP TABLE IF EXISTS t1;

CREATE TABLE T1 (id INT);

SHOW PROFILES;
+----------+------------+--------------------------+
| Query_ID | Duration   | Query                    |
+----------+------------+--------------------------+
|        1 | 0.00009200 | SELECT DATABASE()        |
|        2 | 0.00023800 | show databases           |
|        3 | 0.00018900 | show tables              |
|        4 | 0.00014700 | DROP TABLE IF EXISTS t1  |
|        5 | 0.24476900 | CREATE TABLE T1 (id INT) |
+----------+------------+--------------------------+

SHOW PROFILE;
+----------------------+----------+
| Status               | Duration |
+----------------------+----------+
| starting             | 0.000042 |
| checking permissions | 0.000044 |
| creating table       | 0.244645 |
| After create         | 0.000013 |
| query end            | 0.000003 |
| freeing items        | 0.000016 |
| logging slow query   | 0.000003 |
| cleaning up          | 0.000003 |
+----------------------+----------+

SHOW PROFILE FOR QUERY 4;
+--------------------+----------+
| Status             | Duration |
+--------------------+----------+
| starting           | 0.000126 |
| query end          | 0.000004 |
| freeing items      | 0.000012 |
| logging slow query | 0.000003 |
| cleaning up        | 0.000002 |
+--------------------+----------+

SHOW PROFILE CPU FOR QUERY 5;
+----------------------+----------+----------+------------+
| Status               | Duration | CPU_user | CPU_system |
+----------------------+----------+----------+------------+
| starting             | 0.000042 | 0.000000 |   0.000000 |
| checking permissions | 0.000044 | 0.000000 |   0.000000 |
| creating table       | 0.244645 | 0.000000 |   0.000000 |
| After create         | 0.000013 | 0.000000 |   0.000000 |
| query end            | 0.000003 | 0.000000 |   0.000000 |
| freeing items        | 0.000016 | 0.000000 |   0.000000 |
| logging slow query   | 0.000003 | 0.000000 |   0.000000 |
| cleaning up          | 0.000003 | 0.000000 |   0.000000 |
+----------------------+----------+----------+------------+
```

1.1.1.2.8.45 SHOW PROFILES

Syntax

```
SHOW PROFILES
```

Description

The `SHOW PROFILES` statement displays profiling information that indicates resource usage for statements executed during the course of the current session. It is used together with `SHOW PROFILE`.

1.1.1.2.8.46 SHOW QUERY_RESPONSE_TIME

It is possible to use `SHOW QUERY_RESPONSE_TIME` as an alternative for retrieving information from the QUERY_RESPONSE_TIME plugin.

This was introduced as part of the Information Schema plugin extension.

1.1.1.2.5.6 SHOW RELAYLOG EVENTS

1.1.1.2.5.9 SHOW REPLICA HOSTS

1.1.1.2.5.7 SHOW REPLICA STATUS

1.1.1.2.8.50 SHOW STATUS

Syntax

```
SHOW [GLOBAL | SESSION] STATUS
    [LIKE 'pattern' | WHERE expr]
```

Description

`SHOW STATUS` provides server status information. This information also can be obtained using the mariadb-admin extended-status command, or by querying the Information Schema GLOBAL_STATUS and SESSION_STATUS tables. The `LIKE` clause, if present, indicates which variable names to match. The `WHERE` clause can be given to select rows using more general conditions.

With the `GLOBAL` modifier, `SHOW STATUS` displays the status values for all connections to MariaDB. With `SESSION`, it displays the status values for the current connection. If no modifier is present, the default is `SESSION`. `LOCAL` is a synonym for `SESSION`. If you see a lot of 0 values, the reason is probably that you have used `SHOW STATUS` with a new connection instead of `SHOW GLOBAL STATUS`.

Some status variables have only a global value. For these, you get the same value for both `GLOBAL` and `SESSION`.

See Server Status Variables for a full list, scope and description of the variables that can be viewed with `SHOW STATUS`.

The `LIKE` clause, if present on its own, indicates which variable name to match.

The `WHERE` and `LIKE` clauses can be given to select rows using more general conditions, as discussed in Extended SHOW.

Examples

```
SHOW GLOBAL STATUS;
+-----------------------------------------------------------------+----------------------------
----------+
| Variable_name                                                   | Value
|
+-----------------------------------------------------------------+----------------------------
----------+
| Aborted_clients                                                 | 0
|
| Aborted_connects                                                | 0
|
| Access_denied_errors                                            | 0
|
| Acl_column_grants                                               | 0
|
| Acl_database_grants                                             | 2
|
| Acl_function_grants                                             | 0
|
| Acl_procedure_grants                                            | 0
|
| Acl_proxy_users                                                 | 2
|
| Acl_role_grants                                                 | 0
|
| Acl_roles                                                       | 0
|
| Acl_table_grants                                                | 0
|
| Acl_users                                                       | 6
|
| Aria_pagecache_blocks_not_flushed                               | 0
|
| Aria_pagecache_blocks_unused                                    | 15706
|
...
| wsrep_local_index                                               | 18446744073709551615
|
| wsrep_provider_name                                             |
|
| wsrep_provider_vendor                                           |
|
| wsrep_provider_version                                          |
|
| wsrep_ready                                                     | OFF
|
| wsrep_thread_count                                              | 0
|
+-----------------------------------------------------------------+----------------------------
----------+
516 rows in set (0.00 sec)
```

Example of filtered output:

```
SHOW STATUS LIKE 'Key%';
+------------------------+--------+
| Variable_name          | Value  |
+------------------------+--------+
| Key_blocks_not_flushed | 0      |
| Key_blocks_unused      | 107163 |
| Key_blocks_used        | 0      |
| Key_blocks_warm        | 0      |
| Key_read_requests      | 0      |
| Key_reads              | 0      |
| Key_write_requests     | 0      |
| Key_writes             | 0      |
+------------------------+--------+
8 rows in set (0.00 sec)
```

1.1.1.2.8.51 SHOW TABLE STATUS

Syntax

```
SHOW TABLE STATUS [{FROM | IN} db_name]
    [LIKE 'pattern' | WHERE expr]
```

Contents

Description

`SHOW TABLE STATUS` works like SHOW TABLES, but provides more extensive information about each table (until MariaDB 11.2.0, only non-TEMPORARY tables are shown).

The `LIKE` clause, if present on its own, indicates which table names to match. The `WHERE` and `LIKE` clauses can be given to select rows using more general conditions, as discussed in Extended SHOW.

The following information is returned:

Column	Description
Name	Table name.
Engine	Table storage engine.
Version	Version number from the table's .frm file.
Row_format	Row format (see InnoDB, Aria and MyISAM row formats).
Rows	Number of rows in the table. Some engines, such as InnoDB may store an estimate.
Avg_row_length	Average row length in the table.
Data_length	For InnoDB, the index size, in pages, multiplied by the page size. For Aria and MyISAM, length of the data file, in bytes. For MEMORY, the approximate allocated memory.
Max_data_length	Maximum length of the data file, ie the total number of bytes that could be stored in the table. Not used in InnoDB.
Index_length	Length of the index file.
Data_free	Bytes allocated but unused. For InnoDB tables in a shared tablespace, the free space of the shared tablespace with small safety margin. An estimate in the case of partitioned tables - see the PARTITIONS table.
Auto_increment	Next AUTO_INCREMENT value.
Create_time	Time the table was created. Some engines just return the ctime information from the file system layer here, in that case the value is not necessarily the table creation time but rather the time the file system metadata for it had last changed.
Update_time	Time the table was last updated. On Windows, the timestamp is not updated on update, so MyISAM values will be inaccurate. In InnoDB, if shared tablespaces are used, will be `NULL`, while buffering can also delay the update, so the value will differ from the actual time of the last `UPDATE`, `INSERT` or `DELETE`.
Check_time	Time the table was last checked. Not kept by all storage engines, in which case will be `NULL`.
Collation	Character set and collation.
Checksum	Live checksum value, if any.
Create_options	Extra CREATE TABLE options.
Comment	Table comment provided when MariaDB created the table.
Max_index_length	Maximum index length (supported by MyISAM and Aria tables).
Temporary	Until MariaDB 11.2.0, placeholder to signal that a table is a temporary table and always "N", except "Y" for generated information_schema tables and NULL for views. From MariaDB 11.2.0, will also be set to "Y" for local temporary tables.

Similar information can be found in the information_schema.TABLES table as well as by using mariadb-show:

```
mariadb-show --status db_name
```

Views

For views, all columns in `SHOW TABLE STATUS` are `NULL` except 'Name' and 'Comment'

Example

```
show table status\G
*************************** 1. row ***************************
           Name: bus_routes
         Engine: InnoDB
        Version: 10
     Row_format: Dynamic
           Rows: 5
 Avg_row_length: 3276
    Data_length: 16384
Max_data_length: 0
   Index_length: 0
      Data_free: 0
 Auto_increment: NULL
    Create_time: 2017-05-24 11:17:46
    Update_time: NULL
     Check_time: NULL
      Collation: latin1_swedish_ci
       Checksum: NULL
 Create_options:
        Comment:
```

1.1.1.2.8.52 SHOW TABLES

Syntax

```
SHOW [FULL] TABLES [FROM db_name]
    [LIKE 'pattern' | WHERE expr]
```

Contents

Description

`SHOW TABLES` lists the tables (until MariaDB 11.2.0, only non- `TEMPORARY` tables are shown), sequences and views in a given database.

The `LIKE` clause, if present on its own, indicates which table names to match. The `WHERE` and `LIKE` clauses can be given to select rows using more general conditions, as discussed in Extended SHOW. For example, when searching for tables in the `test` database, the column name for use in the `WHERE` and `LIKE` clauses will be `Tables_in_test`

The `FULL` modifier is supported such that `SHOW FULL TABLES` displays a second output column. Values for the second column, `Table_type`, are `BASE TABLE` for a table, `VIEW` for a view and `SEQUENCE` for a sequence.

You can also get this information using:

```
mariadb-show db_name
```

See mariadb-show for more details.

If you have no privileges for a base table or view, it does not show up in the output from `SHOW TABLES` or `mariadb-show db_name`.

The information_schema.TABLES table, as well as the SHOW TABLE STATUS statement, provide extended information about tables.

Examples

```
SHOW TABLES;
+----------------------+
| Tables_in_test       |
+----------------------+
| animal_count         |
| animals              |
| are_the_mooses_loose |
| aria_test2           |
| t1                   |
| view1                |
+----------------------+
```

Showing the tables beginning with *a* only.

```
SHOW TABLES WHERE Tables_in_test LIKE 'a%';
+----------------------+
| Tables_in_test       |
+----------------------+
| animal_count         |
| animals              |
| are_the_mooses_loose |
| aria_test2           |
+----------------------+
```

Showing tables and table types:

```
SHOW FULL TABLES;
+----------------+------------+
| Tables_in_test | Table_type |
+----------------+------------+
| s1             | SEQUENCE   |
| student        | BASE TABLE |
| v1             | VIEW       |
+----------------+------------+
```

Showing temporary tables: <= MariaDB 11.1

```
CREATE TABLE t (t int(11));
CREATE TEMPORARY TABLE t (t int(11));
CREATE TEMPORARY TABLE te (t int(11));

SHOW TABLES;
+----------------+
| Tables_in_test |
+----------------+
| t              |
+----------------+
```

From MariaDB 11.2.0:

```
CREATE TABLE t (t int(11));
CREATE TEMPORARY TABLE t (t int(11));
CREATE TEMPORARY TABLE te (t int(11));

SHOW TABLES;
+----------------+
| Tables_in_test |
+----------------+
| te             |
| t              |
| t              |
+----------------+
```

1.1.1.2.8.53 SHOW TABLE_STATISTICS

Syntax

```
SHOW TABLE_STATISTICS
```

Description

The `SHOW TABLE_STATISTICS` statementis part of the User Statistics feature. It was removed as a separate statement in MariaDB 10.1.1, but effectively replaced by the generic SHOW information_schema_table statement. The information_schema.TABLE_STATISTICS table shows statistics on table usage

The userstat system variable must be set to 1 to activate this feature. See the User Statistics and information_schema.TABLE_STATISTICS articles for more information.

Example

```
SHOW TABLE_STATISTICS\G
*************************** 1. row ***************************
           Table_schema: mysql
             Table_name: proxies_priv
              Rows_read: 2
           Rows_changed: 0
Rows_changed_x_#indexes: 0
*************************** 2. row ***************************
           Table_schema: test
             Table_name: employees_example
              Rows_read: 7
           Rows_changed: 0
Rows_changed_x_#indexes: 0
*************************** 3. row ***************************
           Table_schema: mysql
             Table_name: user
              Rows_read: 16
           Rows_changed: 0
Rows_changed_x_#indexes: 0
*************************** 4. row ***************************
           Table_schema: mysql
             Table_name: db
              Rows_read: 2
           Rows_changed: 0
Rows_changed_x_#indexes: 0
```

1.1.1.2.8.54 SHOW TRIGGERS

Syntax

```
SHOW TRIGGERS [FROM db_name]
    [LIKE 'pattern' | WHERE expr]
```

Contents

Description

`SHOW TRIGGERS` lists the triggers currently defined for tables in a database (the default database unless a `FROM` clause is given). This statement requires the `TRIGGER` privilege (prior to MySQL 5.1.22, it required the `SUPER` privilege).

The `LIKE` clause, if present on its own, indicates which table names to match and causes the statement to display triggers for those tables. The `WHERE` and `LIKE` clauses can be given to select rows using more general conditions, as discussed in Extended SHOW.

Similar information is stored in the `information_schema.TRIGGERS` table.

MariaDB starting with 10.2.3

If there are multiple triggers for the same action, then the triggers are shown in action order.

Examples

For the trigger defined at Trigger Overview:

```
SHOW triggers Like 'animals' \G
*************************** 1. row ***************************
             Trigger: the_mooses_are_loose
               Event: INSERT
               Table: animals
           Statement: BEGIN
 IF NEW.name = 'Moose' THEN
  UPDATE animal_count SET animal_count.animals = animal_count.animals+100;
 ELSE
  UPDATE animal_count SET animal_count.animals = animal_count.animals+1;
 END IF;
END
              Timing: AFTER
             Created: 2016-09-29 13:53:34.35
            sql_mode:
             Definer: root@localhost
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
```

Listing all triggers associated with a certain table:

```
SHOW TRIGGERS FROM test WHERE `Table` = 'user' \G
*************************** 1. row ***************************
             Trigger: user_ai
               Event: INSERT
               Table: user
           Statement: BEGIN END
              Timing: AFTER
             Created:  2016-09-29 13:53:34.35
            sql_mode:
             Definer: root@%
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
```

```
SHOW triggers WHERE Event Like 'Insert' \G
*************************** 1. row ***************************
             Trigger: the_mooses_are_loose
               Event: INSERT
               Table: animals
           Statement: BEGIN
 IF NEW.name = 'Moose' THEN
  UPDATE animal_count SET animal_count.animals = animal_count.animals+100;
 ELSE
  UPDATE animal_count SET animal_count.animals = animal_count.animals+1;
 END IF;
END
              Timing: AFTER
             Created: 2016-09-29 13:53:34.35
            sql_mode:
             Definer: root@localhost
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
```

- `character_set_client` is the session value of the `character_set_client` system variable when the trigger was created.
- `collation_connection` is the session value of the `collation_connection` system variable when the trigger was created.
- `Database Collation` is the collation of the database with which the trigger is associated.

These columns were added in MariaDB/MySQL 5.1.21.

Old triggers created before MySQL 5.7 and MariaDB 10.2.3 has NULL in the `Created` column.

1.1.1.2.8.55 SHOW USER_STATISTICS

Syntax

```
SHOW USER_STATISTICS
```

Description

The `SHOW USER_STATISTICS` statement is part of the User Statistics feature. It was removed as a separate statement in MariaDB 10.1.1, but effectively replaced by the generic SHOW information_schema_table statement. The information_schema.USER_STATISTICS table holds statistics about user activity. You can use this table to find out such things as which user is causing the most load and which users are being abusive. You can also use this table to measure how close to capacity the server may be.

The userstat system variable must be set to 1 to activate this feature. See the User Statistics and information_schema.USER_STATISTICS table for more information.

Example

```
SHOW USER_STATISTICS\G
*************************** 1. row ***************************
                  User: root
     Total_connections: 1
Concurrent_connections: 0
        Connected_time: 3297
             Busy_time: 0.14113400000000006
              Cpu_time: 0.017637000000000003
        Bytes_received: 969
            Bytes_sent: 22355
  Binlog_bytes_written: 0
             Rows_read: 10
             Rows_sent: 67
          Rows_deleted: 0
         Rows_inserted: 0
          Rows_updated: 0
       Select_commands: 7
       Update_commands: 0
        Other_commands: 0
   Commit_transactions: 1
 Rollback_transactions: 0
    Denied_connections: 0
      Lost_connections: 0
         Access_denied: 0
         Empty_queries: 7
```

1.1.1.2.8.56 SHOW VARIABLES

Syntax

```
SHOW [GLOBAL | SESSION] VARIABLES
    [LIKE 'pattern' | WHERE expr]
```

Contents

Description

`SHOW VARIABLES` shows the values of MariaDB system variables. This information also can be obtained using the mariadb-admin variables command. The `LIKE` clause, if present, indicates which variable names to match. The `WHERE` clause can be given to select rows using more general conditions.

With the `GLOBAL` modifier, `SHOW VARIABLES` displays the values that are used for new connections to MariaDB. With `SESSION`, it displays the values that are in effect for the current connection. If no modifier is present, the default is `SESSION`. `LOCAL` is a synonym for `SESSION`. With a `LIKE` clause, the statement displays only rows for those variables with names that match the pattern. To obtain the row for a specific variable, use a `LIKE` clause as shown:

```
SHOW VARIABLES LIKE 'maria_group_commit';
SHOW SESSION VARIABLES LIKE 'maria_group_commit';
```

To get a list of variables whose name match a pattern, use the " `%` " wildcard character in a `LIKE` clause:

```
SHOW VARIABLES LIKE '%maria%';
SHOW GLOBAL VARIABLES LIKE '%maria%';
```

Wildcard characters can be used in any position within the pattern to be matched. Strictly speaking, because " `_` " is a wildcard that matches any single character, you should escape it as " `\_` " to match it literally. In practice, this is rarely necessary.

The `WHERE` and `LIKE` clauses can be given to select rows using more general conditions, as discussed in Extended SHOW.

See `SET` for information on setting server system variables.

See Server System Variables for a list of all the variables that can be set.

You can also see the server variables by querying the Information Schema GLOBAL_VARIABLES and SESSION_VARIABLES tables.

Examples

```
SHOW VARIABLES LIKE 'aria%';
+-----------------------------------------+---------------------+
| Variable_name                           | Value               |
+-----------------------------------------+---------------------+
| aria_block_size                         | 8192                |
| aria_checkpoint_interval                | 30                  |
| aria_checkpoint_log_activity            | 1048576             |
| aria_force_start_after_recovery_failures | 0                   |
| aria_group_commit                       | none                |
| aria_group_commit_interval              | 0                   |
| aria_log_file_size                      | 1073741824          |
| aria_log_purge_type                     | immediate           |
| aria_max_sort_file_size                 | 9223372036853727232 |
| aria_page_checksum                      | ON                  |
| aria_pagecache_age_threshold            | 300                 |
| aria_pagecache_buffer_size              | 134217728           |
| aria_pagecache_division_limit           | 100                 |
| aria_recover                            | NORMAL              |
| aria_repair_threads                     | 1                   |
| aria_sort_buffer_size                   | 134217728           |
| aria_stats_method                       | nulls_unequal       |
| aria_sync_log_dir                       | NEWFILE             |
| aria_used_for_temp_tables               | ON                  |
+-----------------------------------------+---------------------+
```

```
SELECT VARIABLE_NAME, SESSION_VALUE, GLOBAL_VALUE FROM
  INFORMATION_SCHEMA.SYSTEM_VARIABLES WHERE
  VARIABLE_NAME LIKE 'max_error_count' OR
  VARIABLE_NAME LIKE 'innodb_sync_spin_loops';
+--------------------------+---------------+--------------+
| VARIABLE_NAME            | SESSION_VALUE | GLOBAL_VALUE |
+--------------------------+---------------+--------------+
| MAX_ERROR_COUNT          | 64            | 64           |
| INNODB_SYNC_SPIN_LOOPS   | NULL          | 30           |
+--------------------------+---------------+--------------+

SET GLOBAL max_error_count=128;

SELECT VARIABLE_NAME, SESSION_VALUE, GLOBAL_VALUE FROM
  INFORMATION_SCHEMA.SYSTEM_VARIABLES WHERE
  VARIABLE_NAME LIKE 'max_error_count' OR
  VARIABLE_NAME LIKE 'innodb_sync_spin_loops';
+--------------------------+---------------+--------------+
| VARIABLE_NAME            | SESSION_VALUE | GLOBAL_VALUE |
+--------------------------+---------------+--------------+
| MAX_ERROR_COUNT          | 64            | 128          |
| INNODB_SYNC_SPIN_LOOPS   | NULL          | 30           |
+--------------------------+---------------+--------------+

SET GLOBAL max_error_count=128;

SHOW VARIABLES LIKE 'max_error_count';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| max_error_count | 64    |
+-----------------+-------+

SHOW GLOBAL VARIABLES LIKE 'max_error_count';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| max_error_count | 128   |
+-----------------+-------+
```

Because the following variable only has a global scope, the global value is returned even when specifying SESSION (in this case by default):

```
SHOW VARIABLES LIKE 'innodb_sync_spin_loops';
+------------------------+-------+
| Variable_name          | Value |
+------------------------+-------+
| innodb_sync_spin_loops | 30    |
+------------------------+-------+
```

1.1.1.2.8.57 SHOW WARNINGS

Syntax

```
SHOW WARNINGS [LIMIT [offset,] row_count]
SHOW ERRORS [LIMIT row_count OFFSET offset]
SHOW COUNT(*) WARNINGS
```

Contents

Description

`SHOW WARNINGS` shows the error, warning, and note messages that resulted from the last statement that generated messages in the current session. It shows nothing if the last statement used a table and generated no messages. (That is, a statement that uses a table but generates no messages clears the message list.) Statements that do not use tables and do not generate messages have no effect on the message list.

A note is different to a warning in that it only appears if the sql_notes variable is set to 1 (the default), and is not converted to an error if strict mode is enabled.

A related statement, `SHOW ERRORS`, shows only the errors.

The `SHOW COUNT(*) WARNINGS` statement displays the total number of errors, warnings, and notes. You can also retrieve this number from the warning_count variable:

```
SHOW COUNT(*) WARNINGS;
SELECT @@warning_count;
```

The value of warning_count might be greater than the number of messages displayed by `SHOW WARNINGS` if the max_error_count system variable is set so low that not all messages are stored.

The `LIMIT` clause has the same syntax as for the `SELECT statement`.

`SHOW WARNINGS` can be used after EXPLAIN EXTENDED to see how a query is internally rewritten by MariaDB.

If the sql_notes server variable is set to 1, Notes are included in the output of `SHOW WARNINGS`; if it is set to 0, this statement will not show (or count) Notes.

The results of `SHOW WARNINGS` and `SHOW COUNT(*) WARNINGS` are directly sent to the client. If you need to access those information in a stored program, you can use the GET DIAGNOSTICS statement instead.

For a list of MariaDB error codes, see MariaDB Error Codes.

The mariadb client also has a number of options related to warnings. The `\W` command will show warnings after every statement, while `\w` will disable this. Starting the client with the `--show-warnings` option will show warnings after every statement.

MariaDB implements a stored routine error stack trace. `SHOW WARNINGS` can also be used to show more information. See the example below.

Examples

```
SELECT 1/0;
+------+
| 1/0  |
+------+
| NULL |
+------+

SHOW COUNT(*) WARNINGS;
+--------------------------+
| @@session.warning_count |
+--------------------------+
|                        1 |
+--------------------------+

SHOW WARNINGS;
+---------+------+---------------+
| Level   | Code | Message       |
+---------+------+---------------+
| Warning | 1365 | Division by 0 |
+---------+------+---------------+
```

Stack Trace

Displaying a stack trace:

```
DELIMITER $$
CREATE OR REPLACE PROCEDURE p1()
  BEGIN
    DECLARE c CURSOR FOR SELECT * FROM not_existing;
    OPEN c;
    CLOSE c;
  END;
$$
CREATE OR REPLACE PROCEDURE p2()
  BEGIN
    CALL p1;
  END;
$$
DELIMITER ;
CALL p2;
ERROR 1146 (42S02): Table 'test.not_existing' doesn't exist

SHOW WARNINGS;
+-------+------+----------------------------------------+
| Level | Code | Message                                |
+-------+------+----------------------------------------+
| Error | 1146 | Table 'test.not_existing' doesn't exist |
| Note  | 4091 | At line 6 in test.p1                   |
| Note  | 4091 | At line 4 in test.p2                   |
+-------+------+----------------------------------------+
```

`SHOW WARNINGS` displays a stack trace, showing where the error actually happened:

- Line 4 in test.p1 is the OPEN command which actually raised the error
- Line 3 in test.p2 is the CALL statement, calling p1 from p2.

1.1.1.2.8.58 SHOW WSREP_MEMBERSHIP

`SHOW WSREP_MEMBERSHIP` is part of the `WSREP_INFO` plugin.

Syntax

```
SHOW WSREP_MEMBERSHIP
```

Description

The `SHOW WSREP_MEMBERSHIP` statement returns Galera node cluster membership information. It returns the same information as found in the `information_schema.WSREP_MEMBERSHIP` table. Only users with the `SUPER` privilege can access this information.

Examples

```
SHOW WSREP_MEMBERSHIP;
+-------+--------------------------------------+----------+-----------------+
| Index | Uuid                                 | Name     | Address         |
+-------+--------------------------------------+----------+-----------------+
|     0 | 19058073-8940-11e4-8570-16af7bf8fced | my_node1 | 10.0.2.15:16001 |
|     1 | 19f2b0e0-8942-11e4-9cb8-b39e8ee0b5dd | my_node3 | 10.0.2.15:16003 |
|     2 | d85e62db-8941-11e4-b1ef-4bc9980e476d | my_node2 | 10.0.2.15:16002 |
+-------+--------------------------------------+----------+-----------------+
```

1.1.1.2.8.59 SHOW WSREP_STATUS

`SHOW WSREP_STATUS` is part of the `WSREP_INFO` plugin.

Syntax

```
SHOW WSREP_STATUS
```

Description

The `SHOW WSREP_STATUS` statement returns Galera node and cluster status information. It returns the same information as found in the `information_schema.WSREP_STATUS` table. Only users with the `SUPER` privilege can access this information.

Examples

```
SHOW WSREP_STATUS;
+------------+-------------+----------------+--------------+
| Node_Index | Node_Status | Cluster_Status | Cluster_Size |
+------------+-------------+----------------+--------------+
|          0 | Synced      | Primary        |            3 |
+------------+-------------+----------------+--------------+
```

1.1.1.2.9 System Tables

Information Schema
Articles about the Information Schema

Performance Schema
Monitoring server performance.

The mysql Database Tables
mysql database tables.

Sys Schema
Collection of views, functions and procedures to help administrators get in...

mariadb_schema
mariadb_schema is used to enforce MariaDB native types independent of SQL_MODE.

Writing Logs Into Tables
The general query log and the slow query log can be written into system tables

1.1.1.2.9.1 Information Schema

Articles about the Information Schema

Information Schema Tables
Tables in the Information_Schema database

Extended Show
Extended SHOW with WHERE and LIKE.

TIME_MS column in INFORMATION_SCHEMA.PROCESSLIST
Microseconds in the INFORMATION_SCHEMA.PROCESSLIST table

There are 1 related questions.

1.1.1.2.9.1.1 Information Schema Tables

Information Schema InnoDB Tables
All InnoDB-specific Information Schema tables.

Information Schema MyRocks Tables

List of Information Schema tables specifically related to MyRocks.

Information Schema XtraDB Tables

All XtraDB-specific Information Schema tables.

ColumnStore Information Schema Tables

ColumnStore-related Information Schema tables

Information Schema ALL_PLUGINS Table

Information about server plugins, whether installed or not.

Information Schema APPLICABLE_ROLES Table

Roles available to be used.

Information Schema CHARACTER_SETS Table

Supported character sets.

Information Schema CHECK_CONSTRAINTS Table

Supported check constraints.

Information Schema CLIENT_STATISTICS Table

Statistics about client connections.

Information Schema COLLATION_CHARACTER_SET_APPLICABILITY Table

Collations and associated character sets

Information Schema COLLATIONS Table

Supported collations.

Information Schema COLUMN_PRIVILEGES Table

Column privileges

Information Schema COLUMNS Table

Information about table fields.

Information Schema DISKS Table

Plugin that allows the disk space situation to be monitored.

Information Schema ENABLED_ROLES Table

Enabled roles for the current session.

Information Schema ENGINES Table

Storage engine information.

Information Schema EVENTS Table

Server event information

Information Schema FEEDBACK Table

Contents submitted by the Feedback Plugin

Information Schema FILES Table

The FILES tables is unused in MariaDB.

Information Schema GEOMETRY_COLUMNS Table

Support for Spatial Reference systems for GIS data

Information Schema GLOBAL_STATUS and SESSION_STATUS Tables

Global and session status variables

Information Schema GLOBAL_VARIABLES and SESSION_VARIABLES Tables

Global and session system variables

Information Schema INDEX_STATISTICS Table

Statistics on index usage

Information Schema KEY_CACHES Table

Segmented key cache statistics.

Information Schema KEY_COLUMN_USAGE Table

Key columns that have constraints.

Information Schema KEY_PERIOD_USAGE Table

Information about Application-Time Periods.

Information Schema KEYWORDS Table

MariaDB keywords.

Information Schema LOCALES Table

Compiled-in server locales.

Information Schema METADATA_LOCK_INFO Table

Active metadata locks.

Information Schema MROONGA_STATS Table

Mroonga activities statistics.

Information Schema OPTIMIZER_TRACE Table

Contains Optimizer Trace information.

Information Schema PARAMETERS Table

Information about stored procedures and stored functions parameters.

Information Schema PARTITIONS Table

Table partition information.

Information Schema PERIODS Table

Information about application time periods.

Information Schema PLUGINS Table

Information Schema table containing information on plugins installed on a server.

Information Schema PROCESSLIST Table

Thread information.

Information Schema PROFILING Table

Statement resource usage

Information Schema QUERY_CACHE_INFO Table

View the contents of the query cache.

Information Schema QUERY_RESPONSE_TIME Table

Query time information.

Information Schema REFERENTIAL_CONSTRAINTS Table

Foreign key information

Information Schema ROUTINES Table

Stored procedures and stored functions information

Information Schema SCHEMA_PRIVILEGES Table

Database privilege information

Information Schema SCHEMATA Table

Information about databases.

Information Schema SPATIAL_REF_SYS Table

Information on each spatial reference system used in the database.

Information Schema SPIDER_ALLOC_MEM Table

Information about Spider's memory usage.

Information Schema SPIDER_WRAPPER_PROTOCOLS Table

Installed along with the Spider storage engine.

Information Schema SQL_FUNCTIONS Table

Functions in MariaDB.

Information Schema STATISTICS Table

Table index information.

Information Schema SYSTEM_VARIABLES Table

Current global and session values and various metadata of all system variables.

Information Schema TABLE_CONSTRAINTS Table

Tables containing constraints.

Information Schema TABLE_PRIVILEGES Table

Table privileges

Information Schema TABLE_STATISTICS Table

Statistics on table usage.

Information Schema TABLES Table

Database table information.

Information Schema TABLESPACES Table

Information about active tablespaces.

Information Schema THREAD_POOL_GROUPS Table

Information Schema THREAD_POOL_GROUPS Table.

Information Schema THREAD_POOL_QUEUES Table

Information Schema THREAD_POOL_QUEUES Table.

Information Schema THREAD_POOL_STATS Table

Information Schema THREAD_POOL_STATS Table.

Information Schema THREAD_POOL_WAITS Table

Information Schema THREAD_POOL_WAITS Table.

Information Schema TRIGGERS Table

Information about triggers

Information Schema USER_PRIVILEGES Table

Global user privilege information derived from the mysql.global_priv grant table

Information Schema USER_STATISTICS Table

User activity

Information Schema USER_VARIABLES Table

User-defined variable information.

Information Schema VIEWS Table

Information about views.

Information Schema WSREP_MEMBERSHIP Table

Galera node cluster membership information.

Information Schema WSREP_STATUS Table

Galera node cluster status information.

There are 3 related questions.

1.1.1.2.9.1.1.1 Information Schema InnoDB Tables

List of Information Schema tables specifically related to InnoDB. Tables that are specific to XtraDB shares with InnoDB are listed in Information Schema XtraDB Tables.

Information Schema INNODB_BUFFER_PAGE Table

Buffer pool page information.

Information Schema INNODB_BUFFER_PAGE_LRU Table

Buffer pool pages and their eviction order.

Information Schema INNODB_BUFFER_POOL_PAGES Table

XtraDB buffer pool page information.

Information Schema INNODB_BUFFER_POOL_PAGES_BLOB Table

XtraDB buffer pool blob pages.

Information Schema INNODB_BUFFER_POOL_PAGES_INDEX Table

XtraDB buffer pool index pages.

Information Schema INNODB_BUFFER_POOL_STATS Table

InnoDB buffer pool information.

Information Schema INNODB_CHANGED_PAGES Table

Modified pages from the bitmap file data.

Information Schema INNODB_CMP and INNODB_CMP_RESET Tables

XtraDB/InnoDB compression performances with different page sizes.

Information Schema INNODB_CMPMEM and INNODB_CMPMEM_RESET Tables

Number of InnoDB compressed pages of different page sizes.

Information Schema INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET Tables

XtraDB/InnoDB compression performances for different indexes and tables.

Information Schema INNODB_FT_BEING_DELETED Table

Fulltext being deleted.

Information Schema INNODB_FT_CONFIG Table

InnoDB fulltext metadata.

Information Schema INNODB_FT_DEFAULT_STOPWORD Table

Default InnoDB stopwords.

Information Schema INNODB_FT_DELETED Table

Deleted InnoDB fulltext rows.

Information Schema INNODB_FT_INDEX_CACHE Table

Newly added fulltext row information.

Information Schema INNODB_FT_INDEX_TABLE Table

InnoDB fulltext information.

Information Schema INNODB_LOCK_WAITS Table

Blocked InnoDB transactions.

Information Schema INNODB_LOCKS Table

InnoDB lock information.

Information Schema INNODB_METRICS Table

InnoDB performance metrics.

Information Schema INNODB_MUTEXES Table

Monitor mutex waits.

Information Schema INNODB_SYS_COLUMNS Table

InnoDB column information.

Information Schema INNODB_SYS_DATAFILES Table

InnoDB tablespace paths.

Information Schema INNODB_SYS_FIELDS Table

Fields part of an InnoDB index.

Information Schema INNODB_SYS_FOREIGN Table

InnoDB foreign key information.

Information Schema INNODB_SYS_FOREIGN_COLS Table

Foreign key column information.

Information Schema INNODB_SYS_INDEXES Table

InnoDB index information.

Information Schema INNODB_SYS_SEMAPHORE_WAITS Table

Information about current semaphore waits.

Information Schema INNODB_SYS_TABLES Table

InnoDB table information.

Information Schema INNODB_SYS_TABLESPACES Table

InnoDB tablespace information.

Information Schema INNODB_SYS_TABLESTATS Table

InnoDB status for high-level performance monitoring.

Information Schema INNODB_SYS_VIRTUAL Table

Information about base columns of virtual columns.

Information Schema INNODB_TABLESPACES_ENCRYPTION Table

Encryption metadata for InnoDB tablespaces.

Information Schema INNODB_TABLESPACES_SCRUBBING Table

Data scrubbing information.

Information Schema INNODB_TRX Table

Currently-executing InnoDB locks.

Information Schema TEMP_TABLES_INFO Table

Information about active InnoDB temporary tables.

1.1.1.2.9.1.1.1.1 Information Schema INNODB_BUFFER_PAGE Table

The Information Schema `INNODB_BUFFER_PAGE` table contains information about pages in the buffer pool.

The `PROCESS` privilege is required to view the table.

It has the following columns:

Column	Description
`POOL_ID`	Buffer Pool identifier. From MariaDB 10.5.1 returns a value of 0, since multiple InnoDB buffer pool instances has been removed.
`BLOCK_ID`	Buffer Pool Block identifier.
`SPACE`	Tablespace identifier. Matches the `SPACE` value in the `INNODB_SYS_TABLES` table.
`PAGE_NUMBER`	Buffer pool page number.
`PAGE_TYPE`	Page type; one of `allocated` (newly-allocated page), `index` (B-tree node), `undo_log` (undo log page), `inode` (index node), `ibuf_free_list` (insert buffer free list), `ibuf_bitmap` (insert buffer bitmap), `system` (system page), `trx_system` (transaction system data), `file_space_header` (file space header), `extent_descriptor` (extent descriptor page), `blob` (uncompressed blob page), `compressed_blob` (first compressed blob page), `compressed_blob2` (subsequent compressed blob page) or `unknown`.
`FLUSH_TYPE`	Flush type.
`FIX_COUNT`	Count of the threads using this block in the buffer pool. When it is zero, the block can be evicted from the buffer pool.
`IS_HASHED`	Whether or not a hash index has been built on this page.
`NEWEST_MODIFICATION`	Most recent modification's Log Sequence Number.
`OLDEST_MODIFICATION`	Oldest modification's Log Sequence Number.
`ACCESS_TIME`	Abstract number representing the time the page was first accessed.
`TABLE_NAME`	Table that the page belongs to.
`INDEX_NAME`	Index that the page belongs to, either a clustered index or a secondary index.
`NUMBER_RECORDS`	Number of records the page contains.
`DATA_SIZE`	Size in bytes of all the records contained in the page.
`COMPRESSED_SIZE`	Compressed size in bytes of the page, or `NULL` for pages that aren't compressed.
`PAGE_STATE`	Page state; one of `FILE_PAGE` (page from a file) or `MEMORY` (page from an in-memory object) for valid data, or one of `NULL`, `READY_FOR_USE`, `NOT_USED`, `REMOVE_HASH`.
`IO_FIX`	Whether there is I/O pending for the page; one of `IO_NONE` (no pending I/O), `IO_READ` (read pending), `IO_WRITE` (write pending).
`IS_OLD`	Whether the page is old or not.
`FREE_PAGE_CLOCK`	Freed_page_clock counter, which tracks the number of blocks removed from the end of the least recently used (LRU) list, at the time the block was last placed at the head of the list.

The related INFORMATION_SCHEMA.INNODB_BUFFER_PAGE_LRU table contains the same information, but with an LRU (least recently used) position rather than block id.

Examples

```
DESC information_schema.innodb_buffer_page;
+--------------------+---------------------+------+-----+---------+-------+
| Field              | Type                | Null | Key | Default | Extra |
+--------------------+---------------------+------+-----+---------+-------+
| POOL_ID            | bigint(21) unsigned | NO   |     | 0       |       |
| BLOCK_ID           | bigint(21) unsigned | NO   |     | 0       |       |
| SPACE              | bigint(21) unsigned | NO   |     | 0       |       |
| PAGE_NUMBER        | bigint(21) unsigned | NO   |     | 0       |       |
| PAGE_TYPE          | varchar(64)         | YES  |     | NULL    |       |
| FLUSH_TYPE         | bigint(21) unsigned | NO   |     | 0       |       |
| FIX_COUNT          | bigint(21) unsigned | NO   |     | 0       |       |
| IS_HASHED          | varchar(3)          | YES  |     | NULL    |       |
| NEWEST_MODIFICATION | bigint(21) unsigned | NO   |     | 0       |       |
| OLDEST_MODIFICATION | bigint(21) unsigned | NO   |     | 0       |       |
| ACCESS_TIME        | bigint(21) unsigned | NO   |     | 0       |       |
| TABLE_NAME         | varchar(1024)       | YES  |     | NULL    |       |
| INDEX_NAME         | varchar(1024)       | YES  |     | NULL    |       |
| NUMBER_RECORDS     | bigint(21) unsigned | NO   |     | 0       |       |
| DATA_SIZE          | bigint(21) unsigned | NO   |     | 0       |       |
| COMPRESSED_SIZE    | bigint(21) unsigned | NO   |     | 0       |       |
| PAGE_STATE         | varchar(64)         | YES  |     | NULL    |       |
| IO_FIX             | varchar(64)         | YES  |     | NULL    |       |
| IS_OLD             | varchar(3)          | YES  |     | NULL    |       |
| FREE_PAGE_CLOCK    | bigint(21) unsigned | NO   |     | 0       |       |
+--------------------+---------------------+------+-----+---------+-------+
```

```
SELECT * FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE\G
...
*************************** 6. row ***************************
            POOL_ID: 0
          BLOCK_ID: 5
             SPACE: 0
       PAGE_NUMBER: 11
         PAGE_TYPE: INDEX
        FLUSH_TYPE: 1
         FIX_COUNT: 0
         IS_HASHED: NO
NEWEST_MODIFICATION: 2046835
OLDEST_MODIFICATION: 0
       ACCESS_TIME: 2585566280
        TABLE_NAME: `SYS_INDEXES`
        INDEX_NAME: CLUST_IND
    NUMBER_RECORDS: 57
         DATA_SIZE: 4016
   COMPRESSED_SIZE: 0
        PAGE_STATE: FILE_PAGE
            IO_FIX: IO_NONE
            IS_OLD: NO
   FREE_PAGE_CLOCK: 0
...
```

1.1.1.2.9.1.1.1.2 Information Schema INNODB_BUFFER_PAGE_LRU Table

The Information Schema `INNODB_BUFFER_PAGE_LRU` table contains information about pages in the buffer pool and how they are ordered for eviction purposes.

The `PROCESS` privilege is required to view the table.

It has the following columns:

Column	Description
`POOL_ID`	Buffer Pool identifier. From MariaDB 10.5.1 returns a value of 0, since multiple InnoDB buffer pool instances has been removed.
`LRU_POSITION`	LRU (Least recently-used), for determining eviction order from the buffer pool.
`SPACE`	Tablespace identifier. Matches the `SPACE` value on the `INNODB_SYS_TABLES` table.

PAGE_NUMBER	Buffer pool page number.
PAGE_TYPE	Page type; one of `allocated` (newly-allocated page), `index` (B-tree node), `undo_log` (undo log page), `inode` (index node), `ibuf_free_list` (insert buffer free list), `ibuf_bitmap` (insert buffer bitmap), `system` (system page), `trx_system` (transaction system data), `file_space_header` (file space header), `extent_descriptor` (extent descriptor page), `blob` (uncompressed blob page), `compressed_blob` (first compressed blob page), `compressed_blob2` (subsequent compressed blob page) or `unknown` .
FLUSH_TYPE	Flush type.
FIX_COUNT	Count of the threads using this block in the buffer pool. When it is zero, the block can be evicted from the buffer pool.
IS_HASHED	Whether or not a hash index has been built on this page.
NEWEST_MODIFICATION	Most recent modification's Log Sequence Number.
OLDEST_MODIFICATION	Oldest modification's Log Sequence Number.
ACCESS_TIME	Abstract number representing the time the page was first accessed.
TABLE_NAME	Table that the page belongs to.
INDEX_NAME	Index that the page belongs to, either a clustered index or a secondary index.
NUMBER_RECORDS	Number of records the page contains.
DATA_SIZE	Size in bytes of all the records contained in the page.
COMPRESSED_SIZE	Compressed size in bytes of the page, or `NULL` for pages that aren't compressed.
PAGE_STATE	Page state; one of `FILE_PAGE` (page from a file) or `MEMORY` (page from an in-memory object) for valid data, or one of `NULL` , `READY_FOR_USE` , `NOT_USED` , `REMOVE_HASH` .
IO_FIX	Whether there is I/O pending for the page; one of `IO_NONE` (no pending I/O), `IO_READ` (read pending), `IO_WRITE` (write pending).
IS_OLD	Whether the page is old or not.
FREE_PAGE_CLOCK	Freed_page_clock counter, which tracks the number of blocks removed from the end of the LRU list, at the time the block was last placed at the head of the list.

The related INFORMATION_SCHEMA.INNODB_BUFFER_PAGE table contains the same information, but with a block id rather than LRU position.

Example

```
DESC information_schema.innodb_buffer_page_lru;
+---------------------+---------------------+------+-----+---------+-------+
| Field               | Type                | Null | Key | Default | Extra |
+---------------------+---------------------+------+-----+---------+-------+
| POOL_ID             | bigint(21) unsigned | NO   |     | 0       |       |
| LRU_POSITION        | bigint(21) unsigned | NO   |     | 0       |       |
| SPACE               | bigint(21) unsigned | NO   |     | 0       |       |
| PAGE_NUMBER         | bigint(21) unsigned | NO   |     | 0       |       |
| PAGE_TYPE           | varchar(64)         | YES  |     | NULL    |       |
| FLUSH_TYPE          | bigint(21) unsigned | NO   |     | 0       |       |
| FIX_COUNT           | bigint(21) unsigned | NO   |     | 0       |       |
| IS_HASHED           | varchar(3)          | YES  |     | NULL    |       |
| NEWEST_MODIFICATION | bigint(21) unsigned | NO   |     | 0       |       |
| OLDEST_MODIFICATION | bigint(21) unsigned | NO   |     | 0       |       |
| ACCESS_TIME         | bigint(21) unsigned | NO   |     | 0       |       |
| TABLE_NAME          | varchar(1024)       | YES  |     | NULL    |       |
| INDEX_NAME          | varchar(1024)       | YES  |     | NULL    |       |
| NUMBER_RECORDS      | bigint(21) unsigned | NO   |     | 0       |       |
| DATA_SIZE           | bigint(21) unsigned | NO   |     | 0       |       |
| COMPRESSED_SIZE     | bigint(21) unsigned | NO   |     | 0       |       |
| COMPRESSED          | varchar(3)          | YES  |     | NULL    |       |
| IO_FIX              | varchar(64)         | YES  |     | NULL    |       |
| IS_OLD              | varchar(3)          | YES  |     | NULL    |       |
| FREE_PAGE_CLOCK     | bigint(21) unsigned | NO   |     | 0       |       |
+---------------------+---------------------+------+-----+---------+-------+
```

```
SELECT * FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE_LRU\G
...
*************************** 6. row ***************************
            POOL_ID: 0
       LRU_POSITION: 5
              SPACE: 0
        PAGE_NUMBER: 11
          PAGE_TYPE: INDEX
         FLUSH_TYPE: 1
          FIX_COUNT: 0
          IS_HASHED: NO
NEWEST_MODIFICATION: 2046835
OLDEST_MODIFICATION: 0
        ACCESS_TIME: 2585566280
         TABLE_NAME: `SYS_INDEXES`
         INDEX_NAME: CLUST_IND
     NUMBER_RECORDS: 57
          DATA_SIZE: 4016
    COMPRESSED_SIZE: 0
         COMPRESSED: NO
             IO_FIX: IO_NONE
             IS_OLD: NO
    FREE_PAGE_CLOCK: 0
...
```

1.1.1.2.9.1.1.1.3 Information Schema INNODB_BUFFER_POOL_PAGES Table

The Information Schema `INNODB_BUFFER_POOL_PAGES` table is a Percona enhancement, and is only available for XtraDB, not InnoDB (see XtraDB and InnoDB). It contains a record for each page in the buffer pool.

It has the following columns:

Column	Description
`PAGE_TYPE`	Type of page; one of `index`, `undo_log`, `inode`, `ibuf_free_list`, `allocated`, `bitmap`, `sys`, `trx_sys`, `fsp_hdr`, `xdes`, `blob`, `zblob`, `zblob2` and `unknown`.
`SPACE_ID`	Tablespace ID.
`PAGE_NO`	Page offset within tablespace.
`LRU_POSITION`	Page position in the LRU (least-recently-used) list.
`FIX_COUNT`	Page reference count, incremented each time the page is accessed. `0` if the page is not currently being accessed.
`FLUSH_TYPE`	Flush type of the most recent flush. `0` (LRU), `2` (flush_list)

1.1.1.2.9.1.1.1.4 Information Schema INNODB_BUFFER_POOL_PAGES_BLOB Table

The Information Schema `INNODB_BUFFER_POOL_PAGES_BLOB` table is a Percona enchancement, and is only available for XtraDB, not InnoDB (see XtraDB and InnoDB). It contains information about buffer pool blob pages.

It has the following columns:

Column	Description
`SPACE_ID`	Tablespace ID.
`PAGE_NO`	Page offset within tablespace.
`COMPRESSED`	`1` if the blob contains compressed data, `0` if not.
`PART_LEN`	Page data length.

NEXT_PAGE_NO	Next page number.
LRU_POSITION	Page position in the LRU (least-recently-used) list.
FIX_COUNT	Page reference count, incremented each time the page is accessed. 0 if the page is not currently being accessed.
FLUSH_TYPE	Flush type of the most recent flush. 0 (LRU), 2 (flush_list)

1.1.1.2.9.1.1.1.5 Information Schema INNODB_BUFFER_POOL_PAGES_INDEX Table

The Information Schema INNODB_BUFFER_POOL_PAGES table is a Percona enhancement, and is only available for XtraDB, not InnoDB (see XtraDB and InnoDB). It contains information about buffer pool index pages.

It has the following columns:

Column	Description
INDEX_ID	Index name
SPACE_ID	Tablespace ID
PAGE_NO	Page offset within tablespace.
N_RECS	Number of user records on the page.
DATA_SIZE	Total data size in bytes of records in the page.
HASHED	1 if the block is in the adaptive hash index, 0 if not.
ACCESS_TIME	Page's last access time.
MODIFIED	1 if the page has been modified since being loaded, 0 if not.
DIRTY	1 if the page has been modified since it was last flushed, 0 if not
OLD	1 if the page in the in the *old* blocks of the LRU (least-recently-used) list, 0 if not.
LRU_POSITION	Position in the LRU (least-recently-used) list.
FIX_COUNT	Page reference count, incremented each time the page is accessed. 0 if the page is not currently being accessed.
FLUSH_TYPE	Flush type of the most recent flush. 0 (LRU), 2 (flush_list)

1.1.1.2.9.1.1.1.6 Information Schema INNODB_BUFFER_POOL_STATS Table

The Information Schema INNODB_BUFFER_POOL_STATS table contains information about pages in the buffer pool, similar to what is returned with the SHOW ENGINE INNODB STATUS statement.

The PROCESS privilege is required to view the table.

It has the following columns:

Column	Description
POOL_ID	Buffer Pool identifier. From MariaDB 10.5.1 returns a value of 0, since multiple InnoDB buffer pool instances has been removed.
POOL_SIZE	Size in pages of the buffer pool.
FREE_BUFFERS	Number of free pages in the buffer pool.
DATABASE_PAGES	Total number of pages in the buffer pool.
OLD_DATABASE_PAGES	Number of pages in the *old* sublist.

MODIFIED_DATABASE_PAGES	Number of dirty pages.
PENDING_DECOMPRESS	Number of pages pending decompression.
PENDING_READS	Pending buffer pool level reads.
PENDING_FLUSH_LRU	Number of pages in the LRU pending flush.
PENDING_FLUSH_LIST	Number of pages in the flush list pending flush.
PAGES_MADE_YOUNG	Pages moved from the *old* sublist to the *new* sublist.
PAGES_NOT_MADE_YOUNG	Pages that have remained in the *old* sublist without moving to the *new* sublist.
PAGES_MADE_YOUNG_RATE	Hits that cause blocks to move to the top of the *new* sublist.
PAGES_MADE_NOT_YOUNG_RATE	Hits that do not cause blocks to move to the top of the *new* sublist due to the `innodb_old_blocks` delay not being met.
NUMBER_PAGES_READ	Number of pages read.
NUMBER_PAGES_CREATED	Number of pages created.
NUMBER_PAGES_WRITTEN	Number of pages written.
PAGES_READ_RATE	Number of pages read since the last printout divided by the time elapsed, giving pages read per second.
PAGES_CREATE_RATE	Number of pages created since the last printout divided by the time elapsed, giving pages created per second.
PAGES_WRITTEN_RATE	Number of pages written since the last printout divided by the time elapsed, giving pages written per second.
NUMBER_PAGES_GET	Number of logical read requests.
HIT_RATE	Buffer pool hit rate.
YOUNG_MAKE_PER_THOUSAND_GETS	For every 1000 gets, the number of pages made young.
NOT_YOUNG_MAKE_PER_THOUSAND_GETS	For every 1000 gets, the number of pages not made young.
NUMBER_PAGES_READ_AHEAD	Number of pages read ahead.
NUMBER_READ_AHEAD_EVICTED	Number of pages read ahead by the read-ahead thread that were later evicted without being accessed by any queries.
READ_AHEAD_RATE	Pages read ahead since the last printout divided by the time elapsed, giving read-ahead rate per second.
READ_AHEAD_EVICTED_RATE	Read-ahead pages not accessed since the last printout divided by time elapsed, giving the number of read-ahead pages evicted without access per second.
LRU_IO_TOTAL	Total least-recently used I/O.
LRU_IO_CURRENT	Least-recently used I/O for the current interval.
UNCOMPRESS_TOTAL	Total number of pages decompressed.
UNCOMPRESS_CURRENT	Number of pages decompressed in the current interval

Examples

```
DESC information_schema.innodb_buffer_pool_stats;
+---------------------------------+---------------------+------+-----+---------+-------+
| Field                           | Type                | Null | Key | Default | Extra |
+---------------------------------+---------------------+------+-----+---------+-------+
| POOL_ID                         | bigint(21) unsigned | NO   |     | 0       |       |
| POOL_SIZE                       | bigint(21) unsigned | NO   |     | 0       |       |
| FREE_BUFFERS                    | bigint(21) unsigned | NO   |     | 0       |       |
| DATABASE_PAGES                  | bigint(21) unsigned | NO   |     | 0       |       |
| OLD_DATABASE_PAGES              | bigint(21) unsigned | NO   |     | 0       |       |
| MODIFIED_DATABASE_PAGES         | bigint(21) unsigned | NO   |     | 0       |       |
| PENDING_DECOMPRESS              | bigint(21) unsigned | NO   |     | 0       |       |
| PENDING_READS                   | bigint(21) unsigned | NO   |     | 0       |       |
| PENDING_FLUSH_LRU               | bigint(21) unsigned | NO   |     | 0       |       |
| PENDING_FLUSH_LIST              | bigint(21) unsigned | NO   |     | 0       |       |
| PAGES_MADE_YOUNG                | bigint(21) unsigned | NO   |     | 0       |       |
| PAGES_NOT_MADE_YOUNG            | bigint(21) unsigned | NO   |     | 0       |       |
| PAGES_MADE_YOUNG_RATE           | double              | NO   |     | 0       |       |
| PAGES_MADE_NOT_YOUNG_RATE       | double              | NO   |     | 0       |       |
| NUMBER_PAGES_READ               | bigint(21) unsigned | NO   |     | 0       |       |
| NUMBER_PAGES_CREATED            | bigint(21) unsigned | NO   |     | 0       |       |
| NUMBER_PAGES_WRITTEN            | bigint(21) unsigned | NO   |     | 0       |       |
| PAGES_READ_RATE                 | double              | NO   |     | 0       |       |
| PAGES_CREATE_RATE               | double              | NO   |     | 0       |       |
| PAGES_WRITTEN_RATE              | double              | NO   |     | 0       |       |
| NUMBER_PAGES_GET                | bigint(21) unsigned | NO   |     | 0       |       |
| HIT_RATE                        | bigint(21) unsigned | NO   |     | 0       |       |
| YOUNG_MAKE_PER_THOUSAND_GETS    | bigint(21) unsigned | NO   |     | 0       |       |
| NOT_YOUNG_MAKE_PER_THOUSAND_GETS | bigint(21) unsigned | NO   |     | 0       |       |
| NUMBER_PAGES_READ_AHEAD         | bigint(21) unsigned | NO   |     | 0       |       |
| NUMBER_READ_AHEAD_EVICTED       | bigint(21) unsigned | NO   |     | 0       |       |
| READ_AHEAD_RATE                 | double              | NO   |     | 0       |       |
| READ_AHEAD_EVICTED_RATE         | double              | NO   |     | 0       |       |
| LRU_IO_TOTAL                    | bigint(21) unsigned | NO   |     | 0       |       |
| LRU_IO_CURRENT                  | bigint(21) unsigned | NO   |     | 0       |       |
| UNCOMPRESS_TOTAL                | bigint(21) unsigned | NO   |     | 0       |       |
| UNCOMPRESS_CURRENT              | bigint(21) unsigned | NO   |     | 0       |       |
+---------------------------------+---------------------+------+-----+---------+-------+
```

1.1.1.2.9.1.1.1.7 Information Schema INNODB_CHANGED_PAGES Table

The Information Schema `INNODB_CHANGED_PAGES` Table contains data about modified pages from the bitmap file. It is updated at checkpoints by the log tracking thread parsing the log, so does not contain real-time data.

The number of records is limited by the value of the innodb_max_changed_pages system variable.

The `PROCESS` privilege is required to view the table.

It has the following columns:

Column	Description
`SPACE_ID`	Modified page space id
`PAGE_ID`	Modified page id
`START_LSN`	Interval start after which page was changed (equal to checkpoint LSN)
`END_LSN`	Interval end before which page was changed (equal to checkpoint LSN)

1.1.1.2.9.1.1.1.8 Information Schema INNODB_CMP and INNODB_CMP_RESET Tables

The `INNODB_CMP` and `INNODB_CMP_RESET` tables contain status information on compression operations related to compressed XtraDB/InnoDB tables.

The `PROCESS` privilege is required to query this table.

These tables contain the following columns:

Column Name	Description
`PAGE_SIZE`	Compressed page size, in bytes. This value is unique in the table; other values are totals which refer to pages of this size.
`COMPRESS_OPS`	How many times a page of the size `PAGE_SIZE` has been compressed. This happens when a new page is created because the compression log runs out of space. This value includes both successful operations and *compression failures*.
`COMPRESS_OPS_OK`	How many times a page of the size `PAGE_SIZE` has been successfully compressed. This value should be as close as possible to `COMPRESS_OPS`. If it is notably lower, either avoid compressing some tables, or increase the `KEY_BLOCK_SIZE` for some compressed tables.
`COMPRESS_TIME`	Time (in seconds) spent to compress pages of the size `PAGE_SIZE`. This value includes time spent in *compression failures*.
`UNCOMPRESS_OPS`	How many times a page of the size `PAGE_SIZE` has been uncompressed. This happens when an uncompressed version of a page is created in the buffer pool, or when a *compression failure* occurs.
`UNCOMPRESS_TIME`	Time (in seconds) spent to uncompress pages of the size `PAGE_SIZE`.

These tables can be used to measure the effectiveness of XtraDB/InnoDB table compression. When you have to decide a value for `KEY_BLOCK_SIZE`, you can create more than one version of the table (one for each candidate value) and run a realistic workload on them. Then, these tables can be used to see how the operations performed with different page sizes.

`INNODB_CMP` and `INNODB_CMP_RESET` have the same columns and always contain the same values, but when `INNODB_CMP_RESET` is queried, both the tables are cleared. `INNODB_CMP_RESET` can be used, for example, if a script periodically logs the performances of compression in the last period of time. `INNODB_CMP` can be used to see the cumulated statistics.

Examples

```
SELECT * FROM information_schema.INNODB_CMP\G
*************************** 1. row ***************************
      page_size: 1024
   compress_ops: 0
compress_ops_ok: 0
  compress_time: 0
 uncompress_ops: 0
uncompress_time: 0
...
```

1.1.1.2.9.1.1.1.9 Information Schema INNODB_CMPMEM and INNODB_CMPMEM_RESET Tables

The `INNODB_CMPMEM` and `INNODB_CMPMEM_RESET` tables contain status information on compressed pages in the buffer pool (see InnoDB COMPRESSED format).

The PROCESS privilege is required to query this table.

These tables contain the following columns:

Column Name	Description
`PAGE_SIZE`	Compressed page size, in bytes. This value is unique in the table; other values are totals which refer to pages of this size.
`BUFFER_POOL_INSTANCE`	Buffer Pool identifier. From MariaDB 10.5.1 returns a value of 0, since multiple InnoDB buffer pool instances has been removed.
`PAGES_USED`	Number of pages of the size `PAGE_SIZE` which are currently in the buffer pool.

PAGES_FREE	Number of pages of the size PAGE_SIZE which are currently free, and thus are available for allocation. This value represents the buffer pool's fragmentation. A totally unfragmented buffer pool has at most 1 free page.
RELOCATION_OPS	How many times a page of the size PAGE_SIZE has been relocated. This happens when data exceeds a page (because a row must be copied into a new page) and when two pages are merged (because their data shrunk and can now be contained in one page).
RELOCATION_TIME	Time (in seconds) spent in relocation operations for pages of the size PAGE_SIZE . This column is reset when the INNODB_CMPMEM_RESET table is queried.

These tables can be used to measure the effectiveness of InnoDB table compression. When you have to decide a value for KEY_BLOCK_SIZE , you can create more than one version of the table (one for each candidate value) and run a realistic workload on them. Then, these tables can be used to see how the operations performed with different page sizes.

INNODB_CMPMEM and INNODB_CMPMEM_RESET have the same columns and always contain the same values, but when INNODB_CMPMEM_RESET is queried, the RELOCATION_TIME column from both the tables are cleared. INNODB_CMPMEM_RESET can be used, for example, if a script periodically logs the performances of compression in the last period of time. INNODB_CMPMEM can be used to see the cumulated statistics.

Example

```
SELECT * FROM information_schema.INNODB_CMPMEM\G
********************* 1. row *********************
            page_size: 1024
 buffer_pool_instance: 0
           pages_used: 0
           pages_free: 0
      reloacation_ops: 0
      relocation_time: 0
```

1.1.1.2.9.1.1.1.10 Information Schema INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET Tables

The INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET tables contain status information on compression operations related to compressed XtraDB/InnoDB tables, grouped by individual indexes. These tables are only populated if the innodb_cmp_per_index_enabled system variable is set to ON .

The PROCESS privilege is required to query this table.

These tables contains the following columns:

Column Name	Description
DATABASE_NAME	Database containing the index.
TABLE_NAME	Table containing the index.
INDEX_NAME	Other values are totals which refer to this index's compression.
COMPRESS_OPS	How many times a page of INDEX_NAME has been compressed. This happens when a new page is created because the compression log runs out of space. This value includes both successful operations and *compression failures*.
COMPRESS_OPS_OK	How many times a page of INDEX_NAME has been successfully compressed. This value should be as close as possible to COMPRESS_OPS . If it is notably lower, either avoid compressing some tables, or increase the KEY_BLOCK_SIZE for some compressed tables.
COMPRESS_TIME	Time (in seconds) spent to compress pages of the size PAGE_SIZE . This value includes time spent in *compression failures*.
UNCOMPRESS_OPS	How many times a page of INDEX_NAME has been uncompressed. This happens when an uncompressed version of a page is created in the buffer pool, or when a *compression failure* occurs.
UNCOMPRESS_TIME	Time (in seconds) spent to uncompress pages of INDEX_NAME .

These tables can be used to measure the effectiveness of XtraDB/InnoDB compression, per table or per index. The values

in these tables show which tables perform better with index compression, and which tables cause too many *compression failures* or perform too many compression/uncompression operations. When compression performs badly for a table, this might mean that you should change its `KEY_BLOCK_SIZE`, or that the table should not be compressed.

`INNODB_CMP_PER_INDEX` and `INNODB_CMP_PER_INDEX_RESET` have the same columns and always contain the same values, but when `INNODB_CMP_PER_INDEX_RESET` is queried, both the tables are cleared. `INNODB_CMP_PER_INDEX_RESET` can be used, for example, if a script periodically logs the performances of compression in the last period of time. `INNODB_CMP_PER_INDEX` can be used to see the cumulated statistics.

1.1.1.2.9.1.1.1.11 Information Schema INNODB_FT_BEING_DELETED Table

The Information Schema `INNODB_FT_BEING_DELETED` table is only used while document ID's in the related INNODB_FT_DELETED are being removed from an InnoDB fulltext index while an OPTIMIZE TABLE is underway. At all other times the table will be empty.

The `SUPER` privilege is required to view the table, and it also requires the innodb_ft_aux_table system variable to be set.

It has the following column:

Column	Description
`DOC_ID`	Document ID of the row being deleted. Either an underlying ID value, or a sequence value generated by InnoDB if no usable option exists.

1.1.1.2.9.1.1.1.12 Information Schema INNODB_FT_CONFIG Table

The Information Schema `INNODB_FT_CONFIG` table contains InnoDB fulltext index metadata.

The `SUPER` privilege is required to view the table, and it also requires the innodb_ft_aux_table system variable to be set.

It has the following columns:

Column	Description
`KEY`	Metadata item name.
`VALUE`	Associated value.

Example

```
SELECT * FROM INNODB_FT_CONFIG;
+---------------------------+-------+
| KEY                       | VALUE |
+---------------------------+-------+
| optimize_checkpoint_limit | 180   |
| synced_doc_id             | 6     |
| last_optimized_word       |       |
| deleted_doc_count         | 0     |
| total_word_count          |       |
| optimize_start_time       |       |
| optimize_end_time         |       |
| stopword_table_name       |       |
| use_stopword              | 1     |
| table_state               | 0     |
+---------------------------+-------+
```

1.1.1.2.9.1.1.1.13 Information Schema INNODB_FT_DEFAULT_STOPWORD Table

The Information Schema `INNODB_FT_DEFAULT_STOPWORD` table contains a list of default stopwords used when creating an InnoDB fulltext index.

The `PROCESS` privilege is required to view the table.

It has the following column:

Column	Description
`VALUE`	Default `stopword` for an InnoDB fulltext index. Setting either the innodb_ft_server_stopword_table or the innodb_ft_user_stopword_table system variable will override this.

Example

```
SELECT * FROM information_schema.INNODB_FT_DEFAULT_STOPWORD\G
*************************** 1. row ***************************
value: a
*************************** 2. row ***************************
value: about
*************************** 3. row ***************************
value: an
*************************** 4. row ***************************
value: are
...
*************************** 36. row ***************************
value: www
```

1.1.1.2.9.1.1.1.14 Information Schema INNODB_FT_DELETED Table

The Information Schema `INNODB_FT_DELETED` table contains rows that have been deleted from an InnoDB fulltext index. This information is then used to filter results on subsequent searches, removing the need to expensively reorganise the index each time a row is deleted.

The fulltext index is then only reorganized when an OPTIMIZE TABLE statement is underway. The related INNODB_FT_BEING_DELETED table contains rows being deleted while an `OPTIMIZE TABLE` is in the process of running.

The `SUPER` privilege is required to view the table, and it also requires the innodb_ft_aux_table system variable to be set.

It has the following column:

Column	Description
`DOC_ID`	Document ID of the deleted row deleted. Either an underlying ID value, or a sequence value generated by InnoDB if no usable option exists.

Example

```
SELECT * FROM INFORMATION_SCHEMA.INNODB_FT_DELETED;
+--------+
| DOC_ID |
+--------+
|      2 |
+--------+

DELETE FROM test.ft_innodb LIMIT 1;

SELECT * FROM INFORMATION_SCHEMA.INNODB_FT_DELETED;
+--------+
| DOC_ID |
+--------+
|      2 |
|      3 |
+--------+
```

1.1.1.2.9.1.1.1.15 Information Schema

INNODB_FT_INDEX_CACHE Table

The Information Schema `INNODB_FT_INDEX_CACHE` table contains information about rows that have recently been inserted into an InnoDB fulltext index. To avoid re-organizing the fulltext index each time a change is made, which would be very expensive, new changes are stored separately and only integrated when an OPTIMIZE TABLE is run.

The `SUPER` privilege is required to view the table, and it also requires the innodb_ft_aux_table system variable to be set.

It has the following columns:

Column	Description
`WORD`	Word from the text of a newly added row. Words can appear multiple times in the table, once per `DOC_ID` and `POSITION` combination.
`FIRST_DOC_ID`	First document ID where this word appears in the index.
`LAST_DOC_ID`	Last document ID where this word appears in the index.
`DOC_COUNT`	Number of rows containing this word in the index.
`DOC_ID`	Document ID of the newly added row, either an appropriate ID column or an internal InnoDB value.
`POSITION`	Position of this word instance within the `DOC_ID`, as an offset added to the previous `POSITION` instance.

Note that for `OPTIMIZE TABLE` to process InnoDB fulltext index data, the innodb_optimize_fulltext_only system variable needs to be set to `1`. When this is done, and an `OPTIMIZE TABLE` statement run, the `INNODB_FT_INDEX_CACHE` table will be emptied, and the INNODB_FT_INDEX_TABLE table will be updated.

Examples

```
SELECT * FROM INNODB_FT_INDEX_CACHE;
+------------+--------------+-------------+-----------+--------+----------+
| WORD       | FIRST_DOC_ID | LAST_DOC_ID | DOC_COUNT | DOC_ID | POSITION |
+------------+--------------+-------------+-----------+--------+----------+
| and        |            4 |           4 |         1 |      4 |        0 |
| arrived    |            4 |           4 |         1 |      4 |       20 |
| ate        |            1 |           1 |         1 |      1 |        4 |
| everybody  |            1 |           1 |         1 |      1 |        8 |
| goldilocks |            4 |           4 |         1 |      4 |        9 |
| hungry     |            3 |           3 |         1 |      3 |        8 |
| then       |            4 |           4 |         1 |      4 |        4 |
| wicked     |            2 |           2 |         1 |      2 |        4 |
| witch      |            2 |           2 |         1 |      2 |       11 |
+------------+--------------+-------------+-----------+--------+----------+
9 rows in set (0.00 sec)

INSERT INTO test.ft_innodb VALUES(3,'And she ate a pear');

SELECT * FROM INNODB_FT_INDEX_CACHE;
+------------+--------------+-------------+-----------+--------+----------+
| WORD       | FIRST_DOC_ID | LAST_DOC_ID | DOC_COUNT | DOC_ID | POSITION |
+------------+--------------+-------------+-----------+--------+----------+
| and        |            4 |           5 |         2 |      4 |        0 |
| and        |            4 |           5 |         2 |      5 |        0 |
| arrived    |            4 |           4 |         1 |      4 |       20 |
| ate        |            1 |           5 |         2 |      1 |        4 |
| ate        |            1 |           5 |         2 |      5 |        8 |
| everybody  |            1 |           1 |         1 |      1 |        8 |
| goldilocks |            4 |           4 |         1 |      4 |        9 |
| hungry     |            3 |           3 |         1 |      3 |        8 |
| pear       |            5 |           5 |         1 |      5 |       14 |
| she        |            5 |           5 |         1 |      5 |        4 |
| then       |            4 |           4 |         1 |      4 |        4 |
| wicked     |            2 |           2 |         1 |      2 |        4 |
| witch      |            2 |           2 |         1 |      2 |       11 |
+------------+--------------+-------------+-----------+--------+----------+
```

```
OPTIMIZE TABLE test.ft_innodb\G
*************************** 1. row ***************************
   Table: test.ft_innodb
      Op: optimize
Msg_type: note
Msg_text: Table does not support optimize, doing recreate + analyze instead
*************************** 2. row ***************************
   Table: test.ft_innodb
      Op: optimize
Msg_type: status
Msg_text: OK
2 rows in set (2.24 sec)

SELECT * FROM INNODB_FT_INDEX_CACHE;
+------------+--------------+-------------+-----------+--------+----------+
| WORD       | FIRST_DOC_ID | LAST_DOC_ID | DOC_COUNT | DOC_ID | POSITION |
+------------+--------------+-------------+-----------+--------+----------+
| and        |            4 |           5 |         2 |      4 |        0 |
| and        |            4 |           5 |         2 |      5 |        0 |
| arrived    |            4 |           4 |         1 |      4 |       20 |
| ate        |            1 |           5 |         2 |      1 |        4 |
| ate        |            1 |           5 |         2 |      5 |        8 |
| everybody  |            1 |           1 |         1 |      1 |        8 |
| goldilocks |            4 |           4 |         1 |      4 |        9 |
| hungry     |            3 |           3 |         1 |      3 |        8 |
| pear       |            5 |           5 |         1 |      5 |       14 |
| she        |            5 |           5 |         1 |      5 |        4 |
| then       |            4 |           4 |         1 |      4 |        4 |
| wicked     |            2 |           2 |         1 |      2 |        4 |
| witch      |            2 |           2 |         1 |      2 |       11 |
+------------+--------------+-------------+-----------+--------+----------+
13 rows in set (0.00 sec)
```

The `OPTIMIZE TABLE` statement has no effect, because the innodb_optimize_fulltext_only variable wasn't set:

```
SHOW VARIABLES LIKE 'innodb_optimize_fulltext_only';
+--------------------------------+-------+
| Variable_name                  | Value |
+--------------------------------+-------+
| innodb_optimize_fulltext_only | OFF   |
+--------------------------------+-------+

SET GLOBAL innodb_optimize_fulltext_only =1;

OPTIMIZE TABLE test.ft_innodb;
+----------------+----------+----------+----------+
| Table          | Op       | Msg_type | Msg_text |
+----------------+----------+----------+----------+
| test.ft_innodb | optimize | status   | OK       |
+----------------+----------+----------+----------+

SELECT * FROM INNODB_FT_INDEX_CACHE;
Empty set (0.00 sec)
```

1.1.1.2.9.1.1.1.16 Information Schema INNODB_FT_INDEX_TABLE Table

The Information Schema `INNODB_FT_INDEX_TABLE` table contains information about InnoDB fulltext indexes. To avoid re-organizing the fulltext index each time a change is made, which would be very expensive, new changes are stored separately and only integrated when an OPTIMIZE TABLE is run. See the INNODB_FT_INDEX_CACHE table.

The `SUPER` privilege is required to view the table, and it also requires the innodb_ft_aux_table system variable to be set.

It has the following columns:

Column	Description
`WORD`	Word from the text of a column with a fulltext index. Words can appear multiple times in the table, once per `DOC_ID` and `POSITION` combination.

FIRST_DOC_ID	First document ID where this word appears in the index.
LAST_DOC_ID	Last document ID where this word appears in the index.
DOC_COUNT	Number of rows containing this word in the index.
DOC_ID	Document ID of the newly added row, either an appropriate ID column or an internal InnoDB value.
POSITION	Position of this word instance within the DOC_ID , as an offset added to the previous POSITION instance.

Note that for OPTIMIZE TABLE to process InnoDB fulltext index data, the innodb_optimize_fulltext_only system variable needs to be set to 1 . When this is done, and an OPTIMIZE TABLE statement run, the INNODB_FT_INDEX_CACHE table will be emptied, and the INNODB_FT_INDEX_TABLE table will be updated.

Examples

```
SELECT * FROM INNODB_FT_INDEX_TABLE;
Empty set (0.00 sec)

SET GLOBAL innodb_optimize_fulltext_only =1;

OPTIMIZE TABLE test.ft_innodb;
+----------------+----------+----------+----------+
| Table          | Op       | Msg_type | Msg_text |
+----------------+----------+----------+----------+
| test.ft_innodb | optimize | status   | OK       |
+----------------+----------+----------+----------+

SELECT * FROM INNODB_FT_INDEX_TABLE;
+------------+--------------+-------------+-----------+--------+----------+
| WORD       | FIRST_DOC_ID | LAST_DOC_ID | DOC_COUNT | DOC_ID | POSITION |
+------------+--------------+-------------+-----------+--------+----------+
| and        |            4 |           5 |         2 |      4 |        0 |
| and        |            4 |           5 |         2 |      5 |        0 |
| arrived    |            4 |           4 |         1 |      4 |       20 |
| ate        |            1 |           5 |         2 |      1 |        4 |
| ate        |            1 |           5 |         2 |      5 |        8 |
| everybody  |            1 |           1 |         1 |      1 |        8 |
| goldilocks |            4 |           4 |         1 |      4 |        9 |
| hungry     |            3 |           3 |         1 |      3 |        8 |
| pear       |            5 |           5 |         1 |      5 |       14 |
| she        |            5 |           5 |         1 |      5 |        4 |
| then       |            4 |           4 |         1 |      4 |        4 |
| wicked     |            2 |           2 |         1 |      2 |        4 |
| witch      |            2 |           2 |         1 |      2 |       11 |
+------------+--------------+-------------+-----------+--------+----------+
```

1.1.1.2.9.1.1.1.17 Information Schema INNODB_LOCK_WAITS Table

The Information Schema INNODB_LOCK_WAITS table contains information about blocked InnoDB transactions. The PROCESS privilege is required to view the table.

It contains the following columns:

Column	Description
REQUESTING_TRX_ID	Requesting transaction ID from the INNODB_TRX table.
REQUESTED_LOCK_ID	Lock ID from the INNODB.LOCKS table for the waiting transaction.
BLOCKING_TRX_ID	Blocking transaction ID from the INNODB_TRX table.
BLOCKING_LOCK_ID	Lock ID from the INNODB.LOCKS table of a lock held by a transaction that is blocking another transaction.

The table is often used in conjunction with the INNODB_LOCKS and INNODB_TRX tables to diagnose problematic locks and transactions.

1.1.1.2.9.1.1.1.18 Information Schema INNODB_LOCKS Table

The Information Schema `INNODB_LOCKS` table stores information about locks that InnoDB transactions have requested but not yet acquired, or that are blocking another transaction.

It has the following columns:

Column	Description
`LOCK_ID`	Lock ID number - the format is not fixed, so do not rely upon the number for information.
`LOCK_TRX_ID`	Lock's transaction ID. Matches the INNODB_TRX.TRX_ID column.
`LOCK_MODE`	Lock mode. One of `S` (shared), `X` (exclusive), `IS` (intention shared), `IX` (intention exclusive row lock), `S_GAP` (shared gap lock), `X_GAP` (exclusive gap lock), `IS_GAP` (intention shared gap lock), `IX_GAP` (intention exclusive gap lock) or `AUTO_INC` (auto-increment table level lock).
`LOCK_TYPE`	Whether the lock is `RECORD` (row level) or `TABLE` level.
`LOCK_TABLE`	Name of the locked table,or table containing locked rows.
`LOCK_INDEX`	Index name if a `RECORD LOCK_TYPE`, or `NULL` if not.
`LOCK_SPACE`	Tablespace ID if a `RECORD LOCK_TYPE`, or `NULL` if not.
`LOCK_PAGE`	Locked record page number if a `RECORD LOCK_TYPE`, or `NULL` if not.
`LOCK_REC`	Locked record heap number if a `RECORD LOCK_TYPE`, or `NULL` if not.
`LOCK_DATA`	Locked record primary key as an SQL string if a `RECORD LOCK_TYPE`, or `NULL` if not. If no primary key exists, the internal InnoDB row_id number is instead used. To avoid unnecessary IO, also `NULL` if the locked record page is not in the buffer pool

The table is often used in conjunction with the INNODB_LOCK_WAITS and INNODB_TRX tables to diagnose problematic locks and transactions

Example

```
-- session 1
START TRANSACTION;
UPDATE t SET id = 15 WHERE id = 10;

-- session 2
DELETE FROM t WHERE id = 10;

-- session 1
USE information_schema;
SELECT l.*, t.*
    FROM information_schema.INNODB_LOCKS l
    JOIN information_schema.INNODB_TRX t
        ON l.lock_trx_id = t.trx_id
    WHERE trx_state = 'LOCK WAIT' \G
*************************** 1. row ***************************
                    lock_id: 840:40:3:2
                lock_trx_id: 840
                  lock_mode: X
                  lock_type: RECORD
                 lock_table: `test`.`t`
                 lock_index: PRIMARY
                 lock_space: 40
                  lock_page: 3
                   lock_rec: 2
                  lock_data: 10
                     trx_id: 840
                  trx_state: LOCK WAIT
              trx_started: 2019-12-23 18:43:46
     trx_requested_lock_id: 840:40:3:2
          trx_wait_started: 2019-12-23 18:43:46
                trx_weight: 2
       trx_mysql_thread_id: 46
                 trx_query: DELETE FROM t WHERE id = 10
       trx_operation_state: starting index read
         trx_tables_in_use: 1
         trx_tables_locked: 1
          trx_lock_structs: 2
     trx_lock_memory_bytes: 1136
           trx_rows_locked: 1
         trx_rows_modified: 0
   trx_concurrency_tickets: 0
       trx_isolation_level: REPEATABLE READ
         trx_unique_checks: 1
    trx_foreign_key_checks: 1
trx_last_foreign_key_error: NULL
          trx_is_read_only: 0
trx_autocommit_non_locking: 0
```

.

1.1.1.2.9.1.1.1.19 Information Schema INNODB_METRICS Table

Contents

The Information Schema `INNODB_METRICS` table contains a list of useful InnoDB performance metrics. Each row in the table represents an instrumented counter that can be stopped, started and reset, and which can be grouped together by module.

The `PROCESS` privilege is required to view the table.

It has the following columns:

Column	Description
`NAME`	Unique counter name.

SUBSYSTEM	InnoDB subsystem. See below for the matching module to use to enable/disable monitoring this subsytem with the innodb_monitor_enable and innodb_monitor_disable system variables.
COUNT	Count since being enabled.
MAX_COUNT	Maximum value since being enabled.
MIN_COUNT	Minimum value since being enabled.
AVG_COUNT	Average value since being enabled.
COUNT_RESET	Count since last being reset.
MAX_COUNT_RESET	Maximum value since last being reset.
MIN_COUNT_RESET	Minimum value since last being reset.
AVG_COUNT_RESET	Average value since last being reset.
TIME_ENABLED	Time last enabled.
TIME_DISABLED	Time last disabled
TIME_ELAPSED	Time since enabled
TIME_RESET	Time last reset.
STATUS	Whether the counter is currently enabled to disabled.
TYPE	Item type; one of `counter`, `value`, `status_counter`, `set_owner`, `set_member`.
COMMENT	Counter description.

Enabling and Disabling Counters

Most of the counters are disabled by default. To enable them, use the innodb_monitor_enable system variable. You can either enable a variable by its name, for example:

```
SET GLOBAL innodb_monitor_enable = icp_match;
```

or enable a number of counters grouped by module. The `SUBSYSTEM` field indicates which counters are grouped together, but the following module names need to be used:

Module Name	Subsytem Field
module_metadata	metadata
module_lock	lock
module_buffer	buffer
module_buf_page	buffer_page_io
module_os	os
module_trx	transaction
module_purge	purge
module_compress	compression
module_file	file_system
module_index	index
module_adaptive_hash	adaptive_hash_index From MariaDB 10.6.2, if innodb_adaptive_hash_index is disabled (the default), adaptive_hash_index will not be updated.
module_ibuf_system	change_buffer
module_srv	server
module_ddl	ddl
module_dml	dml
module_log	recovery

module_icp	icp

There are four counters in the `icp` subsystem:

```
SELECT NAME, SUBSYSTEM FROM INNODB_METRICS WHERE SUBSYSTEM='icp';
+------------------+-----------+
| NAME             | SUBSYSTEM |
+------------------+-----------+
| icp_attempts     | icp       |
| icp_no_match     | icp       |
| icp_out_of_range | icp       |
| icp_match        | icp       |
+------------------+-----------+
```

To enable them all, use the associated module name from the table above, `module_icp`.

```
SET GLOBAL innodb_monitor_enable = module_icp;
```

The `%` wildcard, used to represent any number of characters, can also be used when naming counters, for example:

```
SET GLOBAL innodb_monitor_enable = 'buffer%'
```

To disable counters, use the innodb_monitor_disable system variable, using the same naming rules as described above for enabling.

Counter status is not persistent, and will be reset when the server restarts. It is possible to use the options on the command line, or the `innodb_monitor_enable` option only in a configuration file.

Resetting Counters

Counters can also be reset. Resetting sets all the `*_COUNT_RESET` values to zero, while leaving the `*_COUNT` values, which perform counts since the counter was enabled, untouched. Resetting is performed with the innodb_monitor_reset (for individual counters) and innodb_monitor_reset_all (for all counters) system variables.

Simplifying from MariaDB 10.6

MariaDB starting with 10.6

From MariaDB 10.6, the interface was simplified by removing the following:

- buffer_LRU_batches_flush
- buffer_LRU_batch_flush_pages
- buffer_LRU_batches_evict
- buffer_LRU_batch_evict_pages

and by making the following reflect the status variables:

- buffer_LRU_batch_flush_total_pages: innodb_buffer_pool_pages_LRU_flushed
- buffer_LRU_batch_evict_total_pages: innodb_buffer_pool_pages_LRU_freed

The intention is to eventually remove the interface entirely (see MDEV-15706).

Examples

MariaDB 10.8:

```
SELECT name,subsystem,type,comment FROM INFORMATION_SCHEMA.INNODB_METRICS\G
*************************** 1. row ***************************
     name: metadata_table_handles_opened
subsystem: metadata
     type: counter
  comment: Number of table handles opened
*************************** 2. row ***************************
     name: lock_deadlocks
subsystem: lock
     type: value
  comment: Number of deadlocks
*************************** 3. row ***************************
     name: lock_timeouts
```

```
     name: lock_timeouts
subsystem: lock
     type: value
  comment: Number of lock timeouts
*************************** 4. row ***************************
     name: lock_rec_lock_waits
subsystem: lock
     type: counter
  comment: Number of times enqueued into record lock wait queue
*************************** 5. row ***************************
     name: lock_table_lock_waits
subsystem: lock
     type: counter
  comment: Number of times enqueued into table lock wait queue
*************************** 6. row ***************************
     name: lock_rec_lock_requests
subsystem: lock
     type: counter
  comment: Number of record locks requested
*************************** 7. row ***************************
     name: lock_rec_lock_created
subsystem: lock
     type: counter
  comment: Number of record locks created
*************************** 8. row ***************************
     name: lock_rec_lock_removed
subsystem: lock
     type: counter
  comment: Number of record locks removed from the lock queue
*************************** 9. row ***************************
     name: lock_rec_locks
subsystem: lock
     type: counter
  comment: Current number of record locks on tables
*************************** 10. row ***************************
     name: lock_table_lock_created
subsystem: lock
     type: counter
  comment: Number of table locks created

...

*************************** 207. row ***************************
     name: icp_attempts
subsystem: icp
     type: counter
  comment: Number of attempts for index push-down condition checks
*************************** 208. row ***************************
     name: icp_no_match
subsystem: icp
     type: counter
  comment: Index push-down condition does not match
*************************** 209. row ***************************
     name: icp_out_of_range
subsystem: icp
     type: counter
  comment: Index push-down condition out of range
*************************** 210. row ***************************
     name: icp_match
subsystem: icp
     type: counter
  comment: Index push-down condition matches
```

1.1.1.2.9.1.1.1.20 Information Schema INNODB_MUTEXES Table

The `INNODB_MUTEXES` table monitors mutex and rw locks waits. It has the following columns:

Column	Description
`NAME`	Name of the lock, as it appears in the source code.

CREATE_FILE	File name of the mutex implementation.
CREATE_LINE	Line number of the mutex implementation.
OS_WAITS	How many times the mutex occurred.

The CREATE_FILE and CREATE_LINE columns depend on the InnoDB/XtraDB version.

Note that since MariaDB 10.2.2, the table has only been providing information about rw_lock_t, not any mutexes. From MariaDB 10.2.2 until MariaDB 10.2.32, MariaDB 10.3.23, MariaDB 10.4.13 and MariaDB 10.5.1, the NAME column was not populated (MDEV-21636).

The SHOW ENGINE INNODB STATUS statement provides similar information.

Examples

```
SELECT * FROM INNODB_MUTEXES;
+-------------------------------+---------------------+-------------+----------+
| NAME                          | CREATE_FILE         | CREATE_LINE | OS_WAITS |
+-------------------------------+---------------------+-------------+----------+
| &dict_sys->mutex              | dict0dict.cc        |         989 |        2 |
| &buf_pool->flush_state_mutex | buf0buf.cc          |        1388 |        1 |
| &log_sys->checkpoint_lock     | log0log.cc          |        1014 |        2 |
| &block->lock                  | combined buf0buf.cc |        1120 |        1 |
+-------------------------------+---------------------+-------------+----------+
```

1.1.1.2.9.1.1.1.21 Information Schema INNODB_SYS_COLUMNS Table

The Information Schema INNODB_SYS_COLUMNS table contains information about InnoDB fields.

The PROCESS privilege is required to view the table.

It has the following columns:

Column	Description
TABLE_ID	Table identifier, matching the value from INNODB_SYS_TABLES.TABLE_ID.
NAME	Column name.
POS	Ordinal position of the column in the table, starting from 0. This value is adjusted when columns are added or removed.
MTYPE	Numeric column type identifier, (see the table below for an explanation of its values).
PRTYPE	Binary value of the InnoDB precise type, representing the data type, character set code and nullability.
LEN	Column length. For multi-byte character sets, represents the length in bytes.

The column MTYPE uses a numeric column type identifier, which has the following values:

Column Type Identifier	Description
1	VARCHAR
2	CHAR
3	FIXBINARY
4	BINARY
5	BLOB
6	INT
7	SYS_CHILD
8	SYS
9	FLOAT

10	DOUBLE
11	DECIMAL
12	VARMYSQL
13	MYSQL

Example

```
SELECT * FROM information_schema.INNODB_SYS_COLUMNS LIMIT 3\G
*************************** 1. row ***************************
TABLE_ID: 11
    NAME: ID
     POS: 0
   MTYPE: 1
  PRTYPE: 524292
     LEN: 0
*************************** 2. row ***************************
TABLE_ID: 11
    NAME: FOR_NAME
     POS: 0
   MTYPE: 1
  PRTYPE: 524292
    LEN: 0
*************************** 3. row ***************************
TABLE_ID: 11
    NAME: REF_NAME
     POS: 0
   MTYPE: 1
  PRTYPE: 524292
     LEN: 0
3 rows in set (0.00 sec)
```

1.1.1.2.9.1.1.1.22 Information Schema INNODB_SYS_DATAFILES Table

MariaDB until 10.5

The `INNODB_SYS_DATAFILES` table was added in MariaDB 10.0.4, and removed in MariaDB 10.6.0.

The Information Schema `INNODB_SYS_DATAFILES` table contains information about InnoDB datafile paths. It was intended to provide metadata for tablespaces inside InnoDB tables, which was never implemented in MariaDB and was removed in MariaDB 10.6. The `PROCESS` privilege is required to view the table.

It contains the following columns:

Column	Description
`SPACE`	Numeric tablespace. Matches the INNODB_SYS_TABLES.SPACE value.
`PATH`	Tablespace datafile path.

Example

```
SELECT * FROM INNODB_SYS_DATAFILES;
+-------+---------------------------------+
| SPACE | PATH                            |
+-------+---------------------------------+
|    19 | ./test/t2.ibd                   |
|    20 | ./test/t3.ibd                   |
...
|    68 | ./test/animals.ibd              |
|    69 | ./test/animal_count.ibd         |
|    70 | ./test/t.ibd                    |
+-------+---------------------------------+
```

1.1.1.2.9.1.1.1.23 Information Schema INNODB_SYS_FIELDS Table

The Information Schema `INNODB_SYS_FIELDS` table contains information about fields that are part of an InnoDB index.

The `PROCESS` privilege is required to view the table.

It has the following columns:

Column	Description
`INDEX_ID`	Index identifier, matching the value from INNODB_SYS_INDEXES.INDEX_ID.
`NAME`	Field name, matching the value from INNODB_SYS_COLUMNS.NAME.
`POS`	Ordinal position of the field within the index, starting from `0` . This is adjusted as columns are removed.

Example

```
SELECT * FROM information_schema.INNODB_SYS_FIELDS LIMIT 3\G
*************************** 1. row ***************************
INDEX_ID: 11
    NAME: ID
     POS: 0
*************************** 2. row ***************************
INDEX_ID: 12
    NAME: FOR_NAME
     POS: 0
*************************** 3. row ***************************
INDEX_ID: 13
    NAME: REF_NAME
     POS: 0
3 rows in set (0.00 sec)
```

1.1.1.2.9.1.1.1.24 Information Schema INNODB_SYS_FOREIGN Table

The Information Schema `INNODB_SYS_FOREIGN` table contains information about InnoDB foreign keys.

The `PROCESS` privilege is required to view the table.

It has the following columns:

Column	Description
`ID`	Database name and foreign key name.
`FOR_NAME`	Database and table name of the foreign key child.
`REF_NAME`	Database and table name of the foreign key parent.
`N_COLS`	Number of foreign key index columns.
`TYPE`	Bit flag providing information about the foreign key.

The `TYPE` column provides a bit flag with information about the foreign key. This information is `OR` 'ed together to read:

Bit Flag	Description
1	ON DELETE CASCADE
2	ON UPDATE SET NULL
4	ON UPDATE CASCADE
8	ON UPDATE SET NULL
16	ON DELETE NO ACTION

32	ON UPDATE NO ACTION

Example

```
SELECT * FROM INNODB_SYS_FOREIGN\G
*************************** 1. row ***************************
      ID: mysql/innodb_index_stats_ibfk_1
FOR_NAME: mysql/innodb_index_stats
REF_NAME: mysql/innodb_table_stats
  N_COLS: 2
    TYPE: 0
...
```

1.1.1.2.9.1.1.1.25 Information Schema INNODB_SYS_FOREIGN_COLS Table

The Information Schema `INNODB_SYS_FOREIGN_COLS` table contains information about InnoDB foreign key columns.

The `PROCESS` privilege is required to view the table.

It has the following columns:

Column	Description
`ID`	Foreign key index associated with this column, matching the INNODB_SYS_FOREIGN.ID field.
`FOR_COL_NAME`	Child column name.
`REF_COL_NAME`	Parent column name.
`POS`	Ordinal position of the column in the table, starting from 0.

1.1.1.2.9.1.1.1.26 Information Schema INNODB_SYS_INDEXES Table

The Information Schema `INNODB_SYS_INDEXES` table contains information about InnoDB indexes.

The `PROCESS` privilege is required to view the table.

It has the following columns:

Field	Type	Null	Key	Default	Description
INDEX_ID	bigint(21) unsigned	NO		0	A unique index identifier.
NAME	varchar(64)	NO			Index name, lowercase for all user-created indexes, or uppercase for implicitly-created indexes; `PRIMARY` (primary key), `GEN_CLUST_INDEX` (index representing primary key where there isn't one), `ID_IND` , `FOR_IND` (validating foreign key constraint) , `REF_IND` .
TABLE_ID	bigint(21) unsigned	NO		0	Table identifier, matching the value from INNODB_SYS_TABLES.TABLE_ID.
TYPE	int(11)	NO		0	Numeric type identifier; one of `0` (secondary index), `1` (clustered index), `2` (unique index), `3` (primary index), `32` (full-text index).
N_FIELDS	int(11)	NO		0	Number of columns in the index. GEN_CLUST_INDEX's have a value of 0 as the index is not based on an actual column in the table.
PAGE_NO	int(11)	NO		0	Index B-tree's root page number. `-1` (unused) for full-text indexes, as they are laid out over several auxiliary tables.

SPACE	int(11)	NO		0	Tablespace identifier where the index resides. `0` represents the InnoDB system tablespace, while any other value represents a table created in file-per-table mode (see the innodb_file_per_table system variable). Remains unchanged after a TRUNCATE TABLE statement, and not necessarily unique.
MERGE_THRESHOLD	int(11)	NO		0	

Example

```
SELECT * FROM information_schema.INNODB_SYS_INDEXES LIMIT 3\G
*************************** 1. row ***************************
       INDEX_ID: 11
           NAME: ID_IND
       TABLE_ID: 11
           TYPE: 3
       N_FIELDS: 1
        PAGE_NO: 302
          SPACE: 0
MERGE_THRESHOLD: 50
*************************** 2. row ***************************
       INDEX_ID: 12
           NAME: FOR_IND
       TABLE_ID: 11
           TYPE: 0
       N_FIELDS: 1
        PAGE_NO: 303
          SPACE: 0
MERGE_THRESHOLD: 50
*************************** 3. row ***************************
       INDEX_ID: 13
           NAME: REF_IND
       TABLE_ID: 11
           TYPE: 3
       N_FIELDS: 1
        PAGE_NO: 304
          SPACE: 0
MERGE_THRESHOLD: 50
3 rows in set (0.00 sec)
```

1.1.1.2.9.1.1.1.27 Information Schema INNODB_SYS_SEMAPHORE_WAITS Table

The Information Schema INNODB_SYS_SEMAPHORE_WAITS table is meant to contain information about current semaphore waits. At present it is not correctly populated. See MDEV-21330.

The PROCESS privilege is required to view the table.

It contains the following columns:

Column	Description
THREAD_ID	Thread id waiting for semaphore
OBJECT_NAME	Semaphore name
FILE	File name where semaphore was requested
LINE	Line number on above file
WAIT_TIME	Wait time
WAIT_OBJECT	
WAIT_TYPE	Object type (mutex, rw-lock)
HOLDER_THREAD_ID	Holder thread id
HOLDER_FILE	File name where semaphore was acquired

HOLDER_LINE	Line number for above
CREATED_FILE	Creation file name
CREATED_LINE	Line number for above
WRITER_THREAD	Last write request thread id
RESERVATION_MODE	Reservation mode (shared, exclusive)
READERS	Number of readers if only shared mode
WAITERS_FLAG	Flags
LOCK_WORD	Lock word (for developers)
LAST_READER_FILE	Removed
LAST_READER_LINE	Removed
LAST_WRITER_FILE	Last writer file name
LAST_WRITER_LINE	Above line number
OS_WAIT_COUNT	Wait count

1.1.1.2.9.1.1.1.28 Information Schema INNODB_SYS_TABLES Table

The Information Schema `INNODB_SYS_TABLES` table contains information about InnoDB tables.

The `PROCESS` privilege is required to view the table.

It has the following columns:

Field	Type	Null	Key	Default	Description
TABLE_ID	bigint(21) unsigned	NO		0	Unique InnoDB table identifier.
NAME	varchar(655)	NO			Database and table name, or the uppercase InnoDB system table name.
FLAG	int(11)	NO		0	See Flag below
N_COLS	int(11) unsigned (>= MariaDB 10.5) int(11) (<= MariaDB 10.4)	NO		0	Number of columns in the table. The count includes two or three hidden InnoDB system columns, appended to the end of the column list: DB_ROW_ID (if there is no primary key or unique index on NOT NULL columns), DB_TRX_ID, DB_ROLL_PTR.
SPACE	int(11) unsigned (>= MariaDB 10.5) int(11) (<= MariaDB 10.4)	NO		0	Tablespace identifier where the index resides. `0` represents the InnoDB system tablespace, while any other value represents a table created in file-per-table mode (see the innodb_file_per_table system variable). Remains unchanged after a TRUNCATE TABLE statement.
FILE_FORMAT	varchar(10)	YES		NULL	InnoDB file format (Antelope or Barracuda). Removed in MariaDB 10.3.
ROW_FORMAT	enum('Redundant', 'Compact', 'Compressed', 'Dynamic') (>= MariaDB 10.5) varchar(12) (<= MariaDB 10.4)	YES		NULL	InnoDB storage format (Compact, Redundant, Dynamic, or Compressed).
ZIP_PAGE_SIZE	int(11) unsigned	NO		0	For Compressed tables, the zipped page size.
SPACE_TYPE	enum('Single','System') (>= MariaDB 10.5) varchar(10) (<= MariaDB 10.4)	YES		NULL	

Flag

The flag field returns the dict_table_t::flags that correspond to the data dictionary record.

Bit	Description
0	Set if ROW_FORMAT is not REDUNDANT.
1 to 4	0, except for ROW_FORMAT=COMPRESSED, where they will determine the KEY_BLOCK_SIZE (the compressed page size).
5	Set for ROW_FORMAT=DYNAMIC or ROW_FORMAT=COMPRESSED.
6	Set if the DATA DIRECTORY attribute was present when the table was originally created.
7	Set if the page_compressed attribute is present.
8 to 11	Determine the page_compression_level.
12 13	Normally 00, but 11 for "no-rollback tables" (MariaDB 10.3 CREATE SEQUENCE). In MariaDB 10.1, these bits could be 01 or 10 for ATOMIC_WRITES=ON or ATOMIC_WRITES=OFF.

Note that the table flags returned here are not the same as tablespace flags (FSP_SPACE_FLAGS).

Example

```
SELECT * FROM information_schema.INNODB_SYS_TABLES LIMIT 2\G
*************************** 1. row ***************************
     TABLE_ID: 14
         NAME: SYS_DATAFILES
         FLAG: 0
       N_COLS: 5
        SPACE: 0
   ROW_FORMAT: Redundant
ZIP_PAGE_SIZE: 0
   SPACE_TYPE: System
*************************** 2. row ***************************
     TABLE_ID: 11
         NAME: SYS_FOREIGN
         FLAG: 0
       N_COLS: 7
        SPACE: 0
   ROW_FORMAT: Redundant
ZIP_PAGE_SIZE: 0
   SPACE_TYPE: System
2 rows in set (0.00 sec)
```

1.1.1.2.9.1.1.1.29 Information Schema INNODB_SYS_TABLESPACES Table

The Information Schema INNODB_SYS_TABLESPACES table contains information about InnoDB tablespaces. Until MariaDB 10.5 it was based on the internal SYS_TABLESPACES table. This internal table was removed in MariaDB 10.6.0, so this Information Schema table has been repurposed to directly reflect the filesystem (fil_system.space_list).

The PROCESS privilege is required to view the table.

It has the following columns:

Column	Description
SPACE	Unique InnoDB tablespace identifier.
NAME	Database and table name separated by a backslash, or the uppercase InnoDB system table name.
FLAG	1 if a DATA DIRECTORY option has been specified in CREATE TABLE, otherwise 0.
FILE_FORMAT	InnoDB file format. Removed in MariaDB 10.3.1

ROW_FORMAT	InnoDB storage format used for this tablespace. If the Antelope file format is used, this value is always `Compact or Redundant`. When a table's checksum algorithm is `full_crc32` (the default from MariaDB 10.5), the value can only be `Compressed` or `NULL`.
PAGE_SIZE	Page size in bytes for this tablespace. Until MariaDB 10.5.0, this was the value of the innodb_page_size variable. From MariaDB 10.6.0, contains the physical page size of a page (previously `ZIP_PAGE_SIZE`).
ZIP_PAGE_SIZE	Zip page size for this tablespace. Removed in MariaDB 10.6.0.
SPACE_TYPE	Tablespace type. Can be `General` for general tablespaces or `Single` for file-per-table tablespaces. Introduced MariaDB 10.2.1. Removed MariaDB 10.5.0.
FS_BLOCK_SIZE	File system block size. Introduced MariaDB 10.2.1.
FILE_SIZE	Maximum size of the file, uncompressed. Introduced MariaDB 10.2.1.
ALLOCATED_SIZE	Actual size of the file as per space allocated on disk. Introduced MariaDB 10.2.1.
FILENAME	Tablespace datafile path, previously part of the INNODB_SYS_DATAFILES table. Added in MariaDB 10.6.0.

Examples

MariaDB 10.4:

```
DESC information_schema.innodb_sys_tablespaces;
+---------------+---------------------+------+-----+---------+-------+
| Field         | Type                | Null | Key | Default | Extra |
+---------------+---------------------+------+-----+---------+-------+
| SPACE         | int(11) unsigned    | NO   |     | 0       |       |
| NAME          | varchar(655)        | NO   |     |         |       |
| FLAG          | int(11) unsigned    | NO   |     | 0       |       |
| ROW_FORMAT    | varchar(22)         | YES  |     | NULL    |       |
| PAGE_SIZE     | int(11) unsigned    | NO   |     | 0       |       |
| ZIP_PAGE_SIZE | int(11) unsigned    | NO   |     | 0       |       |
| SPACE_TYPE    | varchar(10)         | YES  |     | NULL    |       |
| FS_BLOCK_SIZE | int(11) unsigned    | NO   |     | 0       |       |
| FILE_SIZE     | bigint(21) unsigned | NO   |     | 0       |       |
| ALLOCATED_SIZE | bigint(21) unsigned | NO   |     | 0       |       |
+---------------+---------------------+------+-----+---------+-------+
```

From MariaDB 10.4:

```
SELECT * FROM information_schema.INNODB_SYS_TABLESPACES LIMIT 2\G
*************************** 1. row ***************************
         SPACE: 2
          NAME: mysql/innodb_table_stats
          FLAG: 33
    ROW_FORMAT: Dynamic
     PAGE_SIZE: 16384
 ZIP_PAGE_SIZE: 0
    SPACE_TYPE: Single
 FS_BLOCK_SIZE: 4096
     FILE_SIZE: 98304
ALLOCATED_SIZE: 98304
*************************** 2. row ***************************
         SPACE: 3
          NAME: mysql/innodb_index_stats
          FLAG: 33
    ROW_FORMAT: Dynamic
     PAGE_SIZE: 16384
 ZIP_PAGE_SIZE: 0
    SPACE_TYPE: Single
 FS_BLOCK_SIZE: 4096
     FILE_SIZE: 98304
ALLOCATED_SIZE: 98304
```

1.1.1.2.9.1.1.1.30 Information Schema

INNODB_SYS_TABLESTATS Table

The Information Schema `INNODB_SYS_TABLESTATS` table contains InnoDB status information. It can be used for developing new performance-related extensions, or high-level performance monitoring.

The `PROCESS` privilege is required to view the table.

Note that the MySQL InnoDB and Percona XtraDB versions of the tables differ (see XtraDB and InnoDB).

It contains the following columns:

Column	Description
`TABLE_ID`	Table ID, matching the INNODB_SYS_TABLES.TABLE_ID value.
`SCHEMA`	Database name (XtraDB only).
`NAME`	Table name, matching the INNODB_SYS_TABLES.NAME value.
`STATS_INITIALIZED`	`Initialized` if statistics have already been collected, otherwise `Uninitialized`.
`NUM_ROWS`	Estimated number of rows currently in the table. Updated after each statement modifying the data, but uncommited transactions mean it may not be accurate.
`CLUST_INDEX_SIZE`	Number of pages on disk storing the clustered index, holding InnoDB table data in primary key order, or `NULL` if not statistics yet collected.
`OTHER_INDEX_SIZE`	Number of pages on disk storing secondary indexes for the table, or `NULL` if not statistics yet collected.
`MODIFIED_COUNTER`	Number of rows modified by statements modifying data.
`AUTOINC`	Auto_increment value.
`REF_COUNT`	Countdown to zero, when table metadata can be removed from the table cache. (InnoDB only)
`MYSQL_HANDLES_OPENED`	(XtraDB only).

1.1.1.2.9.1.1.1.31 Information Schema INNODB_SYS_VIRTUAL Table

MariaDB starting with 10.2

The `INNODB_SYS_VIRTUAL` table was added in MariaDB 10.2.

The Information Schema `INNODB_SYS_VIRTUAL` table contains information about base columns of virtual columns. The `PROCESS` privilege is required to view the table.

It contains the following columns:

Field	Type	Null	Key	Default	Description
TABLE_ID	bigint(21) unsigned	NO		0	
POS	int(11) unsigned	NO		0	
BASE_POS	int(11) unsigned	NO		0	

1.1.1.2.9.1.1.1.32 Information Schema INNODB_TABLESPACES_ENCRYPTION Table

The Information Schema `INNODB_TABLESPACES_ENCRYPTION` table contains metadata about encrypted InnoDB tablespaces. When you enable encryption for an InnoDB tablespace, an entry for the tablespace is added to this table. If you later disable encryption for the InnoDB tablespace, then the row still remains in this table, but the `ENCRYPTION_SCHEME` and `CURRENT_KEY_VERSION` columns will be set to `0`.

Viewing this table requires the PROCESS privilege, although a bug in versions before MariaDB 10.1.46, 10.2.33, 10.3.24, 10.4.14 and 10.5.5 mean the SUPER privilege was required (MDEV-23003).

It has the following columns:

Column	Description	Added
`SPACE`	InnoDB tablespace ID.	
`NAME`	Path to the InnoDB tablespace file, without the extension.	
`ENCRYPTION_SCHEME`	Key derivation algorithm. Only `1` is currently used to represent an algorithm. If this value is `0`, then the tablespace is unencrypted.	
`KEYSERVER_REQUESTS`	Number of times InnoDB has had to request a key from the encryption key management plugin. The three most recent keys are cached internally.	
`MIN_KEY_VERSION`	Minimum key version used to encrypt a page in the tablespace. Different pages may be encrypted with different key versions.	
`CURRENT_KEY_VERSION`	Key version that will be used to encrypt pages. If this value is `0`, then the tablespace is unencrypted.	
`KEY_ROTATION_PAGE_NUMBER`	Page that a background encryption thread is currently rotating. If key rotation is not enabled, then the value will be `NULL`.	
`KEY_ROTATION_MAX_PAGE_NUMBER`	When a background encryption thread starts rotating a tablespace, the field contains its current size. If key rotation is not enabled, then the value will be `NULL`.	
`CURRENT_KEY_ID`	Key ID for the encryption key currently in use.	MariaDB 10.1.13
`ROTATING_OR_FLUSHING`	Current key rotation status. If this value is `1`, then the background encryption threads are working on the tablespace. See MDEV-11738.	MariaDB 10.2.5, MariaDB 10.1.23

When the InnoDB system tablespace is encrypted, it is represented in this table with the special name: `innodb_system`.

Example

```
SELECT * FROM information_schema.INNODB_TABLESPACES_ENCRYPTION
WHERE NAME LIKE 'db_encrypt%';
+-------+-----------------------------------------------+------------------+-----------------
---+-----------------+--------------------+--------------------------+----------------------
--------+
| SPACE | NAME                                          | ENCRYPTION_SCHEME |
KEYSERVER_REQUESTS | MIN_KEY_VERSION | CURRENT_KEY_VERSION | KEY_ROTATION_PAGE_NUMBER |
KEY_ROTATION_MAX_PAGE_NUMBER |
+-------+-----------------------------------------------+------------------+-----------------
---+-----------------+--------------------+--------------------------+----------------------
--------+
|    18 | db_encrypt/t_encrypted_existing_key           |                1 |
1 |               1 |                  1 |                     NULL |
NULL |
|    19 | db_encrypt/t_not_encrypted_existing_key       |                1 |
0 |               1 |                  1 |                     NULL |
NULL |
|    20 | db_encrypt/t_not_encrypted_non_existing_key   |                1 |
0 |      4294967295 |          4294967295 |                     NULL |
NULL |
|    21 | db_encrypt/t_default_encryption_existing_key  |                1 |
1 |               1 |                  1 |                     NULL |
NULL |
|    22 | db_encrypt/t_encrypted_default_key            |                1 |
1 |               1 |                  1 |                     NULL |
NULL |
|    23 | db_encrypt/t_not_encrypted_default_key        |                1 |
0 |               1 |                  1 |                     NULL |
NULL |
|    24 | db_encrypt/t_defaults                         |                1 |
1 |               1 |                  1 |                     NULL |
NULL |
+-------+-----------------------------------------------+------------------+-----------------
---+-----------------+--------------------+--------------------------+----------------------
--------+
7 rows in set (0.00 sec)
```

1.1.1.2.9.1.1.1.33 Information Schema INNODB_TABLESPACES_SCRUBBING Table

MariaDB 10.1.3 - 10.5.1

InnoDB and XtraDB data scrubbing was introduced in MariaDB 10.1.3. The table was removed in MariaDB 10.5.2 - see MDEV-15528.

The Information Schema `INNODB_TABLESPACES_SCRUBBING` table contains data scrubbing information.

The `PROCESS` privilege is required to view the table.

It has the following columns:

Column	Description
`SPACE`	InnoDB table space id number.
`NAME`	Path to the table space file, without the extension.
`COMPRESSED`	The compressed page size, or zero if uncompressed.
`LAST_SCRUB_COMPLETED`	Date and time when the last scrub was completed, or `NULL` if never been performed.
`CURRENT_SCRUB_STARTED`	Date and time when the current scrub started, or `NULL` if never been performed.
`CURRENT_SCRUB_ACTIVE_THREADS`	Number of threads currently scrubbing the tablespace.
`CURRENT_SCRUB_PAGE_NUMBER`	Page that the scrubbing thread is currently scrubbing, or `NULL` if not enabled.
`CURRENT_SCRUB_MAX_PAGE_NUMBER`	When a scrubbing starts rotating a table space, the field contains its current size. `NULL` if not enabled.

ON_SSD	The field contains 1 when MariaDB detects that the table space is on a SSD based storage. 0 if not SSD or it could not be determined (since MariaDB 10.4.4)

Example

```
SELECT * FROM information_schema.INNODB_TABLESPACES_SCRUBBING LIMIT 1\G
*************************** 1. row ***************************
                       SPACE: 1
                        NAME: mysql/innodb_table_stats
                  COMPRESSED: 0
         LAST_SCRUB_COMPLETED: NULL
        CURRENT_SCRUB_STARTED: NULL
    CURRENT_SCRUB_PAGE_NUMBER: NULL
CURRENT_SCRUB_MAX_PAGE_NUMBER: 0
         ROTATING_OR_FLUSHING: 0
1 rows in set (0.00 sec)
```

1.1.1.2.9.1.1.1.34 Information Schema INNODB_TRX Table

The Information Schema INNODB_TRX table stores information about all currently executing InnoDB transactions.

It has the following columns:

Column	Description
TRX_ID	Unique transaction ID number.
TRX_STATE	Transaction execution state; one of RUNNING, LOCK WAIT, ROLLING BACK or COMMITTING.
TRX_STARTED	Time that the transaction started.
TRX_REQUESTED_LOCK_ID	If TRX_STATE is LOCK_WAIT, the INNODB_LOCKS.LOCK_ID value of the lock being waited on. NULL if any other state.
TRX_WAIT_STARTED	If TRX_STATE is LOCK_WAIT, the time the transaction started waiting for the lock, otherwise NULL.
TRX_WEIGHT	Transaction weight, based on the number of locked rows and the number of altered rows. To resolve deadlocks, lower weighted transactions are rolled back first. Transactions that have affected non-transactional tables are always treated as having a heavier weight.
TRX_MYSQL_THREAD_ID	Thread ID from the PROCESSLIST table (note that the locking and transaction information schema tables use a different snapshot from the processlist, so records may appear in one but not the other).
TRX_QUERY	SQL that the transaction is currently running.
TRX_OPERATION_STATE	Transaction's current state, or NULL.
TRX_TABLES_IN_USE	Number of InnoDB tables currently being used for processing the current SQL statement.
TRX_TABLES_LOCKED	Number of InnoDB tables that that have row locks held by the current SQL statement.
TRX_LOCK_STRUCTS	Number of locks reserved by the transaction.
TRX_LOCK_MEMORY_BYTES	Total size in bytes of the memory used to hold the lock structures for the current transaction in memory.
TRX_ROWS_LOCKED	Number of rows the current transaction has locked. locked by this transaction. An approximation, and may include rows not visible to the current transaction that are delete-marked but physically present.
TRX_ROWS_MODIFIED	Number of rows added or changed in the current transaction.
TRX_CONCURRENCY_TICKETS	Indicates how much work the current transaction can do before being swapped out, see the innodb_concurrency_tickets system variable.

TRX_ISOLATION_LEVEL	Isolation level of the current transaction.
TRX_UNIQUE_CHECKS	Whether unique checks are `on` or `off` for the current transaction. Bulk data are a case where unique checks would be off.
TRX_FOREIGN_KEY_CHECKS	Whether foreign key checks are `on` or `off` for the current transaction. Bulk data are a case where foreign keys checks would be off.
TRX_LAST_FOREIGN_KEY_ERROR	Error message for the most recent foreign key error, or `NULL` if none.
TRX_ADAPTIVE_HASH_LATCHED	Whether the adaptive hash index is locked by the current transaction or not. One transaction at a time can change the adaptive hash index.
TRX_ADAPTIVE_HASH_TIMEOUT	Whether the adaptive hash index search latch shoild be relinquished immediately or reserved across all MariaDB calls. `0` if there is no contention on the adaptive hash index, in which case the latch is reserved until completion, otherwise counts down to zero and the latch is released after each row lookup.
TRX_IS_READ_ONLY	`1` if a read-only transaction, otherwise `0`.
TRX_AUTOCOMMIT_NON_LOCKING	`1` if the transaction only contains this one statement, that is, a `SELECT` statement not using `FOR UPDATE` or `LOCK IN SHARED MODE`, and with autocommit on. If this and `TRX_IS_READ_ONLY` are both 1, the transaction can be optimized by the storrage engine to reduce some overheads

The table is often used in conjunction with the INNODB_LOCKS and INNODB_LOCK_WAITS tables to diagnose problematic locks and transactions.

XA transactions are not stored in this table. To see them, `XA RECOVER` can be used.

Example

```
-- session 1
START TRANSACTION;
UPDATE t SET id = 15 WHERE id = 10;

-- session 2
DELETE FROM t WHERE id = 10;

-- session 1
USE information_schema;
SELECT l.*, t.*
    FROM information_schema.INNODB_LOCKS l
    JOIN information_schema.INNODB_TRX t
        ON l.lock_trx_id = t.trx_id
    WHERE trx_state = 'LOCK WAIT' \G
*************************** 1. row ***************************
                    lock_id: 840:40:3:2
                lock_trx_id: 840
                  lock_mode: X
                  lock_type: RECORD
                 lock_table: `test`.`t`
                 lock_index: PRIMARY
                 lock_space: 40
                  lock_page: 3
                   lock_rec: 2
                  lock_data: 10
                     trx_id: 840
                  trx_state: LOCK WAIT
                trx_started: 2019-12-23 18:43:46
     trx_requested_lock_id: 840:40:3:2
          trx_wait_started: 2019-12-23 18:43:46
                trx_weight: 2
       trx_mysql_thread_id: 46
                 trx_query: DELETE FROM t WHERE id = 10
       trx_operation_state: starting index read
         trx_tables_in_use: 1
         trx_tables_locked: 1
          trx_lock_structs: 2
     trx_lock_memory_bytes: 1136
           trx_rows_locked: 1
         trx_rows_modified: 0
   trx_concurrency_tickets: 0
       trx_isolation_level: REPEATABLE READ
         trx_unique_checks: 1
    trx_foreign_key_checks: 1
trx_last_foreign_key_error: NULL
          trx_is_read_only: 0
trx_autocommit_non_locking: 0
```

1.1.1.2.9.1.1.1.35 Information Schema TEMP_TABLES_INFO Table

MariaDB 10.2.2 - 10.2.3

The `TEMP_TABLES_INFO` table was introduced in MariaDB 10.2.2 and was removed in MariaDB 10.2.4. See MDEV-12459 progress on an alternative.

The Information Schema `TEMP_TABLES_INFO` table contains information about active InnoDB temporary tables. All user and system-created temporary tables are reported when querying this table, with the exception of optimized internal temporary tables. The data is stored in memory.

Previously, InnoDB temp table metadata was rather stored in InnoDB system tables.

It has the following columns:

Column	Description
`TABLE_ID`	Table ID.
`NAME`	Table name.

N_COLS	Number of columns in the temporary table, including three hidden columns that InnoDB creates (DB_ROW_ID , DB_TRX_ID , and DB_ROLL_PTR).
SPACE	Numerical identifier for the tablespace identifier holding the temporary table. Compressed temporary tables are stored by default in separate per-table tablespaces in the temporary file directory. For non-compressed tables, the shared temporary table is named ibtmp1 , found in the data directory. Always a non-zero value, and regenerated on server restart.
PER_TABLE_TABLESPACE	If TRUE , the temporary table resides in a separate per-table tablespace. If FALSE , it resides in the shared temporary tablespace.
IS_COMPRESSED	TRUE if the table is compressed.

The PROCESS privilege is required to view the table.

Examples

```
CREATE TEMPORARY TABLE t (i INT) ENGINE=INNODB;

SELECT * FROM INFORMATION_SCHEMA.INNODB_TEMP_TABLE_INFO;
+----------+--------------+--------+-------+----------------------+----------------+
| TABLE_ID | NAME         | N_COLS | SPACE | PER_TABLE_TABLESPACE | IS_COMPRESSED |
+----------+--------------+--------+-------+----------------------+----------------+
|       39 | #sql1c93_3_1 |      4 |    64 | FALSE                | FALSE          |
+----------+--------------+--------+-------+----------------------+----------------+
```

Adding a compressed table:

```
SET GLOBAL innodb_file_format="Barracuda";

CREATE TEMPORARY TABLE t2 (i INT) ROW_FORMAT=COMPRESSED ENGINE=INNODB;

SELECT * FROM INFORMATION_SCHEMA.INNODB_TEMP_TABLE_INFO;
+----------+--------------+--------+-------+----------------------+----------------+
| TABLE_ID | NAME         | N_COLS | SPACE | PER_TABLE_TABLESPACE | IS_COMPRESSED |
+----------+--------------+--------+-------+----------------------+----------------+
|       40 | #sql1c93_3_3 |      4 |    65 | TRUE                 | TRUE           |
|       39 | #sql1c93_3_1 |      4 |    64 | FALSE                | FALSE          |
+----------+--------------+--------+-------+----------------------+----------------+
```

1.1.1.2.9.1.1.2 Information Schema MyRocks Tables

List of Information Schema tables specifically related to MyRocks.

Information Schema ROCKSDB_CFSTATS Table

The Information Schema ROCKSDB_CFSTATS table is included as part of the MyR...

Information Schema ROCKSDB_CF_OPTIONS Table

Information about MyRocks Column Families.

Information Schema ROCKSDB_COMPACTION_STATS Table

The Information Schema ROCKSDB_COMPACTION_STATS table is included as part o...

Information Schema ROCKSDB_DBSTATS Table

The Information Schema ROCKSDB_DBSTATS table is included as part of the MyR...

Information Schema ROCKSDB_DDL Table

The Information Schema ROCKSDB_DDL table is included as part of the MyRocks...

Information Schema ROCKSDB_DEADLOCK Table

The Information Schema ROCKSDB_DEADLOCK table is included as part of the My...

Information Schema ROCKSDB_GLOBAL_INFO Table

The Information Schema ROCKSDB_GLOBAL_INFO table is included as part of the...

Information Schema ROCKSDB_INDEX_FILE_MAP Table

The Information Schema ROCKSDB_INDEX_FILE_MAP table is included as part of ...

Information Schema ROCKSDB_LOCKS Table

The Information Schema ROCKSDB_LOCKS table is included as part of the MyRoc...

Information Schema ROCKSDB_PERF_CONTEXT Table

Per-table/partition counters.

Information Schema ROCKSDB_PERF_CONTEXT_GLOBAL Table

Global counters.

Information Schema ROCKSDB_SST_PROPS Table

The Information Schema ROCKSDB_SST_PROPS table is included as part of the M...

Information Schema ROCKSDB_TRX Table

The Information Schema ROCKSDB_TRX table is included as part of the MyRocks...

1.1.1.2.9.1.1.2.1 Information Schema ROCKSDB_CFSTATS Table

The Information Schema `ROCKSDB_CFSTATS` table is included as part of the MyRocks storage engine.

The `PROCESS` privilege is required to view the table.

It contains the following columns:

Column	Description
`CF_NAME`	
`STAT_TYPE`	
`VALUE`	

1.1.1.2.9.1.1.2.2 Information Schema ROCKSDB_CF_OPTIONS Table

The Information Schema `ROCKSDB_CF_OPTIONS` table is included as part of the MyRocks storage engine, and contains infomation about MyRocks column families.

The `PROCESS` privilege is required to view the table.

It contains the following columns:

Column	Description
`CF_NAME`	Column family name.
`OPTION_TYPE`	
`VALUE`	

1.1.1.2.9.1.1.2.3 Information Schema ROCKSDB_COMPACTION_STATS Table

The Information Schema `ROCKSDB_COMPACTION_STATS` table is included as part of the MyRocks storage engine.

The `PROCESS` privilege is required to view the table.

It contains the following columns:

Column	Description
CF_NAME	
LEVEL	
TYPE	
VALUE	

1.1.1.2.9.1.1.2.4 Information Schema ROCKSDB_DBSTATS Table

The Information Schema `ROCKSDB_DBSTATS` table is included as part of the MyRocks storage engine.

The `PROCESS` privilege is required to view the table.

It contains the following columns:

Column	Description
STAT_TYPE	
VALUE	

1.1.1.2.9.1.1.2.5 Information Schema ROCKSDB_DDL Table

The Information Schema `ROCKSDB_DDL` table is included as part of the MyRocks storage engine.

The `PROCESS` privilege is required to view the table.

It contains the following columns:

Column	Description
TABLE_SCHEMA	
TABLE_NAME	
PARTITION_NAME	
INDEX_NAME	
COLUMN_FAMILY	
INDEX_NUMBER	
INDEX_TYPE	
KV_FORMAT_VERSION	
TTL_DURATION	
INDEX_FLAGS	
CF	
AUTO_INCREMENT	

1.1.1.2.9.1.1.2.6 Information Schema ROCKSDB_DEADLOCK Table

The Information Schema `ROCKSDB_DEADLOCK` table is included as part of the MyRocks storage engine.

The `PROCESS` privilege is required to view the table.

It contains the following columns:

Column	Description

DEADLOCK_ID	
TIMESTAMP	
TRANSACTION_ID	
CF_NAME	
WAITING_KEY	
LOCK_TYPE	
INDEX_NAME	
TABLE_NAME	
ROLLED_BACK	

1.1.1.2.9.1.1.2.7 Information Schema ROCKSDB_GLOBAL_INFO Table

The Information Schema `ROCKSDB_GLOBAL_INFO` table is included as part of the MyRocks storage engine.

The `PROCESS` privilege is required to view the table.

It contains the following columns:

Column	Description
TYPE	
NAME	
VALUE	

1.1.1.2.9.1.1.2.8 Information Schema ROCKSDB_INDEX_FILE_MAP Table

The Information Schema `ROCKSDB_INDEX_FILE_MAP` table is included as part of the MyRocks storage engine.

The `PROCESS` privilege is required to view the table.

It contains the following columns:

Column	Description
COLUMN_FAMILY	
INDEX_NUMBER	
SST_NAME	
NUM_ROWS	
DATA_SIZE	
ENTRY_DELETES	
ENTRY_SINGLEDELETES	
ENTRY_MERGES	
ENTRY_OTHERS	
DISTINCT_KEYS_PREFIX	

1.1.1.2.9.1.1.2.9 Information Schema ROCKSDB_LOCKS Table

The Information Schema `ROCKSDB_LOCKS` table is included as part of the MyRocks storage engine.

The PROCESS privilege is required to view the table.

It contains the following columns:

Column	Description
COLUMN_FAMILY_ID	
TRANSACTION_ID	
KEY	
MODE	

1.1.1.2.9.1.1.2.10 Information Schema ROCKSDB_PERF_CONTEXT Table

The Information Schema ROCKSDB_PERF_CONTEXT table is included as part of the MyRocks storage engine and includes per-table/partition counters .

The PROCESS privilege is required to view the table.

It contains the following columns:

Column	Description
TABLE_SCHEMA	
TABLE_NAME	
PARTITION_NAME	
STAT_TYPE	
VALUE	

Note: for multi-table queries, all counter increments are "billed" to the first table in the query:
https://github.com/facebook/mysql-5.6/issues/1018

1.1.1.2.9.1.1.2.11 Information Schema ROCKSDB_PERF_CONTEXT_GLOBAL Table

The Information Schema ROCKSDB_PERF_CONTEXT_GLOBAL table is included as part of the MyRocks storage engine and includes global counter information.

The PROCESS privilege is required to view the table.

It contains the following columns:

Column	Description
STAT_TYPE	
VALUE	

1.1.1.2.9.1.1.2.12 Information Schema ROCKSDB_SST_PROPS Table

The Information Schema ROCKSDB_SST_PROPS table is included as part of the MyRocks storage engine.

The PROCESS privilege is required to view the table.

It contains the following columns:

Column	Description
SST_NAME	
COLUMN_FAMILY	

DATA_BLOCKS	
ENTRIES	
RAW_KEY_SIZE	
RAW_VALUE_SIZE	
DATA_BLOCK_SIZE	
INDEX_BLOCK_SIZE	
INDEX_PARTITIONS	
TOP_LEVEL_INDEX_SIZE	
FILTER_BLOCK_SIZE	
COMPRESSION_ALGO	
CREATION_TIME	

1.1.1.2.9.1.1.2.13 Information Schema ROCKSDB_TRX Table

The Information Schema `ROCKSDB_TRX` table is included as part of the MyRocks storage engine.

The `PROCESS` privilege is required to view the table.

It contains the following columns:

Column	Description
TRANSACTION_ID	
STATE	
NAME	
WRITE_COUNT	
LOCK_COUNT	
TIMEOUT_SEC	
WAITING_KEY	
WAITING_COLUMN_FAMILY_ID	
IS_REPLICATION	
SKIP_TRX_API	
READ_ONLY	
HAS_DEADLOCK_DETECTION	
NUM_ONGOING_BULKLOAD	
THREAD_ID	
QUERY	

1.1.1.2.9.1.1.3 ColumnStore Information Schema Tables

MariaDB ColumnStore has four Information Schema tables that expose information about the table and column storage. These tables were added in version 1.0.5 of ColumnStore and were heavily modified for 1.0.6.

COLUMNSTORE_TABLES

The first table is the INFORMATION_SCHEMA.COLUMNSTORE_TABLES. This contains information about the tables inside ColumnStore. The table layout is as follows:

Column	Description
TABLE_SCHEMA	The database schema for the table
TABLE_NAME	The table name
OBJECT_ID	The ColumnStore object ID for the table
CREATION_DATE	The date the table was created
COLUMN_COUNT	The number of columns in the table
AUTOINCREMENT	The start autoincrement value for the table set during CREATE TABLE

Note: Tables created with ColumnStore 1.0.4 or lower will have the year field of the creation data set incorrectly by 1900 years.

COLUMNSTORE_COLUMNS

The INFORMATION_SCHEMA.COLUMNSTORE_COLUMNS table contains information about every single column inside ColumnStore. The table layout is as follows:

Column	Description
TABLE_SCHEMA	The database schema for the table
TABLE_NAME	The table name for the column
COLUMN_NAME	The column name
OBJECT_ID	The object ID for the column
DICTIONARY_OBJECT_ID	The dictionary object ID for the column (NULL if there is no dictionary object
LIST_OBJECT_ID	Placeholder for future information
TREE_OBJECT_ID	Placeholder for future information
DATA_TYPE	The data type for the column
COLUMN_LENGTH	The data length for the column
COLUMN_POSITION	The position of the column in the table, starting at 0
COLUMN_DEFAULT	The default value for the column
IS_NULLABLE	Whether or not the column can be set to NULL
NUMERIC_PRECISION	The numeric precision for the column
NUMERIC_SCALE	The numeric scale for the column
IS_AUTOINCREMENT	Set to 1 if the column is an autoincrement column
COMPRESSION_TYPE	The type of compression (either "None" or "Snappy")

COLUMNSTORE_EXTENTS

This table displays the extent map in a user consumable form. An extent is a collection of details about a section of data related to a columnstore column. A majority of columns in ColumnStore will have multiple extents and the columns table above can be joined to this one to filter results by table or column. The table layout is as follows:

Column	Description
OBJECT_ID	The object ID for the extent
OBJECT_TYPE	Whether this is a "Column" or "Dictionary" extent
LOGICAL_BLOCK_START	ColumnStore's internal start LBID for this extent
LOGICAL_BLOCK_END	ColumnStore's internal end LBID for this extent
MIN_VALUE	This minimum value stored in this extent
MAX_VALUE	The maximum value stored in this extent
WIDTH	The data width for the extent
DBROOT	The DBRoot number for the extent
PARTITION_ID	The parition ID for the extent
SEGMENT_ID	The segment ID for the extent
BLOCK_OFFSET	The block offset for the data file, each data file can contain multiple extents for a column
MAX_BLOCKS	The maximum number of blocks for the extent
HIGH_WATER_MARK	The last block committed to the extent (starting at 0)
STATE	The state of the extent (see below)
STATUS	The availability status for the column which is either "Available", "Unavailable" or "Out of service"
DATA_SIZE	The uncompressed data size for the extent calculated as (HWM + 1) * BLOCK_SIZE

Notes:

1. The state is "Valid" for a normal state, "Invalid" if a cpimport has completed but the table has not yet been accessed (min/max values will be invalid) or "Updating" if there is a DML statement writing to the column
2. In ColumnStore the block size is 8192 bytes
3. By default ColumnStore will write create an extent file of 256*1024*WIDTH bytes for the first partition, if this is too small then for uncompressed data it will create a file of the maximum size for the extent (MAX_BLOCKS * BLOCK_SIZE). Snappy always compression adds a header block.
4. Object IDs of less than 3000 are for internal tables and will not appear in any of the information schema tables
5. Prior to 1.0.12 / 1.1.2 DATA_SIZE was incorrectly calculated
6. HWM is set to zero for the lower segments when there are multiple segments in an extent file, these can be observed when BLOCK_OFFSET > 0
7. When HWM is 0 the DATA_SIZE will show 0 instead of 8192 to avoid confusion when there is multiple segments in an extent file

COLUMNSTORE_FILES

The columnstore_files table provides information about each file associated with extensions. Each extension can reuse a file at different block offsets so this is not a 1:1 relationship to the columnstore_extents table.

Column	Description
OBJECT_ID	The object ID for the extent
SEGMENT_ID	The segment ID for the extent
PARTITION_ID	The partition ID for the extent
FILENAME	The full path and filename for the extent file, multiple extents for the same column can point to this file with different BLOCK_OFFSETs
FILE_SIZE	The disk file size for the extent

COMPRESSED_DATA_SIZE	The amount of the compressed file used, NULL if this is an uncompressed file

Stored Procedures

A few stored procedures were added in 1.0.6 to provide summaries based on the information schema tables. These can be accessed from the COLUMNSTORE_INFO schema.

total_usage()

The total_usage() procedure gives a total disk usage summary for all the columns in ColumnStore with the exception of the columns used for internal maintenance. It is executed using the following query:

```
> call columnstore_info.total_usage();
```

table_usage()

The table_usage() procedure gives a the total data disk usage, dictionary disk usage and grand total disk usage per-table. It can be called in several ways, the first gives a total for each table:

```
> call columnstore_info.table_usage(NULL, NULL);
```

Or for a specific table, my_table in my_schema in this example:

```
> call columnstore_info.table_usage('my_schema', 'my_table');
```

You can also request all tables for a specified schema:

```
> call columnstore_info.table_usage('my_schema', NULL);
```

Note: The quotes around the table name are required, an error will occur without them.

compression_ratio()

The compression_ratio() procedure calculates the average compression ratio across all the compressed extents in ColumnStore. It is called using:

```
> call columnstore_info.compression_ratio();
```

Note: The compression ratio is incorrectly calculated before versions 1.0.12 / 1.1.2

1.1.1.2.9.1.1.4 Information Schema ALL_PLUGINS Table

Description

The Information Schema `ALL_PLUGINS` table contains information about server plugins, whether installed or not.

It contains the following columns:

Column	Description
`PLUGIN_NAME`	Name of the plugin.
`PLUGIN_VERSION`	Version from the plugin's general type descriptor.
`PLUGIN_STATUS`	Plugin status, one of `ACTIVE`, `INACTIVE`, `DISABLED`, `DELETED` or `NOT INSTALLED`.

PLUGIN_TYPE	Plugin type; STORAGE ENGINE, INFORMATION_SCHEMA, AUTHENTICATION, REPLICATION, DAEMON or AUDIT.
PLUGIN_TYPE_VERSION	Version from the plugin's type-specific descriptor.
PLUGIN_LIBRARY	Plugin's shared object file name, located in the directory specified by the plugin_dir system variable, and used by the INSTALL PLUGIN and UNINSTALL PLUGIN statements. NULL if the plugin is complied in and cannot be uninstalled.
PLUGIN_LIBRARY_VERSION	Version from the plugin's API interface.
PLUGIN_AUTHOR	Author of the plugin.
PLUGIN_DESCRIPTION	Description.
PLUGIN_LICENSE	Plugin's licence.
LOAD_OPTION	How the plugin was loaded; one of OFF, ON, FORCE or FORCE_PLUS_PERMANENT. See Installing Plugins.
PLUGIN_MATURITY	Plugin's maturity level; one of Unknown, Experimental, Alpha, Beta, 'Gamma, and Stable.
PLUGIN_AUTH_VERSION	Plugin's version as determined by the plugin author. An example would be '0.99 beta 1'.

It provides a superset of the information shown by the SHOW PLUGINS SONAME statement, as well as the information_schema.PLUGINS table. For specific information about storage engines (a particular type of plugin), see the Information Schema ENGINES table and the SHOW ENGINES statement.

The table is not a standard Information Schema table, and is a MariaDB extension.

Example

```
SELECT * FROM information_schema.all_plugins\G
*************************** 1. row ***************************
           PLUGIN_NAME: binlog
        PLUGIN_VERSION: 1.0
         PLUGIN_STATUS: ACTIVE
           PLUGIN_TYPE: STORAGE ENGINE
   PLUGIN_TYPE_VERSION: 100314.0
        PLUGIN_LIBRARY: NULL
PLUGIN_LIBRARY_VERSION: NULL
         PLUGIN_AUTHOR: MySQL AB
    PLUGIN_DESCRIPTION: This is a pseudo storage engine to represent the binlog in a transaction
        PLUGIN_LICENSE: GPL
           LOAD_OPTION: FORCE
       PLUGIN_MATURITY: Stable
   PLUGIN_AUTH_VERSION: 1.0
*************************** 2. row ***************************
           PLUGIN_NAME: mysql_native_password
        PLUGIN_VERSION: 1.0
         PLUGIN_STATUS: ACTIVE
           PLUGIN_TYPE: AUTHENTICATION
   PLUGIN_TYPE_VERSION: 2.1
        PLUGIN_LIBRARY: NULL
PLUGIN_LIBRARY_VERSION: NULL
         PLUGIN_AUTHOR: R.J.Silk, Sergei Golubchik
    PLUGIN_DESCRIPTION: Native MySQL authentication
        PLUGIN_LICENSE: GPL
           LOAD_OPTION: FORCE
       PLUGIN_MATURITY: Stable
   PLUGIN_AUTH_VERSION: 1.0
*************************** 3. row ***************************
           PLUGIN_NAME: mysql_old_password
        PLUGIN_VERSION: 1.0
         PLUGIN_STATUS: ACTIVE
           PLUGIN_TYPE: AUTHENTICATION
   PLUGIN_TYPE_VERSION: 2.1
        PLUGIN_LIBRARY: NULL
PLUGIN_LIBRARY_VERSION: NULL
         PLUGIN_AUTHOR: R.J.Silk, Sergei Golubchik
    PLUGIN_DESCRIPTION: Old MySQL-4.0 authentication
        PLUGIN_LICENSE: GPL
           LOAD_OPTION: FORCE
       PLUGIN_MATURITY: Stable
   PLUGIN_AUTH_VERSION: 1.0
...
*************************** 104. row ***************************
           PLUGIN_NAME: WSREP_MEMBERSHIP
        PLUGIN_VERSION: 1.0
         PLUGIN_STATUS: NOT INSTALLED
           PLUGIN_TYPE: INFORMATION SCHEMA
   PLUGIN_TYPE_VERSION: 100314.0
        PLUGIN_LIBRARY: wsrep_info.so
PLUGIN_LIBRARY_VERSION: 1.13
         PLUGIN_AUTHOR: Nirbhay Choubey
    PLUGIN_DESCRIPTION: Information about group members
        PLUGIN_LICENSE: GPL
           LOAD_OPTION: OFF
       PLUGIN_MATURITY: Stable
   PLUGIN_AUTH_VERSION: 1.0
*************************** 105. row ***************************
           PLUGIN_NAME: WSREP_STATUS
        PLUGIN_VERSION: 1.0
         PLUGIN_STATUS: NOT INSTALLED
           PLUGIN_TYPE: INFORMATION SCHEMA
   PLUGIN_TYPE_VERSION: 100314.0
        PLUGIN_LIBRARY: wsrep_info.so
PLUGIN_LIBRARY_VERSION: 1.13
         PLUGIN_AUTHOR: Nirbhay Choubey
    PLUGIN_DESCRIPTION: Group view information
        PLUGIN_LICENSE: GPL
           LOAD_OPTION: OFF
       PLUGIN_MATURITY: Stable
```

1.1.1.2.9.1.1.5 Information Schema APPLICABLE_ROLES Table

The Information Schema `APPLICABLE_ROLES` table shows the role authorizations that the current user may use.

It contains the following columns:

Column	Description	Added
`GRANTEE`	Account that the role was granted to.	
`ROLE_NAME`	Name of the role.	
`IS_GRANTABLE`	Whether the role can be granted or not.	
`IS_DEFAULT`	Whether the role is the user's default role or not	MariaDB 10.1.3

The current role is in the ENABLED_ROLES Information Schema table.

Example

```
SELECT * FROM information_schema.APPLICABLE_ROLES;
+----------------+-------------+--------------+------------+
| GRANTEE        | ROLE_NAME   | IS_GRANTABLE | IS_DEFAULT |
+----------------+-------------+--------------+------------+
| root@localhost | journalist  | YES          | NO         |
| root@localhost | staff       | YES          | NO         |
| root@localhost | dd          | YES          | NO         |
| root@localhost | dog         | YES          | NO         |
+----------------+-------------+--------------+------------+
```

1.1.1.2.9.1.1.6 Information Schema CHARACTER_SETS Table

The Information Schema `CHARACTER_SETS` table contains a list of supported character sets, their default collations and maximum lengths.

It contains the following columns:

Column	Description
`CHARACTER_SET_NAME`	Name of the character set.
`DEFAULT_COLLATE_NAME`	Default collation used.
`DESCRIPTION`	Character set description.
`MAXLEN`	Maximum length.

The SHOW CHARACTER SET statement returns the same results (although in a different order), and both can be refined in the same way. For example, the following two statements return the same results:

```
SHOW CHARACTER SET WHERE Maxlen LIKE '2';
```

and

```
SELECT * FROM information_schema.CHARACTER_SETS
WHERE MAXLEN LIKE '2';
```

See Setting Character Sets and Collations for details on specifying the character set at the server, database, table and column levels, and Supported Character Sets and Collations for a full list of supported characters sets and collations.

Example

```
SELECT CHARACTER_SET_NAME FROM information_schema.CHARACTER_SETS
WHERE DEFAULT_COLLATE_NAME LIKE '%chinese%';
+--------------------+
| CHARACTER_SET_NAME |
+--------------------+
| big5               |
| gb2312             |
| gbk                |
+--------------------+
```

1.1.1.2.9.1.1.7 Information Schema CHECK_CONSTRAINTS Table

The Information Schema `CHECK_CONSTRAINTS` table stores metadata about the constraints defined for tables in all databases.

It contains the following columns:

Column	Description
`CONSTRAINT_CATALOG`	Always contains the string 'def'.
`CONSTRAINT_SCHEMA`	Database name.
`CONSTRAINT_NAME`	Constraint name.
`TABLE_NAME`	Table name.
`LEVEL`	Type of the constraint ('Column' or 'Table'). From MariaDB 10.5.10
`CHECK_CLAUSE`	Constraint clause.

Example

A table with a numeric table check constraint and with a default check constraint name:

```
CREATE TABLE t ( a int, CHECK (a>10));
```

To see check constraint call `check_constraints` table from information schema.

```
SELECT * from INFORMATION_SCHEMA.CHECK_CONSTRAINTS\G
```

```
*************************** 1. row ***************************
CONSTRAINT_CATALOG: def
 CONSTRAINT_SCHEMA: test
   CONSTRAINT_NAME: CONSTRAINT_1
        TABLE_NAME: t
      CHECK_CLAUSE: `a` > 10
```

A new table check constraint called `a_upper` :

```
ALTER TABLE t ADD CONSTRAINT a_upper CHECK (a<100);
```

```
SELECT * from INFORMATION_SCHEMA.CHECK_CONSTRAINTS\G
```

```
*************************** 1. row ***************************
CONSTRAINT_CATALOG: def
 CONSTRAINT_SCHEMA: test
   CONSTRAINT_NAME: CONSTRAINT_1
        TABLE_NAME: t
      CHECK_CLAUSE: `a` > 10
*************************** 2. row ***************************
CONSTRAINT_CATALOG: def
 CONSTRAINT_SCHEMA: test
   CONSTRAINT_NAME: a_upper
        TABLE_NAME: t
      CHECK_CLAUSE: `a` < 100
```

A new table `tt` with a field check constraint called `b`, as well as a table check constraint called `b_upper`:

```
CREATE TABLE tt(b int CHECK(b>0),CONSTRAINT b_upper CHECK(b<50));

SELECT * from INFORMATION_SCHEMA.CHECK_CONSTRAINTS;
+--------------------+-------------------+-----------------+------------+--------------+
| CONSTRAINT_CATALOG | CONSTRAINT_SCHEMA | CONSTRAINT_NAME | TABLE_NAME | CHECK_CLAUSE |
+--------------------+-------------------+-----------------+------------+--------------+
| def                | test              | b               | tt         | `b` > 0      |
| def                | test              | b_upper         | tt         | `b` < 50     |
| def                | test              | CONSTRAINT_1    | t          | `a` > 10     |
| def                | test              | a_upper         | t          | `a` < 100    |
+--------------------+-------------------+-----------------+------------+--------------+
```

Note: The name of the field constraint is the same as the field name.

After dropping the default table constraint called `CONSTRAINT_1`:

```
ALTER TABLE t DROP CONSTRAINT CONSTRAINT_1;

SELECT * from INFORMATION_SCHEMA.CHECK_CONSTRAINTS;
+--------------------+-------------------+-----------------+------------+--------------+
| CONSTRAINT_CATALOG | CONSTRAINT_SCHEMA | CONSTRAINT_NAME | TABLE_NAME | CHECK_CLAUSE |
+--------------------+-------------------+-----------------+------------+--------------+
| def                | test              | b               | tt         | `b` > 0      |
| def                | test              | b_upper         | tt         | `b` < 50     |
| def                | test              | a_upper         | t          | `a` < 100    |
+--------------------+-------------------+-----------------+------------+--------------+
```

Trying to insert invalid arguments into table `t` and `tt` generates an error.

```
INSERT INTO t VALUES (10),(20),(100);
ERROR 4025 (23000): CONSTRAINT `a_upper` failed for `test`.`t`

INSERT INTO tt VALUES (10),(-10),(100);
ERROR 4025 (23000): CONSTRAINT `b` failed for `test`.`tt`

INSERT INTO tt VALUES (10),(20),(100);
ERROR 4025 (23000): CONSTRAINT `b_upper` failed for `test`.`tt`
```

From MariaDB 10.5.10:

```
create table majra(check(x>0), x int, y int check(y < 0), z int,
                           constraint z check(z>0), constraint xyz check(x<10 and y<10 and
z<10));
Query OK, 0 rows affected (0.036 sec)

show create table majra;
+-------+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
----------------------------------+
| Table | Create Table
|
+-------+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
----------------------------------+
| majra | CREATE TABLE `majra` (
  `x` int(11) DEFAULT NULL,
  `y` int(11) DEFAULT NULL CHECK (`y` < 0),
  `z` int(11) DEFAULT NULL,
  CONSTRAINT `CONSTRAINT_1` CHECK (`x` > 0),
  CONSTRAINT `z` CHECK (`z` > 0),
  CONSTRAINT `xyz` CHECK (`x` < 10 and `y` < 10 and `z` < 10)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+-------+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
----------------------------------+
1 row in set (0.000 sec)


select * from information_schema.check_constraints where table_name='majra';
+--------------------+--------------------+------------+-----------------+--------+-----------
-------------------------+
| CONSTRAINT_CATALOG | CONSTRAINT_SCHEMA | TABLE_NAME | CONSTRAINT_NAME | LEVEL  | CHECK_CLAUSE
|
+--------------------+--------------------+------------+-----------------+--------+-----------
-------------------------+
| def                | test               | majra      | y               | Column | `y` < 0
|
| def                | test               | majra      | CONSTRAINT_1    | Table  | `x` > 0
|
| def                | test               | majra      | z               | Table  | `z` > 0
|
| def                | test               | majra      | xyz             | Table  | `x` < 10
and `y` < 10 and `z` < 10 |
+--------------------+--------------------+------------+-----------------+--------+-----------
-------------------------+
4 rows in set (0.001 sec)
```

1.1.1.2.9.1.1.8 Information Schema CLIENT_STATISTICS Table

The Information Schema `CLIENT_STATISTICS` table holds statistics about client connections. This is part of the User Statistics feature, which is not enabled by default.

It contains the following columns:

Field	Type	Notes
`CLIENT`	`VARCHAR(64)`	The IP address or hostname the connection originated from.
`TOTAL_CONNECTIONS`	`INT(21)`	The number of connections created for this client.
`CONCURRENT_CONNECTIONS`	`INT(21)`	The number of concurrent connections for this client.
`CONNECTED_TIME`	`INT(21)`	The cumulative number of seconds elapsed while there were connections from this client.

BUSY_TIME	DOUBLE	The cumulative number of seconds there was activity on connections from this client.
CPU_TIME	DOUBLE	The cumulative CPU time elapsed while servicing this client's connections. Note that this number may be wrong on SMP system if there was a CPU migration for the thread during the execution of the query.
BYTES_RECEIVED	INT(21)	The number of bytes received from this client's connections.
BYTES_SENT	INT(21)	The number of bytes sent to this client's connections.
BINLOG_BYTES_WRITTEN	INT(21)	The number of bytes written to the binary log from this client's connections.
ROWS_READ	INT(21)	The number of rows read by this client's connections.
ROWS_SENT	INT(21)	The number of rows sent by this client's connections.
ROWS_DELETED	INT(21)	The number of rows deleted by this client's connections.
ROWS_INSERTED	INT(21)	The number of rows inserted by this client's connections.
ROWS_UPDATED	INT(21)	The number of rows updated by this client's connections.
SELECT_COMMANDS	INT(21)	The number of `SELECT` commands executed from this client's connections.
UPDATE_COMMANDS	INT(21)	The number of `UPDATE` commands executed from this client's connections.
OTHER_COMMANDS	INT(21)	The number of other commands executed from this client's connections.
COMMIT_TRANSACTIONS	INT(21)	The number of `COMMIT` commands issued by this client's connections.
ROLLBACK_TRANSACTIONS	INT(21)	The number of `ROLLBACK` commands issued by this client's connections.
DENIED_CONNECTIONS	INT(21)	The number of connections denied to this client.
LOST_CONNECTIONS	INT(21)	The number of this client's connections that were terminated uncleanly.
ACCESS_DENIED	INT(21)	The number of times this client's connections issued commands that were denied.
EMPTY_QUERIES	INT(21)	The number of times this client's connections sent queries that returned no results to the server.
TOTAL_SSL_CONNECTIONS	INT(21)	The number of TLS connections created for this client. (>= MariaDB 10.1.1)
MAX_STATEMENT_TIME_EXCEEDED	INT(21)	The number of times a statement was aborted, because it was executed longer than its `MAX_STATEMENT_TIME` threshold. (>= MariaDB 10.1.1)

Example

```
SELECT * FROM information_schema.CLIENT_STATISTICS\G
*************************** 1. row ***************************
                CLIENT: localhost
     TOTAL_CONNECTIONS: 3
CONCURRENT_CONNECTIONS: 0
        CONNECTED_TIME: 4883
             BUSY_TIME: 0.009722
              CPU_TIME: 0.0102131
        BYTES_RECEIVED: 841
            BYTES_SENT: 13897
  BINLOG_BYTES_WRITTEN: 0
             ROWS_READ: 0
             ROWS_SENT: 214
          ROWS_DELETED: 0
         ROWS_INSERTED: 207
          ROWS_UPDATED: 0
       SELECT_COMMANDS: 10
       UPDATE_COMMANDS: 0
        OTHER_COMMANDS: 13
   COMMIT_TRANSACTIONS: 0
 ROLLBACK_TRANSACTIONS: 0
    DENIED_CONNECTIONS: 0
      LOST_CONNECTIONS: 0
         ACCESS_DENIED: 0
         EMPTY_QUERIES: 1
```

1.1.1.2.9.1.1.9 Information Schema COLLATION_CHARACTER_SET_APPLICABILIT Table

The Information Schema `COLLATION_CHARACTER_SET_APPLICABILITY` table shows which character sets are associated with which collations.

It contains the following columns:

Column	Description
`COLLATION_NAME`	Collation name.
`CHARACTER_SET_NAME`	Name of the associated character set.

`COLLATION_CHARACTER_SET_APPLICABILITY` is essentially a subset of the `COLLATIONS` table.

```
SELECT COLLATION_NAME,CHARACTER_SET_NAME FROM information_schema.COLLATIONS;
```

and

```
SELECT * FROM information_schema.COLLATION_CHARACTER_SET_APPLICABILITY;
```

will return identical results.

See Setting Character Sets and Collations for details on specifying the character set at the server, database, table and column levels.

Example

```
SELECT * FROM information_schema.COLLATION_CHARACTER_SET_APPLICABILITY
WHERE CHARACTER_SET_NAME='utf32';
+---------------------+--------------------+
| COLLATION_NAME      | CHARACTER_SET_NAME |
+---------------------+--------------------+
| utf32_general_ci    | utf32              |
| utf32_bin           | utf32              |
| utf32_unicode_ci    | utf32              |
| utf32_icelandic_ci  | utf32              |
| utf32_latvian_ci    | utf32              |
| utf32_romanian_ci   | utf32              |
| utf32_slovenian_ci  | utf32              |
| utf32_polish_ci     | utf32              |
| utf32_estonian_ci   | utf32              |
| utf32_spanish_ci    | utf32              |
| utf32_swedish_ci    | utf32              |
| utf32_turkish_ci    | utf32              |
| utf32_czech_ci      | utf32              |
| utf32_danish_ci     | utf32              |
| utf32_lithuanian_ci | utf32              |
| utf32_slovak_ci     | utf32              |
| utf32_spanish2_ci   | utf32              |
| utf32_roman_ci      | utf32              |
| utf32_persian_ci    | utf32              |
| utf32_esperanto_ci  | utf32              |
| utf32_hungarian_ci  | utf32              |
| utf32_sinhala_ci    | utf32              |
| utf32_german2_ci    | utf32              |
| utf32_croatian_ci   | utf32              |
+---------------------+--------------------+
```

1.1.1.2.9.1.1.10 Information Schema COLLATIONS Table

Contents

The Information Schema `COLLATIONS` table contains a list of supported collations.

It contains the following columns:

Column	Description
`COLLATION_NAME`	Name of the collation.
`CHARACTER_SET_NAME`	Associated character set.
`ID`	Collation id.
`IS_DEFAULT`	Whether the collation is the character set's default.
`IS_COMPILED`	Whether the collation is compiled into the server.
`SORTLEN`	Sort length, used for determining the memory used to sort strings in this collation.

The SHOW COLLATION statement returns the same results and both can be reduced in a similar way.

For example, in MariaDB 10.6, the following two statements return the same results:

```
SHOW COLLATION WHERE Charset LIKE 'utf8mb3';
```

and

```
SELECT * FROM information_schema.COLLATIONS
WHERE CHARACTER_SET_NAME LIKE 'utf8mb3';
```

In MariaDB 10.5 and before, `utf8` should be specified instead of `utf8mb3`.

NO PAD collations

`NO PAD` collations regard trailing spaces as normal characters. You can get a list of all `NO PAD` collations as follows:

```
SELECT collation_name FROM information_schema.COLLATIONS
WHERE collation_name LIKE "%nopad%";
+----------------------------+
| collation_name             |
+----------------------------+
| big5_chinese_nopad_ci      |
| big5_nopad_bin             |
...
```

Example

```
SELECT * FROM information_schema.COLLATIONS;
+----------------------------+--------------------+------+------------+-------------+---------+
| COLLATION_NAME             | CHARACTER_SET_NAME | ID   | IS_DEFAULT | IS_COMPILED | SORTLEN |
+----------------------------+--------------------+------+------------+-------------+---------+
| big5_chinese_ci            | big5               |    1 | Yes        | Yes         |       1 |
| big5_bin                   | big5               |   84 |            | Yes         |       1 |
| big5_chinese_nopad_ci      | big5               | 1025 |            | Yes         |       1 |
| big5_nopad_bin             | big5               | 1108 |            | Yes         |       1 |
| dec8_swedish_ci            | dec8               |    3 | Yes        | Yes         |       1 |
| dec8_bin                   | dec8               |   69 |            | Yes         |       1 |
| dec8_swedish_nopad_ci      | dec8               | 1027 |            | Yes         |       1 |
| dec8_nopad_bin             | dec8               | 1093 |            | Yes         |       1 |
| cp850_general_ci           | cp850              |    4 | Yes        | Yes         |       1 |
| cp850_bin                  | cp850              |   80 |            | Yes         |       1 |
...
```

1.1.1.2.9.1.1.11 Information Schema COLUMN_PRIVILEGES Table

The Information Schema `COLUMN_PRIVILEGES` table contains column privilege information derived from the `mysql.columns_priv` grant table.

It has the following columns:

Column	Description
`GRANTEE`	In the format `user_name@host_name`.
`TABLE_CATALOG`	Always `def`.
`TABLE_SCHEMA`	Database name.
`TABLE_NAME`	Table name.
`COLUMN_NAME`	Column name.
`PRIVILEGE_TYPE`	One of `SELECT`, `INSERT`, `UPDATE` or `REFERENCES`.
`IS_GRANTABLE`	Whether the user has the `GRANT OPTION` for this privilege.

Similar information can be accessed with the `SHOW FULL COLUMNS` and `SHOW GRANTS` statements. See the `GRANT` article for more about privileges.

This information is also stored in the `columns_priv` table, in the `mysql` system database.

For a description of the privileges that are shown in this table, see column privileges.

Example

In the following example, no column-level privilege has been explicitly assigned:

```
SELECT * FROM information_schema.COLUMN_PRIVILEGES;
Empty set
```

1.1.1.2.9.1.1.12 Information Schema COLUMNS Table

The Information Schema `COLUMNS` table provides information about columns in each table on the server.

It contains the following columns:

Column	Description
`TABLE_CATALOG`	Always contains the string 'def'.
`TABLE_SCHEMA`	Database name.
`TABLE_NAME`	Table name.
`COLUMN_NAME`	Column name.
`ORDINAL_POSITION`	Column position in the table. Can be used for ordering.
`COLUMN_DEFAULT`	Default value for the column. Literals are quoted to distinguish them from expressions. `NULL` means that the column has no default. In MariaDB 10.2.6 and earlier, no quotes were used for any type of default and `NULL` can either mean that there is no default, or that the default column value is `NULL`.
`IS_NULLABLE`	Whether the column can contain `NULL`s.
`DATA_TYPE`	The column's data type.
`CHARACTER_MAXIMUM_LENGTH`	Maximum length.
`CHARACTER_OCTET_LENGTH`	Same as the `CHARACTER_MAXIMUM_LENGTH` except for multi-byte character sets.
`NUMERIC_PRECISION`	For numeric types, the precision (number of significant digits) for the column. `NULL` if not a numeric field.
`NUMERIC_SCALE`	For numeric types, the scale (significant digits to the right of the decimal point). NULL if not a numeric field.
`DATETIME_PRECISION`	Fractional-seconds precision, or `NULL` if not a time data type.
`CHARACTER_SET_NAME`	Character set if a non-binary string data type, otherwise NULL.
`COLLATION_NAME`	Collation if a non-binary string data type, otherwise NULL.
`COLUMN_TYPE`	Column definition, a MySQL and MariaDB extension.
`COLUMN_KEY`	Index type. `PRI` for primary key, `UNI` for unique index, `MUL` for multiple index. A MySQL and MariaDB extension.
`EXTRA`	Additional information about a column, for example whether the column is an invisible column, or `WITHOUT SYSTEM VERSIONING` if the table is not a system-versioned table. A MySQL and MariaDB extension.
`PRIVILEGES`	Which privileges you have for the column. A MySQL and MariaDB extension.
`COLUMN_COMMENT`	Column comments.
`IS_GENERATED`	Indicates whether the column value is generated (virtual, or computed). Can be `ALWAYS` or `NEVER`.
`GENERATION_EXPRESSION`	The expression used for computing the column value in a generated (virtual, or computed) column.
`IS_SYSTEM_TIME_PERIOD_START`	From MariaDB 11.3.0.
`IS_SYSTEM_TIME_PERIOD_END`	From MariaDB 11.3.0.

It provides information similar to, but more complete, than SHOW COLUMNS and mariadb-show.

Examples

```
SELECT * FROM information_schema.COLUMNS\G
...
*************************** 9. row ***************************
           TABLE_CATALOG: def
            TABLE_SCHEMA: test
              TABLE_NAME: t2
             COLUMN_NAME: j
        ORDINAL_POSITION: 1
          COLUMN_DEFAULT: NULL
             IS_NULLABLE: YES
               DATA_TYPE: longtext
CHARACTER_MAXIMUM_LENGTH: 4294967295
  CHARACTER_OCTET_LENGTH: 4294967295
       NUMERIC_PRECISION: NULL
           NUMERIC_SCALE: NULL
      DATETIME_PRECISION: NULL
      CHARACTER_SET_NAME: utf8mb4
          COLLATION_NAME: utf8mb4_bin
             COLUMN_TYPE: longtext
              COLUMN_KEY:
                   EXTRA:
              PRIVILEGES: select,insert,update,references
          COLUMN_COMMENT:
            IS_GENERATED: NEVER
   GENERATION_EXPRESSION: NULL
...
```

```
CREATE TABLE t (
  s1 VARCHAR(20) DEFAULT 'ABC',
  s2 VARCHAR(20) DEFAULT (concat('A','B')),
  s3 VARCHAR(20) DEFAULT ("concat('A','B')"),
  s4 VARCHAR(20),
  s5 VARCHAR(20) DEFAULT NULL,
  s6 VARCHAR(20) NOT NULL,
  s7 VARCHAR(20) DEFAULT 'NULL' NULL,
  s8 VARCHAR(20) DEFAULT 'NULL' NOT NULL
);

SELECT
  table_name,
  column_name,
  ordinal_position,
  column_default,
  column_default IS NULL
FROM information_schema.COLUMNS
WHERE table_schema=DATABASE()
AND TABLE_NAME='t';
+------------+-------------+------------------+----------------------+------------------------+
| table_name | column_name | ordinal_position | column_default       | column_default IS NULL |
+------------+-------------+------------------+----------------------+------------------------+
| t          | s1          |                1 | 'ABC'                |                      0 |
| t          | s2          |                2 | concat('A','B')      |                      0 |
| t          | s3          |                3 | 'concat(''A'',''B'')' |                      0 |
| t          | s4          |                4 | NULL                 |                      0 |
| t          | s5          |                5 | NULL                 |                      0 |
| t          | s6          |                6 | NULL                 |                      1 |
| t          | s7          |                7 | 'NULL'               |                      0 |
| t          | s8          |                8 | 'NULL'               |                      0 |
+------------+-------------+------------------+----------------------+------------------------+
```

In the results above, the two single quotes in `concat(''A'',''B'')` indicate an escaped single quote - see string-literals. Note that while mariadb client appears to show the same default value for columns `s5` and `s6`, the first is a 4-character string "NULL", while the second is the SQL NULL value.

From MariaDB 11.3:

```
CREATE TABLE t(
     x INT,
     start_timestamp TIMESTAMP(6) GENERATED ALWAYS AS ROW START,
     end_timestamp TIMESTAMP(6) GENERATED ALWAYS AS ROW END,
     PERIOD FOR SYSTEM_TIME(start_timestamp, end_timestamp)
) WITH SYSTEM VERSIONING;

SELECT TABLE_NAME, COLUMN_NAME, ORDINAL_POSITION,
  IS_SYSTEM_TIME_PERIOD_START, IS_SYSTEM_TIME_PERIOD_END
  FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME='t'\G
*************************** 1. row ***************************
                TABLE_NAME: t
               COLUMN_NAME: x
          ORDINAL_POSITION: 1
IS_SYSTEM_TIME_PERIOD_START: NO
  IS_SYSTEM_TIME_PERIOD_END: NO
*************************** 2. row ***************************
                TABLE_NAME: t
               COLUMN_NAME: start_timestamp
          ORDINAL_POSITION: 2
IS_SYSTEM_TIME_PERIOD_START: YES
  IS_SYSTEM_TIME_PERIOD_END: NO
*************************** 3. row ***************************
                TABLE_NAME: t
               COLUMN_NAME: end_timestamp
          ORDINAL_POSITION: 3
IS_SYSTEM_TIME_PERIOD_START: NO
  IS_SYSTEM_TIME_PERIOD_END: YES
```

1.1.1.2.9.1.1.13 Information Schema DISKS Table

Contents

Description

The `DISKS` table is created when the DISKS plugin is enabled, and shows metadata about disks on the system.

Before MariaDB 10.4.7, MariaDB 10.3.17, MariaDB 10.2.26 and MariaDB 10.1.41, this plugin did **not** check user privileges. When it is enabled, **any** user can query the `INFORMATION_SCHEMA.DISKS` table and see all the information it provides.

Since MariaDB 10.4.7, MariaDB 10.3.17, MariaDB 10.2.26 and MariaDB 10.1.41, it requires the FILE privilege.

The plugin only works on Linux.

The table contains the following columns:

Column	Description
`DISK`	Name of the disk itself.
`PATH`	Mount point of the disk.
`TOTAL`	Total space in KiB.
`USED`	Used amount of space in KiB.
`AVAILABLE`	Amount of space in KiB available to non-root users.

Note that as the amount of space available to root (OS user) may be more that what is available to non-root users, 'available' + 'used' may be less than 'total'.

All paths to which a particular disk has been mounted are reported. The rationale is that someone might want to take different action e.g. depending on which disk is relevant for a particular path. This leads to the same disk being reported multiple times.

Example

```
SELECT * FROM information_schema.DISKS;

+-----------+-------+----------+---------+-----------+
| Disk      | Path  | Total    | Used    | Available |
+-----------+-------+----------+---------+-----------+
| /dev/vda1 | /     | 26203116 | 2178424 |  24024692 |
| /dev/vda1 | /boot | 26203116 | 2178424 |  24024692 |
| /dev/vda1 | /etc  | 26203116 | 2178424 |  24024692 |
+-----------+-------+----------+---------+-----------+
```

1.1.1.2.9.1.1.14 Information Schema ENABLED_ROLES Table

The Information Schema `ENABLED_ROLES` table shows the enabled roles for the current session.

It contains the following column:

Column	Description
`ROLE_NAME`	The enabled role name, or `NULL`.

This table lists all roles that are currently enabled, one role per row — the current role, roles granted to the current role, roles granted to these roles and so on. If no role is set, the row contains a `NULL` value.

The roles that the current user can enable are listed in the APPLICABLE_ROLES Information Schema table.

See also CURRENT_ROLE().

Examples

```
SELECT * FROM information_schema.ENABLED_ROLES;
+-----------+
| ROLE_NAME |
+-----------+
| NULL      |
+-----------+

SET ROLE staff;

SELECT * FROM information_schema.ENABLED_ROLES;
+-----------+
| ROLE_NAME |
+-----------+
| staff     |
+-----------+
```

1.1.1.2.9.1.1.15 Information Schema ENGINES Table

The Information Schema `ENGINES` table displays status information about the server's storage engines.

It contains the following columns:

Column	Description
`ENGINE`	Name of the storage engine.
`SUPPORT`	Whether the engine is the default, or is supported or not.
`COMMENT`	Storage engine comments.
`TRANSACTIONS`	Whether or not the engine supports transactions.
`XA`	Whether or not the engine supports XA transactions.

SAVEPOINTS	Whether or not savepoints are supported.

It provides identical information to the `SHOW ENGINES` statement. Since storage engines are plugins, different information about them is also shown in the `information_schema.PLUGINS` table and by the `SHOW PLUGINS` statement.

The table is not a standard Information Schema table, and is a MySQL and MariaDB extension.

Note that both MySQL's InnoDB and Percona's XtraDB replacement are labeled as `InnoDB`. However, if XtraDB is in use, it will be specified in the `COMMENT` field. See XtraDB and InnoDB. The same applies to FederatedX.

Example

```
SELECT * FROM information_schema.ENGINES\G;
*************************** 1. row ***************************
      ENGINE: InnoDB
     SUPPORT: DEFAULT
     COMMENT: Supports transactions, row-level locking, and foreign keys
TRANSACTIONS: YES
          XA: YES
  SAVEPOINTS: YES
*************************** 2. row ***************************
      ENGINE: CSV
     SUPPORT: YES
     COMMENT: CSV storage engine
TRANSACTIONS: NO
          XA: NO
  SAVEPOINTS: NO
*************************** 3. row ***************************
      ENGINE: MyISAM
     SUPPORT: YES
     COMMENT: MyISAM storage engine
TRANSACTIONS: NO
          XA: NO
  SAVEPOINTS: NO
*************************** 4. row ***************************
      ENGINE: BLACKHOLE
     SUPPORT: YES
     COMMENT: /dev/null storage engine (anything you write to it disappears)
TRANSACTIONS: NO
          XA: NO
  SAVEPOINTS: NO
*************************** 5. row ***************************
      ENGINE: FEDERATED
     SUPPORT: YES
     COMMENT: FederatedX pluggable storage engine
TRANSACTIONS: YES
          XA: NO
  SAVEPOINTS: YES
*************************** 6. row ***************************
      ENGINE: MRG_MyISAM
     SUPPORT: YES
     COMMENT: Collection of identical MyISAM tables
TRANSACTIONS: NO
          XA: NO
  SAVEPOINTS: NO
*************************** 7. row ***************************
      ENGINE: ARCHIVE
     SUPPORT: YES
     COMMENT: Archive storage engine
TRANSACTIONS: NO
          XA: NO
  SAVEPOINTS: NO
*************************** 8. row ***************************
      ENGINE: MEMORY
     SUPPORT: YES
     COMMENT: Hash based, stored in memory, useful for temporary tables
TRANSACTIONS: NO
          XA: NO
  SAVEPOINTS: NO
*************************** 9. row ***************************
      ENGINE: PERFORMANCE_SCHEMA
     SUPPORT: YES
     COMMENT: Performance Schema
TRANSACTIONS: NO
          XA: NO
  SAVEPOINTS: NO
*************************** 10. row ***************************
      ENGINE: Aria
     SUPPORT: YES
     COMMENT: Crash-safe tables with MyISAM heritage
TRANSACTIONS: NO
          XA: NO
  SAVEPOINTS: NO
10 rows in set (0.00 sec)
```

Check if a given storage engine is available:

```
SELECT SUPPORT FROM information_schema.ENGINES WHERE ENGINE LIKE 'tokudb';
Empty set
```

Check which storage engine supports XA transactions:

```
SELECT ENGINE FROM information_schema.ENGINES WHERE XA = 'YES';
+--------+
| ENGINE |
+--------+
| InnoDB |
+--------+
```

1.1.1.2.9.1.1.16 Information Schema EVENTS Table

The Information Schema `EVENTS` table stores information about Events on the server.

It contains the following columns:

Column	Description
`EVENT_CATALOG`	Always `def`.
`EVENT_SCHEMA`	Database where the event was defined.
`EVENT_NAME`	Event name.
`DEFINER`	Event definer.
`TIME_ZONE`	Time zone used for the event's scheduling and execution, by default `SYSTEM`.
`EVENT_BODY`	`SQL`.
`EVENT_DEFINITION`	The SQL defining the event.
`EVENT_TYPE`	Either `ONE TIME` or `RECURRING`.
`EXECUTE_AT`	`DATETIME` when the event is set to execute, or `NULL` if recurring.
`INTERVAL_VALUE`	Numeric interval between event executions for a recurring event, or `NULL` if not recurring.
`INTERVAL_FIELD`	Interval unit (e.g., `HOUR`)
`SQL_MODE`	The `SQL_MODE` at the time the event was created.
`STARTS`	Start `DATETIME` for a recurring event, `NULL` if not defined or not recurring.
`ENDS`	End `DATETIME` for a recurring event, `NULL` if not defined or not recurring.
`STATUS`	One of `ENABLED`, `DISABLED` or / `SLAVESIDE_DISABLED`.
`ON_COMPLETION`	The `ON COMPLETION` clause, either `PRESERVE` or `NOT PRESERVE`.
`CREATED`	When the event was created.
`LAST_ALTERED`	When the event was last changed.
`LAST_EXECUTED`	When the event was last run.
`EVENT_COMMENT`	The comment provided in the `CREATE EVENT` statement, or an empty string if none.
`ORIGINATOR`	MariaDB server ID on which the event was created.
`CHARACTER_SET_CLIENT`	`character_set_client` system variable session value at the time the event was created.
`COLLATION_CONNECTION`	collation_connection system variable session value at the time the event was created.
`DATABASE_COLLATION`	Database collation with which the event is linked.

The `SHOW EVENTS` and `SHOW CREATE EVENT` statements provide similar information.

1.1.1.2.9.1.1.17 Information Schema FEEDBACK Table

The Information Schema `FEEDBACK` table is created when the Feedback Plugin is enabled, and contains the complete contents submitted by the plugin.

It contains two columns:

Column	Description
`VARIABLE_NAME`	Name of the item of information being collected.
`VARIABLE_VALUE`	Contents of the item of information being collected.

It is possible to disable automatic collection, by setting the `feedback_url` variable to an empty string, and to submit the contents manually, as follows:

```
$ mysql -e 'SELECT * FROM information_schema.FEEDBACK' > report.txt
```

Then you can send it by opening `https://mariadb.org/feedback_plugin/post` in your browser, and uploading your generated `report.txt`. Or you can do it from the command line with (for example):

```
$ curl -F data=@report.txt https://mariadb.org/feedback_plugin/post
```

Manual uploading allows you to be absolutely sure that we receive only the data shown in the `information_schema.FEEDBACK` table and that no private or sensitive information is being sent.

Example

```
SELECT * FROM information_schema.FEEDBACK\G
...
*************************** 906. row ***************************
 VARIABLE_NAME: Uname_sysname
VARIABLE_VALUE: Linux
*************************** 907. row ***************************
 VARIABLE_NAME: Uname_release
VARIABLE_VALUE: 3.13.0-53-generic
*************************** 908. row ***************************
 VARIABLE_NAME: Uname_version
VARIABLE_VALUE: #89-Ubuntu SMP Wed May 20 10:34:39 UTC 2015
*************************** 909. row ***************************
 VARIABLE_NAME: Uname_machine
VARIABLE_VALUE: x86_64
*************************** 910. row ***************************
 VARIABLE_NAME: Uname_distribution
VARIABLE_VALUE: lsb: Ubuntu 14.04.2 LTS
*************************** 911. row ***************************
 VARIABLE_NAME: Collation used latin1_german1_ci
VARIABLE_VALUE: 1
*************************** 912. row ***************************
 VARIABLE_NAME: Collation used latin1_swedish_ci
VARIABLE_VALUE: 18
*************************** 913. row ***************************
 VARIABLE_NAME: Collation used utf8_general_ci
VARIABLE_VALUE: 567
*************************** 914. row ***************************
 VARIABLE_NAME: Collation used latin1_bin
VARIABLE_VALUE: 1
*************************** 915. row ***************************
 VARIABLE_NAME: Collation used binary
VARIABLE_VALUE: 16
*************************** 916. row ***************************
 VARIABLE_NAME: Collation used utf8_bin
VARIABLE_VALUE: 4044
```

1.1.1.2.9.1.1.18 Information Schema FILES

Table

The `FILES` tables is unused in MariaDB. See MDEV-11426.

1.1.1.2.9.1.1.19 Information Schema GEOMETRY_COLUMNS Table

Description

The Information Schema `GEOMETRY_COLUMNS` table provides support for Spatial Reference systems for GIS data.

It contains the following columns:

Column	Type	Null	Description
`F_TABLE_CATALOG`	`VARCHAR(512)`	NO	Together with `F_TABLE_SCHEMA` and `F_TABLE_NAME`, the fully qualified name of the featured table containing the geometry column.
`F_TABLE_SCHEMA`	`VARCHAR(64)`	NO	Together with `F_TABLE_CATALOG` and `F_TABLE_NAME`, the fully qualified name of the featured table containing the geometry column.
`F_TABLE_NAME`	`VARCHAR(64)`	NO	Together with `F_TABLE_CATALOG` and `F_TABLE_SCHEMA`, the fully qualified name of the featured table containing the geometry column.
`F_GEOMETRY_COLUMN`	`VARCHAR(64)`	NO	Name of the column in the featured table that is the geometry golumn.
`G_TABLE_CATALOG`	`VARCHAR(512)`	NO	
`G_TABLE_SCHEMA`	`VARCHAR(64)`	NO	Database name of the table implementing the geometry column.
`G_TABLE_NAME`	`VARCHAR(64)`	NO	Table name that is implementing the geometry column.
`G_GEOMETRY_COLUMN`	`VARCHAR(64)`	NO	
`STORAGE_TYPE`	`TINYINT(2)`	NO	Binary geometry implementation. Always 1 in MariaDB.
`GEOMETRY_TYPE`	`INT(7)`	NO	Integer reflecting the type of geometry stored in this column (see table below).
`COORD_DIMENSION`	`TINYINT(2)`	NO	Number of dimensions in the spatial reference system. Always 2 in MariaDB.
`MAX_PPR`	`TINYINT(2)`	NO	Always 0 in MariaDB.
`SRID`	`SMALLINT(5)`	NO	ID of the Spatial Reference System used for the coordinate geometry in this table. It is a foreign key reference to the `SPATIAL_REF_SYS table`.

Storage_type

The integers in the `storage_type` field match the geometry types as follows:

Integer	Type
0	`GEOMETRY`
1	`POINT`
3	`LINESTRING`
5	`POLYGON`
7	`MULTIPOINT`
9	`MULTILINESTRING`
11	`MULTIPOLYGON`

Example

```
CREATE TABLE g1(g GEOMETRY(9,4) REF_SYSTEM_ID=101);

SELECT * FROM information_schema.GEOMETRY_COLUMNS\G
*************************** 1. row ***************************
  F_TABLE_CATALOG: def
   F_TABLE_SCHEMA: test
     F_TABLE_NAME: g1
F_GEOMETRY_COLUMN:
  G_TABLE_CATALOG: def
   G_TABLE_SCHEMA: test
     G_TABLE_NAME: g1
G_GEOMETRY_COLUMN: g
     STORAGE_TYPE: 1
    GEOMETRY_TYPE: 0
  COORD_DIMENSION: 2
          MAX_PPR: 0
             SRID: 101
```

1.1.1.2.9.1.1.20 Information Schema GLOBAL_STATUS and SESSION_STATUS Tables

The Information Schema `GLOBAL_STATUS` and `SESSION_STATUS` tables store a record of all status variables and their global and session values respectively. This is the same information as displayed by the `SHOW STATUS` commands `SHOW GLOBAL STATUS` and `SHOW SESSION STATUS`.

They contain the following columns:

Column	Description
`VARIABLE_NAME`	Status variable name.
`VARIABLE_VALUE`	Global or session value.

Example

```
SELECT * FROM information_schema.GLOBAL_STATUS;
+-----------------------------------------------+--------------------+
| VARIABLE_NAME                                 | VARIABLE_VALUE     |
+-----------------------------------------------+--------------------+
...
| BINLOG_SNAPSHOT_FILE                          | mariadb-bin.000208 |
| BINLOG_SNAPSHOT_POSITION                      | 369                |
...
| THREADS_CONNECTED                             | 1                  |
| THREADS_CREATED                               | 1                  |
| THREADS_RUNNING                               | 1                  |
| UPTIME                                        | 57358              |
| UPTIME_SINCE_FLUSH_STATUS                     | 57358              |
+-----------------------------------------------+--------------------+
```

1.1.1.2.9.1.1.21 Information Schema GLOBAL_VARIABLES and SESSION_VARIABLES Tables

The Information Schema `GLOBAL_VARIABLES` and `SESSION_VARIABLES` tables stores a record of all system variables and their global and session values respectively. This is the same information as displayed by the `SHOW VARIABLES` commands `SHOW GLOBAL VARIABLES` and `SHOW SESSION VARIABLES`.

It contains the following columns:

Column	Description

VARIABLE_NAME	System variable name.
VARIABLE_VALUE	Global or session value.

Example

```
SELECT * FROM information_schema.GLOBAL_VARIABLES ORDER BY VARIABLE_NAME\G
*************************** 1. row ***************************
 VARIABLE_NAME: ARIA_BLOCK_SIZE
VARIABLE_VALUE: 8192
*************************** 2. row ***************************
 VARIABLE_NAME: ARIA_CHECKPOINT_LOG_ACTIVITY
VARIABLE_VALUE: 1048576
*************************** 3. row ***************************
 VARIABLE_NAME: ARIA_CHECKPOINT_INTERVAL
VARIABLE_VALUE: 30
...
*************************** 455. row ***************************
 VARIABLE_NAME: VERSION_COMPILE_MACHINE
VARIABLE_VALUE: x86_64
*************************** 456. row ***************************
 VARIABLE_NAME: VERSION_COMPILE_OS
VARIABLE_VALUE: debian-linux-gnu
*************************** 457. row ***************************
 VARIABLE_NAME: WAIT_TIMEOUT
VARIABLE_VALUE: 600
```

1.1.1.2.9.1.1.22 Information Schema INDEX_STATISTICS Table

The Information Schema `INDEX_STATISTICS` table shows statistics on index usage and makes it possible to do such things as locating unused indexes and generating the commands to remove them.

This is part of the User Statistics feature, which is not enabled by default.

It contains the following columns:

Field	Type	Notes
TABLE_SCHEMA	VARCHAR(192)	The schema (database) name.
TABLE_NAME	VARCHAR(192)	The table name.
INDEX_NAME	VARCHAR(192)	The index name (as visible in `SHOW CREATE TABLE`).
ROWS_READ	INT(21)	The number of rows read from this index.

Example

```
SELECT * FROM information_schema.INDEX_STATISTICS
WHERE TABLE_NAME = "author";
+--------------+------------+------------+-----------+
| TABLE_SCHEMA | TABLE_NAME | INDEX_NAME | ROWS_READ |
+--------------+------------+------------+-----------+
| books        | author     | by_name    |        15 |
+--------------+------------+------------+-----------+
```

1.1.1.2.9.1.1.23 Information Schema KEY_CACHES Table

The Information Schema `KEY_CACHES` table shows statistics about the segmented key cache,.

It contains the following columns:

Column Name	Description
KEY_CACHE_NAME	The name of the key cache
SEGMENTS	total number of segments (set to NULL for regular key caches)
SEGMENT_NUMBER	segment number (set to NULL for any regular key caches and for rows containing aggregation statistics for segmented key caches)
FULL_SIZE	memory for cache buffers/auxiliary structures
BLOCK_SIZE	size of the blocks
USED_BLOCKS	number of currently used blocks
UNUSED_BLOCKS	number of currently unused blocks
DIRTY_BLOCKS	number of currently dirty blocks
READ_REQUESTS	number of read requests
READS	number of actual reads from files into buffers
WRITE_REQUESTS	number of write requests
WRITES	number of actual writes from buffers into files

Example

```
SELECT * FROM information_schema.KEY_CACHES \G
*********************** 1. row ***********************
KEY_CACHE_NAME: default
SEGMENTS: NULL
SEGMENT_NUMBER: NULL
     FULL_SIZE: 134217728
    BLOCK_SIZE: 1024
   USED_BLOCKS: 36
 UNUSED_BLOCKS: 107146
  DIRTY_BLOCKS: 0
 READ_REQUESTS: 40305
         READS: 21
WRITE_REQUESTS: 19239
        WRITES: 358
```

1.1.1.2.9.1.1.24 Information Schema KEY_COLUMN_USAGE Table

The Information Schema KEY_COLUMN_USAGE table shows which key columns have constraints.

It contains the following columns:

Column	Description
CONSTRAINT_CATALOG	Always def .
CONSTRAINT_SCHEMA	Database name of the constraint.
CONSTRAINT_NAME	Name of the constraint (PRIMARY for the primary key).
TABLE_CATALOG	Always #def .
TABLE_SCHEMA	Database name of the column constraint.
TABLE_NAME	Table name of the column constraint.
COLUMN_NAME	Column name of the constraint.
ORDINAL_POSITION	Position of the column within the constraint.
POSITION_IN_UNIQUE_CONSTRAINT	For foreign keys, the position in the unique constraint.
REFERENCED_TABLE_SCHEMA	For foreign keys, the referenced database name.

REFERENCED_TABLE_NAME	For foreign keys, the referenced table name.
REFERENCED_COLUMN_NAME	For foreign keys, the referenced column name.

Example

```
SELECT * FROM information_schema.KEY_COLUMN_USAGE LIMIT 1 \G
*********************** 1. row ***********************
           CONSTRAINT_CATALOG: def
            CONSTRAINT_SCHEMA: my_website
              CONSTRAINT_NAME: PRIMARY
                TABLE_CATALOG: def
                 TABLE_SCHEMA: users
                  COLUMN_NAME: user_id
             ORDINAL_POSITION: 1
POSITION_IN_UNIQUE_CONSTRAINT: NULL
      REFERENCED_TABLE_SCHEMA: NULL
        REFERENCED_TABLE_NAME: NULL
       REFERENCED_COLUMN_NAME: NULL
```

1.1.1.2.9.1.1.25 Information Schema KEY_PERIOD_USAGE Table

MariaDB starting with 11.3.0

The Information Schema `KEY_PERIOD_USAGE` table shows information about Application-Time Periods.

It contains the following columns:

Column	Description
CONSTRAINT_CATALOG	
CONSTRAINT_SCHEMA	
CONSTRAINT_NAME	
TABLE_CATALOG	
TABLE_SCHEMA	
TABLE_NAME	
PERIOD_NAME	

Example

1.1.1.2.9.1.1.26 Information Schema KEYWORDS Table

MariaDB starting with 10.6.3

The `KEYWORDS` table was added in MariaDB 10.6.3.

Description

The Information Schema `KEYWORDS` table contains the list of MariaDB keywords.

It contains a single column:

Column	Description
WORD	Keyword

The table is not a standard Information Schema table, and is a MariaDB extension.

Example

```
SELECT * FROM INFORMATION_SCHEMA.KEYWORDS;
+-------------------------------+
| WORD                          |
+-------------------------------+
| &&                            |
| <=                            |
| <>                            |
| !=                            |
| >=                            |
| <<                            |
| >>                            |
| <=>                           |
| ACCESSIBLE                    |
| ACCOUNT                       |
| ACTION                        |
| ADD                           |
| ADMIN                         |
| AFTER                         |
| AGAINST                       |
| AGGREGATE                     |
| ALL                           |
| ALGORITHM                     |
| ALTER                         |
| ALWAYS                        |
| ANALYZE                       |
| AND                           |
| ANY                           |
| AS                            |
| ASC                           |
| ASCII                         |
| ASENSITIVE                    |
| AT                            |
| ATOMIC                        |
| AUTHORS                       |
| AUTO_INCREMENT                |
| AUTOEXTEND_SIZE               |
| AUTO                          |
| AVG                           |
| AVG_ROW_LENGTH                |
| BACKUP                        |
| BEFORE                        |
| BEGIN                         |
| BETWEEN                       |
| BIGINT                        |
| BINARY                        |
| BINLOG                        |
| BIT                           |
| BLOB                          |
| BLOCK                         |
| BODY                          |
| BOOL                          |
| BOOLEAN                       |
| BOTH                          |
| BTREE                         |
| BY                            |
| BYTE                          |
| CACHE                         |
| CALL                          |
| CASCADE                       |
| CASCADED                      |
| CASE                          |
| CATALOG_NAME                  |
| CHAIN                         |
| CHANGE                        |
| CHANGED                       |
| CHAR                          |
| CHARACTER                     |
| CHARSET                       |
| CHECK                         |
```

| CHECKPOINT |
| CHECKSUM |
| CIPHER |
| CLASS_ORIGIN |
| CLIENT |
| CLOB |
| CLOSE |
| COALESCE |
| CODE |
| COLLATE |
| COLLATION |
| COLUMN |
| COLUMN_NAME |
| COLUMNS |
| COLUMN_ADD |
| COLUMN_CHECK |
| COLUMN_CREATE |
| COLUMN_DELETE |
| COLUMN_GET |
| COMMENT |
| COMMIT |
| COMMITTED |
| COMPACT |
| COMPLETION |
| COMPRESSED |
| CONCURRENT |
| CONDITION |
| CONNECTION |
| CONSISTENT |
| CONSTRAINT |
| CONSTRAINT_CATALOG |
| CONSTRAINT_NAME |
| CONSTRAINT_SCHEMA |
| CONTAINS |
| CONTEXT |
| CONTINUE |
| CONTRIBUTORS |
| CONVERT |
| CPU |
| CREATE |
| CROSS |
| CUBE |
| CURRENT |
| CURRENT_DATE |
| CURRENT_POS |
| CURRENT_ROLE |
| CURRENT_TIME |
| CURRENT_TIMESTAMP |
| CURRENT_USER |
| CURSOR |
| CURSOR_NAME |
| CYCLE |
| DATA |
| DATABASE |
| DATABASES |
| DATAFILE |
| DATE |
| DATETIME |
| DAY |
| DAY_HOUR |
| DAY_MICROSECOND |
| DAY_MINUTE |
| DAY_SECOND |
| DEALLOCATE |
| DEC |
| DECIMAL |
| DECLARE |
| DEFAULT |
| DEFINER |
| DELAYED |
| DELAY_KEY_WRITE |
| DELETE |
| DELETE_DOMAIN_ID |
| DESC |
| DESCRIBE |

```
| DES_KEY_FILE                  |
| DETERMINISTIC                 |
| DIAGNOSTICS                   |
| DIRECTORY                     |
| DISABLE                       |
| DISCARD                       |
| DISK                          |
| DISTINCT                      |
| DISTINCTROW                   |
| DIV                           |
| DO                            |
| DOUBLE                        |
| DO_DOMAIN_IDS                 |
| DROP                          |
| DUAL                          |
| DUMPFILE                      |
| DUPLICATE                     |
| DYNAMIC                       |
| EACH                          |
| ELSE                          |
| ELSEIF                        |
| ELSIF                         |
| EMPTY                         |
| ENABLE                        |
| ENCLOSED                      |
| END                           |
| ENDS                          |
| ENGINE                        |
| ENGINES                       |
| ENUM                          |
| ERROR                         |
| ERRORS                        |
| ESCAPE                        |
| ESCAPED                       |
| EVENT                         |
| EVENTS                        |
| EVERY                         |
| EXAMINED                      |
| EXCEPT                        |
| EXCHANGE                      |
| EXCLUDE                       |
| EXECUTE                       |
| EXCEPTION                     |
| EXISTS                        |
| EXIT                          |
| EXPANSION                     |
| EXPIRE                        |
| EXPORT                        |
| EXPLAIN                       |
| EXTENDED                      |
| EXTENT_SIZE                   |
| FALSE                         |
| FAST                          |
| FAULTS                        |
| FEDERATED                     |
| FETCH                         |
| FIELDS                        |
| FILE                          |
| FIRST                         |
| FIXED                         |
| FLOAT                         |
| FLOAT4                        |
| FLOAT8                        |
| FLUSH                         |
| FOLLOWING                     |
| FOLLOWS                       |
| FOR                           |
| FORCE                         |
| FOREIGN                       |
| FORMAT                        |
| FOUND                         |
| FROM                          |
| FULL                          |
| FULLTEXT                      |
| FUNCTION                      |
```

| GENERAL |
| GENERATED |
| GET_FORMAT |
| GET |
| GLOBAL |
| GOTO |
| GRANT |
| GRANTS |
| GROUP |
| HANDLER |
| HARD |
| HASH |
| HAVING |
| HELP |
| HIGH_PRIORITY |
| HISTORY |
| HOST |
| HOSTS |
| HOUR |
| HOUR_MICROSECOND |
| HOUR_MINUTE |
| HOUR_SECOND |
| ID |
| IDENTIFIED |
| IF |
| IGNORE |
| IGNORED |
| IGNORE_DOMAIN_IDS |
| IGNORE_SERVER_IDS |
| IMMEDIATE |
| IMPORT |
| INTERSECT |
| IN |
| INCREMENT |
| INDEX |
| INDEXES |
| INFILE |
| INITIAL_SIZE |
| INNER |
| INOUT |
| INSENSITIVE |
| INSERT |
| INSERT_METHOD |
| INSTALL |
| INT |
| INT1 |
| INT2 |
| INT3 |
| INT4 |
| INT8 |
| INTEGER |
| INTERVAL |
| INVISIBLE |
| INTO |
| IO |
| IO_THREAD |
| IPC |
| IS |
| ISOLATION |
| ISOPEN |
| ISSUER |
| ITERATE |
| INVOKER |
| JOIN |
| JSON |
| JSON_TABLE |
| KEY |
| KEYS |
| KEY_BLOCK_SIZE |
| KILL |
| LANGUAGE |
| LAST |
| LAST_VALUE |
| LASTVAL |
| LEADING |

| LEADING |
| LEAVE |
| LEAVES |
| LEFT |
| LESS |
| LEVEL |
| LIKE |
| LIMIT |
| LINEAR |
| LINES |
| LIST |
| LOAD |
| LOCAL |
| LOCALTIME |
| LOCALTIMESTAMP |
| LOCK |
| LOCKED |
| LOCKS |
| LOGFILE |
| LOGS |
| LONG |
| LONGBLOB |
| LONGTEXT |
| LOOP |
| LOW_PRIORITY |
| MASTER |
| MASTER_CONNECT_RETRY |
| MASTER_DELAY |
| MASTER_GTID_POS |
| MASTER_HOST |
| MASTER_LOG_FILE |
| MASTER_LOG_POS |
| MASTER_PASSWORD |
| MASTER_PORT |
| MASTER_SERVER_ID |
| MASTER_SSL |
| MASTER_SSL_CA |
| MASTER_SSL_CAPATH |
| MASTER_SSL_CERT |
| MASTER_SSL_CIPHER |
| MASTER_SSL_CRL |
| MASTER_SSL_CRLPATH |
| MASTER_SSL_KEY |
| MASTER_SSL_VERIFY_SERVER_CERT |
| MASTER_USER |
| MASTER_USE_GTID |
| MASTER_HEARTBEAT_PERIOD |
| MATCH |
| MAX_CONNECTIONS_PER_HOUR |
| MAX_QUERIES_PER_HOUR |
| MAX_ROWS |
| MAX_SIZE |
| MAX_STATEMENT_TIME |
| MAX_UPDATES_PER_HOUR |
| MAX_USER_CONNECTIONS |
| MAXVALUE |
| MEDIUM |
| MEDIUMBLOB |
| MEDIUMINT |
| MEDIUMTEXT |
| MEMORY |
| MERGE |
| MESSAGE_TEXT |
| MICROSECOND |
| MIDDLEINT |
| MIGRATE |
| MINUS |
| MINUTE |
| MINUTE_MICROSECOND |
| MINUTE_SECOND |
| MINVALUE |
| MIN_ROWS |
| MOD |
| MODE |
| MODIFIES |
| MODIFY |

| MODIFY |
| MONITOR |
| MONTH |
| MUTEX |
| MYSQL |
| MYSQL_ERRNO |
| NAME |
| NAMES |
| NATIONAL |
| NATURAL |
| NCHAR |
| NESTED |
| NEVER |
| NEW |
| NEXT |
| NEXTVAL |
| NO |
| NOMAXVALUE |
| NOMINVALUE |
| NOCACHE |
| NOCYCLE |
| NO_WAIT |
| NOWAIT |
| NODEGROUP |
| NONE |
| NOT |
| NOTFOUND |
| NO_WRITE_TO_BINLOG |
| NULL |
| NUMBER |
| NUMERIC |
| NVARCHAR |
| OF |
| OFFSET |
| OLD_PASSWORD |
| ON |
| ONE |
| ONLINE |
| ONLY |
| OPEN |
| OPTIMIZE |
| OPTIONS |
| OPTION |
| OPTIONALLY |
| OR |
| ORDER |
| ORDINALITY |
| OTHERS |
| OUT |
| OUTER |
| OUTFILE |
| OVER |
| OVERLAPS |
| OWNER |
| PACKAGE |
| PACK_KEYS |
| PAGE |
| PAGE_CHECKSUM |
| PARSER |
| PARSE_VCOL_EXPR |
| PATH |
| PERIOD |
| PARTIAL |
| PARTITION |
| PARTITIONING |
| PARTITIONS |
| PASSWORD |
| PERSISTENT |
| PHASE |
| PLUGIN |
| PLUGINS |
| PORT |
| PORTION |
| PRECEDES |
| PRECEDING |

| PRECISION |
| PREPARE |
| PRESERVE |
| PREV |
| PREVIOUS |
| PRIMARY |
| PRIVILEGES |
| PROCEDURE |
| PROCESS |
| PROCESSLIST |
| PROFILE |
| PROFILES |
| PROXY |
| PURGE |
| QUARTER |
| QUERY |
| QUICK |
| RAISE |
| RANGE |
| RAW |
| READ |
| READ_ONLY |
| READ_WRITE |
| READS |
| REAL |
| REBUILD |
| RECOVER |
| RECURSIVE |
| REDO_BUFFER_SIZE |
| REDOFILE |
| REDUNDANT |
| REFERENCES |
| REGEXP |
| RELAY |
| RELAYLOG |
| RELAY_LOG_FILE |
| RELAY_LOG_POS |
| RELAY_THREAD |
| RELEASE |
| RELOAD |
| REMOVE |
| RENAME |
| REORGANIZE |
| REPAIR |
| REPEATABLE |
| REPLACE |
| REPLAY |
| REPLICA |
| REPLICAS |
| REPLICA_POS |
| REPLICATION |
| REPEAT |
| REQUIRE |
| RESET |
| RESIGNAL |
| RESTART |
| RESTORE |
| RESTRICT |
| RESUME |
| RETURNED_SQLSTATE |
| RETURN |
| RETURNING |
| RETURNS |
| REUSE |
| REVERSE |
| REVOKE |
| RIGHT |
| RLIKE |
| ROLE |
| ROLLBACK |
| ROLLUP |
| ROUTINE |
| ROW |
| ROWCOUNT |
| ROWNUM |

| ROWS |
| ROWTYPE |
| ROW_COUNT |
| ROW_FORMAT |
| RTREE |
| SAVEPOINT |
| SCHEDULE |
| SCHEMA |
| SCHEMA_NAME |
| SCHEMAS |
| SECOND |
| SECOND_MICROSECOND |
| SECURITY |
| SELECT |
| SENSITIVE |
| SEPARATOR |
| SEQUENCE |
| SERIAL |
| SERIALIZABLE |
| SESSION |
| SERVER |
| SET |
| SETVAL |
| SHARE |
| SHOW |
| SHUTDOWN |
| SIGNAL |
| SIGNED |
| SIMPLE |
| SKIP |
| SLAVE |
| SLAVES |
| SLAVE_POS |
| SLOW |
| SNAPSHOT |
| SMALLINT |
| SOCKET |
| SOFT |
| SOME |
| SONAME |
| SOUNDS |
| SOURCE |
| STAGE |
| STORED |
| SPATIAL |
| SPECIFIC |
| REF_SYSTEM_ID |
| SQL |
| SQLEXCEPTION |
| SQLSTATE |
| SQLWARNING |
| SQL_BIG_RESULT |
| SQL_BUFFER_RESULT |
| SQL_CACHE |
| SQL_CALC_FOUND_ROWS |
| SQL_NO_CACHE |
| SQL_SMALL_RESULT |
| SQL_THREAD |
| SQL_TSI_SECOND |
| SQL_TSI_MINUTE |
| SQL_TSI_HOUR |
| SQL_TSI_DAY |
| SQL_TSI_WEEK |
| SQL_TSI_MONTH |
| SQL_TSI_QUARTER |
| SQL_TSI_YEAR |
| SSL |
| START |
| STARTING |
| STARTS |
| STATEMENT |
| STATS_AUTO_RECALC |
| STATS_PERSISTENT |
| STATS_SAMPLE_PAGES |
| STATUS |

```
| STOP                          |
| STORAGE                       |
| STRAIGHT_JOIN                 |
| STRING                        |
| SUBCLASS_ORIGIN               |
| SUBJECT                       |
| SUBPARTITION                  |
| SUBPARTITIONS                 |
| SUPER                         |
| SUSPEND                       |
| SWAPS                         |
| SWITCHES                      |
| SYSDATE                       |
| SYSTEM                        |
| SYSTEM_TIME                   |
| TABLE                         |
| TABLE_NAME                    |
| TABLES                        |
| TABLESPACE                    |
| TABLE_CHECKSUM                |
| TEMPORARY                     |
| TEMPTABLE                     |
| TERMINATED                    |
| TEXT                          |
| THAN                          |
| THEN                          |
| TIES                          |
| TIME                          |
| TIMESTAMP                     |
| TIMESTAMPADD                  |
| TIMESTAMPDIFF                 |
| TINYBLOB                      |
| TINYINT                       |
| TINYTEXT                      |
| TO                            |
| TRAILING                      |
| TRANSACTION                   |
| TRANSACTIONAL                 |
| THREADS                       |
| TRIGGER                       |
| TRIGGERS                      |
| TRUE                          |
| TRUNCATE                      |
| TYPE                          |
| TYPES                         |
| UNBOUNDED                     |
| UNCOMMITTED                   |
| UNDEFINED                     |
| UNDO_BUFFER_SIZE              |
| UNDOFILE                      |
| UNDO                          |
| UNICODE                       |
| UNION                         |
| UNIQUE                        |
| UNKNOWN                       |
| UNLOCK                        |
| UNINSTALL                     |
| UNSIGNED                      |
| UNTIL                         |
| UPDATE                        |
| UPGRADE                       |
| USAGE                         |
| USE                           |
| USER                          |
| USER_RESOURCES                |
| USE_FRM                       |
| USING                         |
| UTC_DATE                      |
| UTC_TIME                      |
| UTC_TIMESTAMP                 |
| VALUE                         |
| VALUES                        |
| VARBINARY                     |
| VARCHAR                       |
| VARCHARACTER                  |
```

```
| VARCHAR2                       |
| VARIABLES                      |
| VARYING                        |
| VIA                            |
| VIEW                           |
| VIRTUAL                        |
| VISIBLE                        |
| VERSIONING                     |
| WAIT                           |
| WARNINGS                       |
| WEEK                           |
| WEIGHT_STRING                  |
| WHEN                           |
| WHERE                          |
| WHILE                          |
| WINDOW                         |
| WITH                           |
| WITHIN                         |
| WITHOUT                        |
| WORK                           |
| WRAPPER                        |
| WRITE                          |
| X509                           |
| XOR                            |
| XA                             |
| XML                            |
| YEAR                           |
| YEAR_MONTH                     |
| ZEROFILL                       |
| ||                             |
+--------------------------------+
694 rows in set (0.000 sec)
```

1.1.1.2.9.1.1.27 Information Schema LOCALES Table

Description

The Information Schema `LOCALES` table contains a list of all compiled-in locales. It is only available if the LOCALES plugin has been installed.

It contains the following columns:

Column	Description
`ID`	Row ID.
`NAME`	Locale name, for example `en_GB`.
`DESCRIPTION`	Locale description, for example `English - United Kingdom`.
`MAX_MONTH_NAME_LENGTH`	Numeric length of the longest month in the locale
`MAX_DAY_NAME_LENGTH`	Numeric length of the longest day name in the locale.
`DECIMAL_POINT`	Decimal point character (some locales use a comma).
`THOUSAND_SEP`	Thousand's character separator,
`ERROR_MESSAGE_LANGUAGE`	Error message language.

The table is not a standard Information Schema table, and is a MariaDB extension.

The SHOW LOCALES statement returns a subset of the information.

Example

```
SELECT * FROM information_schema.LOCALES;
+-----+-------+--------------------------------------+----------------------+----------------
-----+---------------+-------------+------------------------+
| ID  | NAME  | DESCRIPTION                          | MAX_MONTH_NAME_LENGTH |
MAX_DAY_NAME_LENGTH | DECIMAL_POINT | THOUSAND_SEP | ERROR_MESSAGE_LANGUAGE |
+-----+-------+--------------------------------------+----------------------+----------------
-----+---------------+-------------+------------------------+
|   0 | en_US | English - United States              |                    9 |
9 | .             | ,           | english                |
|   1 | en_GB | English - United Kingdom             |                    9 |
9 | .             | ,           | english                |
|   2 | ja_JP | Japanese - Japan                     |                    3 |
3 | .             | ,           | japanese               |
|   3 | sv_SE | Swedish - Sweden                     |                    9 |
7 | ,             |             | swedish                |
|   4 | de_DE | German - Germany                     |                    9 |
10 | ,            | .           | german                  |
|   5 | fr_FR | French - France                      |                    9 |
8 | ,             |             | french                 |
|   6 | ar_AE | Arabic - United Arab Emirates        |                    6 |
8 | .             | ,           | english                |
|   7 | ar_BH | Arabic - Bahrain                     |                    6 |
8 | .             | ,           | english                |
|   8 | ar_JO | Arabic - Jordan                      |                   12 |
8 | .             | ,           | english                |
...
| 106 | no_NO | Norwegian - Norway                   |                    9 |
7 | ,             | .           | norwegian              |
| 107 | sv_FI | Swedish - Finland                    |                    9 |
7 | ,             |             | swedish                |
| 108 | zh_HK | Chinese - Hong Kong SAR              |                    3 |
3 | .             | ,           | english                |
| 109 | el_GR | Greek - Greece                       |                   11 |
9 | ,             | .           | greek                  |
+-----+-------+--------------------------------------+----------------------+----------------
-----+---------------+-------------+------------------------+
```

1.1.1.2.9.1.1.28 Information Schema METADATA_LOCK_INFO Table

The Information Schema `METADATA_LOCK_INFO` table is created by the metadata_lock_info plugin. It shows active metadata locks and user locks (the locks acquired with GET_LOCK).

It has the following columns:

Column	Description
`THREAD_ID`	
`LOCK_MODE`	One of `MDL_INTENTION_EXCLUSIVE`, `MDL_SHARED`, `MDL_SHARED_HIGH_PRIO`, `MDL_SHARED_READ`, `MDL_SHARED_READ_ONLY`, `MDL_SHARED_WRITE`, `MDL_SHARED_NO_WRITE`, `MDL_SHARED_NO_READ_WRITE`, `MDL_SHARED_UPGRADABLE` or `MDL_EXCLUSIVE`.
`LOCK_DURATION`	One of `MDL_STATEMENT`, `MDL_TRANSACTION` or `MDL_EXPLICIT`
`LOCK_TYPE`	One of `Global read lock`, `Schema metadata lock`, `Table metadata lock`, `Stored function metadata lock`, `Stored procedure metadata lock`, `Trigger metadata lock`, `Event metadata lock`, `Commit lock` or `User lock`.
`TABLE_SCHEMA`	
`TABLE_NAME`	

"LOCK_MODE" Descriptions

The `LOCK_MODE` column can have the following values:

Value	Description

MDL_INTENTION_EXCLUSIVE	An intention exclusive metadata lock (IX). Used only for scoped locks. Owner of this type of lock can acquire upgradable exclusive locks on individual objects. Compatible with other IX locks, but is incompatible with scoped S and X locks. IX lock is taken in SCHEMA namespace when we intend to modify object metadata. Object may refer table, stored procedure, trigger, view/etc.
MDL_SHARED	A shared metadata lock (S). To be used in cases when we are interested in object metadata only and there is no intention to access object data (e.g. for stored routines or during preparing prepared statements). We also mis-use this type of lock for open HANDLERs, since lock acquired by this statement has to be compatible with lock acquired by LOCK TABLES ... WRITE statement, i.e. SNRW (We can't get by by acquiring S lock at HANDLER ... OPEN time and upgrading it to SR lock for HANDLER ... READ as it doesn't solve problem with need to abort DML statements which wait on table level lock while having open HANDLER in the same connection). To avoid deadlock which may occur when SNRW lock is being upgraded to X lock for table on which there is an active S lock which is owned by thread which waits in its turn for table-level lock owned by thread performing upgrade we have to use thr_abort_locks_for_thread() facility in such situation. This problem does not arise for locks on stored routines as we don't use SNRW locks for them. It also does not arise when S locks are used during PREPARE calls as table-level locks are not acquired in this case. This lock is taken for global read lock, when caching a stored procedure in memory for the duration of the transaction and for tables used by prepared statements.
MDL_SHARED_HIGH_PRIO	A high priority shared metadata lock. Used for cases when there is no intention to access object data (i.e. data in the table). "High priority" means that, unlike other shared locks, it is granted ignoring pending requests for exclusive locks. Intended for use in cases when we only need to access metadata and not data, e.g. when filling an INFORMATION_SCHEMA table. Since SH lock is compatible with SNRW lock, the connection that holds SH lock lock should not try to acquire any kind of table-level or row-level lock, as this can lead to a deadlock. Moreover, after acquiring SH lock, the connection should not wait for any other resource, as it might cause starvation for X locks and a potential deadlock during upgrade of SNW or SNRW to X lock (e.g. if the upgrading connection holds the resource that is being waited for).
MDL_SHARED_READ	A shared metadata lock (SR) for cases when there is an intention to read data from table. A connection holding this kind of lock can read table metadata and read table data (after acquiring appropriate table and row-level locks). This means that one can only acquire TL_READ, TL_READ_NO_INSERT, and similar table-level locks on table if one holds SR MDL lock on it. To be used for tables in SELECTs, subqueries, and LOCK TABLE ... READ statements.
MDL_SHARED_WRITE	A shared metadata lock (SW) for cases when there is an intention to modify (and not just read) data in the table. A connection holding SW lock can read table metadata and modify or read table data (after acquiring appropriate table and row-level locks). To be used for tables to be modified by INSERT, UPDATE, DELETE statements, but not LOCK TABLE ... WRITE or DDL). Also taken by SELECT ... FOR UPDATE.
MDL_SHARED_UPGRADABLE	An upgradable shared metadata lock for cases when there is an intention to modify (and not just read) data in the table. Can be upgraded to MDL_SHARED_NO_WRITE and MDL_EXCLUSIVE. A connection holding SU lock can read table metadata and modify or read table data (after acquiring appropriate table and row-level locks). To be used for the first phase of ALTER TABLE.
MDL_SHARED_READ_ONLY	A shared metadata lock for cases when we need to read data from table and block all concurrent modifications to it (for both data and metadata). Used by LOCK TABLES READ statement.
MDL_SHARED_NO_WRITE	An upgradable shared metadata lock which blocks all attempts to update table data, allowing reads. A connection holding this kind of lock can read table metadata and read table data. Can be upgraded to X metadata lock. Note, that since this type of lock is not compatible with SNRW or SW lock types, acquiring appropriate engine-level locks for reading (TL_READ* for MyISAM, shared row locks in InnoDB) should be contention-free. To be used for the first phase of ALTER TABLE, when copying data between tables, to allow concurrent SELECTs from the table, but not UPDATEs.

MDL_SHARED_NO_READ_WRITE	An upgradable shared metadata lock which allows other connections to access table metadata, but not data. It blocks all attempts to read or update table data, while allowing INFORMATION_SCHEMA and SHOW queries. A connection holding this kind of lock can read table metadata modify and read table data. Can be upgraded to X metadata lock. To be used for LOCK TABLES WRITE statement. Not compatible with any other lock type except S and SH.
MDL_EXCLUSIVE	An exclusive metadata lock (X). A connection holding this lock can modify both table's metadata and data. No other type of metadata lock can be granted while this lock is held. To be used for CREATE/DROP/RENAME TABLE statements and for execution of certain phases of other DDL statements.

Examples

User lock:

```
SELECT GET_LOCK('abc',1000);
+----------------------+
| GET_LOCK('abc',1000) |
+----------------------+
|                    1 |
+----------------------+

SELECT * FROM information_schema.METADATA_LOCK_INFO;
+-----------+---------------------------+---------------+-----------+--------------+------------+
| THREAD_ID | LOCK_MODE                 | LOCK_DURATION | LOCK_TYPE | TABLE_SCHEMA | TABLE_NAME |
+-----------+---------------------------+---------------+-----------+--------------+------------+
|        61 | MDL_SHARED_NO_READ_WRITE | MDL_EXPLICIT  | User lock | abc          |            |
+-----------+---------------------------+---------------+-----------+--------------+------------+
```

Table metadata lock:

```
START TRANSACTION;

INSERT INTO t VALUES (1,2);

SELECT * FROM information_schema.METADATA_LOCK_INFO \G
*************************** 1. row ***************************
    THREAD_ID: 4
    LOCK_MODE: MDL_SHARED_WRITE
LOCK_DURATION: MDL_TRANSACTION
    LOCK_TYPE: Table metadata lock
 TABLE_SCHEMA: test
   TABLE_NAME: t
```

```
SELECT * FROM information_schema.METADATA_LOCK_INFO;
+-----------+---------------------------+---------------+---------------------+--------------
---+-------------+
| THREAD_ID | LOCK_MODE | LOCK_DURATION | LOCK_TYPE | TABLE_SCHEMA | TABLE_NAME |
+-----------+---------------------------+---------------+---------------------+--------------
---+-------------+
| 31 | MDL_INTENTION_EXCLUSIVE | MDL_EXPLICIT | Global read lock | | |
| 31 | MDL_INTENTION_EXCLUSIVE | MDL_EXPLICIT | Commit lock | | |
| 31 | MDL_INTENTION_EXCLUSIVE | MDL_EXPLICIT | Schema metadata lock | dbname | |
| 31 | MDL_SHARED_NO_READ_WRITE | MDL_EXPLICIT | Table metadata lock | dbname | exotics |
+-----------+---------------------------+---------------+---------------------+--------------
---+-------------+
```

1.1.1.2.9.1.1.29 Information Schema MROONGA_STATS Table

The Information Schema `MROONGA_STATS` table only exists if the Mroonga storage engine is installed, and contains information about its activities.

Column	Description
VERSION	Mroonga version.
rows_written	Number of rows written into Mroonga tables.
rows_read	Number of rows read from all Mroonga tables.

This table always contains 1 row.

1.1.1.2.9.1.1.30 Information Schema OPTIMIZER_TRACE Table

MariaDB starting with 10.4.3

Optimizer Trace was introduced in MariaDB 10.4.3.

Description

The Information Schema OPTIMIZER_TRACE table contains Optimizer Trace information.

It contains the following columns:

Column	Description
QUERY	Displays the query that was asked to be traced.
TRACE	A JSON document displaying the stats we collected when the query was run.
MISSING_BYTES_BEYOND_MAX_MEM_SIZE	For huge trace, where the trace is truncated due to the optimizer_trace_max_mem_size limit being reached, displays the bytes that are missing in the trace
INSUFFICENT_PRIVILEGES	Set to 1 if the user running the trace does not have the privileges to see the trace.

Structure:

```
SHOW CREATE TABLE INFORMATION_SCHEMA.OPTIMIZER_TRACE \G
*************************** 1. row ***************************
       Table: OPTIMIZER_TRACE
Create Table: CREATE TEMPORARY TABLE `OPTIMIZER_TRACE` (
  `QUERY` longtext NOT NULL DEFAULT '',
  `TRACE` longtext NOT NULL DEFAULT '',
  `MISSING_BYTES_BEYOND_MAX_MEM_SIZE` int(20) NOT NULL DEFAULT 0,
  `INSUFFICIENT_PRIVILEGES` tinyint(1) NOT NULL DEFAULT 0
) ENGINE=Aria DEFAULT CHARSET=utf8 PAGE_CHECKSUM=0
```

1.1.1.2.9.1.1.31 Information Schema PARAMETERS Table

The Information Schema PARAMETERS table stores information about stored procedures and stored functions parameters.

It contains the following columns:

Column	Description
SPECIFIC_CATALOG	Always def.
SPECIFIC_SCHEMA	Database name containing the stored routine parameter.
SPECIFIC_NAME	Stored routine name.
ORDINAL_POSITION	Ordinal position of the parameter, starting at 1. 0 for a function RETURNS clause.
PARAMETER_MODE	One of IN, OUT, INOUT or NULL for RETURNS.
PARAMETER_NAME	Name of the parameter, or NULL for RETURNS.

DATA_TYPE	The column's data type.
CHARACTER_MAXIMUM_LENGTH	Maximum length.
CHARACTER_OCTET_LENGTH	Same as the CHARACTER_MAXIMUM_LENGTH except for multi-byte character sets.
NUMERIC_PRECISION	For numeric types, the precision (number of significant digits) for the column. NULL if not a numeric field.
NUMERIC_SCALE	For numeric types, the scale (significant digits to the right of the decimal point). NULL if not a numeric field.
DATETIME_PRECISION	Fractional-seconds precision, or NULL if not a time data type.
CHARACTER_SET_NAME	Character set if a non-binary string data type, otherwise NULL.
COLLATION_NAME	Collation if a non-binary string data type, otherwise NULL.
DTD_IDENTIFIER	Description of the data type.
ROUTINE_TYPE	PROCEDURE or FUNCTION.

Information from this table is similar to that found in the param_list column in the mysql.proc table, and the output of the SHOW CREATE PROCEDURE and SHOW CREATE FUNCTION statements.

To obtain information about the routine itself, you can query the Information Schema ROUTINES table.

Example

```
SELECT * FROM information_schema.PARAMETERS
LIMIT 1 \G
********************* 1. row *********************
        SPECIFIC_CATALOG: def
         SPECIFIC_SCHEMA: accounts
           SPECIFIC_NAME: user_counts
        ORDINAL_POSITION: 1
          PARAMETER_MODE: IN
          PARAMETER_NAME: user_order
               DATA_TYPE: varchar
CHARACTER_MAXIMUM_LENGTH: 255
  CHARACTER_OCTET_LENGTH: 765
       NUMERIC_PRECISION: NULL
           NUMERIC_SCALE: NULL
      DATETIME_PRECISION: NULL
      CHARACTER_SET_NAME: utf8
          COLLATION_NAME: utf8_general_ci
          DTD_IDENTIFIER: varchar(255)
            ROUTINE_TYPE: PROCEDURE
```

1.1.1.2.9.1.1.32 Information Schema PARTITIONS Table

The Information Schema PARTITIONS contains information about table partitions, with each record corresponding to a single partition or subpartition of a partitioned table. Each non-partitioned table also has a record in the PARTITIONS table, but most of the values are NULL.

It contains the following columns:

Column	Description
TABLE_CATALOG	Always def.
TABLE_SCHEMA	Database name.
TABLE_NAME	Table name containing the partition.
PARTITION_NAME	Partition name.
SUBPARTITION_NAME	Subpartition name, or NULL if not a subpartition.
PARTITION_ORDINAL_POSITION	Order of the partition starting from 1.

SUBPARTITION_ORDINAL_POSITION	Order of the subpartition starting from 1.
PARTITION_METHOD	The partitioning type; one of RANGE, LIST, HASH, LINEAR HASH, KEY or LINEAR KEY.
SUBPARTITION_METHOD	Subpartition type; one of HASH, LINEAR HASH, KEY or LINEAR KEY, or NULL if not a subpartition.
PARTITION_EXPRESSION	Expression used to create the partition by the CREATE TABLE or ALTER TABLE statement.
SUBPARTITION_EXPRESSION	Expression used to create the subpartition by the CREATE TABLE or ALTER TABLE statement, or NULL if not a subpartition.
PARTITION_DESCRIPTION	For a RANGE partition, contains either MAXINTEGER or an integer, as set in the VALUES LESS THAN clause. For a LIST partition, contains a comma-separated list of integers, as set in the VALUES IN. For a SYSTEM_TIME INTERVAL partition, shows a defined upper boundary timestamp for historical values (the last history partition can contain values above the upper boundary). NULL if another type of partition.
TABLE_ROWS	Number of rows in the table (may be an estimate for some storage engines).
AVG_ROW_LENGTH	Average row length, that is DATA_LENGTH divided by TABLE_ROWS
DATA_LENGTH	Total number of bytes stored in all rows of the partition.
MAX_DATA_LENGTH	Maximum bytes that could be stored in the partition.
INDEX_LENGTH	Size in bytes of the partition index file.
DATA_FREE	Unused bytes allocated to the partition.
CREATE_TIME	Time the partition was created
UPDATE_TIME	Time the partition was last modified.
CHECK_TIME	Time the partition was last checked, or NULL for storage engines that don't record this information.
CHECKSUM	Checksum value, or NULL if none.
PARTITION_COMMENT	Partition comment, truncated to 80 characters, or an empty string if no comment.
NODEGROUP	Node group, only used for MySQL Cluster, defaults to 0.
TABLESPACE_NAME	Always default.

1.1.1.2.9.1.1.33 Information Schema PERIODS Table

MariaDB starting with 11.3.0

The Information Schema PERIODS table provides information about Application-Time Periods.

It contains the following columns:

Column	Description
TABLE_CATALOG	
TABLE_SCHEMA	
TABLE_NAME	
PERIOD	
START_COLUMN_NAME	
END_COLUMN_NAME	

Example

```
CREATE OR REPLACE TABLE t1(
 name VARCHAR(50),
 date_1 DATE,
 date_2 DATE,
 PERIOD FOR date_period(date_1, date_2)
);

SELECT * FROM INFORMATION_SCHEMA.PERIODS;
+---------------+--------------+------------+-------------+-------------------+-----------------+
| TABLE_CATALOG | TABLE_SCHEMA | TABLE_NAME | PERIOD      | START_COLUMN_NAME | END_COLUMN_NAME |
+---------------+--------------+------------+-------------+-------------------+-----------------+
| def           | test         | t1         | date_period | date_1            | date_2          |
+---------------+--------------+------------+-------------+-------------------+-----------------+
```

1.1.1.2.9.1.1.34 Information Schema PLUGINS Table

The Information Schema `PLUGINS` table contains information about server plugins.

It contains the following columns:

Column	Description
`PLUGIN_NAME`	Name of the plugin.
`PLUGIN_VERSION`	Version from the plugin's general type descriptor.
`PLUGIN_STATUS`	Plugin status, one of `ACTIVE`, `INACTIVE`, `DISABLED` or `DELETED`.
`PLUGIN_TYPE`	Plugin type; `STORAGE ENGINE`, `INFORMATION_SCHEMA`, `AUTHENTICATION`, `REPLICATION`, `DAEMON` or `AUDIT`.
`PLUGIN_TYPE_VERSION`	Version from the plugin's type-specific descriptor.
`PLUGIN_LIBRARY`	Plugin's shared object file name, located in the directory specified by the plugin_dir system variable, and used by the INSTALL PLUGIN and UNINSTALL PLUGIN statements. `NULL` if the plugin is complied in and cannot be uninstalled.
`PLUGIN_LIBRARY_VERSION`	Version from the plugin's API interface.
`PLUGIN_AUTHOR`	Author of the plugin.
`PLUGIN_DESCRIPTION`	Description.
`PLUGIN_LICENSE`	Plugin's licence.
`LOAD_OPTION`	How the plugin was loaded; one of `OFF`, `ON`, `FORCE` or `FORCE_PLUS_PERMANENT`. See Installing Plugins.
`PLUGIN_MATURITY`	Plugin's maturity level; one of `Unknown`, `Experimental`, `Alpha`, `Beta`, `'Gamma`, and `Stable`.
`PLUGIN_AUTH_VERSION`	Plugin's version as determined by the plugin author. An example would be '0.99 beta 1'.

It provides a superset of the information shown by the SHOW PLUGINS statement. For specific information about storage engines (a particular type of plugins), see the information_schema.ENGINES table and the SHOW ENGINES statement.

This table provides a subset of the Information Schema information_schema.ALL_PLUGINS table, which contains all available plugins, installed or not.

The table is not a standard Information Schema table, and is a MariaDB extension.

Examples

The easiest way to get basic information on plugins is with SHOW PLUGINS:

```
SHOW PLUGINS;

+----------------------------+----------+--------------------+-------------+---------+
| Name                       | Status   | Type               | Library     | License |
+----------------------------+----------+--------------------+-------------+---------+
| binlog                     | ACTIVE   | STORAGE ENGINE     | NULL        | GPL     |
| mysql_native_password      | ACTIVE   | AUTHENTICATION     | NULL        | GPL     |
| mysql_old_password         | ACTIVE   | AUTHENTICATION     | NULL        | GPL     |
| MRG_MyISAM                 | ACTIVE   | STORAGE ENGINE     | NULL        | GPL     |
| MyISAM                     | ACTIVE   | STORAGE ENGINE     | NULL        | GPL     |
| CSV                        | ACTIVE   | STORAGE ENGINE     | NULL        | GPL     |
| MEMORY                     | ACTIVE   | STORAGE ENGINE     | NULL        | GPL     |
| FEDERATED                  | ACTIVE   | STORAGE ENGINE     | NULL        | GPL     |
| PERFORMANCE_SCHEMA         | ACTIVE   | STORAGE ENGINE     | NULL        | GPL     |
| Aria                       | ACTIVE   | STORAGE ENGINE     | NULL        | GPL     |
| InnoDB                     | ACTIVE   | STORAGE ENGINE     | NULL        | GPL     |
| INNODB_TRX                 | ACTIVE   | INFORMATION SCHEMA | NULL        | GPL     |
| INNODB_LOCKS               | ACTIVE   | INFORMATION SCHEMA | NULL        | GPL     |
| INNODB_LOCK_WAITS          | ACTIVE   | INFORMATION SCHEMA | NULL        | GPL     |
| INNODB_CMP                 | ACTIVE   | INFORMATION SCHEMA | NULL        | GPL     |
| INNODB_CMP_RESET           | ACTIVE   | INFORMATION SCHEMA | NULL        | GPL     |
| INNODB_CMPMEM              | ACTIVE   | INFORMATION SCHEMA | NULL        | GPL     |
| INNODB_CMPMEM_RESET        | ACTIVE   | INFORMATION SCHEMA | NULL        | GPL     |
| INNODB_BUFFER_PAGE         | ACTIVE   | INFORMATION SCHEMA | NULL        | GPL     |
| INNODB_BUFFER_PAGE_LRU     | ACTIVE   | INFORMATION SCHEMA | NULL        | GPL     |
| INNODB_BUFFER_POOL_STATS   | ACTIVE   | INFORMATION SCHEMA | NULL        | GPL     |
| INNODB_METRICS             | ACTIVE   | INFORMATION SCHEMA | NULL        | GPL     |
| INNODB_FT_DEFAULT_STOPWORD | ACTIVE   | INFORMATION SCHEMA | NULL        | GPL     |
| INNODB_FT_INSERTED         | ACTIVE   | INFORMATION SCHEMA | NULL        | GPL     |
| INNODB_FT_DELETED          | ACTIVE   | INFORMATION SCHEMA | NULL        | GPL     |
| INNODB_FT_BEING_DELETED    | ACTIVE   | INFORMATION SCHEMA | NULL        | GPL     |
| INNODB_FT_CONFIG           | ACTIVE   | INFORMATION SCHEMA | NULL        | GPL     |
| INNODB_FT_INDEX_CACHE      | ACTIVE   | INFORMATION SCHEMA | NULL        | GPL     |
| INNODB_FT_INDEX_TABLE      | ACTIVE   | INFORMATION SCHEMA | NULL        | GPL     |
| INNODB_SYS_TABLES          | ACTIVE   | INFORMATION SCHEMA | NULL        | GPL     |
| INNODB_SYS_TABLESTATS      | ACTIVE   | INFORMATION SCHEMA | NULL        | GPL     |
| INNODB_SYS_INDEXES         | ACTIVE   | INFORMATION SCHEMA | NULL        | GPL     |
| INNODB_SYS_COLUMNS         | ACTIVE   | INFORMATION SCHEMA | NULL        | GPL     |
| INNODB_SYS_FIELDS          | ACTIVE   | INFORMATION SCHEMA | NULL        | GPL     |
| INNODB_SYS_FOREIGN         | ACTIVE   | INFORMATION SCHEMA | NULL        | GPL     |
| INNODB_SYS_FOREIGN_COLS    | ACTIVE   | INFORMATION SCHEMA | NULL        | GPL     |
| SPHINX                     | ACTIVE   | STORAGE ENGINE     | NULL        | GPL     |
| ARCHIVE                    | ACTIVE   | STORAGE ENGINE     | NULL        | GPL     |
| BLACKHOLE                  | ACTIVE   | STORAGE ENGINE     | NULL        | GPL     |
| FEEDBACK                   | DISABLED | INFORMATION SCHEMA | NULL        | GPL     |
| partition                  | ACTIVE   | STORAGE ENGINE     | NULL        | GPL     |
| pam                        | ACTIVE   | AUTHENTICATION     | auth_pam.so | GPL     |
+----------------------------+----------+--------------------+-------------+---------+
```

```
SELECT LOAD_OPTION
FROM INFORMATION_SCHEMA.PLUGINS
WHERE PLUGIN_NAME LIKE 'tokudb';
Empty set
```

The equivalent SELECT query would be:

```
SELECT PLUGIN_NAME, PLUGIN_STATUS,
PLUGIN_TYPE, PLUGIN_LIBRARY, PLUGIN_LICENSE
FROM INFORMATION_SCHEMA.PLUGINS;
```

Other SELECT queries can be used to see additional information. For example:

```
SELECT PLUGIN_NAME, PLUGIN_DESCRIPTION,
PLUGIN_MATURITY, PLUGIN_AUTH_VERSION
FROM INFORMATION_SCHEMA.PLUGINS
WHERE PLUGIN_TYPE='STORAGE ENGINE'
ORDER BY PLUGIN_MATURITY \G

*************************** 1. row ***************************
        PLUGIN_NAME: FEDERATED
 PLUGIN_DESCRIPTION: FederatedX pluggable storage engine
    PLUGIN_MATURITY: Beta
PLUGIN_AUTH_VERSION: 2.1
*************************** 2. row ***************************
        PLUGIN_NAME: Aria
 PLUGIN_DESCRIPTION: Crash-safe tables with MyISAM heritage
    PLUGIN_MATURITY: Gamma
PLUGIN_AUTH_VERSION: 1.5
*************************** 3. row ***************************
        PLUGIN_NAME: PERFORMANCE_SCHEMA
 PLUGIN_DESCRIPTION: Performance Schema
    PLUGIN_MATURITY: Gamma
PLUGIN_AUTH_VERSION: 0.1
*************************** 4. row ***************************
        PLUGIN_NAME: binlog
 PLUGIN_DESCRIPTION: This is a pseudo storage engine to represent the binlog in a transaction
    PLUGIN_MATURITY: Stable
PLUGIN_AUTH_VERSION: 1.0
*************************** 5. row ***************************
        PLUGIN_NAME: MEMORY
 PLUGIN_DESCRIPTION: Hash based, stored in memory, useful for temporary tables
    PLUGIN_MATURITY: Stable
PLUGIN_AUTH_VERSION: 1.0
*************************** 6. row ***************************
        PLUGIN_NAME: MyISAM
 PLUGIN_DESCRIPTION: MyISAM storage engine
    PLUGIN_MATURITY: Stable
PLUGIN_AUTH_VERSION: 1.0
*************************** 7. row ***************************
        PLUGIN_NAME: MRG_MyISAM
 PLUGIN_DESCRIPTION: Collection of identical MyISAM tables
    PLUGIN_MATURITY: Stable
PLUGIN_AUTH_VERSION: 1.0
*************************** 8. row ***************************
        PLUGIN_NAME: CSV
 PLUGIN_DESCRIPTION: CSV storage engine
    PLUGIN_MATURITY: Stable
PLUGIN_AUTH_VERSION: 1.0
*************************** 9. row ***************************
        PLUGIN_NAME: InnoDB
 PLUGIN_DESCRIPTION: Supports transactions, row-level locking, and foreign keys
    PLUGIN_MATURITY: Stable
PLUGIN_AUTH_VERSION: 1.2.5
*************************** 10. row ***************************
        PLUGIN_NAME: BLACKHOLE
 PLUGIN_DESCRIPTION: /dev/null storage engine (anything you write to it disappears)
    PLUGIN_MATURITY: Stable
PLUGIN_AUTH_VERSION: 1.0
*************************** 11. row ***************************
        PLUGIN_NAME: ARCHIVE
 PLUGIN_DESCRIPTION: Archive storage engine
    PLUGIN_MATURITY: Stable
PLUGIN_AUTH_VERSION: 1.0
*************************** 12. row ***************************
        PLUGIN_NAME: partition
 PLUGIN_DESCRIPTION: Partition Storage Engine Helper
    PLUGIN_MATURITY: Stable
PLUGIN_AUTH_VERSION: 1.0
```

Check if a given plugin is available:

```
SELECT LOAD_OPTION
FROM INFORMATION_SCHEMA.PLUGINS
WHERE PLUGIN_NAME LIKE 'tokudb';
Empty set
```

Show authentication plugins:

```
SELECT PLUGIN_NAME, LOAD_OPTION
FROM INFORMATION_SCHEMA.PLUGINS
WHERE PLUGIN_TYPE LIKE 'authentication' \G

*************************** 1. row ***************************
PLUGIN_NAME: mysql_native_password
LOAD_OPTION: FORCE
*************************** 2. row ***************************
PLUGIN_NAME: mysql_old_password
LOAD_OPTION: FORCE
```

1.1.1.2.9.1.1.35 Information Schema PROCESSLIST Table

Contents

The Information Schema `PROCESSLIST` table contains information about running threads.

Similar information can also be returned with the `SHOW [FULL] PROCESSLIST` statement, or the mariadb-admin processlist command.

It contains the following columns:

Column	Description
`ID`	Connection identifier.
`USER`	MariaDB User.
`HOST`	The hostname from which this thread is connected. For Unix socket connections, `localhost`. For TCP/IP connections, the TCP port is appended (e.g. `192.168.1.17:58061` or `other-host.company.com:58061`). For `system user`, this column is blank (`''`).
`DB`	Default database, or `NULL` if none.
`COMMAND`	Type of command running, corresponding to the `Com_` status variables. See Thread Command Values.
`TIME`	Seconds that the thread has spent on the current `COMMAND` so far.
`STATE`	Current state of the thread. See Thread States.
`INFO`	Statement the thread is executing, or `NULL` if none.
`TIME_MS`	Time in milliseconds with microsecond precision that the thread has spent on the current `COMMAND` so far (see more).
`STAGE`	The stage the process is currently in.
`MAX_STAGE`	The maximum number of stages.
`PROGRESS`	The progress of the process within the current stage (0-100%).
`MEMORY_USED`	Memory in bytes used by the thread.
`MAX_MEMORY_USED`	Maximum memory in bytes used by the thread.
`EXAMINED_ROWS`	Rows examined by the thread. Only updated by UPDATE, DELETE, and similar statements. For SELECT and other statements, the value remains zero.
`SENT_ROWS`	Number of rows sent by the statement being executed. From MariaDB 11.3.0.

QUERY_ID	Query ID.
INFO_BINARY	Binary data information
TID	Thread ID (MDEV-6756)

Note that as a difference to MySQL, in MariaDB the TIME column (and also the TIME_MS column) are not affected by any setting of @TIMESTAMP. This means that it can be reliably used also for threads that change @TIMESTAMP (such as the replication SQL thread). See also MySQL Bug #22047.

As a consequence of this, the TIME column of SHOW FULL PROCESSLIST and INFORMATION_SCHEMA.PROCESSLIST can not be used to determine if a slave is lagging behind. For this, use instead the Seconds_Behind_Master column in the output of SHOW SLAVE STATUS.

Note that the PROGRESS field from the information schema, and the PROGRESS field from SHOW PROCESSLIST display different results. SHOW PROCESSLIST shows the total progress, while the information schema shows the progress for the current stage only.. To retrieve a similar "total" Progress value from information_schema.PROCESSLIST as the one from SHOW PROCESSLIST, use

```
SELECT CASE WHEN Max_Stage < 2 THEN Progress ELSE (Stage-1)/Max_Stage*100+Progress/Max_Stage END
  AS Progress FROM INFORMATION_SCHEMA.PROCESSLIST;
```

Example

```
SELECT * FROM INFORMATION_SCHEMA.PROCESSLIST\G
*************************** 1. row ***************************
           ID: 9
         USER: msandbox
         HOST: localhost
           DB: NULL
      COMMAND: Query
         TIME: 0
        STATE: Filling schema table
         INFO: SELECT * FROM INFORMATION_SCHEMA.PROCESSLIST
      TIME_MS: 0.351
        STAGE: 0
    MAX_STAGE: 0
     PROGRESS: 0.000
  MEMORY_USED: 85392
EXAMINED_ROWS: 0
     QUERY_ID: 15
  INFO_BINARY: SELECT * FROM INFORMATION_SCHEMA.PROCESSLIST
          TID: 11838
*************************** 2. row ***************************
           ID: 5
         USER: system user
         HOST:
           DB: NULL
      COMMAND: Daemon
         TIME: 0
        STATE: InnoDB shutdown handler
         INFO: NULL
      TIME_MS: 0.000
        STAGE: 0
    MAX_STAGE: 0
     PROGRESS: 0.000
  MEMORY_USED: 24160
EXAMINED_ROWS: 0
     QUERY_ID: 0
  INFO_BINARY: NULL
          TID: 3856
...
```

1.1.1.2.9.1.1.36 Information Schema PROFILING Table

The Information Schema `PROFILING` table contains information about statement resource usage. Profiling information is only recorded if the `profiling` session variable is set to 1.

It contains the following columns:

Column Name	Description
`QUERY_ID`	Query_ID.
`SEQ`	Sequence number showing the display order for rows with the same `QUERY_ID`.
`STATE`	Profiling state.
`DURATION`	Time in seconds that the statement has been in the current state.
`CPU_USER`	User CPU usage in seconds.
`CPU_SYSTEM`	System CPU usage in seconds.
`CONTEXT_VOLUNTARY`	Number of voluntary context switches.
`CONTEXT_INVOLUNTARY`	Number of involuntary context switches.
`BLOCK_OPS_IN`	Number of block input operations.
`BLOCK_OPS_OUT`	Number of block output operations.
`MESSAGES_SENT`	Number of communications sent.
`MESSAGES_RECEIVED`	Number of communications received.
`PAGE_FAULTS_MAJOR`	Number of major page faults.
`PAGE_FAULTS_MINOR`	Number of minor page faults.
`SWAPS`	Number of swaps.
`SOURCE_FUNCTION`	Function in the source code executed by the profiled state.
`SOURCE_FILE`	File in the source code executed by the profiled state.
`SOURCE_LINE`	Line in the source code executed by the profiled state.

It contains similar information to the `SHOW PROFILE and SHOW PROFILES` statements.

Profiling is enabled per session. When a session ends, its profiling information is lost.

1.1.1.2.9.1.1.37 Information Schema QUERY_CACHE_INFO Table

Description

The table is not a standard Information Schema table, and is a MariaDB extension.

The `QUERY_CACHE_INFO` table is created by the QUERY_CACHE_INFO plugin, and allows you to see the contents of the query cache. It creates a table in the information_schema database that shows all queries that are in the cache. You must have the `PROCESS` privilege (see GRANT) to use this table.

It contains the following columns:

Column	Description
`STATEMENT_SCHEMA`	Database used when query was included
`STATEMENT_TEXT`	Query text
`RESULT_BLOCKS_COUNT`	Number of result blocks
`RESULT_BLOCKS_SIZE`	Size in bytes of result blocks
`RESULT_BLOCKS_SIZE_USED`	Size in bytes of used blocks
`LIMIT`	Added MariaDB 10.1.8.
`MAX_SORT_LENGTH`	Added MariaDB 10.1.8.

GROUP_CONCAT_MAX_LENGTH	Added MariaDB 10.1.8.
CHARACTER_SET_CLIENT	Added MariaDB 10.1.8.
CHARACTER_SET_RESULT	Added MariaDB 10.1.8.
COLLATION	Added MariaDB 10.1.8.
TIMEZONE	Added MariaDB 10.1.8.
DEFAULT_WEEK_FORMAT	Added MariaDB 10.1.8.
DIV_PRECISION_INCREMENT	Added MariaDB 10.1.8.
SQL_MODE	Added MariaDB 10.1.8.
LC_TIME_NAMES	Added MariaDB 10.1.8.
CLIENT_LONG_FLAG	Added MariaDB 10.1.8.
CLIENT_PROTOCOL_41	Added MariaDB 10.1.8.
PROTOCOL_TYPE	Added MariaDB 10.1.8.
MORE_RESULTS_EXISTS	Added MariaDB 10.1.8.
IN_TRANS	Added MariaDB 10.1.8.
AUTOCOMMIT	Added MariaDB 10.1.8.
PACKET_NUMBER	Added MariaDB 10.1.8.
HITS	Incremented each time the query cache is hit. Added MariaDB 10.3.2.

For example:

```
SELECT * FROM information_schema.QUERY_CACHE_INFO;
+------------------+-----------------+---------------------+--------------------+------------
-------------+
| STATEMENT_SCHEMA | STATEMENT_TEXT  | RESULT_BLOCKS_COUNT | RESULT_BLOCKS_SIZE |
RESULT_BLOCKS_SIZE_USED |
+------------------+-----------------+---------------------+--------------------+------------
-------------+
...
| test             | SELECT * FROM a |                   1 |                512 |
143 |
| test             | select * FROM a |                   1 |                512 |
143 |
...
+------------------+-----------------+---------------------+--------------------+------------
-------------
```

1.1.1.2.9.1.1.38 Information Schema QUERY_RESPONSE_TIME Table

Description

The Information Schema `QUERY_RESPONSE_TIME` table contains information about queries that take a long time to execute. It is only available if the QUERY_RESPONSE_TIME plugin has been installed.

It contains the following columns:

Column	Description
TIME	Time interval
COUNT	Count of queries falling into the time interval
TOTAL	Total execution time of all queries for this interval

See QUERY_RESPONSE_TIME plugin for a full description.

The table is not a standard Information Schema table, and is a MariaDB extension.

`SHOW QUERY_RESPONSE_TIME` is available from MariaDB 10.1.1 as an alternative for retrieving the data.

Example

```
SELECT * FROM information_schema.QUERY_RESPONSE_TIME;
+----------------+-------+----------------+
| TIME           | COUNT | TOTAL          |
+----------------+-------+----------------+
|       0.000001 |     0 |       0.000000 |
|       0.000010 |    17 |       0.000094 |
|       0.000100 |  4301 |       0.236555 |
|       0.001000 |  1499 |       0.824450 |
|       0.010000 | 14851 |      81.680502 |
|       0.100000 |  8066 |     443.635693 |
|       1.000000 |     0 |       0.000000 |
|      10.000000 |     0 |       0.000000 |
|     100.000000 |     1 |      55.937094 |
|    1000.000000 |     0 |       0.000000 |
|   10000.000000 |     0 |       0.000000 |
|  100000.000000 |     0 |       0.000000 |
| 1000000.000000 |     0 |       0.000000 |
| TOO LONG       |     0 | TOO LONG       |
+----------------+-------+----------------+
```

1.1.1.2.9.1.1.39 Information Schema REFERENTIAL_CONSTRAINTS Table

The Information Schema `REFERENTIAL_CONSTRAINTS` table contains information about foreign keys. The single columns are listed in the `KEY_COLUMN_USAGE` table.

It has the following columns:

Column	Description
`CONSTRAINT_CATALOG`	Always `def`.
`CONSTRAINT_SCHEMA`	Database name, together with `CONSTRAINT_NAME` identifies the foreign key.
`CONSTRAINT_NAME`	Foreign key name, together with `CONSTRAINT_SCHEMA` identifies the foreign key.
`UNIQUE_CONSTRAINT_CATALOG`	Always `def`.
`UNIQUE_CONSTRAINT_SCHEMA`	Database name, together with `UNIQUE_CONSTRAINT_NAME` and `REFERENCED_TABLE_NAME` identifies the referenced key.
`UNIQUE_CONSTRAINT_NAME`	Referenced key name, together with `UNIQUE_CONSTRAINT_SCHEMA` and `REFERENCED_TABLE_NAME` identifies the referenced key.
`MATCH_OPTION`	Always `NONE`.
`UPDATE_RULE`	The Update Rule; one of `CASCADE`, `SET NULL`, `SET DEFAULT`, `RESTRICT`, `NO ACTION`.
`DELETE_RULE`	The Delete Rule; one of `CASCADE`, `SET NULL`, `SET DEFAULT`, `RESTRICT`, `NO ACTION`.
`TABLE_NAME`	Table name from the `TABLE_CONSTRAINTS` table.
`REFERENCED_TABLE_NAME`	Referenced key table name, together with `UNIQUE_CONSTRAINT_SCHEMA` and `UNIQUE_CONSTRAINT_NAME` identifies the referenced key.

1.1.1.2.9.1.1.40 Information Schema ROUTINES Table

The Information Schema `ROUTINES` table stores information about stored procedures and stored functions.

It contains the following columns:

Column	Description
SPECIFIC_NAME	
ROUTINE_CATALOG	Always def.
ROUTINE_SCHEMA	Database name associated with the routine.
ROUTINE_NAME	Name of the routine.
ROUTINE_TYPE	Whether the routine is a PROCEDURE or a FUNCTION.
DATA_TYPE	The return value's data type (for stored functions).
CHARACTER_MAXIMUM_LENGTH	Maximum length.
CHARACTER_OCTET_LENGTH	Same as the CHARACTER_MAXIMUM_LENGTH except for multi-byte character sets.
NUMERIC_PRECISION	For numeric types, the precision (number of significant digits) for the column. NULL if not a numeric field.
NUMERIC_SCALE	For numeric types, the scale (significant digits to the right of the decimal point). NULL if not a numeric field.
DATETIME_PRECISION	Fractional-seconds precision, or NULL if not a time data type.
CHARACTER_SET_NAME	Character set if a non-binary string data type, otherwise NULL.
COLLATION_NAME	Collation if a non-binary string data type, otherwise NULL.
DATA_TYPE	The column's data type.
ROUTINE_BODY	Always SQL.
ROUTINE_DEFINITION	Definition of the routine.
EXTERNAL_NAME	Always NULL.
EXTERNAL_LANGUAGE	Always SQL.
PARAMETER_STYLE	Always SQL.
IS_DETERMINISTIC	Whether the routine is deterministic (can produce only one result for a given list of parameters) or not.
SQL_DATA_ACCESS	One of READS SQL DATA, MODIFIES SQL DATA, CONTAINS SQL, or NO SQL.
SQL_PATH	Always NULL.
SECURITY_TYPE	INVOKER or DEFINER. Indicates which user's privileges apply to this routine.
CREATED	Date and time the routine was created.
LAST_ALTERED	Date and time the routine was last changed.
SQL_MODE	The SQL_MODE at the time the routine was created.
ROUTINE_COMMENT	Comment associated with the routine.
DEFINER	If the SECURITY_TYPE is DEFINER, this value indicates which user defined this routine.
CHARACTER_SET_CLIENT	The character set used by the client that created the routine.
COLLATION_CONNECTION	The collation (and character set) used by the connection that created the routine.
DATABASE_COLLATION	The default collation (and character set) for the database, at the time the routine was created.

It provides information similar to, but more complete, than the SHOW PROCEDURE STATUS and SHOW FUNCTION STATUS statements.

For information about the parameters accepted by the routine, you can query the information_schema.PARAMETERS table.

1.1.1.2.9.1.1.41 Information Schema SCHEMA_PRIVILEGES Table

The Information Schema `SCHEMA_PRIVILEGES` table contains information about database privileges.

It contains the following columns:

Column	Description
`GRANTEE`	Account granted the privilege in the format `user_name@host_name`.
`TABLE_CATALOG`	Always `def`
`TABLE_SCHEMA`	Database name.
`PRIVILEGE_TYPE`	The granted privilege.
`IS_GRANTABLE`	Whether the privilege can be granted.

The same information in a different format can be found in the `mysql.db` table.

1.1.1.2.9.1.1.42 Information Schema SCHEMATA Table

The Information Schema `SCHEMATA` table stores information about databases on the server.

It contains the following columns:

Column	Description
`CATALOG_NAME`	Always `def`.
`SCHEMA_NAME`	Database name.
`DEFAULT_CHARACTER_SET_NAME`	Default character set for the database.
`DEFAULT_COLLATION_NAME`	Default collation.
`SQL_PATH`	Always `NULL`.
`SCHEMA_COMMENT`	Database comment. From MariaDB 10.5.0.

Example

```
SELECT * FROM INFORMATION_SCHEMA.SCHEMATA\G
*************************** 1. row ***************************
              CATALOG_NAME: def
               SCHEMA_NAME: information_schema
DEFAULT_CHARACTER_SET_NAME: utf8
    DEFAULT_COLLATION_NAME: utf8_general_ci
                  SQL_PATH: NULL
*************************** 2. row ***************************
              CATALOG_NAME: def
               SCHEMA_NAME: mysql
DEFAULT_CHARACTER_SET_NAME: latin1
    DEFAULT_COLLATION_NAME: latin1_swedish_ci
                  SQL_PATH: NULL
*************************** 3. row ***************************
              CATALOG_NAME: def
               SCHEMA_NAME: performance_schema
DEFAULT_CHARACTER_SET_NAME: utf8
    DEFAULT_COLLATION_NAME: utf8_general_ci
                  SQL_PATH: NULL
*************************** 4. row ***************************
              CATALOG_NAME: def
               SCHEMA_NAME: test
DEFAULT_CHARACTER_SET_NAME: latin1
    DEFAULT_COLLATION_NAME: latin1_swedish_ci
                  SQL_PATH: NULL
...
```

From MariaDB 10.5.0:

```
SELECT * FROM INFORMATION_SCHEMA.SCHEMATA\G
...
*************************** 2. row ***************************
              CATALOG_NAME: def
               SCHEMA_NAME: presentations
DEFAULT_CHARACTER_SET_NAME: latin1
    DEFAULT_COLLATION_NAME: latin1_swedish_ci
                  SQL_PATH: NULL
            SCHEMA_COMMENT: Presentations for conferences
...
```

1.1.1.2.9.1.1.43 Information Schema SPATIAL_REF_SYS Table

Description

The Information Schema `SPATIAL_REF_SYS` table stores information on each spatial reference system used in the database.

It contains the following columns:

Column	Type	Null	Description
`SRID`	`smallint(5)`	`NO`	An integer value that uniquely identifies each Spatial Reference System within a database.
`AUTH_NAME`	`varchar(512)`	`NO`	The name of the standard or standards body that is being cited for this reference system.
`AUTH_SRID`	`smallint(5)`	`NO`	The numeric ID of the coordinate system in the above authority's catalog.
`SRTEXT`	`varchar(2048)`	`NO`	The Well-known Text Representation of the Spatial Reference System.

Note: See MDEV-7540.

See Aso

- information_schema.GEOMETRY_COLUMNS table.

1.1.1.2.9.1.1.44 Information Schema SPIDER_ALLOC_MEM Table

The Information Schema `SPIDER_ALLOC_MEM` table is installed along with the Spider storage engine. It shows information about Spider's memory usage.

It contains the following columns:

Column	Description
`ID`	
`FUNC_NAME`	
`FILE_NAME`	
`LINE_NO`	
`TOTAL_ALLOC_MEM`	
`CURRENT_ALLOC_MEM`	
`ALLOC_MEM_COUNT`	
`FREE_MEM_COUNT`	

1.1.1.2.9.1.1.45 Information Schema SPIDER_WRAPPER_PROTOCOLS Table

MariaDB starting with 10.5.4

The Information Schema `SPIDER_WRAPPER_PROTOCOLS` table is installed along with the Spider storage engine from MariaDB 10.5.4.

It contains the following columns:

Column	Type	Description
`WRAPPER_NAME`	varchar(64)	
`WRAPPER_VERSION`	varchar(20)	
`WRAPPER_DESCRIPTION`	longtext	
`WRAPPER_MATURITY`	varchar(12)	

1.1.1.2.9.1.1.46 Information Schema SQL_FUNCTIONS Table

MariaDB starting with 10.6.3

The `SQL_FUNCTIONS` table was added in MariaDB 10.6.3.

Description

The Information Schema `SQL_FUNCTIONS` table contains the list of MariaDB functions.

It contains a single column:

Column	Description
`FUNCTION`	Function name

The table is not a standard Information Schema table, and is a MariaDB extension.

Example

```
SELECT * FROM INFORMATION_SCHEMA.SQL_FUNCTIONS;
+---------------------------+
| FUNCTION                  |
+---------------------------+
| ADDDATE                   |
| ADD_MONTHS                |
| BIT_AND                   |
| BIT_OR                    |
| BIT_XOR                   |
| CAST                      |
| COUNT                     |
| CUME_DIST                 |
| CURDATE                   |
| CURTIME                   |
| DATE_ADD                  |
| DATE_SUB                  |
| DATE_FORMAT               |
| DECODE                    |
| DENSE_RANK                |
| EXTRACT                   |
| FIRST_VALUE               |
| GROUP_CONCAT              |
| JSON_ARRAYAGG             |
| JSON_OBJECTAGG            |
| LAG                       |
| LEAD                      |
```

| MAX |
| MEDIAN |
| MID |
| MIN |
| NOW |
| NTH_VALUE |
| NTILE |
| POSITION |
| PERCENT_RANK |
| PERCENTILE_CONT |
| PERCENTILE_DISC |
| RANK |
| ROW_NUMBER |
| SESSION_USER |
| STD |
| STDDEV |
| STDDEV_POP |
| STDDEV_SAMP |
| SUBDATE |
| SUBSTR |
| SUBSTRING |
| SUM |
| SYSTEM_USER |
| TRIM |
| TRIM_ORACLE |
| VARIANCE |
| VAR_POP |
| VAR_SAMP |
| ABS |
| ACOS |
| ADDTIME |
| AES_DECRYPT |
| AES_ENCRYPT |
| ASIN |
| ATAN |
| ATAN2 |
| BENCHMARK |
| BIN |
| BINLOG_GTID_POS |
| BIT_COUNT |
| BIT_LENGTH |
| CEIL |
| CEILING |
| CHARACTER_LENGTH |
| CHAR_LENGTH |
| CHR |
| COERCIBILITY |
| COLUMN_CHECK |
| COLUMN_EXISTS |
| COLUMN_LIST |
| COLUMN_JSON |
| COMPRESS |
| CONCAT |
| CONCAT_OPERATOR_ORACLE |
| CONCAT_WS |
| CONNECTION_ID |
| CONV |
| CONVERT_TZ |
| COS |
| COT |
| CRC32 |
| DATEDIFF |
| DAYNAME |
| DAYOFMONTH |
| DAYOFWEEK |
| DAYOFYEAR |
| DEGREES |
| DECODE_HISTOGRAM |
| DECODE_ORACLE |
| DES_DECRYPT |
| DES_ENCRYPT |
| ELT |
| ENCODE |
| ENCRYPT |
| EXP |

| EXPORT_SET |
| EXTRACTVALUE |
| FIELD |
| FIND_IN_SET |
| FLOOR |
| FORMAT |
| FOUND_ROWS |
| FROM_BASE64 |
| FROM_DAYS |
| FROM_UNIXTIME |
| GET_LOCK |
| GREATEST |
| HEX |
| IFNULL |
| INSTR |
| ISNULL |
| IS_FREE_LOCK |
| IS_USED_LOCK |
| JSON_ARRAY |
| JSON_ARRAY_APPEND |
| JSON_ARRAY_INSERT |
| JSON_COMPACT |
| JSON_CONTAINS |
| JSON_CONTAINS_PATH |
| JSON_DEPTH |
| JSON_DETAILED |
| JSON_EXISTS |
| JSON_EXTRACT |
| JSON_INSERT |
| JSON_KEYS |
| JSON_LENGTH |
| JSON_LOOSE |
| JSON_MERGE |
| JSON_MERGE_PATCH |
| JSON_MERGE_PRESERVE |
| JSON_QUERY |
| JSON_QUOTE |
| JSON_OBJECT |
| JSON_REMOVE |
| JSON_REPLACE |
| JSON_SET |
| JSON_SEARCH |
| JSON_TYPE |
| JSON_UNQUOTE |
| JSON_VALID |
| JSON_VALUE |
| LAST_DAY |
| LAST_INSERT_ID |
| LCASE |
| LEAST |
| LENGTH |
| LENGTHB |
| LN |
| LOAD_FILE |
| LOCATE |
| LOG |
| LOG10 |
| LOG2 |
| LOWER |
| LPAD |
| LPAD_ORACLE |
| LTRIM |
| LTRIM_ORACLE |
| MAKEDATE |
| MAKETIME |
| MAKE_SET |
| MASTER_GTID_WAIT |
| MASTER_POS_WAIT |
| MD5 |
| MONTHNAME |
| NAME_CONST |
| NVL |
| NVL2 |
| NULLIF |
| OCT |

```
| OCTET_LENGTH               |
| ORD                        |
| PERIOD_ADD                 |
| PERIOD_DIFF                |
| PI                         |
| POW                        |
| POWER                      |
| QUOTE                      |
| REGEXP_INSTR               |
| REGEXP_REPLACE             |
| REGEXP_SUBSTR              |
| RADIANS                    |
| RAND                       |
| RELEASE_ALL_LOCKS          |
| RELEASE_LOCK               |
| REPLACE_ORACLE             |
| REVERSE                    |
| ROUND                      |
| RPAD                       |
| RPAD_ORACLE                |
| RTRIM                      |
| RTRIM_ORACLE               |
| SEC_TO_TIME                |
| SHA                        |
| SHA1                       |
| SHA2                       |
| SIGN                       |
| SIN                        |
| SLEEP                      |
| SOUNDEX                    |
| SPACE                      |
| SQRT                       |
| STRCMP                     |
| STR_TO_DATE                |
| SUBSTR_ORACLE              |
| SUBSTRING_INDEX            |
| SUBTIME                    |
| SYS_GUID                   |
| TAN                        |
| TIMEDIFF                   |
| TIME_FORMAT                |
| TIME_TO_SEC                |
| TO_BASE64                  |
| TO_CHAR                    |
| TO_DAYS                    |
| TO_SECONDS                 |
| UCASE                      |
| UNCOMPRESS                 |
| UNCOMPRESSED_LENGTH        |
| UNHEX                      |
| UNIX_TIMESTAMP             |
| UPDATEXML                  |
| UPPER                      |
| UUID                       |
| UUID_SHORT                 |
| VERSION                    |
| WEEKDAY                    |
| WEEKOFYEAR                 |
| WSREP_LAST_WRITTEN_GTID    |
| WSREP_LAST_SEEN_GTID       |
| WSREP_SYNC_WAIT_UPTO_GTID  |
| YEARWEEK                   |
+----------------------------+
234 rows in set (0.001 sec)
```

1.1.1.2.9.1.1.47 Information Schema STATISTICS Table

The Information Schema `STATISTICS` table provides information about table indexes.

It contains the following columns:

Column	Description
TABLE_CATALOG	Always def.
TABLE_SCHEMA	Database name.
TABLE_NAME	Table name.
NON_UNIQUE	1 if the index can have duplicates, 0 if not.
INDEX_SCHEMA	Database name.
INDEX_NAME	Index name. The primary key is always named PRIMARY.
SEQ_IN_INDEX	The column sequence number, starting at 1.
COLUMN_NAME	Column name.
COLLATION	A for sorted in ascending order, or NULL for unsorted.
CARDINALITY	Estimate of the number of unique values stored in the index based on statistics stored as integers. Higher cardinalities usually mean a greater chance of the index being used in a join. Updated by the ANALYZE TABLE statement or myisamchk -a.
SUB_PART	NULL if the whole column is indexed, or the number of indexed characters if partly indexed.
PACKED	NULL if not packed, otherwise how the index is packed.
NULLABLE	YES if the column may contain NULLs, empty string if not.
INDEX_TYPE	Index type, one of BTREE, RTREE, HASH or FULLTEXT. See Storage Engine Index Types.
COMMENT	Index comments from the CREATE INDEX statement.
IGNORED	Whether or not an index will be ignored by the optimizer. See Ignored Indexes. From MariaDB 10.6.0.

The SHOW INDEX statement produces similar output.

Example

```
SELECT * FROM INFORMATION_SCHEMA.STATISTICS\G
...
*************************** 85. row ***************************
TABLE_CATALOG: def
 TABLE_SCHEMA: test
   TABLE_NAME: table1
   NON_UNIQUE: 1
 INDEX_SCHEMA: test
   INDEX_NAME: col2
 SEQ_IN_INDEX: 1
  COLUMN_NAME: col2
    COLLATION: A
  CARDINALITY: 6
     SUB_PART: NULL
       PACKED: NULL
     NULLABLE:
   INDEX_TYPE: BTREE
      COMMENT:
INDEX_COMMENT:
...
```

1.1.1.2.9.1.1.48 Information Schema SYSTEM_VARIABLES Table

The Information Schema SYSTEM_VARIABLES table shows current values and various metadata of all system variables.

It contains the following columns:

Column	Description

VARIABLE_NAME	System variable name.
SESSION_VALUE	Session value of the variable or NULL if the variable only has a global scope.
GLOBAL_VALUE	Global value of the variable or NULL if the variable only has a session scope.
GLOBAL_VALUE_ORIGIN	How the global value was set — a compile-time default, auto-configured by the server, configuration file (or a command line), with the SQL statement.
DEFAULT_VALUE	Compile-time default value of the variable.
VARIABLE_SCOPE	Global, session, or session-only.
VARIABLE_TYPE	Data type of the variable value.
VARIABLE_COMMENT	Help text, usually shown in `mysqld --help --verbose`.
NUMERIC_MIN_VALUE	For numeric variables — minimal allowed value.
NUMERIC_MAX_VALUE	For numeric variables — maximal allowed value.
NUMERIC_BLOCK_SIZE	For numeric variables — a valid value must be a multiple of the "block size".
ENUM_VALUE_LIST	For `ENUM`, `SET`, and `FLAGSET` variables — the list of recognized values.
READ_ONLY	Whether a variable can be set with the SQL statement. Note that many "read only" variables can still be set on the command line.
COMMAND_LINE_ARGUMENT	Whether an argument is required when setting the variable on the command line. `NULL` when a variable can not be set on the command line.
GLOBAL_VALUE_PATH	Which config file the variable got its value from. `NULL` if not set in any config file. Added in MariaDB 10.5.0.

Example

```
SELECT * FROM information_schema.SYSTEM_VARIABLES
  WHERE VARIABLE_NAME='JOIN_BUFFER_SIZE'\G
*************************** 1. row ***************************
        VARIABLE_NAME: JOIN_BUFFER_SIZE
        SESSION_VALUE: 131072
         GLOBAL_VALUE: 131072
  GLOBAL_VALUE_ORIGIN: COMPILE-TIME
        DEFAULT_VALUE: 131072
       VARIABLE_SCOPE: SESSION
        VARIABLE_TYPE: BIGINT UNSIGNED
     VARIABLE_COMMENT: The size of the buffer that is used for joins
    NUMERIC_MIN_VALUE: 128
    NUMERIC_MAX_VALUE: 18446744073709551615
   NUMERIC_BLOCK_SIZE: 128
      ENUM_VALUE_LIST: NULL
            READ_ONLY: NO
COMMAND_LINE_ARGUMENT: REQUIRED
```

1.1.1.2.9.1.1.49 Information Schema TABLE_CONSTRAINTS Table

The Information Schema `TABLE_CONSTRAINTS` table contains information about tables that have constraints.

It has the following columns:

Column	Description
CONSTRAINT_CATALOG	Always `def`.
CONSTRAINT_SCHEMA	Database name containing the constraint.
CONSTRAINT_NAME	Constraint name.
TABLE_SCHEMA	Database name.

TABLE_NAME	Table name.
CONSTRAINT_TYPE	Type of constraint; one of UNIQUE, PRIMARY KEY, FOREIGN KEY or CHECK.

The REFERENTIAL_CONSTRAINTS table has more information about foreign keys.

1.1.1.2.9.1.1.50 Information Schema TABLE_PRIVILEGES Table

The Information Schema TABLE_PRIVILEGES table contains table privilege information derived from the mysql.tables_priv grant table.

It has the following columns:

Column	Description
GRANTEE	In the format user_name@host_name.
TABLE_CATALOG	Always def.
TABLE_SCHEMA	Database name.
TABLE_NAME	Table name.
PRIVILEGE_TYPE	One of SELECT, INSERT, UPDATE, REFERENCES, ALTER, INDEX, DROP or CREATE VIEW.
IS_GRANTABLE	Whether the user has the GRANT OPTION for this privilege.

Similar information can be accessed with the SHOW GRANTS statement. See the GRANT article for more about privileges.

For a description of the privileges that are shown in this table, see table privileges.

1.1.1.2.9.1.1.51 Information Schema TABLE_STATISTICS Table

The Information Schema TABLE_STATISTICS table shows statistics on table usage.

This is part of the User Statistics feature, which is not enabled by default.

It contains the following columns:

Field	Type	Notes
TABLE_SCHEMA	varchar(192)	The schema (database) name.
TABLE_NAME	varchar(192)	The table name.
ROWS_READ	int(21)	The number of rows read from the table.
ROWS_CHANGED	int(21)	The number of rows changed in the table.
ROWS_CHANGED_X_INDEXES	int(21)	The number of rows changed in the table, multiplied by the number of indexes changed.

Example

```
SELECT * FROM INFORMATION_SCHEMA.TABLE_STATISTICS WHERE TABLE_NAME='user';
+--------------+------------+-----------+--------------+------------------------+
| TABLE_SCHEMA | TABLE_NAME | ROWS_READ | ROWS_CHANGED | ROWS_CHANGED_X_INDEXES |
+--------------+------------+-----------+--------------+------------------------+
| mysql        | user       |         5 |            2 |                      2 |
+--------------+------------+-----------+--------------+------------------------+
```

1.1.1.2.9.1.1.52 Information Schema TABLES Table

Contents

The Information Schema table shows information about the various tables (until MariaDB 11.2.0, only non- `TEMPORARY` tables, except for tables from the `Information Schema` database) and views on the server.

It contains the following columns:

Column	Description
`TABLE_CATALOG`	Always `def`.
`TABLE_SCHEMA`	Database name.
`TABLE_NAME`	Table name.
`TABLE_TYPE`	One of `BASE TABLE` for a regular table, `VIEW` for a view, `SYSTEM VIEW` for Information Schema tables, `SYSTEM VERSIONED` for system-versioned tables, `SEQUENCE` for sequences or, from MariaDB 11.2.0, `TEMPORARY` for local temporary tables.
`ENGINE`	Storage Engine.
`VERSION`	Version number from the table's .frm file
`ROW_FORMAT`	Row format (see InnoDB, Aria and MyISAM row formats).
`TABLE_ROWS`	Number of rows in the table. Some engines, such as XtraDB and InnoDB may store an estimate.
`AVG_ROW_LENGTH`	Average row length in the table.
`DATA_LENGTH`	For InnoDB/XtraDB, the index size, in pages, multiplied by the page size. For Aria and MyISAM, length of the data file, in bytes. For MEMORY, the approximate allocated memory.
`MAX_DATA_LENGTH`	Maximum length of the data file, ie the total number of bytes that could be stored in the table. Not used in XtraDB and InnoDB.
`INDEX_LENGTH`	Length of the index file.
`DATA_FREE`	Bytes allocated but unused. For InnoDB tables in a shared tablespace, the free space of the shared tablespace with small safety margin. An estimate in the case of partitioned tables - see the `PARTITIONS` table.
`AUTO_INCREMENT`	Next AUTO_INCREMENT value.
`CREATE_TIME`	Time the table was created. Some engines just return the `ctime` information from the file system layer here, in that case the value is not necessarily the table creation time but rather the time the file system metadata for it had last changed.
`UPDATE_TIME`	Time the table was last updated. On Windows, the timestamp is not updated on update, so MyISAM values will be inaccurate. In InnoDB, if shared tablespaces are used, will be NULL, while buffering can also delay the update, so the value will differ from the actual time of the last `UPDATE`, `INSERT` or `DELETE`.
`CHECK_TIME`	Time the table was last checked. Not kept by all storage engines, in which case will be `NULL`.
`TABLE_COLLATION`	Character set and collation.
`CHECKSUM`	Live checksum value, if any.
`CREATE_OPTIONS`	Extra `CREATE TABLE` options.
`TABLE_COMMENT`	Table comment provided when MariaDB created the table.
`MAX_INDEX_LENGTH`	Maximum index length (supported by MyISAM and Aria tables). Added in MariaDB 10.3.5.
`TEMPORARY`	Until MariaDB 11.2.0, placeholder to signal that a table is a temporary table and always "N", except "Y" for generated information_schema tables and NULL for views. From MariaDB 11.2.0, will also be set to "Y" for local temporary tables. Added in MariaDB 10.3.5.

Although the table is standard in the Information Schema, all but `TABLE_CATALOG`, `TABLE_SCHEMA`, `TABLE_NAME`, `TABLE_TYPE`, `ENGINE` and `VERSION` are MySQL and MariaDB extensions.

SHOW TABLES lists all tables in a database.

Examples

From MariaDB 10.3.5:

```
SELECT * FROM information_schema.tables WHERE table_schema='test'\G
*************************** 1. row ***************************
   TABLE_CATALOG: def
    TABLE_SCHEMA: test
      TABLE_NAME: xx5
      TABLE_TYPE: BASE TABLE
          ENGINE: InnoDB
         VERSION: 10
      ROW_FORMAT: Dynamic
      TABLE_ROWS: 0
  AVG_ROW_LENGTH: 0
     DATA_LENGTH: 16384
 MAX_DATA_LENGTH: 0
    INDEX_LENGTH: 0
       DATA_FREE: 0
  AUTO_INCREMENT: NULL
     CREATE_TIME: 2020-11-18 15:57:10
     UPDATE_TIME: NULL
      CHECK_TIME: NULL
 TABLE_COLLATION: latin1_swedish_ci
        CHECKSUM: NULL
  CREATE_OPTIONS:
   TABLE_COMMENT:
MAX_INDEX_LENGTH: 0
       TEMPORARY: N
*************************** 2. row ***************************
   TABLE_CATALOG: def
    TABLE_SCHEMA: test
      TABLE_NAME: xx4
      TABLE_TYPE: BASE TABLE
          ENGINE: MyISAM
         VERSION: 10
      ROW_FORMAT: Fixed
      TABLE_ROWS: 0
  AVG_ROW_LENGTH: 0
     DATA_LENGTH: 0
 MAX_DATA_LENGTH: 1970324836974591
    INDEX_LENGTH: 1024
       DATA_FREE: 0
  AUTO_INCREMENT: NULL
     CREATE_TIME: 2020-11-18 15:56:57
     UPDATE_TIME: 2020-11-18 15:56:57
      CHECK_TIME: NULL
 TABLE_COLLATION: latin1_swedish_ci
        CHECKSUM: NULL
  CREATE_OPTIONS:
   TABLE_COMMENT:
MAX_INDEX_LENGTH: 17179868160
       TEMPORARY: N
...
```

Example with temporary = 'y', from MariaDB 10.3.5:

```
SELECT * FROM information_schema.tables WHERE temporary='y'\G
 *************************** 1. row ***************************
   TABLE_CATALOG: def
    TABLE_SCHEMA: information_schema
      TABLE_NAME: INNODB_FT_DELETED
      TABLE_TYPE: SYSTEM VIEW
          ENGINE: MEMORY
         VERSION: 11
      ROW_FORMAT: Fixed
      TABLE_ROWS: NULL
  AVG_ROW_LENGTH: 9
     DATA_LENGTH: 0
 MAX_DATA_LENGTH: 9437184
    INDEX_LENGTH: 0
       DATA_FREE: 0
  AUTO_INCREMENT: NULL
     CREATE_TIME: 2020-11-17 21:54:02
     UPDATE_TIME: NULL
      CHECK_TIME: NULL
 TABLE_COLLATION: utf8_general_ci
        CHECKSUM: NULL
  CREATE_OPTIONS: max_rows=1864135
   TABLE_COMMENT:
MAX_INDEX_LENGTH: 0
       TEMPORARY: Y
...
```

View Tables in Order of Size

Returns a list of all tables in the database, ordered by size:

```
SELECT table_schema as `DB`, table_name AS `Table`,
  ROUND(((data_length + index_length) / 1024 / 1024), 2) `Size (MB)`
  FROM information_schema.TABLES
  ORDER BY (data_length + index_length) DESC;

+--------------------+---------------------------------------+-----------+
| DB                 | Table                                 | Size (MB) |
+--------------------+---------------------------------------+-----------+
| wordpress          | wp_simple_history_contexts            |      7.05 |
| wordpress          | wp_posts                              |      6.59 |
| wordpress          | wp_simple_history                     |      3.05 |
| wordpress          | wp_comments                           |      2.73 |
| wordpress          | wp_commentmeta                        |      2.47 |
| wordpress          | wp_simple_login_log                   |      2.03 |
...
```

From MariaDB 11.2.0

```
CREATE TEMPORARY TABLE foo.t1 (a int);

SELECT * FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA='foo' AND TEMPORARY='y'\G
*************************** 1. row ***************************
   TABLE_CATALOG: def
    TABLE_SCHEMA: foo
      TABLE_NAME: t1
      TABLE_TYPE: TEMPORARY
...
       TEMPORARY: Y
```

1.1.1.2.9.1.1.53 Information Schema TABLESPACES Table

The Information Schema `TABLESPACES` table contains information about active tablespaces..

The table is a MariaDB and MySQL extension, and does not include information about InnoDB tablespaces.

Column	Description

TABLESPACE_NAME	
ENGINE	
TABLESPACE_TYPE	
LOGFILE_GROUP_NAME	
EXTENT_SIZE	
AUTOEXTEND_SIZE	
MAXIMUM_SIZE	
NODEGROUP_ID	
TABLESPACE_COMMENT	

1.1.1.2.9.1.1.54 Information Schema THREAD_POOL_GROUPS Table

MariaDB starting with 10.5

The Information Schema `THREAD_POOL_GROUPS` table was introduced in MariaDB 10.5.0.

The table provides information about thread pool groups, and contains the following columns:

Column	Description
GROUP_ID	the thread group this row is showing data for
CONNECTIONS	the number of clients currently connected to this thread group
THREADS	total number of threads in this group (ACTIVE+STANDBY+LISTENER)
ACTIVE_THREADS	number of threads currently executing a query
STANDBY_THREADS	number of threads in reserve that do not currently execute anything
QUEUE_LENGTH	number of client requests waiting for execution
HAS_LISTENER	whether there is an active listener thread right now, always 1 if thread_pool_dedicated_listener is ON
IS_STALLED	whether there's currently an active worker thread in this group that has exceeded thread_pool_stall_limit time

Setting thread_pool_dedicated_listener will give each group its own dedicated listener, and the listener thread will not pick up work items. As a result, the actual queue size in the table will be more exact, since IO requests are immediately dequeued from poll, without delay.

1.1.1.2.9.1.1.55 Information Schema THREAD_POOL_QUEUES Table

MariaDB starting with 10.5

The Information Schema `THREAD_POOL_QUEUES` table was introduced in MariaDB 10.5.0.

The table provides information about thread pool queues, and contains the following columns:

Column	Description
GROUP_ID	the thread group this row is showing data for
POSITION	position in the groups queue
PRIORITY	request priority, see priority scheduling
CONNECTION_ID	ID of the client connection that submitted the queued request

QUEUEING_TIME_MICROSECONDS	how long the request has already been waiting in the queue in microseconds

Setting thread_poll_exact_stats will provides better queueing time statistics by using a high precision timestamp, at a small performance cost, for the time when the connection was added to the queue. This timestamp helps calculate the queuing time shown in the table.

Setting thread_pool_dedicated_listener will give each group its own dedicated listener, and the listener thread will not pick up work items. As a result, the queueing time in the table will be more exact, since IO requests are immediately dequeued from poll, without delay.

1.1.1.2.9.1.1.56 Information Schema THREAD_POOL_STATS Table

MariaDB starting with 10.5

The Information Schema THREAD_POOL_STATS table was introduced in MariaDB 10.5.0.

The table provides performance counter information for the thread pool, and contains the following columns:

Column	Description
GROUP_ID	the thread group this row is showing data for
THREAD_CREATIONS	number of threads created for this group so far
THREAD_CREATIONS_DUE_TO_STALL	number of threads created due to detected stalls
WAKES	
WAKES_DUE_TO_STALL	
THROTTLES	how often thread creation was throttled, see also: thread-creation-throttling
STALLS	number of detected stalls
POLLS_BY_LISTENER	
POLLS_BY_WORKER	
DEQUEUES_BY_LISTENER	
DEQUEUES_BY_WORKER	

1.1.1.2.9.1.1.57 Information Schema THREAD_POOL_WAITS Table

MariaDB starting with 10.5

The Information Schema THREAD_POOL_WAITS table was introduced in MariaDB 10.5.0.

The table provides wait counters for the thread pool, and contains the following columns:

Column	Description
REASON	name of the reason for waiting, e.g. ROW_LOCK, DISKIO, NET ...
COUNT	how often a wait for this specific reason has happened so far

1.1.1.2.9.1.1.58 Information Schema TRIGGERS Table

Contents

The Information Schema TRIGGERS table contains information about triggers.

It has the following columns:

Column	Description
TRIGGER_CATALOG	Always def.
TRIGGER_SCHEMA	Database name in which the trigger occurs.
TRIGGER_NAME	Name of the trigger.
EVENT_MANIPULATION	The event that activates the trigger. One of INSERT, UPDATE or 'DELETE.
EVENT_OBJECT_CATALOG	Always def.
EVENT_OBJECT_SCHEMA	Database name on which the trigger acts.
EVENT_OBJECT_TABLE	Table name on which the trigger acts.
ACTION_ORDER	Indicates the order that the action will be performed in (of the list of a table's triggers with identical EVENT_MANIPULATION and ACTION_TIMING values). Before MariaDB 10.2.3 introduced the FOLLOWS and PRECEDES clauses, always 0
ACTION_CONDITION	NULL
ACTION_STATEMENT	Trigger body, UTF-8 encoded.
ACTION_ORIENTATION	Always ROW.
ACTION_TIMING	Whether the trigger acts BEFORE or AFTER the event that triggers it.
ACTION_REFERENCE_OLD_TABLE	Always NULL.
ACTION_REFERENCE_NEW_TABLE	Always NULL.
ACTION_REFERENCE_OLD_ROW	Always OLD.
ACTION_REFERENCE_NEW_ROW	Always NEW.
CREATED	Always NULL.
SQL_MODE	The SQL_MODE when the trigger was created, and which it uses for execution.
DEFINER	The account that created the trigger, in the form user_name@host_name
CHARACTER_SET_CLIENT	The client character set when the trigger was created, from the session value of the character_set_client system variable.
COLLATION_CONNECTION	The client collation when the trigger was created, from the session value of the collation_connection system variable.
DATABASE_COLLATION	Collation of the associated database.

Queries to the TRIGGERS table will return information only for databases and tables for which you have the TRIGGER privilege. Similar information is returned by the SHOW TRIGGERS statement.

1.1.1.2.9.1.1.59 Information Schema USER_PRIVILEGES Table

The Information Schema USER_PRIVILEGES table contains global user privilege information derived from the mysql.global_priv grant table.

It contains the following columns:

Column	Description
GRANTEE	In the format user_name@host_name.
TABLE_CATALOG	Always def.
PRIVILEGE_TYPE	The specific privilege, for example CREATE USER, RELOAD, SHUTDOWN, SELECT, INSERT, UPDATE or REFERENCES.
IS_GRANTABLE	Whether the user has the GRANT OPTION for this privilege.

The database, table and column privileges returned here are the ones granted on all databases and tables, and by implication all columns.

Similar information can be accessed with the SHOW GRANTS statement. See the GRANT article for more about privileges.

This information is also stored in the mysql.global_priv table, in the `mysql` system database.

1.1.1.2.9.1.1.60 Information Schema USER_STATISTICS Table

The Information Schema `USER_STATISTICS` table holds statistics about user activity. This is part of the User Statistics feature, which is not enabled by default.

You can use this table to find out such things as which user is causing the most load and which users are being abusive. You can also use this table to measure how close to capacity the server may be.

It contains the following columns:

Field	Type	Notes
USER	varchar(48)	The username. The value `'#mysql_system_user#'` appears when there is no username (such as for the slave SQL thread).
TOTAL_CONNECTIONS	int(21)	The number of connections created for this user.
CONCURRENT_CONNECTIONS	int(21)	The number of concurrent connections for this user.
CONNECTED_TIME	int(21)	The cumulative number of seconds elapsed while there were connections from this user.
BUSY_TIME	double	The cumulative number of seconds there was activity on connections from this user.
CPU_TIME	double	The cumulative CPU time elapsed while servicing this user's connections.
BYTES_RECEIVED	int(21)	The number of bytes received from this user's connections.
BYTES_SENT	int(21)	The number of bytes sent to this user's connections.
BINLOG_BYTES_WRITTEN	int(21)	The number of bytes written to the binary log from this user's connections.
ROWS_READ	int(21)	The number of rows read by this user's connections.
ROWS_SENT	int(21)	The number of rows sent by this user's connections.
ROWS_DELETED	int(21)	The number of rows deleted by this user's connections.
ROWS_INSERTED	int(21)	The number of rows inserted by this user's connections.
ROWS_UPDATED	int(21)	The number of rows updated by this user's connections.
SELECT_COMMANDS	int(21)	The number of `SELECT` commands executed from this user's connections.
UPDATE_COMMANDS	int(21)	The number of `UPDATE` commands executed from this user's connections.
OTHER_COMMANDS	int(21)	The number of other commands executed from this user's connections.
COMMIT_TRANSACTIONS	int(21)	The number of `COMMIT` commands issued by this user's connections.
ROLLBACK_TRANSACTIONS	int(21)	The number of `ROLLBACK` commands issued by this user's connections.
DENIED_CONNECTIONS	int(21)	The number of connections denied to this user.
LOST_CONNECTIONS	int(21)	The number of this user's connections that were terminated uncleanly.
ACCESS_DENIED	int(21)	The number of times this user's connections issued commands that were denied.
EMPTY_QUERIES	int(21)	The number of times this user's connections sent queries to the server that did not return data to the client (a per-user aggregate of the empty_queries status variable).

TOTAL_SSL_CONNECTIONS	int(21)	The number of TLS connections created for this user.
MAX_STATEMENT_TIME_EXCEEDED	int(21)	The number of times a statement was aborted, because it was executed longer than its MAX_STATEMENT_TIME threshold.

Example

```
SELECT * FROM information_schema.USER_STATISTICS\G
*************************** 1. row ***************************
                  USER: root
     TOTAL_CONNECTIONS: 1
CONCURRENT_CONNECTIONS: 0
        CONNECTED_TIME: 297
             BUSY_TIME: 0.001725
              CPU_TIME: 0.001982
        BYTES_RECEIVED: 388
            BYTES_SENT: 2327
  BINLOG_BYTES_WRITTEN: 0
             ROWS_READ: 0
             ROWS_SENT: 12
          ROWS_DELETED: 0
         ROWS_INSERTED: 13
          ROWS_UPDATED: 0
       SELECT_COMMANDS: 4
       UPDATE_COMMANDS: 0
        OTHER_COMMANDS: 3
   COMMIT_TRANSACTIONS: 0
 ROLLBACK_TRANSACTIONS: 0
    DENIED_CONNECTIONS: 0
      LOST_CONNECTIONS: 0
         ACCESS_DENIED: 0
         EMPTY_QUERIES: 1
```

1.1.1.2.9.1.1.61 Information Schema USER_VARIABLES Table

Contents

Description

The USER_VARIABLES table is created when the user_variables plugin is enabled, and contains information about user-defined variables.

The table contains the following columns:

Column	Description
VARIABLE_NAME	Variable name.
VARIABLE_VALUE	Variable value.
VARIABLE_TYPE	Variable type.
CHARACTER_SET_NAME	Character set.

User variables are reset and the table emptied with the FLUSH USER_VARIABLES statement.

Example

```
SELECT * FROM information_schema.USER_VARIABLES ORDER BY VARIABLE_NAME;
+---------------+----------------+---------------+--------------------+
| VARIABLE_NAME | VARIABLE_VALUE | VARIABLE_TYPE | CHARACTER_SET_NAME |
+---------------+----------------+---------------+--------------------+
| var           | 0              | INT           | utf8               |
| var2          | abc            | VARCHAR       | utf8               |
+---------------+----------------+---------------+--------------------+
```

1.1.1.2.9.1.1.62 Information Schema VIEWS Table

The Information Schema `VIEWS` table contains information about views. The `SHOW VIEW` privilege is required to view the table.

It has the following columns:

Column	Description
`TABLE_CATALOG`	Aways `def`.
`TABLE_SCHEMA`	Database name containing the view.
`TABLE_NAME`	View table name.
`VIEW_DEFINITION`	Definition of the view.
`CHECK_OPTION`	`YES` if the `WITH CHECK_OPTION` clause has been specified, `NO` otherwise.
`IS_UPDATABLE`	Whether the view is updatable or not.
`DEFINER`	Account specified in the DEFINER clause (or the default when created).
`SECURITY_TYPE`	`SQL SECURITY` characteristic, either `DEFINER` or `INVOKER`.
`CHARACTER_SET_CLIENT`	The client character set when the view was created, from the session value of the character_set_client system variable.
`COLLATION_CONNECTION`	The client collation when the view was created, from the session value of the collation_connection system variable.
`ALGORITHM`	The algorithm used in the view. See View Algorithms.

Example

```
SELECT * FROM information_schema.VIEWS\G
*************************** 1. row ***************************
       TABLE_CATALOG: def
        TABLE_SCHEMA: test
          TABLE_NAME: v
     VIEW_DEFINITION: select `test`.`t`.`qty` AS `qty`,`test`.`t`.`price` AS `price`,
(`test`.`t`.`qty` * `test`.`t`.`price`) AS `value` from `test`.`t`
        CHECK_OPTION: NONE
        IS_UPDATABLE: YES
             DEFINER: root@localhost
       SECURITY_TYPE: DEFINER
CHARACTER_SET_CLIENT: utf8
COLLATION_CONNECTION: utf8_general_ci
           ALGORITHM: UNDEFINED
```

1.1.1.2.9.1.1.63 Information Schema WSREP_MEMBERSHIP Table

The `WSREP_STATUS` table makes Galera node cluster membership information available through the Information Schema. The same information can be returned using the SHOW WSREP_MEMBERSHIP statement. Only users with the SUPER can access information from this table.

The `WSREP_MEMBERSHIP` table is part of the WSREP_INFO plugin.

Example

```
SELECT * FROM information_schema.WSREP_MEMBERSHIP;
+-------+--------------------------------------+-------+-----------------+
| INDEX | UUID                                 | NAME  | ADDRESS         |
+-------+--------------------------------------+-------+-----------------+
|     0 | 46da96e3-6e9e-11e4-95a2-f609aa5444b3 | node1 | 10.0.2.15:16000 |
|     1 | 5f6bc72a-6e9e-11e4-84ed-57ec6780a3d3 | node2 | 10.0.2.15:16001 |
|     2 | 7473fd75-6e9e-11e4-91de-0b541ad91bd0 | node3 | 10.0.2.15:16002 |
+-------+--------------------------------------+-------+-----------------+
```

1.1.1.2.9.1.1.64 Information Schema WSREP_STATUS Table

The WSREP_STATUS table makes Galera node cluster status information available through the Information Schema. The same information can be returned using the SHOW WSREP_STATUS statement. Only users with the SUPER privilege can access information from this table.

The `WSREP_STATUS` table is part of the WSREP_INFO plugin.

Example

```
SELECT * FROM information_schema.WSREP_STATUS\G
*************************** 1. row ***************************
         NODE_INDEX: 0
        NODE_STATUS: Synced
     CLUSTER_STATUS: Primary
       CLUSTER_SIZE: 3
 CLUSTER_STATE_UUID: 00b0fbad-6e84-11e4-8a8b-376f19ce8ee7
CLUSTER_STATE_SEQNO: 2
    CLUSTER_CONF_ID: 3
                GAP: NO
   PROTOCOL_VERSION: 3
```

1.1.1.2.8.2 Extended SHOW

1.1.1.2.9.1.3 TIME_MS column in INFORMATION_SCHEMA.PROCESSLIST

In MariaDB, an extra column `TIME_MS` has been added to the INFORMATION_SCHEMA.PROCESSLIST table. This column shows the same information as the column ' `TIME` ', but in units of milliseconds with microsecond precision (the unit and precision of the `TIME` column is one second).

For details about microseconds support in MariaDB, see microseconds in MariaDB.

The value displayed in the `TIME` and `TIME_MS` columns is the period of time that the given thread has been in its current state. Thus it can be used to check for example how long a thread has been executing the current query, or for how long it has been idle.

```
select id, time, time_ms, command, state from
   information_schema.processlist, (select sleep(2)) t;
+----+------+----------+---------+-----------+
| id | time | time_ms  | command | state     |
+----+------+----------+---------+-----------+
| 37 |    2 | 2000.493 | Query   | executing |
+----+------+----------+---------+-----------+
```

Note that as a difference to MySQL, in MariaDB the `TIME` column (and also the `TIME_MS` column) are not affected by any setting of @TIMESTAMP. This means that it can be reliably used also for threads that change `@TIMESTAMP` (such as the replication SQL thread). See also MySQL Bug #22047.

As a consequence of this, the `TIME` column of `SHOW FULL PROCESSLIST` and `INFORMATION_SCHEMA.PROCESSLIST` can not be used to determine if a slave is lagging behind. For this, use instead the `Seconds_Behind_Master` column in the output of SHOW SLAVE STATUS.

The addition of the TIME_MS column is based on the microsec_process patch, developed by Percona.

1.1.1.2.9.2 Performance Schema

The MariaDB Performance Schema is a feature for monitoring the performance of your MariaDB server. It is not enabled by default - see the overview for details on activating.

Performance Schema Tables

Tables making up the MariaDB Performance Schema.

Performance Schema Overview

Quick overview of the Performance Schema.

Performance Schema Status Variables

Performance Schema status variables.

Performance Schema System Variables

Performance Schema system variables.

Performance Schema Digests

Normalized statements with data values removed

PERFORMANCE_SCHEMA Storage Engine

PERFORMANCE_SCHEMA storage engine, a mechanism for implementing the feature.

There are 4 related questions.

1.1.1.2.9.2.1 Performance Schema Tables

Tables that are part of the MariaDB Performance Schema, a feature for monitoring the performance of MariaDB server.

List of Performance Schema Tables

List and short description of all performance_schema tables.

Performance Schema accounts Table

Account connection information.

Performance Schema cond_instances Table

List of instrumented condition objects.

Performance Schema events_stages_current Table

Current stage events.

Performance Schema events_stages_history Table

Most recent stage events per thread.

Performance Schema events_stages_history_long Table

Most recent completed stage events.

Performance Schema events_stages_summary_by_account_by_event_name Table

Stage events, summarized by account and event name.

Performance Schema events_stages_summary_by_host_by_event_name Table

Stage events summarized by host and event name.

Performance Schema events_stages_summary_by_thread_by_event_name Table

Stage events summarized by thread and event name.

Performance Schema events_stages_summary_by_user_by_event_name Table

Stage events summarized by user and event name.

Performance Schema events_stages_summary_global_by_event_name Table

Event summaries.

Performance Schema events_statements_current Table

Current statement events.

Performance Schema events_statements_history Table

Most recent statement events per thread

Performance Schema events_statements_history_long Table

Most recent statement events.

Performance Schema events_statements_summary_by_account_by_event_name Table

Statement events summarized by account and event name.

Performance Schema events_statements_summary_by_digest Table

Statement events summarized by schema and digest.

Performance Schema events_statements_summary_by_host_by_event_name Table

Statement events summarized by host and event name.

Performance Schema events_statements_summary_by_program Table

Summarizes events for a particular stored program.

Performance Schema events_statements_summary_by_thread_by_event_name Table

Statement events summarized by thread and event name.

Performance Schema events_statements_summary_by_user_by_event_name Table

Statement events summarized by user and event name.

Performance Schema events_statements_summary_global_by_event_name Table

Statement events summarized by event name.

Performance Schema events_transactions_current Table

Current transaction events for each thread.

Performance Schema events_transactions_history Table

Most recent completed transaction events for each thread.

Performance Schema events_transactions_history_long Table

Most recent completed transaction events that have ended globally.

Performance Schema events_transactions_summary_by_account_by_event_name Table

Transaction events aggregated by account and event name.

Performance Schema events_transactions_summary_by_host_by_event_name Table

Transaction events aggregated by host and event name.

Performance Schema events_transactions_summary_by_thread_by_event_name Table

Transaction events aggregated by thread and event name.

Performance Schema events_transactions_summary_by_user_by_event_name Table

Transaction events aggregated by user and event name.

Performance Schema events_transactions_summary_global_by_event_name Table

Transaction events aggregated by event name.

Performance Schema events_waits_current Table

Current wait events

Performance Schema events_waits_history Table

Most recent wait events per thread

Performance Schema events_waits_history_long Table

Most recent completed wait events

Performance Schema events_waits_summary_by_account_by_event_name Table

Wait events summarized by account and event name.

Performance Schema events_waits_summary_by_host_by_event_name Table

Wait events summarized by host and event name.

Performance Schema events_waits_summary_by_instance Table

Wait events summarized by instance

Performance Schema events_waits_summary_by_thread_by_event_name Table

Wait events summarized by thread and event name.

Performance Schema events_waits_summary_by_user_by_event_name Table

Wait events summarized by user and event name.

Performance Schema events_waits_summary_global_by_event_name Table

Wait events summarized by event name.

Performance Schema file_instances Table

List of file instruments.

Performance Schema file_summary_by_event_name Table

File events summarized by event name.

Performance Schema file_summary_by_instance Table

File events summarized by instance.

Performance Schema global_status Table

Status variables and their global values.

Performance Schema hosts Table

Hosts used to connect to the server.

Performance Schema host_cache Table

Host_cache information.

Performance Schema memory_summary_by_account_by_event_name Table

Memory usage statistics aggregated by account and event.

Performance Schema memory_summary_by_host_by_event_name Table

Memory usage statistics aggregated by host and event.

Performance Schema memory_summary_by_thread_by_event_name Table

Memory usage statistics aggregated by thread and event.

Performance Schema memory_summary_by_user_by_event_name Table

Memory usage statistics aggregated by user and event.

Performance Schema memory_summary_global_by_event_name Table

Memory usage statistics aggregated by event and event.

Performance Schema metadata_locks Table

Metadata lock information.

Performance Schema mutex_instances Table

Seen mutexes

Performance Schema objects_summary_global_by_type Table

Aggregates object wait events.

Performance Schema performance_timers Table

Available event timers

Performance Schema prepared_statements_instances Table

Aggregated statistics of prepared statements.

Performance Schema replication_applier_configuration Table

Configuration settings affecting replica transactions.

Performance Schema replication_applier_status Table

Information about the general transaction execution status on the slave.

Performance Schema replication_applier_status_by_coordinator Table

Coordinator thread status used in multi-threaded replicas to manage multiple workers.

Performance Schema replication_applier_status_by_worker Table

Slave worker thread specific information.

Performance Schema replication_connection_configuration Table

Replica configuration settings used for connecting to the primary.

Performance Schema rwlock_instances Table

Seen read-write locks

Performance Schema session_account_connect_attrs Table

Connection attributes for the current session.

Performance Schema session_connect_attrs Table

Connection attributes for all sessions.

Performance Schema session_status Table

Status variables and their session values.

Performance Schema setup_actors Table

Determines whether monitoring is enabled for host/user combinations.

Performance Schema setup_consumers Table

Lists the types of consumers for which event information is available.

Performance Schema setup_instruments Table

List of instrumented object classes

Performance Schema setup_objects Table

Which objects are monitored.

Performance Schema setup_timers Table

Currently selected event timers

Performance Schema socket_instances Table

Active server connections.

Performance Schema socket_summary_by_event_name Table

Aggregates timer and byte count statistics for all socket I/O operations by socket instrument.

Performance Schema socket_summary_by_instance Table

Aggregates timer and byte count statistics for all socket I/O operations by socket instance.

Performance Schema status_by_account Table

Status variable information by user/host account.

Performance Schema status_by_host Table

Status variable information by host.

Performance Schema status_by_thread Table

Status variable information about active foreground threads.

Performance Schema status_by_user Table

Status variable information by user.

Performance Schema table_handles Table

Table lock information.

Performance Schema table_io_waits_summary_by_index_usage Table

Table I/O waits by index.

Performance Schema table_io_waits_summary_by_table Table

Table I/O waits by table.

Performance Schema table_lock_waits_summary_by_table Table

Table lock waits by table.

Performance Schema threads Table

Each server thread is represented as a row in the threads table.

Performance Schema users Table

User connection information.

Performance Schema user_variables_by_thread Table

User-defined variables and the threads that defined them.

1.1.1.2.9.2.1.1 List of Performance Schema Tables

Below is a list of all Performance Schema tables as well as a brief description of each of them.

Table	Description
accounts	Client account connection statistics.
cond_instances	Synchronization object instances.
events_stages_current	Current stage events.
events_stages_history	Ten most recent stage events per thread.
events_stages_history_long	Ten thousand most recent stage events.
events_stages_summary_by_account_by_event_name	Summarized stage events per account and event name.
events_stages_summary_by_host_by_event_name	Summarized stage events per host and event name.
events_stages_summary_by_thread_by_event_name	Summarized stage events per thread and event name.
events_stages_summary_by_user_by_event_name	Summarized stage events per user name and event name.
events_stages_summary_global_by_event_name	Summarized stage events per event name.
events_statements_current	Current statement events.

events_statements_history	Ten most recent events per thread.
events_statements_history_long	Ten thousand most recent stage events.
events_statements_summary_by_account_by_event_name	Summarized statement events per account and event name.
events_statements_summary_by_digest	Summarized statement events by scheme and digest.
events_statements_summary_by_host_by_event_name	Summarized statement events by host and event name.
events_statements_summary_by_program	Events for a particular stored program.
events_statements_summary_by_thread_by_event_name	Summarized statement events by thread and event name.
events_statements_summary_by_user_by_event_name	Summarized statement events by user and event name.
events_statements_summary_global_by_event_name	Summarized statement events by event name.
events_transactions_current	Current transaction events for each thread.
events_transactions_history	Most recent completed transaction events for each thread.
events_transactions_history_long	Most recent completed transaction events that have ended globally.
events_transactions_summary_by_account_by_event_name	Transaction events aggregated by account and event.
events_transactions_summary_by_host_by_event_name	Transaction events aggregated by host and event..
events_transactions_summary_by_thread_by_event_name	Transaction events aggregated by thread and event..
events_transactions_summary_by_user_by_event_name	Transaction events aggregated by user and event..
events_transactions_summary_global_by_event_name	Transaction events aggregated by event name.
events_waits_current	Current wait events.
events_waits_history	Ten most recent wait events per thread.
events_waits_history_long	Ten thousand most recent wait events per thread.
events_waits_summary_by_account_by_event_name	Summarized wait events by account and event name.
events_waits_summary_by_host_by_event_name	Summarized wait events by host and event name.
events_waits_summary_by_instance	Summarized wait events by instance.
events_waits_summary_by_thread_by_event_name	Summarized wait events by thread and event name.
events_waits_summary_by_user_by_event_name	Summarized wait events by user and event name.
events_waits_summary_global_by_event_name	Summarized wait events by event name.
file_instances	Seen files.
file_summary_by_event_name	File events summarized by event name.
file_summary_by_instance	File events summarized by instance.
global_status	Global status variables and values.
host_cache	Host and IP information.
hosts	Connections by host.
memory_summary_by_account_by_event_name	Memory usage statistics aggregated by account and event.
memory_summary_by_host_by_event_name	Memory usage statistics aggregated by host. and event.

memory_summary_by_thread_by_event_name	Memory usage statistics aggregated by thread and event..
memory_summary_by_user_by_event_name	Memory usage statistics aggregated by user and event..
memory_summary_global_by_event_name	Memory usage statistics aggregated by event.
metadata_locks	Metadata locks.
mutex_instances	Seen mutexes.
objects_summary_global_by_type	Object wait events.
performance_timers	Available event timers.
prepared_statements_instances	Aggregate statistics of prepared statements.
replication_applier_configuration	Configuration settings affecting replica transactions.
replication_applier_status	General transaction execution status on the replica.
replication_applier_status_by_coordinator	Coordinator thread specific information.
replication_applier_status_by_worker	Replica worker thread specific information.
replication_connection_configuration	Rreplica's configuration settings used for connecting to the primary.
rwlock_instances	Seen read-write locks.
session_account_connect_attrs	Current session connection attributes.
session_connect_attrs	All session connection attributes.
session_status	Session status variables and values.
setup_actors	Details on foreground thread monitoring.
setup_consumers	Consumers for which event information is stored.
setup_instruments	Instrumented objects for which events are collected.
setup_objects	Objects to be monitored.
setup_timers	Currently selected event timers.
socket_instances	Active connections.
socket_summary_by_event_name	Timer and byte count statistics by socket instrument.
socket_summary_by_instance	Timer and byte count statistics by socket instance.
status_by_account	Status variable info by host/user account.
status_by_host	Status variable info by host.
status_by_thread	Status variable info about active foreground threads.
status_by_user	Status variable info by user.
table_handles	Table lock information.
table_io_waits_summary_by_index_usage	Aggregate table I/O wait events by index.
table_io_waits_summary_by_table	Aggregate table I/O wait events by table.
table_lock_waits_summary_by_table	Aggregate table lock wait events by table.
threads	Server thread information.
user_variables_by_thread	User-defined variables by thread.
users	Connection statistics by user.

1.1.1.2.9.2.1.2 Performance Schema accounts Table

Description

Each account that connects to the server is stored as a row in the accounts table, along with current and total connections.

The table size is determined at startup by the value of the performance_schema_accounts_size system variable. If this is set to 0, account statistics will be disabled.

Column	Description
USER	The connection's client user name for the connection, or NULL if an internal thread.
HOST	The connection client's host name, or NULL if an internal thread.
CURRENT_CONNECTIONS	Current connections for the account.
TOTAL_CONNECTIONS	Total connections for the account.

The USER and HOST values shown here are the username and host used for user connections, not the patterns used to check permissions.

Example

```
SELECT * FROM performance_schema.accounts;
+------------------+-----------+---------------------+-------------------+
| USER             | HOST      | CURRENT_CONNECTIONS | TOTAL_CONNECTIONS |
+------------------+-----------+---------------------+-------------------+
| root             | localhost |                   1 |                 2 |
| NULL             | NULL      |                  20 |                23 |
| debian-sys-maint | localhost |                   0 |                35 |
+------------------+-----------+---------------------+-------------------+
```

1.1.1.2.9.2.1.3 Performance Schema cond_instances Table

Description

The cond_instances table lists all conditions while the server is executing. A condition, or instrumented condition object, is an internal code mechanism used for signalling that a specific event has occurred so that any threads waiting for this condition can continue.

The maximum number of conditions stored in the performance schema is determined by the performance_schema_max_cond_instances system variable.

Column	Description
NAME	Client user name for the connection, or NULL if an internal thread.
OBJECT_INSTANCE_BEGIN	Address in memory of the instrumented condition.

1.1.1.2.9.2.1.4 Performance Schema events_stages_current Table

The events_stages_current table contains current stage events, with each row being a record of a thread and its most recent stage event.

The table contains the following columns:

Column	Description
THREAD_ID	Thread associated with the event. Together with EVENT_ID uniquely identifies the row.
EVENT_ID	Thread's current event number at the start of the event. Together with THREAD_ID uniquely identifies the row.

END_EVENT_ID	NULL when the event starts, set to the thread's current event number at the end of the event.
EVENT_NAME	Event instrument name and a NAME from the setup_instruments table
SOURCE	Name and line number of the source file containing the instrumented code that produced the event.
TIMER_START	Value in picoseconds when the event timing started or NULL if timing is not collected.
TIMER_END	Value in picoseconds when the event timing ended, or NULL if the event has not ended or timing is not collected.
TIMER_WAIT	Value in picoseconds of the event's duration or NULL if the event has not ended or timing is not collected.
NESTING_EVENT_ID	EVENT_ID of event within which this event nests.
NESTING_EVENT_TYPE	Nesting event type. One of transaction, statement, stage or wait.

It is possible to empty this table with a TRUNCATE TABLE statement.

The related tables, events_stages_history and events_stages_history_long derive their values from the current events.

1.1.1.2.9.2.1.5 Performance Schema events_stages_history Table

The events_stages_history table by default contains the ten most recent completed stage events per thread. This number can be adjusted by setting the performance_schema_events_stages_history_size system variable when the server starts up.

The table structure is identical to the events_stage_current table structure, and contains the following columns:

Column	Description
THREAD_ID	Thread associated with the event. Together with EVENT_ID uniquely identifies the row.
EVENT_ID	Thread's current event number at the start of the event. Together with THREAD_ID uniquely identifies the row.
END_EVENT_ID	NULL when the event starts, set to the thread's current event number at the end of the event.
EVENT_NAME	Event instrument name and a NAME from the setup_instruments table
SOURCE	Name and line number of the source file containing the instrumented code that produced the event.
TIMER_START	Value in picoseconds when the event timing started or NULL if timing is not collected.
TIMER_END	Value in picoseconds when the event timing ended, or NULL if timing is not collected.
TIMER_WAIT	Value in picoseconds of the event's duration or NULL if timing is not collected.
NESTING_EVENT_ID	EVENT_ID of event within which this event nests.
NESTING_EVENT_TYPE	Nesting event type. One of transaction, statement, stage or wait.

It is possible to empty this table with a TRUNCATE TABLE statement.

events_stages_current and events_stages_history_long are related tables.

1.1.1.2.9.2.1.6 Performance Schema events_stages_history_long Table

The events_stages_history_long table by default contains the ten thousand most recent completed stage events. This number can be adjusted by setting the performance_schema_events_stages_history_long_size system variable when the server starts up.

The table structure is identical to the events_stage_current table structure, and contains the following columns:

Column	Description

THREAD_ID	Thread associated with the event. Together with EVENT_ID uniquely identifies the row.
EVENT_ID	Thread's current event number at the start of the event. Together with THREAD_ID uniquely identifies the row.
END_EVENT_ID	NULL when the event starts, set to the thread's current event number at the end of the event.
EVENT_NAME	Event instrument name and a NAME from the setup_instruments table
SOURCE	Name and line number of the source file containing the instrumented code that produced the event.
TIMER_START	Value in picoseconds when the event timing started or NULL if timing is not collected.
TIMER_END	Value in picoseconds when the event timing ended, or NULL if timing is not collected.
TIMER_WAIT	Value in picoseconds of the event's duration or NULL if timing is not collected.
NESTING_EVENT_ID	EVENT_ID of event within which this event nests.
NESTING_EVENT_TYPE	Nesting event type. One of transaction , statement , stage or wait .

It is possible to empty this table with a TRUNCATE TABLE statement.

events_stages_current and events_stages_history are related tables.

1.1.1.2.9.2.1.7 Performance Schema events_stages_summary_by_account_by_event_ Table

The table lists stage events, summarized by account and event name.

It contains the following columns:

Column	Description
USER	User. Used together with HOST and EVENT_NAME for grouping events.
HOST	Host. Used together with USER and EVENT_NAME for grouping events.
EVENT_NAME	Event name. Used together with USER and HOST for grouping events.
COUNT_STAR	Number of summarized events, which includes all timed and untimed events.
SUM_TIMER_WAIT	Total wait time of the timed summarized events.
MIN_TIMER_WAIT	Minimum wait time of the timed summarized events.
AVG_TIMER_WAIT	Average wait time of the timed summarized events.
MAX_TIMER_WAIT	Maximum wait time of the timed summarized events.

Example

```
SELECT * FROM events_stages_summary_by_account_by_event_name\G
...
*************************** 325. row ***************************
          USER: NULL
          HOST: NULL
    EVENT_NAME: stage/sql/Waiting for event metadata lock
    COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
*************************** 326. row ***************************
          USER: NULL
          HOST: NULL
    EVENT_NAME: stage/sql/Waiting for commit lock
    COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
*************************** 327. row ***************************
          USER: NULL
          HOST: NULL
    EVENT_NAME: stage/aria/Waiting for a resource
    COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
```

1.1.1.2.9.2.1.8 Performance Schema events_stages_summary_by_host_by_event_nan Table

The table lists stage events, summarized by host and event name.

It contains the following columns:

Column	Description
`HOST`	Host. Used together with `EVENT_NAME` for grouping events.
`EVENT_NAME`	Event name. Used together with `HOST` for grouping events.
`COUNT_STAR`	Number of summarized events, which includes all timed and untimed events.
`SUM_TIMER_WAIT`	Total wait time of the timed summarized events.
`MIN_TIMER_WAIT`	Minimum wait time of the timed summarized events.
`AVG_TIMER_WAIT`	Average wait time of the timed summarized events.
`MAX_TIMER_WAIT`	Maximum wait time of the timed summarized events.

Example

```
SELECT * FROM events_stages_summary_by_host_by_event_name\G
...
*************************** 216. row ***************************
          HOST: NULL
    EVENT_NAME: stage/sql/Waiting for event metadata lock
    COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
*************************** 217. row ***************************
          HOST: NULL
    EVENT_NAME: stage/sql/Waiting for commit lock
    COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
*************************** 218. row ***************************
          HOST: NULL
    EVENT_NAME: stage/aria/Waiting for a resource
    COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
```

1.1.1.2.9.2.1.9 Performance Schema events_stages_summary_by_thread_by_event_n Table

The table lists stage events, summarized by thread and event name.

It contains the following columns:

Column	Description
`THREAD_ID`	Thread associated with the event. Together with `EVENT_NAME` uniquely identifies the row.
`EVENT_NAME`	Event name. Used together with `THREAD_ID` for grouping events.
`COUNT_STAR`	Number of summarized events, which includes all timed and untimed events.
`SUM_TIMER_WAIT`	Total wait time of the timed summarized events.
`MIN_TIMER_WAIT`	Minimum wait time of the timed summarized events.
`AVG_TIMER_WAIT`	Average wait time of the timed summarized events.
`MAX_TIMER_WAIT`	Maximum wait time of the timed summarized events.

Example

```
SELECT * FROM events_stages_summary_by_thread_by_event_name\G
...
*************************** 2287. row ***************************
     THREAD_ID: 64
    EVENT_NAME: stage/sql/Waiting for event metadata lock
    COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
*************************** 2288. row ***************************
     THREAD_ID: 64
    EVENT_NAME: stage/sql/Waiting for commit lock
    COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
*************************** 2289. row ***************************
     THREAD_ID: 64
    EVENT_NAME: stage/aria/Waiting for a resource
    COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
```

1.1.1.2.9.2.1.10 Performance Schema events_stages_summary_by_user_by_event_nan Table

The table lists stage events, summarized by user and event name.

It contains the following columns:

Column	Description
`USER`	User. Used together with `EVENT_NAME` for grouping events.
`EVENT_NAME`	Event name. Used together with `USER` for grouping events.
`COUNT_STAR`	Number of summarized events, which includes all timed and untimed events.
`SUM_TIMER_WAIT`	Total wait time of the timed summarized events.
`MIN_TIMER_WAIT`	Minimum wait time of the timed summarized events.
`AVG_TIMER_WAIT`	Average wait time of the timed summarized events.
`MAX_TIMER_WAIT`	Maximum wait time of the timed summarized events.

Example

```
SELECT * FROM events_stages_summary_by_user_by_event_name\G
...
*************************** 325. row ***************************
          USER: NULL
    EVENT_NAME: stage/sql/Waiting for event metadata lock
    COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
*************************** 326. row ***************************
          USER: NULL
    EVENT_NAME: stage/sql/Waiting for commit lock
    COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
*************************** 327. row ***************************
          USER: NULL
    EVENT_NAME: stage/aria/Waiting for a resource
    COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
```

1.1.1.2.9.2.1.11 Performance Schema events_stages_summary_global_by_event_name Table

The table lists stage events, summarized by thread and event name.

It contains the following columns:

Column	Description
EVENT_NAME	Event name.
COUNT_STAR	Number of summarized events, which includes all timed and untimed events.
SUM_TIMER_WAIT	Total wait time of the timed summarized events.
MIN_TIMER_WAIT	Minimum wait time of the timed summarized events.
AVG_TIMER_WAIT	Average wait time of the timed summarized events.
MAX_TIMER_WAIT	Maximum wait time of the timed summarized events.

Example

```
SELECT * FROM events_stages_summary_global_by_event_name\G
...
*************************** 106. row ***************************
    EVENT_NAME: stage/sql/Waiting for trigger metadata lock
    COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
*************************** 107. row ***************************
    EVENT_NAME: stage/sql/Waiting for event metadata lock
    COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
*************************** 108. row ***************************
    EVENT_NAME: stage/sql/Waiting for commit lock
    COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
*************************** 109. row ***************************
    EVENT_NAME: stage/aria/Waiting for a resource
    COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
```

1.1.1.2.9.2.1.12 Performance Schema events_statements_current Table

The `events_statements_current` table contains current statement events, with each row being a record of a thread and its most recent statement event.

The table contains the following columns:

Column	Description
`THREAD_ID`	Thread associated with the event. Together with `EVENT_ID` uniquely identifies the row.
`EVENT_ID`	Thread's current event number at the start of the event. Together with `THREAD_ID` uniquely identifies the row.
`END_EVENT_ID`	`NULL` when the event starts, set to the thread's current event number at the end of the event.
`EVENT_NAME`	Event instrument name and a `NAME` from the `setup_instruments` table
`SOURCE`	Name and line number of the source file containing the instrumented code that produced the event.
`TIMER_START`	Value in picoseconds when the event timing started or `NULL` if timing is not collected.
`TIMER_END`	Value in picoseconds when the event timing ended, or `NULL` if the event has not ended or timing is not collected.
`TIMER_WAIT`	Value in picoseconds of the event's duration or `NULL` if the event has not ended or timing is not collected.
`LOCK_TIME`	Time in picoseconds spent waiting for locks. The time is calculated in microseconds but stored in picoseconds for compatibility with other timings.
`SQL_TEXT`	The SQL statement, or `NULL` if the command is not associated with an SQL statement.
`DIGEST`	Statement digest.
`DIGEST_TEXT`	Statement digest text.
`CURRENT_SCHEMA`	Statement's default database for the statement, or `NULL` if there was none.

OBJECT_SCHEMA	Reserved, currently NULL
OBJECT_NAME	Reserved, currently NULL
OBJECT_TYPE	Reserved, currently NULL
OBJECT_INSTANCE_BEGIN	Address in memory of the statement object.
MYSQL_ERRNO	Error code. See MariaDB Error Codes for a full list.
RETURNED_SQLSTATE	The SQLSTATE value.
MESSAGE_TEXT	Statement error message. See MariaDB Error Codes.
ERRORS	0 if SQLSTATE signifies completion (starting with 00) or warning (01), otherwise 1.
WARNINGS	Number of warnings from the diagnostics area.
ROWS_AFFECTED	Number of rows affected the statement affected.
ROWS_SENT	Number of rows returned.
ROWS_EXAMINED	Number of rows read during the statement's execution.
CREATED_TMP_DISK_TABLES	Number of on-disk temp tables created by the statement.
CREATED_TMP_TABLES	Number of temp tables created by the statement.
SELECT_FULL_JOIN	Number of joins performed by the statement which did not use an index.
SELECT_FULL_RANGE_JOIN	Number of joins performed by the statement which used a range search of the first table.
SELECT_RANGE	Number of joins performed by the statement which used a range of the first table.
SELECT_RANGE_CHECK	Number of joins without keys performed by the statement that check for key usage after each row.
SELECT_SCAN	Number of joins performed by the statement which used a full scan of the first table.
SORT_MERGE_PASSES	Number of merge passes by the sort algorithm performed by the statement. If too high, you may need to increase the sort_buffer_size.
SORT_RANGE	Number of sorts performed by the statement which used a range.
SORT_ROWS	Number of rows sorted by the statement.
SORT_SCAN	Number of sorts performed by the statement which used a full table scan.
NO_INDEX_USED	0 if the statement performed a table scan with an index, 1 if without an index.
NO_GOOD_INDEX_USED	0 if a good index was found for the statement, 1 if no good index was found. See the Range checked for each record description in the EXPLAIN article.
NESTING_EVENT_ID	Reserved, currently NULL.
NESTING_EVENT_TYPE	Reserved, currently NULL.

It is possible to empty this table with a TRUNCATE TABLE statement.

The related tables, events_statements_history and events_statements_history_long derive their values from the current events table.

1.1.1.2.9.2.1.13 Performance Schema events_statements_history Table

The events_statements_history table by default contains the ten most recent completed statement events per thread. This number can be adjusted by setting the performance_schema_events_statements_history_size system variable when the server starts up.

The table structure is identical to the events_statements_current table structure, and contains the following columns:

The table contains the following columns:

Column	Description
THREAD_ID	Thread associated with the event. Together with EVENT_ID uniquely identifies the row.

EVENT_ID	Thread's current event number at the start of the event. Together with THREAD_ID uniquely identifies the row.
END_EVENT_ID	NULL when the event starts, set to the thread's current event number at the end of the event.
EVENT_NAME	Event instrument name and a NAME from the setup_instruments table
SOURCE	Name and line number of the source file containing the instrumented code that produced the event.
TIMER_START	Value in picoseconds when the event timing started or NULL if timing is not collected.
TIMER_END	Value in picoseconds when the event timing ended, or NULL if timing is not collected.
TIMER_WAIT	Value in picoseconds of the event's duration or NULL if timing is not collected.
LOCK_TIME	Time in picoseconds spent waiting for locks. The time is calculated in microseconds but stored in picoseconds for compatibility with other timings.
SQL_TEXT	The SQL statement, or NULL if the command is not associated with an SQL statement.
DIGEST	Statement digest.
DIGEST_TEXT	Statement digest text.
CURRENT_SCHEMA	Statement's default database for the statement, or NULL if there was none.
OBJECT_SCHEMA	Reserved, currently NULL
OBJECT_NAME	Reserved, currently NULL
OBJECT_TYPE	Reserved, currently NULL
OBJECT_INSTANCE_BEGIN	Address in memory of the statement object.
MYSQL_ERRNO	Error code. See MariaDB Error Codes for a full list.
RETURNED_SQLSTATE	The SQLSTATE value.
MESSAGE_TEXT	Statement error message. See MariaDB Error Codes.
ERRORS	0 if SQLSTATE signifies completion (starting with 00) or warning (01), otherwise 1 .
WARNINGS	Number of warnings from the diagnostics area.
ROWS_AFFECTED	Number of rows affected the statement affected.
ROWS_SENT	Number of rows returned.
ROWS_EXAMINED	Number of rows read during the statement's execution.
CREATED_TMP_DISK_TABLES	Number of on-disk temp tables created by the statement.
CREATED_TMP_TABLES	Number of temp tables created by the statement.
SELECT_FULL_JOIN	Number of joins performed by the statement which did not use an index.
SELECT_FULL_RANGE_JOIN	Number of joins performed by the statement which used a range search of the first table.
SELECT_RANGE	Number of joins performed by the statement which used a range of the first table.
SELECT_RANGE_CHECK	Number of joins without keys performed by the statement that check for key usage after each row.
SELECT_SCAN	Number of joins performed by the statement which used a full scan of the first table.
SORT_MERGE_PASSES	Number of merge passes by the sort algorithm performed by the statement. If too high, you may need to increase the sort_buffer_size.
SORT_RANGE	Number of sorts performed by the statement which used a range.
SORT_ROWS	Number of rows sorted by the statement.
SORT_SCAN	Number of sorts performed by the statement which used a full table scan.
NO_INDEX_USED	0 if the statement performed a table scan with an index, 1 if without an index.
NO_GOOD_INDEX_USED	0 if a good index was found for the statement, 1 if no good index was found. See the Range checked for each record description in the EXPLAIN article.

NESTING_EVENT_ID	Reserved, currently NULL.
NESTING_EVENT_TYPE	Reserved, currently NULL.

It is possible to empty this table with a TRUNCATE TABLE statement.

events_statements_current and events_statements_history_long are related tables.

1.1.1.2.9.2.1.14 Performance Schema events_statements_history_long Table

The events_statements_history_long table by default contains the ten thousand most recent completed statement events. This number can be adjusted by setting the performance_schema_events_statements_history_long_size system variable when the server starts up.

The table structure is identical to the events_statements_current table structure, and contains the following columns:

Column	Description
THREAD_ID	Thread associated with the event. Together with EVENT_ID uniquely identifies the row.
EVENT_ID	Thread's current event number at the start of the event. Together with THREAD_ID uniquely identifies the row.
END_EVENT_ID	NULL when the event starts, set to the thread's current event number at the end of the event.
EVENT_NAME	Event instrument name and a NAME from the setup_instruments table
SOURCE	Name and line number of the source file containing the instrumented code that produced the event.
TIMER_START	Value in picoseconds when the event timing started or NULL if timing is not collected.
TIMER_END	Value in picoseconds when the event timing ended, or NULL if timing is not collected.
TIMER_WAIT	Value in picoseconds of the event's duration or NULL if timing is not collected.
LOCK_TIME	Time in picoseconds spent waiting for locks. The time is calculated in microseconds but stored in picoseconds for compatibility with other timings.
SQL_TEXT	The SQL statement, or NULL if the command is not associated with an SQL statement.
DIGEST	Statement digest.
DIGEST_TEXT	Statement digest text.
CURRENT_SCHEMA	Statement's default database for the statement, or NULL if there was none.
OBJECT_SCHEMA	Reserved, currently NULL
OBJECT_NAME	Reserved, currently NULL
OBJECT_TYPE	Reserved, currently NULL
OBJECT_INSTANCE_BEGIN	Address in memory of the statement object.
MYSQL_ERRNO	Error code. See MariaDB Error Codes for a full list.
RETURNED_SQLSTATE	The SQLSTATE value.
MESSAGE_TEXT	Statement error message. See MariaDB Error Codes.
ERRORS	0 if SQLSTATE signifies completion (starting with 00) or warning (01), otherwise 1.
WARNINGS	Number of warnings from the diagnostics area.
ROWS_AFFECTED	Number of rows affected the statement affected.
ROWS_SENT	Number of rows returned.
ROWS_EXAMINED	Number of rows read during the statement's execution.
CREATED_TMP_DISK_TABLES	Number of on-disk temp tables created by the statement.
CREATED_TMP_TABLES	Number of temp tables created by the statement.

SELECT_FULL_JOIN	Number of joins performed by the statement which did not use an index.
SELECT_FULL_RANGE_JOIN	Number of joins performed by the statement which used a range search of the first table.
SELECT_RANGE	Number of joins performed by the statement which used a range of the first table.
SELECT_RANGE_CHECK	Number of joins without keys performed by the statement that check for key usage after each row.
SELECT_SCAN	Number of joins performed by the statement which used a full scan of the first table.
SORT_MERGE_PASSES	Number of merge passes by the sort algorithm performed by the statement. If too high, you may need to increase the sort_buffer_size.
SORT_RANGE	Number of sorts performed by the statement which used a range.
SORT_ROWS	Number of rows sorted by the statement.
SORT_SCAN	Number of sorts performed by the statement which used a full table scan.
NO_INDEX_USED	0 if the statement performed a table scan with an index, 1 if without an index.
NO_GOOD_INDEX_USED	0 if a good index was found for the statement, 1 if no good index was found. See the Range checked for each record description in the EXPLAIN article.
NESTING_EVENT_ID	Reserved, currently NULL.
NESTING_EVENT_TYPE	Reserved, currently NULL.

It is possible to empty this table with a TRUNCATE TABLE statement.

events_statements_current and events_statements_history are related tables.

1.1.1.2.9.2.1.15 Performance Schema events_statements_summary_by_account_by_ev Table

The Performance Schema events_statements_summary_by_account_by_event_name table contains statement events summarized by account and event name. It contains the following columns:

Column	Description
USER	User. Used together with HOST and EVENT_NAME for grouping events.
HOST	Host. Used together with USER and EVENT_NAME for grouping events.
EVENT_NAME	Event name. Used together with USER and HOST for grouping events.
COUNT_STAR	Number of summarized events
SUM_TIMER_WAIT	Total wait time of the summarized events that are timed.
MIN_TIMER_WAIT	Minimum wait time of the summarized events that are timed.
AVG_TIMER_WAIT	Average wait time of the summarized events that are timed.
MAX_TIMER_WAIT	Maximum wait time of the summarized events that are timed.
SUM_LOCK_TIME	Sum of the LOCK_TIME column in the events_statements_current table.
SUM_ERRORS	Sum of the ERRORS column in the events_statements_current table.
SUM_WARNINGS	Sum of the WARNINGS column in the events_statements_current table.
SUM_ROWS_AFFECTED	Sum of the ROWS_AFFECTED column in the events_statements_current table.
SUM_ROWS_SENT	Sum of the ROWS_SENT column in the events_statements_current table.
SUM_ROWS_EXAMINED	Sum of the ROWS_EXAMINED column in the events_statements_current table.
SUM_CREATED_TMP_DISK_TABLES	Sum of the CREATED_TMP_DISK_TABLES column in the events_statements_current table.
SUM_CREATED_TMP_TABLES	Sum of the CREATED_TMP_TABLES column in the events_statements_current table.

SUM_SELECT_FULL_JOIN	Sum of the SELECT_FULL_JOIN column in the events_statements_current table.
SUM_SELECT_FULL_RANGE_JOIN	Sum of the SELECT_FULL_RANGE_JOIN column in the events_statements_current table.
SUM_SELECT_RANGE	Sum of the SELECT_RANGE column in the events_statements_current table.
SUM_SELECT_RANGE_CHECK	Sum of the SELECT_RANGE_CHECK column in the events_statements_current table.
SUM_SELECT_SCAN	Sum of the SELECT_SCAN column in the events_statements_current table.
SUM_SORT_MERGE_PASSES	Sum of the SORT_MERGE_PASSES column in the events_statements_current table.
SUM_SORT_RANGE	Sum of the SORT_RANGE column in the events_statements_current table.
SUM_SORT_ROWS	Sum of the SORT_ROWS column in the events_statements_current table.
SUM_SORT_SCAN	Sum of the SORT_SCAN column in the events_statements_current table.
SUM_NO_INDEX_USED	Sum of the NO_INDEX_USED column in the events_statements_current table.
SUM_NO_GOOD_INDEX_USED	Sum of the NO_GOOD_INDEX_USED column in the events_statements_current table.

The *_TIMER_WAIT columns only calculate results for timed events, as non-timed events have a NULL wait time.

Example

```
SELECT * FROM events_statements_summary_by_account_by_event_name\G
...
*************************** 521. row ***************************
                    USER: NULL
                    HOST: NULL
              EVENT_NAME: statement/com/Error
              COUNT_STAR: 0
          SUM_TIMER_WAIT: 0
          MIN_TIMER_WAIT: 0
          AVG_TIMER_WAIT: 0
          MAX_TIMER_WAIT: 0
           SUM_LOCK_TIME: 0
              SUM_ERRORS: 0
            SUM_WARNINGS: 0
       SUM_ROWS_AFFECTED: 0
           SUM_ROWS_SENT: 0
       SUM_ROWS_EXAMINED: 0
SUM_CREATED_TMP_DISK_TABLES: 0
     SUM_CREATED_TMP_TABLES: 0
       SUM_SELECT_FULL_JOIN: 0
 SUM_SELECT_FULL_RANGE_JOIN: 0
           SUM_SELECT_RANGE: 0
     SUM_SELECT_RANGE_CHECK: 0
            SUM_SELECT_SCAN: 0
      SUM_SORT_MERGE_PASSES: 0
             SUM_SORT_RANGE: 0
              SUM_SORT_ROWS: 0
              SUM_SORT_SCAN: 0
          SUM_NO_INDEX_USED: 0
     SUM_NO_GOOD_INDEX_USED: 0
*************************** 522. row ***************************
                    USER: NULL
                    HOST: NULL
              EVENT_NAME: statement/com/
              COUNT_STAR: 0
          SUM_TIMER_WAIT: 0
          MIN_TIMER_WAIT: 0
          AVG_TIMER_WAIT: 0
          MAX_TIMER_WAIT: 0
           SUM_LOCK_TIME: 0
              SUM_ERRORS: 0
            SUM_WARNINGS: 0
       SUM_ROWS_AFFECTED: 0
           SUM_ROWS_SENT: 0
       SUM_ROWS_EXAMINED: 0
SUM_CREATED_TMP_DISK_TABLES: 0
     SUM_CREATED_TMP_TABLES: 0
       SUM_SELECT_FULL_JOIN: 0
 SUM_SELECT_FULL_RANGE_JOIN: 0
           SUM_SELECT_RANGE: 0
     SUM_SELECT_RANGE_CHECK: 0
            SUM_SELECT_SCAN: 0
      SUM_SORT_MERGE_PASSES: 0
             SUM_SORT_RANGE: 0
              SUM_SORT_ROWS: 0
              SUM_SORT_SCAN: 0
          SUM_NO_INDEX_USED: 0
     SUM_NO_GOOD_INDEX_USED: 0
```

1.1.1.2.9.2.1.16 Performance Schema events_statements_summary_by_digest Table

The Performance Schema digest is a hashed, normalized form of a statement with the specific data values removed. It allows statistics to be gathered for similar kinds of statements.

The Performance Schema `events_statements_summary_by_digest` table records statement events summarized by schema and digest. It contains the following columns:

Column	Description

SCHEMA_NAME	Database name. Records are summarised together with DIGEST.
DIGEST	Performance Schema digest. Records are summarised together with SCHEMA_NAME.
DIGEST_TEXT	The unhashed form of the digest.
COUNT_STAR	Number of summarized events
SUM_TIMER_WAIT	Total wait time of the summarized events that are timed.
MIN_TIMER_WAIT	Minimum wait time of the summarized events that are timed.
AVG_TIMER_WAIT	Average wait time of the summarized events that are timed.
MAX_TIMER_WAIT	Maximum wait time of the summarized events that are timed.
SUM_LOCK_TIME	Sum of the LOCK_TIME column in the events_statements_current table.
SUM_ERRORS	Sum of the ERRORS column in the events_statements_current table.
SUM_WARNINGS	Sum of the WARNINGS column in the events_statements_current table.
SUM_ROWS_AFFECTED	Sum of the ROWS_AFFECTED column in the events_statements_current table.
SUM_ROWS_SENT	Sum of the ROWS_SENT column in the events_statements_current table.
SUM_ROWS_EXAMINED	Sum of the ROWS_EXAMINED column in the events_statements_current table.
SUM_CREATED_TMP_DISK_TABLES	Sum of the CREATED_TMP_DISK_TABLES column in the events_statements_current table.
SUM_CREATED_TMP_TABLES	Sum of the CREATED_TMP_TABLES column in the events_statements_current table.
SUM_SELECT_FULL_JOIN	Sum of the SELECT_FULL_JOIN column in the events_statements_current table.
SUM_SELECT_FULL_RANGE_JOIN	Sum of the SELECT_FULL_RANGE_JOIN column in the events_statements_current table.
SUM_SELECT_RANGE	Sum of the SELECT_RANGE column in the events_statements_current table.
SUM_SELECT_RANGE_CHECK	Sum of the SELECT_RANGE_CHECK column in the events_statements_current table.
SUM_SELECT_SCAN	Sum of the SELECT_SCAN column in the events_statements_current table.
SUM_SORT_MERGE_PASSES	Sum of the SORT_MERGE_PASSES column in the events_statements_current table.
SUM_SORT_RANGE	Sum of the SORT_RANGE column in the events_statements_current table.
SUM_SORT_ROWS	Sum of the SORT_ROWS column in the events_statements_current table.
SUM_SORT_SCAN	Sum of the SORT_SCAN column in the events_statements_current table.
SUM_NO_INDEX_USED	Sum of the NO_INDEX_USED column in the events_statements_current table.
SUM_NO_GOOD_INDEX_USED	Sum of the NO_GOOD_INDEX_USED column in the events_statements_current table.
FIRST_SEEN	Time at which the digest was first seen.
LAST_SEEN	Time at which the digest was most recently seen.

The *_TIMER_WAIT columns only calculate results for timed events, as non-timed events have a NULL wait time.

The events_statements_summary_by_digest table is limited in size by the performance_schema_digests_size system variable. Once the limit has been reached and the table is full, all entries are aggregated in a row with a NULL digest. The COUNT_STAR value of this NULL row indicates how many digests are recorded in the row and therefore gives an indication of whether performance_schema_digests_size should be increased to provide more accurate statistics.

1.1.1.2.9.2.1.17 Performance Schema events_statements_summary_by_host_by_event_

Table

The Performance Schema events_statements_summary_by_host_by_event_name table contains statement events summarized by host and event name. It contains the following columns:

Column	Description
HOST	Host. Used together with EVENT_NAME for grouping events.
EVENT_NAME	Event name. Used together with HOST for grouping events.
COUNT_STAR	Number of summarized events
SUM_TIMER_WAIT	Total wait time of the summarized events that are timed.
MIN_TIMER_WAIT	Minimum wait time of the summarized events that are timed.
AVG_TIMER_WAIT	Average wait time of the summarized events that are timed.
MAX_TIMER_WAIT	Maximum wait time of the summarized events that are timed.
SUM_LOCK_TIME	Sum of the LOCK_TIME column in the events_statements_currentd table.
SUM_ERRORS	Sum of the ERRORS column in the events_statements_current table.
SUM_WARNINGS	Sum of the WARNINGS column in the events_statements_current table.
SUM_ROWS_AFFECTED	Sum of the ROWS_AFFECTED column in the events_statements_current table.
SUM_ROWS_SENT	Sum of the ROWS_SENT column in the events_statements_current table.
SUM_ROWS_EXAMINED	Sum of the ROWS_EXAMINED column in the events_statements_current table.
SUM_CREATED_TMP_DISK_TABLES	Sum of the CREATED_TMP_DISK_TABLES column in the events_statements_current table.
SUM_CREATED_TMP_TABLES	Sum of the CREATED_TMP_TABLES column in the events_statements_current table.
SUM_SELECT_FULL_JOIN	Sum of the SELECT_FULL_JOIN column in the events_statements_current table.
SUM_SELECT_FULL_RANGE_JOIN	Sum of the SELECT_FULL_RANGE_JOINW column in the events_statements_current table.
SUM_SELECT_RANGE	Sum of the SELECT_RANGE column in the events_statements_current table.
SUM_SELECT_RANGE_CHECK	Sum of the SELECT_RANGE_CHECK column in the events_statements_current table.
SUM_SELECT_SCAN	Sum of the SELECT_SCAN column in the events_statements_current table.
SUM_SORT_MERGE_PASSES	Sum of the SORT_MERGE_PASSES column in the events_statements_current table.
SUM_SORT_RANGE	Sum of the SORT_RANGE column in the events_statements_current table.
SUM_SORT_ROWS	Sum of the SORT_ROWS column in the events_statements_current table.
SUM_SORT_SCAN	Sum of the SORT_SCAN column in the events_statements_current table.
SUM_NO_INDEX_USED	Sum of the NO_INDEX_USED column in the events_statements_current table.
SUM_NO_GOOD_INDEX_USED	Sum of the NO_GOOD_INDEX_USED column in the events_statements_current table.

The *_TIMER_WAIT columns only calculate results for timed events, as non-timed events have a NULL wait time.

Example

```
SELECT * FROM events_statements_summary_by_host_by_event_name\G
...
*************************** 347. row ***************************
                    HOST: NULL
              EVENT_NAME: statement/com/Error
              COUNT_STAR: 0
          SUM_TIMER_WAIT: 0
          MIN_TIMER_WAIT: 0
          AVG_TIMER_WAIT: 0
          MAX_TIMER_WAIT: 0
           SUM_LOCK_TIME: 0
              SUM_ERRORS: 0
            SUM_WARNINGS: 0
       SUM_ROWS_AFFECTED: 0
           SUM_ROWS_SENT: 0
       SUM_ROWS_EXAMINED: 0
SUM_CREATED_TMP_DISK_TABLES: 0
     SUM_CREATED_TMP_TABLES: 0
       SUM_SELECT_FULL_JOIN: 0
 SUM_SELECT_FULL_RANGE_JOIN: 0
           SUM_SELECT_RANGE: 0
     SUM_SELECT_RANGE_CHECK: 0
            SUM_SELECT_SCAN: 0
      SUM_SORT_MERGE_PASSES: 0
             SUM_SORT_RANGE: 0
              SUM_SORT_ROWS: 0
              SUM_SORT_SCAN: 0
          SUM_NO_INDEX_USED: 0
     SUM_NO_GOOD_INDEX_USED: 0
*************************** 348. row ***************************
                    HOST: NULL
              EVENT_NAME: statement/com/
              COUNT_STAR: 0
          SUM_TIMER_WAIT: 0
          MIN_TIMER_WAIT: 0
          AVG_TIMER_WAIT: 0
          MAX_TIMER_WAIT: 0
           SUM_LOCK_TIME: 0
              SUM_ERRORS: 0
            SUM_WARNINGS: 0
       SUM_ROWS_AFFECTED: 0
           SUM_ROWS_SENT: 0
       SUM_ROWS_EXAMINED: 0
SUM_CREATED_TMP_DISK_TABLES: 0
     SUM_CREATED_TMP_TABLES: 0
       SUM_SELECT_FULL_JOIN: 0
 SUM_SELECT_FULL_RANGE_JOIN: 0
           SUM_SELECT_RANGE: 0
     SUM_SELECT_RANGE_CHECK: 0
            SUM_SELECT_SCAN: 0
      SUM_SORT_MERGE_PASSES: 0
             SUM_SORT_RANGE: 0
              SUM_SORT_ROWS: 0
              SUM_SORT_SCAN: 0
          SUM_NO_INDEX_USED: 0
     SUM_NO_GOOD_INDEX_USED: 0
```

1.1.1.2.9.2.1.18 Performance Schema events_statements_summary_by_program Table

MariaDB starting with 10.5.2

The `events_statements_summary_by_program` table, along with many other new Performance Schema tables, was added in MariaDB 10.5.2.

Each row in the the Performance Schema events_statements_summary_by_program table summarizes events for a particular stored program (stored procedure, stored function, trigger or event).

It contains the following fields.

Column	Type	Null	Description
OBJECT_TYPE	enum('EVENT', 'FUNCTION', 'PROCEDURE', 'TABLE', 'TRIGGER')	YES	Object type for which the summary is generated.
OBJECT_SCHEMA	varchar(64)	NO	The schema of the object for which the summary is generated.
OBJECT_NAME	varchar(64)	NO	The name of the object for which the summary is generated.
COUNT_STAR	bigint(20) unsigned	NO	The number of summarized events (from events_statements_current). This value includes all events, whether timed or nontimed.
SUM_TIMER_WAIT	bigint(20) unsigned	NO	The number of summarized events (from events_statements_current). This value includes all events, whether timed or nontimed.
MIN_TIMER_WAIT	bigint(20) unsigned	NO	The minimum wait time of the summarized timed events.
AVG_TIMER_WAIT	bigint(20) unsigned	NO	The average wait time of the summarized timed events.
MAX_TIMER_WAIT	bigint(20) unsigned	NO	The maximum wait time of the summarized timed events.
COUNT_STATEMENTS	bigint(20) unsigned	NO	Total number of nested statements invoked during stored program execution.
SUM_STATEMENTS_WAIT	bigint(20) unsigned	NO	The total wait time of the summarized timed statements. This value is calculated only for timed statements because nontimed statements have a wait time of NULL. The same is true for the other xxx_STATEMENT_WAIT values.
MIN_STATEMENTS_WAIT	bigint(20) unsigned	NO	The minimum wait time of the summarized timed statements.
AVG_STATEMENTS_WAIT	bigint(20) unsigned	NO	The average wait time of the summarized timed statements.
MAX_STATEMENTS_WAIT	bigint(20) unsigned	NO	The maximum wait time of the summarized timed statements.
SUM_LOCK_TIME	bigint(20) unsigned	NO	The total time spent (in picoseconds) waiting for table locks for the summarized statements.
SUM_ERRORS	bigint(20) unsigned	NO	The total number of errors that occurend for the summarized statements.
SUM_WARNINGS	bigint(20) unsigned	NO	The total number of warnings that occurend for the summarized statements.
SUM_ROWS_AFFECTED	bigint(20) unsigned	NO	The total number of affected rows by the summarized statements.
SUM_ROWS_SENT	bigint(20) unsigned	NO	The total number of rows returned by the summarized statements.
SUM_ROWS_EXAMINED	bigint(20) unsigned	NO	The total number of rows examined by the summarized statements.The total number of affected rows by the summarized statements.
SUM_CREATED_TMP_DISK_TABLES	bigint(20) unsigned	NO	The total number of on-disk temporary tables created by the summarized statements.
SUM_CREATED_TMP_TABLES	bigint(20) unsigned	NO	The total number of in-memory temporary tables created by the summarized statements.

SUM_SELECT_FULL_JOIN	bigint(20) unsigned	NO	The total number of full joins executed by the summarized statements.
SUM_SELECT_FULL_RANGE_JOIN	bigint(20) unsigned	NO	The total number of range search joins executed by the summarized statements.
SUM_SELECT_RANGE	bigint(20) unsigned	NO	The total number of joins that used ranges on the first table executed by the summarized statements.
SUM_SELECT_RANGE_CHECK	bigint(20) unsigned	NO	The total number of joins that check for key usage after each row executed by the summarized statements.
SUM_SELECT_SCAN	bigint(20) unsigned	NO	The total number of joins that did a full scan of the first table executed by the summarized statements.
SUM_SORT_MERGE_PASSES	bigint(20) unsigned	NO	The total number of merge passes that the sort algorithm has had to do for the summarized statements.
SUM_SORT_RANGE	bigint(20) unsigned	NO	The total number of sorts that were done using ranges for the summarized statements.
SUM_SORT_ROWS	bigint(20) unsigned	NO	The total number of sorted rows that were sorted by the summarized statements.
SUM_SORT_SCAN	bigint(20) unsigned	NO	The total number of sorts that were done by scanning the table by the summarized statements.
SUM_NO_INDEX_USED	bigint(20) unsigned	NO	The total number of statements that performed a table scan without using an index.
SUM_NO_GOOD_INDEX_USED	bigint(20) unsigned	NO	The total number of statements where no good index was found.

1.1.1.2.9.2.1.19 Performance Schema events_statements_summary_by_thread_by_eve Table

The Performance Schema `events_statements_summary_by_thread_by_event_name` table contains statement events summarized by thread and event name. It contains the following columns:

Column	Description
`THREAD_ID`	Thread associated with the event. Together with `EVENT_NAME` uniquely identifies the row.
`EVENT_NAME`	Event name. Used together with `THREAD_ID` for grouping events.
`COUNT_STAR`	Number of summarized events
`SUM_TIMER_WAIT`	Total wait time of the summarized events that are timed.
`MIN_TIMER_WAIT`	Minimum wait time of the summarized events that are timed.
`AVG_TIMER_WAIT`	Average wait time of the summarized events that are timed.
`MAX_TIMER_WAIT`	Maximum wait time of the summarized events that are timed.
`SUM_LOCK_TIME`	Sum of the `LOCK_TIME` column in the `events_statements_current` table.
`SUM_ERRORS`	Sum of the `ERRORS` column in the `events_statements_current` table.
`SUM_WARNINGS`	Sum of the `WARNINGS` column in the `events_statements_current` table.
`SUM_ROWS_AFFECTED`	Sum of the `ROWS_AFFECTED` column in the `events_statements_current` table.
`SUM_ROWS_SENT`	Sum of the `ROWS_SENT` column in the `events_statements_current` table.
`SUM_ROWS_EXAMINED`	Sum of the `ROWS_EXAMINED` column in the `events_statements_current` table.
`SUM_CREATED_TMP_DISK_TABLES`	Sum of the `CREATED_TMP_DISK_TABLES` column in the `events_statements_current` table.
`SUM_CREATED_TMP_TABLES`	Sum of the `CREATED_TMP_TABLES` column in the `events_statements_current` table.

SUM_SELECT_FULL_JOIN	Sum of the SELECT_FULL_JOIN column in the events_statements_current table.
SUM_SELECT_FULL_RANGE_JOIN	Sum of the SELECT_FULL_RANGE_JOIN column in the events_statements_current table.
SUM_SELECT_RANGE	Sum of the SELECT_RANGE column in the events_statements_current table.
SUM_SELECT_RANGE_CHECK	Sum of the SELECT_RANGE_CHECK column in the events_statements_current table.
SUM_SELECT_SCAN	Sum of the SELECT_SCAN column in the events_statements_current table.
SUM_SORT_MERGE_PASSES	Sum of the SORT_MERGE_PASSES column in the events_statements_current table.
SUM_SORT_RANGE	Sum of the SORT_RANGE column in the events_statements_current table.
SUM_SORT_ROWS	Sum of the SORT_ROWS column in the events_statements_current table.
SUM_SORT_SCAN	Sum of the SORT_SCAN column in the events_statements_current table.
SUM_NO_INDEX_USED	Sum of the NO_INDEX_USED column in the events_statements_current table.
SUM_NO_GOOD_INDEX_USED	Sum of the NO_GOOD_INDEX_USED column in the events_statements_current table.

The *_TIMER_WAIT columns only calculate results for timed events, as non-timed events have a NULL wait time.

Example

```
SELECT * FROM events_statements_summary_by_thread_by_event_name\G
...
*************************** 3653. row ***************************
                  THREAD_ID: 64
                 EVENT_NAME: statement/com/Error
                 COUNT_STAR: 0
             SUM_TIMER_WAIT: 0
             MIN_TIMER_WAIT: 0
             AVG_TIMER_WAIT: 0
             MAX_TIMER_WAIT: 0
              SUM_LOCK_TIME: 0
                 SUM_ERRORS: 0
               SUM_WARNINGS: 0
          SUM_ROWS_AFFECTED: 0
              SUM_ROWS_SENT: 0
          SUM_ROWS_EXAMINED: 0
SUM_CREATED_TMP_DISK_TABLES: 0
     SUM_CREATED_TMP_TABLES: 0
       SUM_SELECT_FULL_JOIN: 0
 SUM_SELECT_FULL_RANGE_JOIN: 0
           SUM_SELECT_RANGE: 0
     SUM_SELECT_RANGE_CHECK: 0
            SUM_SELECT_SCAN: 0
      SUM_SORT_MERGE_PASSES: 0
             SUM_SORT_RANGE: 0
              SUM_SORT_ROWS: 0
              SUM_SORT_SCAN: 0
          SUM_NO_INDEX_USED: 0
     SUM_NO_GOOD_INDEX_USED: 0
*************************** 3654. row ***************************
                  THREAD_ID: 64
                 EVENT_NAME: statement/com/
                 COUNT_STAR: 0
             SUM_TIMER_WAIT: 0
             MIN_TIMER_WAIT: 0
             AVG_TIMER_WAIT: 0
             MAX_TIMER_WAIT: 0
              SUM_LOCK_TIME: 0
                 SUM_ERRORS: 0
               SUM_WARNINGS: 0
          SUM_ROWS_AFFECTED: 0
              SUM_ROWS_SENT: 0
          SUM_ROWS_EXAMINED: 0
SUM_CREATED_TMP_DISK_TABLES: 0
     SUM_CREATED_TMP_TABLES: 0
       SUM_SELECT_FULL_JOIN: 0
 SUM_SELECT_FULL_RANGE_JOIN: 0
           SUM_SELECT_RANGE: 0
     SUM_SELECT_RANGE_CHECK: 0
            SUM_SELECT_SCAN: 0
      SUM_SORT_MERGE_PASSES: 0
             SUM_SORT_RANGE: 0
              SUM_SORT_ROWS: 0
              SUM_SORT_SCAN: 0
          SUM_NO_INDEX_USED: 0
     SUM_NO_GOOD_INDEX_USED: 0
```

1.1.1.2.9.2.1.20 Performance Schema events_statements_summary_by_user_by_event_ Table

The Performance Schema `events_statements_summary_by_user_by_event_name` table contains statement events summarized by user and event name. It contains the following columns:

Column	Description
`USER`	User. Used together with `EVENT_NAME` for grouping events.
`EVENT_NAME`	Event name. Used together with `USER` for grouping events.

COUNT_STAR	Number of summarized events
SUM_TIMER_WAIT	Total wait time of the summarized events that are timed.
MIN_TIMER_WAIT	Minimum wait time of the summarized events that are timed.
AVG_TIMER_WAIT	Average wait time of the summarized events that are timed.
MAX_TIMER_WAIT	Maximum wait time of the summarized events that are timed.
SUM_LOCK_TIME	Sum of the LOCK_TIME column in the events_statements_current table.
SUM_ERRORS	Sum of the ERRORS column in the events_statements_current table.
SUM_WARNINGS	Sum of the WARNINGS column in the events_statements_current table.
SUM_ROWS_AFFECTED	Sum of the ROWS_AFFECTED column in the events_statements_current table.
SUM_ROWS_SENT	Sum of the ROWS_SENT column in the events_statements_current table.
SUM_ROWS_EXAMINED	Sum of the ROWS_EXAMINED column in the events_statements_current table.
SUM_CREATED_TMP_DISK_TABLES	Sum of the CREATED_TMP_DISK_TABLES column in the events_statements_current table.
SUM_CREATED_TMP_TABLES	Sum of the CREATED_TMP_TABLES column in the events_statements_current table.
SUM_SELECT_FULL_JOIN	Sum of the SELECT_FULL_JOIN column in the events_statements_current table.
SUM_SELECT_FULL_RANGE_JOIN	Sum of the SELECT_FULL_RANGE_JOIN column in the events_statements_current table.
SUM_SELECT_RANGE	Sum of the SELECT_RANGE column in the events_statements_current table.
SUM_SELECT_RANGE_CHECK	Sum of the SELECT_RANGE_CHECK column in the events_statements_current table.
SUM_SELECT_SCAN	Sum of the SELECT_SCAN column in the events_statements_current table.
SUM_SORT_MERGE_PASSES	Sum of the SORT_MERGE_PASSES column in the events_statements_current table.
SUM_SORT_RANGE	Sum of the SORT_RANGE column in the events_statements_current table.
SUM_SORT_ROWS	Sum of the SORT_ROWS column in the events_statements_current table.
SUM_SORT_SCAN	Sum of the SORT_SCAN column in the events_statements_current table.
SUM_NO_INDEX_USED	Sum of the NO_INDEX_USED column in the events_statements_current table.
SUM_NO_GOOD_INDEX_USED	Sum of the NO_GOOD_INDEX_USED column in the events_statements_current table.

The *_TIMER_WAIT columns only calculate results for timed events, as non-timed events have a NULL wait time.

Example

```
SELECT * FROM events_statements_summary_by_user_by_event_name\G
...
*************************** 521. row ***************************
                    USER: NULL
              EVENT_NAME: statement/com/Error
              COUNT_STAR: 0
          SUM_TIMER_WAIT: 0
          MIN_TIMER_WAIT: 0
          AVG_TIMER_WAIT: 0
          MAX_TIMER_WAIT: 0
           SUM_LOCK_TIME: 0
              SUM_ERRORS: 0
            SUM_WARNINGS: 0
       SUM_ROWS_AFFECTED: 0
           SUM_ROWS_SENT: 0
       SUM_ROWS_EXAMINED: 0
SUM_CREATED_TMP_DISK_TABLES: 0
     SUM_CREATED_TMP_TABLES: 0
       SUM_SELECT_FULL_JOIN: 0
 SUM_SELECT_FULL_RANGE_JOIN: 0
           SUM_SELECT_RANGE: 0
     SUM_SELECT_RANGE_CHECK: 0
            SUM_SELECT_SCAN: 0
      SUM_SORT_MERGE_PASSES: 0
             SUM_SORT_RANGE: 0
              SUM_SORT_ROWS: 0
              SUM_SORT_SCAN: 0
          SUM_NO_INDEX_USED: 0
     SUM_NO_GOOD_INDEX_USED: 0
*************************** 522. row ***************************
                    USER: NULL
              EVENT_NAME: statement/com/
              COUNT_STAR: 0
          SUM_TIMER_WAIT: 0
          MIN_TIMER_WAIT: 0
          AVG_TIMER_WAIT: 0
          MAX_TIMER_WAIT: 0
           SUM_LOCK_TIME: 0
              SUM_ERRORS: 0
            SUM_WARNINGS: 0
       SUM_ROWS_AFFECTED: 0
           SUM_ROWS_SENT: 0
       SUM_ROWS_EXAMINED: 0
SUM_CREATED_TMP_DISK_TABLES: 0
     SUM_CREATED_TMP_TABLES: 0
       SUM_SELECT_FULL_JOIN: 0
 SUM_SELECT_FULL_RANGE_JOIN: 0
           SUM_SELECT_RANGE: 0
     SUM_SELECT_RANGE_CHECK: 0
            SUM_SELECT_SCAN: 0
      SUM_SORT_MERGE_PASSES: 0
             SUM_SORT_RANGE: 0
              SUM_SORT_ROWS: 0
              SUM_SORT_SCAN: 0
          SUM_NO_INDEX_USED: 0
     SUM_NO_GOOD_INDEX_USED: 0
```

1.1.1.2.9.2.1.21 Performance Schema events_statements_summary_global_by_event_n Table

The Performance Schema `events_statements_summary_global_by_event_name` table contains statement events summarized by event name. It contains the following columns:

Column	Description
`EVENT_NAME`	Event name.
`COUNT_STAR`	Number of summarized events

SUM_TIMER_WAIT	Total wait time of the summarized events that are timed.
MIN_TIMER_WAIT	Minimum wait time of the summarized events that are timed.
AVG_TIMER_WAIT	Average wait time of the summarized events that are timed.
MAX_TIMER_WAIT	Maximum wait time of the summarized events that are timed.
SUM_LOCK_TIME	Sum of the LOCK_TIME column in the events_statements_current table.
SUM_ERRORS	Sum of the ERRORS column in the events_statements_current table.
SUM_WARNINGS	Sum of the WARNINGS column in the events_statements_current table.
SUM_ROWS_AFFECTED	Sum of the ROWS_AFFECTED column in the events_statements_current table.
SUM_ROWS_SENT	Sum of the ROWS_SENT column in the events_statements_current table.
SUM_ROWS_EXAMINED	Sum of the ROWS_EXAMINED column in the events_statements_current table.
SUM_CREATED_TMP_DISK_TABLES	Sum of the CREATED_TMP_DISK_TABLES column in the events_statements_current table.
SUM_CREATED_TMP_TABLES	Sum of the CREATED_TMP_TABLES column in the events_statements_current table.
SUM_SELECT_FULL_JOIN	Sum of the SELECT_FULL_JOIN column in the events_statements_current table.
SUM_SELECT_FULL_RANGE_JOIN	Sum of the SELECT_FULL_RANGE_JOIN column in the events_statements_current table.
SUM_SELECT_RANGE	Sum of the SELECT_RANGE column in the events_statements_current table.
SUM_SELECT_RANGE_CHECK	Sum of the SELECT_RANGE_CHECK column in the events_statements_current table.
SUM_SELECT_SCAN	Sum of the SELECT_SCAN column in the events_statements_current table.
SUM_SORT_MERGE_PASSES	Sum of the SORT_MERGE_PASSES column in the events_statements_current table.
SUM_SORT_RANGE	Sum of the SORT_RANGE column in the events_statements_current table.
SUM_SORT_ROWS	Sum of the SORT_ROWS column in the events_statements_current table.
SUM_SORT_SCAN	Sum of the SORT_SCAN column in the events_statements_current table.
SUM_NO_INDEX_USED	Sum of the NO_INDEX_USED column in the events_statements_current table.
SUM_NO_GOOD_INDEX_USED	Sum of the NO_GOOD_INDEX_USED column in the events_statements_current table.

The *_TIMER_WAIT columns only calculate results for timed events, as non-timed events have a NULL wait time.

Example

```
SELECT * FROM events_statements_summary_global_by_event_name\G
...
*************************** 173. row ***************************
                 EVENT_NAME: statement/com/Error
                 COUNT_STAR: 0
             SUM_TIMER_WAIT: 0
             MIN_TIMER_WAIT: 0
             AVG_TIMER_WAIT: 0
             MAX_TIMER_WAIT: 0
              SUM_LOCK_TIME: 0
                 SUM_ERRORS: 0
               SUM_WARNINGS: 0
          SUM_ROWS_AFFECTED: 0
              SUM_ROWS_SENT: 0
          SUM_ROWS_EXAMINED: 0
SUM_CREATED_TMP_DISK_TABLES: 0
     SUM_CREATED_TMP_TABLES: 0
       SUM_SELECT_FULL_JOIN: 0
 SUM_SELECT_FULL_RANGE_JOIN: 0
           SUM_SELECT_RANGE: 0
     SUM_SELECT_RANGE_CHECK: 0
            SUM_SELECT_SCAN: 0
      SUM_SORT_MERGE_PASSES: 0
             SUM_SORT_RANGE: 0
              SUM_SORT_ROWS: 0
              SUM_SORT_SCAN: 0
          SUM_NO_INDEX_USED: 0
     SUM_NO_GOOD_INDEX_USED: 0
*************************** 174. row ***************************
                 EVENT_NAME: statement/com/
                 COUNT_STAR: 0
             SUM_TIMER_WAIT: 0
             MIN_TIMER_WAIT: 0
             AVG_TIMER_WAIT: 0
             MAX_TIMER_WAIT: 0
              SUM_LOCK_TIME: 0
                 SUM_ERRORS: 0
               SUM_WARNINGS: 0
          SUM_ROWS_AFFECTED: 0
              SUM_ROWS_SENT: 0
          SUM_ROWS_EXAMINED: 0
SUM_CREATED_TMP_DISK_TABLES: 0
     SUM_CREATED_TMP_TABLES: 0
       SUM_SELECT_FULL_JOIN: 0
 SUM_SELECT_FULL_RANGE_JOIN: 0
           SUM_SELECT_RANGE: 0
     SUM_SELECT_RANGE_CHECK: 0
            SUM_SELECT_SCAN: 0
      SUM_SORT_MERGE_PASSES: 0
             SUM_SORT_RANGE: 0
              SUM_SORT_ROWS: 0
              SUM_SORT_SCAN: 0
          SUM_NO_INDEX_USED: 0
     SUM_NO_GOOD_INDEX_USED: 0
```

1.1.1.2.9.2.1.22 Performance Schema events_transactions_current Table

MariaDB starting with 10.5.2

The events_transactions_current table was introduced in MariaDB 10.5.2.

The `events_transactions_current` table contains current transaction events for each thread.

The table size cannot be figured, and always stores one row for each thread, showing the current status of the thread's most recent monitored transaction event.

The table contains the following columns:

Column	Type	Description

THREAD_ID	bigint(20) unsigned	The thread associated with the event.
EVENT_ID	bigint(20) unsigned	The event id associated with the event.
END_EVENT_ID	bigint(20) unsigned	This column is set to NULL when the event starts and updated to the thread current event number when the event ends.
EVENT_NAME	varchar(128)	The name of the instrument from which the event was collected. This is a NAME value from the setup_instruments table.
STATE	enum('ACTIVE', 'COMMITTED', 'ROLLED BACK')	The current transaction state. The value is ACTIVE (after START TRANSACTION or BEGIN), COMMITTED (after COMMIT), or ROLLED BACK (after ROLLBACK).
TRX_ID	bigint(20) unsigned	Unused.
GTID	varchar(64)	Transaction GTID, using the format DOMAIN-SERVER_ID-SEQUENCE_NO.
XID_FORMAT_ID	int(11)	XA transaction format ID for GTRID and BQUAL values.
XID_GTRID	varchar(130)	XA global transaction ID.
XID_BQUAL	varchar(130)	XA transaction branch qualifier.
XA_STATE	varchar(64)	The state of the XA transaction. The value is ACTIVE (after XA START), IDLE (after XA END), PREPARED (after XA PREPARE), ROLLED BACK (after XA ROLLBACK), or COMMITTED (after XA COMMIT).
SOURCE	varchar(64)	The name of the source file containing the instrumented code that produced the event and the line number in the file at which the instrumentation occurs.
TIMER_START	bigint(20) unsigned	The unit is picoseconds. When event timing started. NULL if event has no timing information.
TIMER_END	bigint(20) unsigned	The unit is picoseconds. When event timing ended. NULL if event has no timing information.
TIMER_WAIT	bigint(20) unsigned	The unit is picoseconds. Event duration. NULL if event has not timing information.
ACCESS_MODE	enum('READ ONLY', 'READ WRITE')	Transaction access mode.
ISOLATION_LEVEL	varchar(64)	Transaction isolation level. One of: REPEATABLE READ, READ COMMITTED, READ UNCOMMITTED, or SERIALIZABLE.
AUTOCOMMIT	enum('YES','NO')	Whether autcommit mode was enabled when the transaction started.
NUMBER_OF_SAVEPOINTS	bigint(20) unsigned	The number of SAVEPOINT statements issued during the transaction.
NUMBER_OF_ROLLBACK_TO_SAVEPOINT	bigint(20) unsigned	The number of ROLLBACK_TO_SAVEPOINT statements issued during the transaction.
NUMBER_OF_RELEASE_SAVEPOINT	bigint(20) unsigned	The number of RELEASE_SAVEPOINT statements issued during the transaction.
OBJECT_INSTANCE_BEGIN	bigint(20) unsigned	Unused.
NESTING_EVENT_ID	bigint(20) unsigned	The EVENT_ID value of the event within which this event is nested.
NESTING_EVENT_TYPE	enum('TRANSACTION', 'STATEMENT', 'STAGE', 'WAIT')	The nesting event type.

1.1.1.2.9.2.1.23 Performance Schema events_transactions_history Table

MariaDB starting with 10.5.2

The events_transactions_history table was introduced in MariaDB 10.5.2.

The `events_transactions_history` table contains the most recent completed transaction events for each thread.

The number of records stored per thread in the table is determined by the performance_schema_events_transactions_history_size system variable, which is autosized on startup.

If adding a completed transaction event would cause the table to exceed this limit, the oldest thread row is discarded.

All of a thread's rows are discarded when the thread ends.

The table contains the following columns:

Column	Type	Description
THREAD_ID	bigint(20) unsigned	The thread associated with the event.
EVENT_ID	bigint(20) unsigned	The event id associated with the event.
END_EVENT_ID	bigint(20) unsigned	This column is set to NULL when the event starts and updated to the thread current event number when the event ends.
EVENT_NAME	varchar(128)	The name of the instrument from which the event was collected. This is a NAME value from the setup_instruments table.
STATE	enum('ACTIVE', 'COMMITTED',' ROLLED BACK')	The current transaction state. The value is ACTIVE (after START TRANSACTION or BEGIN), COMMITTED (after COMMIT), or ROLLED BACK (after ROLLBACK).
TRX_ID	bigint(20) unsigned	Unused.
GTID	varchar(64)	Transaction GTID, using the format DOMAIN-SERVER_ID-SEQUENCE_NO.
XID_FORMAT_ID	int(11)	XA transaction format ID for GTRID and BQUAL values.
XID_GTRID	varchar(130)	XA global transaction ID.
XID_BQUAL	varchar(130)	XA transaction branch qualifier.
XA_STATE	varchar(64)	The state of the XA transaction. The value is ACTIVE (after XA START), IDLE (after XA END), PREPARED (after XA PREPARE), ROLLED BACK (after XA ROLLBACK), or COMMITTED (after XA COMMIT).
SOURCE	varchar(64)	The name of the source file containing the instrumented code that produced the event and the line number in the file at which the instrumentation occurs.
TIMER_START	bigint(20) unsigned	The unit is picoseconds. When event timing started. NULL if event has no timing information.
TIMER_END	bigint(20) unsigned	The unit is picoseconds. When event timing ended. NULL if event has no timing information.
TIMER_WAIT	bigint(20) unsigned	The unit is picoseconds. Event duration. NULL if event has not timing information.
ACCESS_MODE	enum('READ ONLY', 'READ WRITE')	Transaction access mode.

ISOLATION_LEVEL	varchar(64)	Transaction isolation level. One of: REPEATABLE READ, READ COMMITTED, READ UNCOMMITTED, or SERIALIZABLE.
AUTOCOMMIT	enum('YES', 'NO')	NO
NUMBER_OF_SAVEPOINTS	bigint(20) unsigned	The number of SAVEPOINT statements issued during the transaction.
NUMBER_OF_ROLLBACK_TO_SAVEPOINT	bigint(20) unsigned	The number of ROLLBACK_TO_SAVEPOINT statements issued during the transaction.
NUMBER_OF_RELEASE_SAVEPOINT	bigint(20) unsigned	The number of RELEASE_SAVEPOINT statements issued during the transaction.
OBJECT_INSTANCE_BEGIN	bigint(20) unsigned	Unused.
NESTING_EVENT_ID	bigint(20) unsigned	The EVENT_ID value of the event within which this event is nested.
NESTING_EVENT_TYPE	enum('TRANSACTION','STATEMENT', 'STAGE', 'WAIT')	The nesting event type.

1.1.1.2.9.2.1.24 Performance Schema events_transactions_history_long Table

MariaDB starting with 10.5.2

The events_transactions_history_long table was introduced in MariaDB 10.5.2.

The `events_transactions_history_long` table contains the most recent completed transaction events that have ended globally, across all threads.

The number of records stored in the table is determined by the performance_schema_events_transactions_history_long_size system variable, which is autosized on startup.

If adding a completed transaction would cause the table to exceed this limit, the oldest row, regardless of thread, is discarded.

The table contains the following columns:

Column	Type	Description
THREAD_ID	bigint(20) unsigned	The thread associated with the event.
EVENT_ID	bigint(20) unsigned	The event id associated with the event.
END_EVENT_ID	bigint(20) unsigned	This column is set to NULL when the event starts and updated to the thread current event number when the event ends.
EVENT_NAME	varchar(128)	The name of the instrument from which the event was collected. This is a NAME value from the setup_instruments table.
STATE	enum('ACTIVE', 'COMMITTED','ROLLED BACK')	The current transaction state. The value is ACTIVE (after START TRANSACTION or BEGIN), COMMITTED (after COMMIT), or ROLLED BACK (after ROLLBACK).
TRX_ID	bigint(20) unsigned	Unused.
GTID	varchar(64)	Transaction GTID, using the format DOMAIN-SERVER_ID-SEQUENCE_NO.
XID_FORMAT_ID	int(11)	XA transaction format ID for GTRID and BQUAL values.
XID_GTRID	varchar(130)	XA global transaction ID.
XID_BQUAL	varchar(130)	XA transaction branch qualifier.

XA_STATE	varchar(64)	The state of the XA transaction. The value is ACTIVE (after XA START), IDLE (after XA END), PREPARED (after XA PREPARE), ROLLED BACK (after XA ROLLBACK), or COMMITTED (after XA COMMIT).
SOURCE	varchar(64)	The name of the source file containing the instrumented code that produced the event and the line number in the file at which the instrumentation occurs.
TIMER_START	bigint(20) unsigned	The unit is picoseconds. When event timing started. NULL if event has no timing information.
TIMER_END	bigint(20) unsigned	The unit is picoseconds. When event timing ended. NULL if event has no timing information.
TIMER_WAIT	bigint(20) unsigned	The unit is picoseconds. Event duration. NULL if event has not timing information.
ACCESS_MODE	enum('READ ONLY', 'READ WRITE')	Transaction access mode.
ISOLATION_LEVEL	varchar(64)	Transaction isolation level. One of: REPEATABLE READ, READ COMMITTED, READ UNCOMMITTED, or SERIALIZABLE.
AUTOCOMMIT	enum('YES', 'NO')	NO
NUMBER_OF_SAVEPOINTS	bigint(20) unsigned	The number of SAVEPOINT statements issued during the transaction.
NUMBER_OF_ROLLBACK_TO_SAVEPOINT	bigint(20) unsigned	The number of ROLLBACK_TO_SAVEPOINT statements issued during the transaction.
NUMBER_OF_RELEASE_SAVEPOINT	bigint(20) unsigned	The number of RELEASE_SAVEPOINT statements issued during the transaction.
OBJECT_INSTANCE_BEGIN	bigint(20) unsigned	Unused.
NESTING_EVENT_ID	bigint(20) unsigned	The EVENT_ID value of the event within which this event is nested.
NESTING_EVENT_TYPE	enum('TRANSACTION','STATEMENT', 'STAGE', 'WAIT')	The nesting event type.

1.1.1.2.9.2.1.25 Performance Schema events_transactions_summary_by_account_by_e Table

MariaDB starting with 10.5.2

The events_transactions_summary_by_account_by_event_name table was introduced in MariaDB 10.5.2.

The `events_transactions_summary_by_account_by_event_name` table contains information on transaction events aggregated by account and event name.

The table contains the following columns:

Column	Type	Description
USER	char(32)	User for which summary is generated.
HOST	char(60)	Host for which summary is generated.
EVENT_NAME	varchar(128)	Event name for which summary is generated.
COUNT_STAR	bigint(20) unsigned	The number of summarized events. This value includes all events, whether timed or nontimed.

SUM_TIMER_WAIT	bigint(20) unsigned	The total wait time of the summarized timed events. This value is calculated only for timed events because nontimed events have a wait time of NULL. The same is true for the other xxx_TIMER_WAIT values.
MIN_TIMER_WAIT	bigint(20) unsigned	The minimum wait time of the summarized timed events.
AVG_TIMER_WAIT	bigint(20) unsigned	The average wait time of the summarized timed events.
MAX_TIMER_WAIT	bigint(20) unsigned	The maximum wait time of the summarized timed events.
COUNT_READ_WRITE	bigint(20) unsigned	The total number of only READ/WRITE transaction events.
SUM_TIMER_READ_WRITE	bigint(20) unsigned	The total wait time of only READ/WRITE transaction events.
MIN_TIMER_READ_WRITE	bigint(20) unsigned	The minimum wait time of only READ/WRITE transaction events.
AVG_TIMER_READ_WRITE	bigint(20) unsigned	The average wait time of only READ/WRITE transaction events.
MAX_TIMER_READ_WRITE	bigint(20) unsigned	The maximum wait time of only READ/WRITE transaction events.
COUNT_READ_ONLY	bigint(20) unsigned	The total number of only READ ONLY transaction events.
SUM_TIMER_READ_ONLY	bigint(20) unsigned	The total wait time of only READ ONLY transaction events.
MIN_TIMER_READ_ONLY	bigint(20) unsigned	The minimum wait time of only READ ONLY transaction events.
AVG_TIMER_READ_ONLY	bigint(20) unsigned	The average wait time of only READ ONLY transaction events.
MAX_TIMER_READ_ONLY	bigint(20) unsigned	The maximum wait time of only READ ONLY transaction events.

1.1.1.2.9.2.1.26 Performance Schema events_transactions_summary_by_host_by_even Table

MariaDB starting with 10.5.2

The events_transactions_summary_by_host_by_event_name table was introduced in MariaDB 10.5.2.

The `events_transactions_summary_by_host_by_event_name` table contains information on transaction events aggregated by host and event name.

The table contains the following columns:

Column	Type	Description
HOST	char(60)	Host for which summary is generated.
EVENT_NAME	varchar(128)	Event name for which summary is generated.
COUNT_STAR	bigint(20) unsigned	The number of summarized events. This value includes all events, whether timed or nontimed.
SUM_TIMER_WAIT	bigint(20) unsigned	The total wait time of the summarized timed events. This value is calculated only for timed events because nontimed events have a wait time of NULL. The same is true for the other xxx_TIMER_WAIT values.
MIN_TIMER_WAIT	bigint(20) unsigned	The minimum wait time of the summarized timed events.

AVG_TIMER_WAIT	bigint(20) unsigned	The average wait time of the summarized timed events.
MAX_TIMER_WAIT	bigint(20) unsigned	The maximum wait time of the summarized timed events.
COUNT_READ_WRITE	bigint(20) unsigned	The total number of only READ/WRITE transaction events.
SUM_TIMER_READ_WRITE	bigint(20) unsigned	The total wait time of only READ/WRITE transaction events.
MIN_TIMER_READ_WRITE	bigint(20) unsigned	The minimum wait time of only READ/WRITE transaction events.
AVG_TIMER_READ_WRITE	bigint(20) unsigned	The average wait time of only READ/WRITE transaction events.
MAX_TIMER_READ_WRITE	bigint(20) unsigned	The maximum wait time of only READ/WRITE transaction events.
COUNT_READ_ONLY	bigint(20) unsigned	The total number of only READ ONLY transaction events.
SUM_TIMER_READ_ONLY	bigint(20) unsigned	The total wait time of only READ ONLY transaction events.
MIN_TIMER_READ_ONLY	bigint(20) unsigned	The minimum wait time of only READ ONLY transaction events.
AVG_TIMER_READ_ONLY	bigint(20) unsigned	The average wait time of only READ ONLY transaction events.
MAX_TIMER_READ_ONLY	bigint(20) unsigned	The maximum wait time of only READ ONLY transaction events.

1.1.1.2.9.2.1.27 Performance Schema events_transactions_summary_by_thread_by_ev Table

MariaDB starting with 10.5.2

The events_transactions_summary_by_thread_by_event_name table was introduced in MariaDB 10.5.2.

The `events_transactions_summary_by_thread_by_event_name` table contains information on transaction events aggregated by thread and event name.

The table contains the following columns:

```
+----------------------+---------------------+------+-----+---------+-------+
| Field                | Type                | Null | Key | Default | Extra |
+----------------------+---------------------+------+-----+---------+-------+
| THREAD_ID            | bigint(20) unsigned | NO   |     | NULL    |       |
| EVENT_NAME           | varchar(128)        | NO   |     | NULL    |       |
| COUNT_STAR           | bigint(20) unsigned | NO   |     | NULL    |       |
| SUM_TIMER_WAIT       | bigint(20) unsigned | NO   |     | NULL    |       |
| MIN_TIMER_WAIT       | bigint(20) unsigned | NO   |     | NULL    |       |
| AVG_TIMER_WAIT       | bigint(20) unsigned | NO   |     | NULL    |       |
| MAX_TIMER_WAIT       | bigint(20) unsigned | NO   |     | NULL    |       |
| COUNT_READ_WRITE     | bigint(20) unsigned | NO   |     | NULL    |       |
| SUM_TIMER_READ_WRITE | bigint(20) unsigned | NO   |     | NULL    |       |
| MIN_TIMER_READ_WRITE | bigint(20) unsigned | NO   |     | NULL    |       |
| AVG_TIMER_READ_WRITE | bigint(20) unsigned | NO   |     | NULL    |       |
| MAX_TIMER_READ_WRITE | bigint(20) unsigned | NO   |     | NULL    |       |
| COUNT_READ_ONLY      | bigint(20) unsigned | NO   |     | NULL    |       |
| SUM_TIMER_READ_ONLY  | bigint(20) unsigned | NO   |     | NULL    |       |
| MIN_TIMER_READ_ONLY  | bigint(20) unsigned | NO   |     | NULL    |       |
| AVG_TIMER_READ_ONLY  | bigint(20) unsigned | NO   |     | NULL    |       |
| MAX_TIMER_READ_ONLY  | bigint(20) unsigned | NO   |     | NULL    |       |
+----------------------+---------------------+------+-----+---------+-------+
```

1.1.1.2.9.2.1.28 Performance Schema events_transactions_summary_by_user_by_even Table

MariaDB starting with 10.5.2

The events_transactions_summary_by_user_by_event_name table was introduced in MariaDB 10.5.2.

The `events_transactions_summary_by_user_by_event_name` table contains information on transaction events aggregated by user and event name.

The table contains the following columns:

```
+----------------------+---------------------+------+-----+---------+-------+
| Field                | Type                | Null | Key | Default | Extra |
+----------------------+---------------------+------+-----+---------+-------+
| USER                 | char(32)            | YES  |     | NULL    |       |
| EVENT_NAME           | varchar(128)        | NO   |     | NULL    |       |
| COUNT_STAR           | bigint(20) unsigned | NO   |     | NULL    |       |
| SUM_TIMER_WAIT       | bigint(20) unsigned | NO   |     | NULL    |       |
| MIN_TIMER_WAIT       | bigint(20) unsigned | NO   |     | NULL    |       |
| AVG_TIMER_WAIT       | bigint(20) unsigned | NO   |     | NULL    |       |
| MAX_TIMER_WAIT       | bigint(20) unsigned | NO   |     | NULL    |       |
| COUNT_READ_WRITE     | bigint(20) unsigned | NO   |     | NULL    |       |
| SUM_TIMER_READ_WRITE | bigint(20) unsigned | NO   |     | NULL    |       |
| MIN_TIMER_READ_WRITE | bigint(20) unsigned | NO   |     | NULL    |       |
| AVG_TIMER_READ_WRITE | bigint(20) unsigned | NO   |     | NULL    |       |
| MAX_TIMER_READ_WRITE | bigint(20) unsigned | NO   |     | NULL    |       |
| COUNT_READ_ONLY      | bigint(20) unsigned | NO   |     | NULL    |       |
| SUM_TIMER_READ_ONLY  | bigint(20) unsigned | NO   |     | NULL    |       |
| MIN_TIMER_READ_ONLY  | bigint(20) unsigned | NO   |     | NULL    |       |
| AVG_TIMER_READ_ONLY  | bigint(20) unsigned | NO   |     | NULL    |       |
| MAX_TIMER_READ_ONLY  | bigint(20) unsigned | NO   |     | NULL    |       |
+----------------------+---------------------+------+-----+---------+-------+
```

1.1.1.2.9.2.1.29 Performance Schema events_transactions_summary_global_by_event_ Table

MariaDB starting with 10.5.2

The events_transactions_summary_global_by_event_name table was introduced in MariaDB 10.5.2.

The `events_transactions_summary_global_by_event_name` table contains information on transaction events aggregated by event name.

The table contains the following columns:

```
+----------------------+---------------------+------+-----+---------+-------+
| Field                | Type                | Null | Key | Default | Extra |
+----------------------+---------------------+------+-----+---------+-------+
| EVENT_NAME           | varchar(128)        | NO   |     | NULL    |       |
| COUNT_STAR           | bigint(20) unsigned | NO   |     | NULL    |       |
| SUM_TIMER_WAIT       | bigint(20) unsigned | NO   |     | NULL    |       |
| MIN_TIMER_WAIT       | bigint(20) unsigned | NO   |     | NULL    |       |
| AVG_TIMER_WAIT       | bigint(20) unsigned | NO   |     | NULL    |       |
| MAX_TIMER_WAIT       | bigint(20) unsigned | NO   |     | NULL    |       |
| COUNT_READ_WRITE     | bigint(20) unsigned | NO   |     | NULL    |       |
| SUM_TIMER_READ_WRITE | bigint(20) unsigned | NO   |     | NULL    |       |
| MIN_TIMER_READ_WRITE | bigint(20) unsigned | NO   |     | NULL    |       |
| AVG_TIMER_READ_WRITE | bigint(20) unsigned | NO   |     | NULL    |       |
| MAX_TIMER_READ_WRITE | bigint(20) unsigned | NO   |     | NULL    |       |
| COUNT_READ_ONLY      | bigint(20) unsigned | NO   |     | NULL    |       |
| SUM_TIMER_READ_ONLY  | bigint(20) unsigned | NO   |     | NULL    |       |
| MIN_TIMER_READ_ONLY  | bigint(20) unsigned | NO   |     | NULL    |       |
| AVG_TIMER_READ_ONLY  | bigint(20) unsigned | NO   |     | NULL    |       |
| MAX_TIMER_READ_ONLY  | bigint(20) unsigned | NO   |     | NULL    |       |
+----------------------+---------------------+------+-----+---------+-------+
```

1.1.1.2.9.2.1.30 Performance Schema events_waits_current Table

The `events_waits_current` table contains the status of a thread's most recently monitored wait event, listing one event per thread.

The table contains the following columns:

Column	Description
`THREAD_ID`	Thread associated with the event. Together with `EVENT_ID` uniquely identifies the row.
`EVENT_ID`	Thread's current event number at the start of the event. Together with `THREAD_ID` uniquely identifies the row.
`END_EVENT_ID`	`NULL` when the event starts, set to the thread's current event number at the end of the event.
`EVENT_NAME`	Event instrument name and a `NAME` from the `setup_instruments` table
`SOURCE`	Name and line number of the source file containing the instrumented code that produced the event.
`TIMER_START`	Value in picoseconds when the event timing started or `NULL` if timing is not collected.
`TIMER_END`	Value in picoseconds when the event timing ended, or `NULL` if the event has not ended or timing is not collected.
`TIMER_WAIT`	Value in picoseconds of the event's duration or `NULL` if the event has not ended or timing is not collected.
`SPINS`	Number of spin rounds for a mutex, or `NULL` if spin rounds are not used, or spinning is not instrumented.
`OBJECT_SCHEMA`	Name of the schema that contains the table for table I/O objects, otherwise `NULL` for file I/O and synchronization objects.
`OBJECT_NAME`	File name for file I/O objects, table name for table I/O objects, the socket's `IP:PORT` value for a socket object or `NULL` for a synchronization object.
`INDEX NAME`	Name of the index, `PRIMARY` for the primary key, or `NULL` for no index used.
`OBJECT_TYPE`	FILE for a file object, `TABLE` or `TEMPORARY TABLE` for a table object, or `NULL` for a synchronization object.
`OBJECT_INSTANCE_BEGIN`	Address in memory of the object.
`NESTING_EVENT_ID`	`EVENT_ID` of event within which this event nests.
`NESTING_EVENT_TYPE`	Nesting event type. Either `statement`, `stage` or `wait`.

OPERATION	Operation type, for example read, write or lock
NUMBER_OF_BYTES	Number of bytes that the operation read or wrote, or NULL for table I/O waits.
FLAGS	Reserved for use in the future.

It is possible to empty this table with a TRUNCATE TABLE statement.

The related tables, events_waits_history and events_waits_history_long derive their values from the current events.

1.1.1.2.9.2.1.31 Performance Schema events_waits_history Table

The events_waits_history table by default contains the ten most recent completed wait events per thread. This number can be adjusted by setting the performance_schema_events_waits_history_size system variable when the server starts up.

The table structure is identical to the events_waits_current table structure, and contains the following columns:

Column	Description
THREAD_ID	Thread associated with the event. Together with EVENT_ID uniquely identifies the row.
EVENT_ID	Thread's current event number at the start of the event. Together with THREAD_ID uniquely identifies the row.
END_EVENT_ID	NULL when the event starts, set to the thread's current event number at the end of the event.
EVENT_NAME	Event instrument name and a NAME from the setup_instruments table
SOURCE	Name and line number of the source file containing the instrumented code that produced the event.
TIMER_START	Value in picoseconds when the event timing started or NULL if timing is not collected.
TIMER_END	Value in picoseconds when the event timing ended, or NULL if timing is not collected.
TIMER_WAIT	Value in picoseconds of the event's duration or NULL if timing is not collected.
SPINS	Number of spin rounds for a mutex, or NULL if spin rounds are not used, or spinning is not instrumented.
OBJECT_SCHEMA	Name of the schema that contains the table for table I/O objects, otherwise NULL for file I/O and synchronization objects.
OBJECT_NAME	File name for file I/O objects, table name for table I/O objects, the socket's IP:PORT value for a socket object or NULL for a synchronization object.
INDEX NAME	Name of the index, PRIMARY for the primary key, or NULL for no index used.
OBJECT_TYPE	FILE for a file object, TABLE or TEMPORARY TABLE for a table object, or NULL for a synchronization object.
OBJECT_INSTANCE_BEGIN	Address in memory of the object.
NESTING_EVENT_ID	EVENT_ID of event within which this event nests.
NESTING_EVENT_TYPE	Nesting event type. Either statement, stage or wait.
OPERATION	Operation type, for example read, write or lock
NUMBER_OF_BYTES	Number of bytes that the operation read or wrote, or NULL for table I/O waits.
FLAGS	Reserved for use in the future.

It is possible to empty this table with a TRUNCATE TABLE statement.

events_waits_current and events_waits_history_long are related tables.

1.1.1.2.9.2.1.32 Performance Schema events_waits_history_long Table

The events_waits_history_long table by default contains the ten thousand most recent completed wait events. This

number can be adjusted by setting the performance_schema_events_waits_history_long_size system variable when the server starts up.

The table structure is identical to the events_waits_current table structure, and contains the following columns:

Column	Description
THREAD_ID	Thread associated with the event. Together with EVENT_ID uniquely identifies the row.
EVENT_ID	Thread's current event number at the start of the event. Together with THREAD_ID uniquely identifies the row.
END_EVENT_ID	NULL when the event starts, set to the thread's current event number at the end of the event.
EVENT_NAME	Event instrument name and a NAME from the setup_instruments table
SOURCE	Name and line number of the source file containing the instrumented code that produced the event.
TIMER_START	Value in picoseconds when the event timing started or NULL if timing is not collected.
TIMER_END	Value in picoseconds when the event timing ended, or NULL if timing is not collected.
TIMER_WAIT	Value in picoseconds of the event's duration or NULL if timing is not collected.
SPINS	Number of spin rounds for a mutex, or NULL if spin rounds are not used, or spinning is not instrumented.
OBJECT_SCHEMA	Name of the schema that contains the table for table I/O objects, otherwise NULL for file I/O and synchronization objects.
OBJECT_NAME	File name for file I/O objects, table name for table I/O objects, the socket's IP:PORT value for a socket object or NULL for a synchronization object.
INDEX_NAME	Name of the index, PRIMARY for the primary key, or NULL for no index used.
OBJECT_TYPE	FILE for a file object, TABLE or TEMPORARY TABLE for a table object, or NULL for a synchronization object.
OBJECT_INSTANCE_BEGIN	Address in memory of the object.
NESTING_EVENT_ID	EVENT_ID of event within which this event nests.
NESTING_EVENT_TYPE	Nesting event type. Either statement, stage or wait.
OPERATION	Operation type, for example read, write or lock
NUMBER_OF_BYTES	Number of bytes that the operation read or wrote, or NULL for table I/O waits.
FLAGS	Reserved for use in the future.

It is possible to empty this table with a TRUNCATE TABLE statement.

events_waits_current and events_waits_history are related tables.

1.1.1.2.9.2.1.33 Performance Schema events_waits_summary_by_account_by_event_n Table

The Performance Schema events_waits_summary_by_account_by_event_name table contains wait events summarized by account and event name. It contains the following columns:

Column	Description
USER	User. Used together with HOST and EVENT_NAME for grouping events.
HOST	Host. Used together with USER and EVENT_NAME for grouping events.
EVENT_NAME	Event name. Used together with USER and HOST for grouping events.
COUNT_STAR	Number of summarized events
SUM_TIMER_WAIT	Total wait time of the summarized events that are timed.

MIN_TIMER_WAIT	Minimum wait time of the summarized events that are timed.
AVG_TIMER_WAIT	Average wait time of the summarized events that are timed.
MAX_TIMER_WAIT	Maximum wait time of the summarized events that are timed.

The *_TIMER_WAIT columns only calculate results for timed events, as non-timed events have a NULL wait time.

Example

```
SELECT * FROM events_waits_summary_by_account_by_event_name\G
...
*************************** 915. row ***************************
          USER: NULL
          HOST: NULL
    EVENT_NAME: wait/io/socket/sql/server_tcpip_socket
    COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
*************************** 916. row ***************************
          USER: NULL
          HOST: NULL
    EVENT_NAME: wait/io/socket/sql/server_unix_socket
    COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
*************************** 917. row ***************************
          USER: NULL
          HOST: NULL
    EVENT_NAME: wait/io/socket/sql/client_connection
    COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
*************************** 918. row ***************************
          USER: NULL
          HOST: NULL
    EVENT_NAME: idle
    COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
```

1.1.1.2.9.2.1.34 Performance Schema events_waits_summary_by_host_by_event_name Table

The Performance Schema events_waits_summary_by_host_by_event_name table contains wait events summarized by host and event name. It contains the following columns:

Column	Description
HOST	Host. Used together with EVENT_NAME for grouping events.
EVENT_NAME	Event name. Used together with USER and HOST for grouping events.
COUNT_STAR	Number of summarized events
SUM_TIMER_WAIT	Total wait time of the summarized events that are timed.
MIN_TIMER_WAIT	Minimum wait time of the summarized events that are timed.

AVG_TIMER_WAIT	Average wait time of the summarized events that are timed.
MAX_TIMER_WAIT	Maximum wait time of the summarized events that are timed.

The *_TIMER_WAIT columns only calculate results for timed events, as non-timed events have a NULL wait time.

Example

```
SELECT * FROM events_waits_summary_by_host_by_event_name\G
...
*************************** 610. row ***************************
          HOST: NULL
    EVENT_NAME: wait/io/socket/sql/server_unix_socket
    COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
*************************** 611. row ***************************
          HOST: NULL
    EVENT_NAME: wait/io/socket/sql/client_connection
    COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
*************************** 612. row ***************************
          HOST: NULL
    EVENT_NAME: idle
    COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
```

1.1.1.2.9.2.1.35 Performance Schema events_waits_summary_by_instance Table

The Performance Schema events_waits_summary_by_instance table contains wait events summarized by instance. It contains the following columns:

Column	Description
EVENT_NAME	Event name. Used together with OBJECT_INSTANCE_BEGIN for grouping events.
OBJECT_INSTANCE_BEGIN	If an instrument creates multiple instances, each instance has a unique OBJECT_INSTANCE_BEGIN value to allow for grouping by instance.
COUNT_STAR	Number of summarized events
SUM_TIMER_WAIT	Total wait time of the summarized events that are timed.
MIN_TIMER_WAIT	Minimum wait time of the summarized events that are timed.
AVG_TIMER_WAIT	Average wait time of the summarized events that are timed.
MAX_TIMER_WAIT	Maximum wait time of the summarized events that are timed.

The *_TIMER_WAIT columns only calculate results for timed events, as non-timed events have a NULL wait time.

Example

```
SELECT * FROM events_waits_summary_by_instance\G
...
*************************** 202. row ***************************
           EVENT_NAME: wait/io/file/sql/binlog
OBJECT_INSTANCE_BEGIN: 140578961969856
           COUNT_STAR: 6
       SUM_TIMER_WAIT: 90478331960
       MIN_TIMER_WAIT: 263344
       AVG_TIMER_WAIT: 15079721848
       MAX_TIMER_WAIT: 67760576376
*************************** 203. row ***************************
           EVENT_NAME: wait/io/file/sql/dbopt
OBJECT_INSTANCE_BEGIN: 140578961970560
           COUNT_STAR: 6
       SUM_TIMER_WAIT: 39891428472
       MIN_TIMER_WAIT: 387168
       AVG_TIMER_WAIT: 6648571412
       MAX_TIMER_WAIT: 24503293304
*************************** 204. row ***************************
           EVENT_NAME: wait/io/file/sql/dbopt
OBJECT_INSTANCE_BEGIN: 140578961971264
           COUNT_STAR: 6
       SUM_TIMER_WAIT: 39902495024
       MIN_TIMER_WAIT: 177888
       AVG_TIMER_WAIT: 6650415692
       MAX_TIMER_WAIT: 21026400404
```

1.1.1.2.9.2.1.36 Performance Schema events_waits_summary_by_thread_by_event_nar Table

The Performance Schema `events_waits_summary_by_thread_by_event_name` table contains wait events summarized by thread and event name. It contains the following columns:

Column	Description
`THREAD_ID`	Thread associated with the event. Together with `EVENT_NAME` uniquely identifies the row.
`EVENT_NAME`	Event name. Used together with `THREAD_ID` for grouping events.
`COUNT_STAR`	Number of summarized events
`SUM_TIMER_WAIT`	Total wait time of the summarized events that are timed.
`MIN_TIMER_WAIT`	Minimum wait time of the summarized events that are timed.
`AVG_TIMER_WAIT`	Average wait time of the summarized events that are timed.
`MAX_TIMER_WAIT`	Maximum wait time of the summarized events that are timed.

The `*_TIMER_WAIT` columns only calculate results for timed events, as non-timed events have a `NULL` wait time.

Example

```
SELECT * FROM events_waits_summary_by_thread_by_event_name\G
...
*************************** 6424. row ***************************
     THREAD_ID: 64
    EVENT_NAME: wait/io/socket/sql/server_unix_socket
    COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
*************************** 6425. row ***************************
     THREAD_ID: 64
    EVENT_NAME: wait/io/socket/sql/client_connection
    COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
*************************** 6426. row ***************************
     THREAD_ID: 64
    EVENT_NAME: idle
    COUNT_STAR: 73
SUM_TIMER_WAIT: 22005252162000000
MIN_TIMER_WAIT: 3000000
AVG_TIMER_WAIT: 301441810000000
MAX_TIMER_WAIT: 4912417573000000
```

1.1.1.2.9.2.1.37 Performance Schema events_waits_summary_by_user_by_event_name Table

The Performance Schema `events_waits_summary_by_user_by_event_name` table contains wait events summarized by user and event name. It contains the following columns:

Column	Description
`USER`	User. Used together with `EVENT_NAME` for grouping events.
`EVENT_NAME`	Event name. Used together with `USER` for grouping events.
`COUNT_STAR`	Number of summarized events
`SUM_TIMER_WAIT`	Total wait time of the summarized events that are timed.
`MIN_TIMER_WAIT`	Minimum wait time of the summarized events that are timed.
`AVG_TIMER_WAIT`	Average wait time of the summarized events that are timed.
`MAX_TIMER_WAIT`	Maximum wait time of the summarized events that are timed.

The `*_TIMER_WAIT` columns only calculate results for timed events, as non-timed events have a `NULL` wait time.

Example

```
SELECT * FROM events_waits_summary_by_user_by_event_name\G
...
*************************** 916. row ***************************
          USER: NULL
    EVENT_NAME: wait/io/socket/sql/server_unix_socket
    COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
*************************** 917. row ***************************
          USER: NULL
    EVENT_NAME: wait/io/socket/sql/client_connection
    COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
*************************** 918. row ***************************
          USER: NULL
    EVENT_NAME: idle
    COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
```

1.1.1.2.9.2.1.38 Performance Schema events_waits_summary_global_by_event_name Table

The Performance Schema `events_waits_summary_global_by_event_name` table contains wait events summarized by event name. It contains the following columns:

Column	Description
`EVENT_NAME`	Event name.
`COUNT_STAR`	Number of summarized events
`SUM_TIMER_WAIT`	Total wait time of the summarized events that are timed.
`MIN_TIMER_WAIT`	Minimum wait time of the summarized events that are timed.
`AVG_TIMER_WAIT`	Average wait time of the summarized events that are timed.
`MAX_TIMER_WAIT`	Maximum wait time of the summarized events that are timed.

The `*_TIMER_WAIT` columns only calculate results for timed events, as non-timed events have a `NULL` wait time.

Example

```
SELECT * FROM events_waits_summary_global_by_event_name\G
...
*************************** 303. row ***************************
    EVENT_NAME: wait/io/socket/sql/server_tcpip_socket
    COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
*************************** 304. row ***************************
    EVENT_NAME: wait/io/socket/sql/server_unix_socket
    COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
*************************** 305. row ***************************
    EVENT_NAME: wait/io/socket/sql/client_connection
    COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
*************************** 306. row ***************************
    EVENT_NAME: idle
    COUNT_STAR: 265
SUM_TIMER_WAIT: 46861125181000000
MIN_TIMER_WAIT: 1000000
AVG_TIMER_WAIT: 176834434000000
MAX_TIMER_WAIT: 4912417573000000
```

1.1.1.2.9.2.1.39 Performance Schema file_instances Table

Description

The `file_instances` table lists instances of instruments seen by the Performance Schema when executing file I/O instrumentation, and the associated files. Only files that have been opened, and that have not been deleted, will be listed in the table.

The performance_schema_max_file_instances system variable specifies the maximum number of instrumented file objects.

Column	Description
`FILE_NAME`	File name.
`EVENT_NAME`	Instrument name associated with the file.
`OPEN_COUNT`	Open handles on the file. A value of greater than zero means that the file is currently open.

Example

```
SELECT * FROM performance_schema.file_instances WHERE OPEN_COUNT>0;
+----------------------------------------------------+-------------------------------------
+------------+
| FILE_NAME                                          | EVENT_NAME                          |
OPEN_COUNT |
+----------------------------------------------------+-------------------------------------
+------------+
| /var/log/mysql/mariadb-bin.index                   | wait/io/file/sql/binlog_index       |
1 |
| /var/lib/mysql/ibdata1                             | wait/io/file/innodb/innodb_data_file |
2 |
| /var/lib/mysql/ib_logfile0                         | wait/io/file/innodb/innodb_log_file  |
2 |
| /var/lib/mysql/ib_logfile1                         | wait/io/file/innodb/innodb_log_file  |
2 |
| /var/lib/mysql/mysql/gtid_slave_pos.ibd            | wait/io/file/innodb/innodb_data_file |
3 |
| /var/lib/mysql/mysql/innodb_index_stats.ibd        | wait/io/file/innodb/innodb_data_file |
3 |
| /var/lib/mysql/mysql/innodb_table_stats.ibd        | wait/io/file/innodb/innodb_data_file |
3 |
...
```

1.1.1.2.9.2.1.40 Performance Schema file_summary_by_event_name Table

The Performance Schema `file_summary_by_event_name` table contains file events summarized by event name. It contains the following columns:

Column	Description
`EVENT_NAME`	Event name.
`COUNT_STAR`	Number of summarized events
`SUM_TIMER_WAIT`	Total wait time of the summarized events that are timed.
`MIN_TIMER_WAIT`	Minimum wait time of the summarized events that are timed.
`AVG_TIMER_WAIT`	Average wait time of the summarized events that are timed.
`MAX_TIMER_WAIT`	Maximum wait time of the summarized events that are timed.
`COUNT_READ`	Number of all read operations, including `FGETS`, `FGETC`, `FREAD`, and `READ`.
`SUM_TIMER_READ`	Total wait time of all read operations that are timed.
`MIN_TIMER_READ`	Minimum wait time of all read operations that are timed.
`AVG_TIMER_READ`	Average wait time of all read operations that are timed.
`MAX_TIMER_READ`	Maximum wait time of all read operations that are timed.
`SUM_NUMBER_OF_BYTES_READ`	Bytes read by read operations.
`COUNT_WRITE`	Number of all write operations, including `FPUTS`, `FPUTC`, `FPRINTF`, `VFPRINTF`, `FWRITE`, and `PWRITE`.
`SUM_TIMER_WRITE`	Total wait time of all write operations that are timed.
`MIN_TIMER_WRITE`	Minimum wait time of all write operations that are timed.
`AVG_TIMER_WRITE`	Average wait time of all write operations that are timed.
`MAX_TIMER_WRITE`	Maximum wait time of all write operations that are timed.
`SUM_NUMBER_OF_BYTES_WRITE`	Bytes written by write operations.
`COUNT_MISC`	Number of all miscellaneous operations not counted above, including `CREATE`, `DELETE`, `OPEN`, `CLOSE`, `STREAM_OPEN`, `STREAM_CLOSE`, `SEEK`, `TELL`, `FLUSH`, `STAT`, `FSTAT`, `CHSIZE`, `RENAME`, and `SYNC`.
`SUM_TIMER_MISC`	Total wait time of all miscellaneous operations that are timed.

MIN_TIMER_MISC	Minimum wait time of all miscellaneous operations that are timed.
AVG_TIMER_MISC	Average wait time of all miscellaneous operations that are timed.
MAX_TIMER_MISC	Maximum wait time of all miscellaneous operations that are timed.

I/O operations can be avoided by caching, in which case they will not be recorded in this table.

You can TRUNCATE the table, which will reset all counters to zero.

Example

```
SELECT * FROM file_summary_by_event_name\G
...
*************************** 49. row ***************************
               EVENT_NAME: wait/io/file/aria/MAD
               COUNT_STAR: 60
           SUM_TIMER_WAIT: 397234368
           MIN_TIMER_WAIT: 0
           AVG_TIMER_WAIT: 6620224
           MAX_TIMER_WAIT: 16808672
               COUNT_READ: 0
           SUM_TIMER_READ: 0
           MIN_TIMER_READ: 0
           AVG_TIMER_READ: 0
           MAX_TIMER_READ: 0
 SUM_NUMBER_OF_BYTES_READ: 0
              COUNT_WRITE: 0
          SUM_TIMER_WRITE: 0
          MIN_TIMER_WRITE: 0
          AVG_TIMER_WRITE: 0
          MAX_TIMER_WRITE: 0
SUM_NUMBER_OF_BYTES_WRITE: 0
               COUNT_MISC: 60
           SUM_TIMER_MISC: 397234368
           MIN_TIMER_MISC: 0
           AVG_TIMER_MISC: 6620224
           MAX_TIMER_MISC: 16808672
*************************** 50. row ***************************
               EVENT_NAME: wait/io/file/aria/control
               COUNT_STAR: 3
           SUM_TIMER_WAIT: 24055778544
           MIN_TIMER_WAIT: 0
           AVG_TIMER_WAIT: 8018592848
           MAX_TIMER_WAIT: 24027262400
               COUNT_READ: 1
           SUM_TIMER_READ: 24027262400
           MIN_TIMER_READ: 0
           AVG_TIMER_READ: 24027262400
           MAX_TIMER_READ: 24027262400
 SUM_NUMBER_OF_BYTES_READ: 52
              COUNT_WRITE: 0
          SUM_TIMER_WRITE: 0
          MIN_TIMER_WRITE: 0
          AVG_TIMER_WRITE: 0
          MAX_TIMER_WRITE: 0
SUM_NUMBER_OF_BYTES_WRITE: 0
               COUNT_MISC: 2
           SUM_TIMER_MISC: 28516144
           MIN_TIMER_MISC: 0
           AVG_TIMER_MISC: 14258072
           MAX_TIMER_MISC: 27262208
```

1.1.1.2.9.2.1.41 Performance Schema file_summary_by_instance Table

The Performance Schema `file_summary_by_instance` table contains file events summarized by instance. It contains the following columns:

Column	Description
FILE_NAME	File name.
EVENT_NAME	Event name.
OBJECT_INSTANCE_BEGIN	Address in memory. Together with FILE_NAME and EVENT_NAME uniquely identifies a row.
COUNT_STAR	Number of summarized events
SUM_TIMER_WAIT	Total wait time of the summarized events that are timed.
MIN_TIMER_WAIT	Minimum wait time of the summarized events that are timed.
AVG_TIMER_WAIT	Average wait time of the summarized events that are timed.
MAX_TIMER_WAIT	Maximum wait time of the summarized events that are timed.
COUNT_READ	Number of all read operations, including FGETS , FGETC , FREAD , and READ .
SUM_TIMER_READ	Total wait time of all read operations that are timed.
MIN_TIMER_READ	Minimum wait time of all read operations that are timed.
AVG_TIMER_READ	Average wait time of all read operations that are timed.
MAX_TIMER_READ	Maximum wait time of all read operations that are timed.
SUM_NUMBER_OF_BYTES_READ	Bytes read by read operations.
COUNT_WRITE	Number of all write operations, including FPUTS , FPUTC , FPRINTF , VFPRINTF , FWRITE , and PWRITE .
SUM_TIMER_WRITE	Total wait time of all write operations that are timed.
MIN_TIMER_WRITE	Minimum wait time of all write operations that are timed.
AVG_TIMER_WRITE	Average wait time of all write operations that are timed.
MAX_TIMER_WRITE	Maximum wait time of all write operations that are timed.
SUM_NUMBER_OF_BYTES_WRITE	Bytes written by write operations.
COUNT_MISC	Number of all miscellaneous operations not counted above, including CREATE , DELETE , OPEN , CLOSE , STREAM_OPEN , STREAM_CLOSE , SEEK , TELL , FLUSH , STAT , FSTAT , CHSIZE , RENAME , and SYNC .
SUM_TIMER_MISC	Total wait time of all miscellaneous operations that are timed.
MIN_TIMER_MISC	Minimum wait time of all miscellaneous operations that are timed.
AVG_TIMER_MISC	Average wait time of all miscellaneous operations that are timed.
MAX_TIMER_MISC	Maximum wait time of all miscellaneous operations that are timed.

I/O operations can be avoided by caching, in which case they will not be recorded in this table.

You can TRUNCATE the table, which will reset all counters to zero.

Example

```
SELECT * FROM file_summary_by_instance\G
...
*************************** 204. row ***************************
                FILE_NAME: /var/lib/mysql/test/db.opt
               EVENT_NAME: wait/io/file/sql/dbopt
    OBJECT_INSTANCE_BEGIN: 140578961971264
               COUNT_STAR: 6
           SUM_TIMER_WAIT: 39902495024
           MIN_TIMER_WAIT: 177888
           AVG_TIMER_WAIT: 6650415692
           MAX_TIMER_WAIT: 21026400404
               COUNT_READ: 1
           SUM_TIMER_READ: 21026400404
           MIN_TIMER_READ: 21026400404
           AVG_TIMER_READ: 21026400404
           MAX_TIMER_READ: 21026400404
 SUM_NUMBER_OF_BYTES_READ: 65
              COUNT_WRITE: 0
          SUM_TIMER_WRITE: 0
          MIN_TIMER_WRITE: 0
          AVG_TIMER_WRITE: 0
          MAX_TIMER_WRITE: 0
SUM_NUMBER_OF_BYTES_WRITE: 0
               COUNT_MISC: 5
           SUM_TIMER_MISC: 18876094620
           MIN_TIMER_MISC: 177888
           AVG_TIMER_MISC: 3775218924
           MAX_TIMER_MISC: 18864558060
*************************** 205. row ***************************
                FILE_NAME: /var/log/mysql/mariadb-bin.000157
               EVENT_NAME: wait/io/file/sql/binlog
    OBJECT_INSTANCE_BEGIN: 140578961971968
               COUNT_STAR: 6
           SUM_TIMER_WAIT: 73985877680
           MIN_TIMER_WAIT: 251136
           AVG_TIMER_WAIT: 12330979468
           MAX_TIMER_WAIT: 73846656340
               COUNT_READ: 0
           SUM_TIMER_READ: 0
           MIN_TIMER_READ: 0
           AVG_TIMER_READ: 0
           MAX_TIMER_READ: 0
 SUM_NUMBER_OF_BYTES_READ: 0
              COUNT_WRITE: 2
          SUM_TIMER_WRITE: 62583004
          MIN_TIMER_WRITE: 27630192
          AVG_TIMER_WRITE: 31291284
          MAX_TIMER_WRITE: 34952812
SUM_NUMBER_OF_BYTES_WRITE: 369
               COUNT_MISC: 4
           SUM_TIMER_MISC: 73923294676
           MIN_TIMER_MISC: 251136
           AVG_TIMER_MISC: 18480823560
           MAX_TIMER_MISC: 73846656340
```

1.1.1.2.9.2.1.42 Performance Schema global_status Table

MariaDB starting with 10.5.2

The `global_status` table was added in MariaDB 10.5.2.

The `global_status` table contains a list of status variables and their global values. The table only stores status variable statistics for threads which are instrumented, and does not collect statistics for `Com_xxx` variables.

The table contains the following columns:

Column	Description
`VARIABLE_NAME`	The global status variable name.

VARIABLE_VALUE	The global status variable value.

TRUNCATE TABLE resets global status variables, including thread, account, host, and user status, but not those that are never reset by the server.

1.1.1.2.9.2.1.43 Performance Schema hosts Table

Description

The `hosts` table contains a row for each host used by clients to connect to the server, containing current and total connections.

The size is determined by the performance_schema_hosts_size system variable, which, if set to zero, will disable connection statistics in the hosts table.

It contains the following columns:

Column	Description
HOST	Host name used by the client to connect, `NULL` for internal threads or user sessions that failed to authenticate.
CURRENT_CONNECTIONS	Current number of the host's connections.
TOTAL_CONNECTIONS	Total number of the host's connections

Example

```
SELECT * FROM hosts;
+-----------+---------------------+-------------------+
| HOST      | CURRENT_CONNECTIONS | TOTAL_CONNECTIONS |
+-----------+---------------------+-------------------+
| localhost |                   1 |                45 |
| NULL      |                  20 |                23 |
+-----------+---------------------+-------------------+
```

1.1.1.2.9.2.1.44 Performance Schema host_cache Table

The `host_cache` table contains host and IP information from the host_cache, used for avoiding DNS lookups for new client connections.

The host_cache table contains the following columns:

Column	Description
IP	Client IP address.
HOST	IP's resolved DNS host name, or `NULL` if unknown.
HOST_VALIDATED	YES if the IP-to-host DNS lookup was successful, and the `HOST` column can be used to avoid DNS calls, or NO if unsuccessful, in which case DNS lookup is performed for each connect until either successful or a permanent error.
SUM_CONNECT_ERRORS	Number of connection errors. Counts only protocol handshake errors for hosts that passed validation. These errors count towards max_connect_errors.
COUNT_HOST_BLOCKED_ERRORS	Number of blocked connections because `SUM_CONNECT_ERRORS` exceeded the max_connect_errors system variable.
COUNT_NAMEINFO_TRANSIENT_ERRORS	Number of transient errors during IP-to-host DNS lookups.

COUNT_NAMEINFO_PERMANENT_ERRORS	Number of permanent errors during IP-to-host DNS lookups.
COUNT_FORMAT_ERRORS	Number of host name format errors, for example a numeric host column.
COUNT_ADDRINFO_TRANSIENT_ERRORS	Number of transient errors during host-to-IP reverse DNS lookups.
COUNT_ADDRINFO_PERMANENT_ERRORS	Number of permanent errors during host-to-IP reverse DNS lookups.
COUNT_FCRDNS_ERRORS	Number of forward-confirmed reverse DNS errors, which occur when IP-to-host DNS lookup does not match the originating IP address.
COUNT_HOST_ACL_ERRORS	Number of errors occurring because no user from the host is permitted to log in. These attempts return error code 1130 ER_HOST_NOT_PRIVILEGED and do not proceed to username and password authentication.
COUNT_NO_AUTH_PLUGIN_ERRORS	Number of errors due to requesting an authentication plugin that was not available. This can be due to the plugin never having been loaded, or the load attempt failing.
COUNT_AUTH_PLUGIN_ERRORS	Number of errors reported by an authentication plugin. Plugins can increment COUNT_AUTHENTICATION_ERRORS or COUNT_HANDSHAKE_ERRORS instead, but, if specified or the error is unknown, this column is incremented.
COUNT_HANDSHAKE_ERRORS	Number of errors detected at the wire protocol level.
COUNT_PROXY_USER_ERRORS	Number of errors detected when a proxy user is proxied to a user that does not exist.
COUNT_PROXY_USER_ACL_ERRORS	Number of errors detected when a proxy user is proxied to a user that exists, but the proxy user doesn't have the PROXY privilege.
COUNT_AUTHENTICATION_ERRORS	Number of errors where authentication failed.
COUNT_SSL_ERRORS	Number of errors due to TLS problems.
COUNT_MAX_USER_CONNECTIONS_ERRORS	Number of errors due to the per-user quota being exceeded.
COUNT_MAX_USER_CONNECTIONS_PER_HOUR_ERRORS	Number of errors due to the per-hour quota being exceeded.
COUNT_DEFAULT_DATABASE_ERRORS	Number of errors due to the user not having permission to access the specified default database, or it not existing.
COUNT_INIT_CONNECT_ERRORS	Number of errors due to statements in the init_connect system variable.
COUNT_LOCAL_ERRORS	Number of local server errors, such as out-of-memory errors, unrelated to network, authentication, or authorization.
COUNT_UNKNOWN_ERRORS	Number of unknown errors that cannot be allocated to another column.
FIRST_SEEN	Timestamp of the first connection attempt by the IP.
LAST_SEEN	Timestamp of the most recent connection attempt by the IP.
FIRST_ERROR_SEEN	Timestamp of the first error seen from the IP.
LAST_ERROR_SEEN	Timestamp of the most recent error seen from the IP.

The host_cache table, along with the host_cache , is cleared with FLUSH HOSTS, TRUNCATE TABLE host_cache or by setting the host_cache_size system variable at runtime.

1.1.1.2.9.2.1.45 Performance Schema memory_summary_by_account_by_event_name Table

MariaDB starting with 10.5.2

The memory_summary_by_account_by_event_name table was introduced in MariaDB 10.5.2.

There are five memory summary tables in the Performance Schema that share a number of fields in common. These include:

- memory_summary_by_account_by_event_name
- memory_summary_by_host_by_event_name
- memory_summary_by_thread_by_event_name
- memory_summary_by_user_by_event_name
- memory_global_by_event_name

The `memory_summary_by_account_by_event_name` table contains memory usage statistics aggregated by account and event.

The table contains the following columns:

Field	Type	Null	Default	Description
USER	char(32)	YES	NULL	User portion of the account.
HOST	char(60)	YES	NULL	Host portion of the account.
EVENT_NAME	varchar(128)	NO	NULL	Event name.
COUNT_ALLOC	bigint(20) unsigned	NO	NULL	Total number of allocations to memory.
COUNT_FREE	bigint(20) unsigned	NO	NULL	Total number of attempts to free the allocated memory.
SUM_NUMBER_OF_BYTES_ALLOC	bigint(20) unsigned	NO	NULL	Total number of bytes allocated.
SUM_NUMBER_OF_BYTES_FREE	bigint(20) unsigned	NO	NULL	Total number of bytes freed
LOW_COUNT_USED	bigint(20)	NO	NULL	Lowest number of allocated blocks (lowest value of CURRENT_COUNT_USED).
CURRENT_COUNT_USED	bigint(20)	NO	NULL	Currently allocated blocks that have not been freed (COUNT_ALLOC minus COUNT_FREE).
HIGH_COUNT_USED	bigint(20)	NO	NULL	Highest number of allocated blocks (highest value of CURRENT_COUNT_USED).
LOW_NUMBER_OF_BYTES_USED	bigint(20)	NO	NULL	Lowest number of bytes used.
CURRENT_NUMBER_OF_BYTES_USED	bigint(20)	NO	NULL	Current number of bytes used (total allocated minus total freed).
HIGH_NUMBER_OF_BYTES_USED	bigint(20)	NO	NULL	Highest number of bytes used.

1.1.1.2.9.2.1.46 Performance Schema memory_summary_by_host_by_event_name Table

MariaDB starting with 10.5.2

The memory_summary_by_host_by_event_name table was introduced in MariaDB 10.5.2.

There are five memory summary tables in the Performance Schema that share a number of fields in common. These include:

- memory_summary_by_account_by_event_name
- memory_summary_by_host_by_event_name
- memory_summary_by_thread_by_event_name
- memory_summary_by_user_by_event_name
- memory_global_by_event_name

The `memory_summary_by_host_by_event_name` table contains memory usage statistics aggregated by host and event.

The table contains the following columns:

Field	Type	Null	Default	Description
HOST	char(60)	YES	NULL	Host portion of the account.
EVENT_NAME	varchar(128)	NO	NULL	Event name.
COUNT_ALLOC	bigint(20) unsigned	NO	NULL	Total number of allocations to memory.
COUNT_FREE	bigint(20) unsigned	NO	NULL	Total number of attempts to free the allocated memory.
SUM_NUMBER_OF_BYTES_ALLOC	bigint(20) unsigned	NO	NULL	Total number of bytes allocated.
SUM_NUMBER_OF_BYTES_FREE	bigint(20) unsigned	NO	NULL	Total number of bytes freed
LOW_COUNT_USED	bigint(20)	NO	NULL	Lowest number of allocated blocks (lowest value of CURRENT_COUNT_USED).
CURRENT_COUNT_USED	bigint(20)	NO	NULL	Currently allocated blocks that have not been freed (COUNT_ALLOC minus COUNT_FREE).
HIGH_COUNT_USED	bigint(20)	NO	NULL	Highest number of allocated blocks (highest value of CURRENT_COUNT_USED).
LOW_NUMBER_OF_BYTES_USED	bigint(20)	NO	NULL	Lowest number of bytes used.
CURRENT_NUMBER_OF_BYTES_USED	bigint(20)	NO	NULL	Current number of bytes used (total allocated minus total freed).
HIGH_NUMBER_OF_BYTES_USED	bigint(20)	NO	NULL	Highest number of bytes used.

1.1.1.2.9.2.1.47 Performance Schema memory_summary_by_thread_by_event_name Table

MariaDB starting with 10.5.2

The memory_summary_by_thread_by_event_name table was introduced in MariaDB 10.5.2.

There are five memory summary tables in the Performance Schema that share a number of fields in common. These include:

- memory_summary_by_account_by_event_name
- memory_summary_by_host_by_event_name
- memory_summary_by_thread_by_event_name
- memory_summary_by_user_by_event_name
- memory_global_by_event_name

The `memory_summary_by_thread_by_event_name` table contains memory usage statistics aggregated by thread and event.

The table contains the following columns:

Field	Type	Null	Default	Description
THREAD_ID	bigint(20) unsigned	NO	NULL	Thread id.
EVENT_NAME	varchar(128)	NO	NULL	Event name.
COUNT_ALLOC	bigint(20) unsigned	NO	NULL	Total number of allocations to memory.
COUNT_FREE	bigint(20) unsigned	NO	NULL	Total number of attempts to free the allocated memory.
SUM_NUMBER_OF_BYTES_ALLOC	bigint(20) unsigned	NO	NULL	Total number of bytes allocated.

SUM_NUMBER_OF_BYTES_FREE	bigint(20) unsigned	NO	NULL	Total number of bytes freed
LOW_COUNT_USED	bigint(20)	NO	NULL	Lowest number of allocated blocks (lowest value of CURRENT_COUNT_USED).
CURRENT_COUNT_USED	bigint(20)	NO	NULL	Currently allocated blocks that have not been freed (COUNT_ALLOC minus COUNT_FREE).
HIGH_COUNT_USED	bigint(20)	NO	NULL	Highest number of allocated blocks (highest value of CURRENT_COUNT_USED).
LOW_NUMBER_OF_BYTES_USED	bigint(20)	NO	NULL	Lowest number of bytes used.
CURRENT_NUMBER_OF_BYTES_USED	bigint(20)	NO	NULL	Current number of bytes used (total allocated minus total freed).
HIGH_NUMBER_OF_BYTES_USED	bigint(20)	NO	NULL	Highest number of bytes used.

1.1.1.2.9.2.1.48 Performance Schema memory_summary_by_user_by_event_name Table

MariaDB starting with 10.5.2

The memory_summary_by_user_by_event_name table was introduced in MariaDB 10.5.2.

There are five memory summary tables in the Performance Schema that share a number of fields in common. These include:

- memory_summary_by_account_by_event_name
- memory_summary_by_host_by_event_name
- memory_summary_by_thread_by_event_name
- memory_summary_by_user_by_event_name
- memory_global_by_event_name

The `memory_summary_by_user_by_event_name` table contains memory usage statistics aggregated by user and event.

The table contains the following columns:

Field	Type	Null	Default	Description
USER	char(32)	YES	NULL	User portion of the account.
EVENT_NAME	varchar(128)	NO	NULL	Event name.
COUNT_ALLOC	bigint(20) unsigned	NO	NULL	Total number of allocations to memory.
COUNT_FREE	bigint(20) unsigned	NO	NULL	Total number of attempts to free the allocated memory.
SUM_NUMBER_OF_BYTES_ALLOC	bigint(20) unsigned	NO	NULL	Total number of bytes allocated.
SUM_NUMBER_OF_BYTES_FREE	bigint(20) unsigned	NO	NULL	Total number of bytes freed
LOW_COUNT_USED	bigint(20)	NO	NULL	Lowest number of allocated blocks (lowest value of CURRENT_COUNT_USED).
CURRENT_COUNT_USED	bigint(20)	NO	NULL	Currently allocated blocks that have not been freed (COUNT_ALLOC minus COUNT_FREE).
HIGH_COUNT_USED	bigint(20)	NO	NULL	Highest number of allocated blocks (highest value of CURRENT_COUNT_USED).
LOW_NUMBER_OF_BYTES_USED	bigint(20)	NO	NULL	Lowest number of bytes used.
CURRENT_NUMBER_OF_BYTES_USED	bigint(20)	NO	NULL	Current number of bytes used (total allocated minus total freed).
HIGH_NUMBER_OF_BYTES_USED	bigint(20)	NO	NULL	Highest number of bytes used.

1.1.1.2.9.2.1.49 Performance Schema memory_summary_global_by_event_name Table

MariaDB starting with 10.5.2

The memory_summary_global_by_event_name table was introduced in MariaDB 10.5.2.

There are five memory summary tables in the Performance Schema that share a number of fields in common. These include:

- memory_summary_by_account_by_event_name
- memory_summary_by_host_by_event_name
- memory_summary_by_thread_by_event_name
- memory_summary_by_user_by_event_name
- memory_global_by_event_name

The `memory_summary_global_by_event_name` table contains memory usage statistics aggregated by event and event.

The table contains the following columns:

Field	Type	Null	Default	Description
EVENT_NAME	varchar(128)	NO	NULL	Event name.
COUNT_ALLOC	bigint(20) unsigned	NO	NULL	Total number of allocations to memory.
COUNT_FREE	bigint(20) unsigned	NO	NULL	Total number of attempts to free the allocated memory.
SUM_NUMBER_OF_BYTES_ALLOC	bigint(20) unsigned	NO	NULL	Total number of bytes allocated.
SUM_NUMBER_OF_BYTES_FREE	bigint(20) unsigned	NO	NULL	Total number of bytes freed
LOW_COUNT_USED	bigint(20)	NO	NULL	Lowest number of allocated blocks (lowest value of CURRENT_COUNT_USED).
CURRENT_COUNT_USED	bigint(20)	NO	NULL	Currently allocated blocks that have not been freed (COUNT_ALLOC minus COUNT_FREE).
HIGH_COUNT_USED	bigint(20)	NO	NULL	Highest number of allocated blocks (highest value of CURRENT_COUNT_USED).
LOW_NUMBER_OF_BYTES_USED	bigint(20)	NO	NULL	Lowest number of bytes used.
CURRENT_NUMBER_OF_BYTES_USED	bigint(20)	NO	NULL	Current number of bytes used (total allocated minus total freed).
HIGH_NUMBER_OF_BYTES_USED	bigint(20)	NO	NULL	Highest number of bytes used.

Example

Seeing what memory was most often allocated for:

```
SELECT * FROM memory_summary_global_by_event_name
  ORDER BY count_alloc DESC LIMIT 1\G
*************************** 1. row ***************************
                  EVENT_NAME: memory/sql/QUICK_RANGE_SELECT::alloc
                 COUNT_ALLOC: 147976
                  COUNT_FREE: 147976
   SUM_NUMBER_OF_BYTES_ALLOC: 600190656
    SUM_NUMBER_OF_BYTES_FREE: 600190656
              LOW_COUNT_USED: 0
          CURRENT_COUNT_USED: 0
             HIGH_COUNT_USED: 68
    LOW_NUMBER_OF_BYTES_USED: 0
CURRENT_NUMBER_OF_BYTES_USED: 0
   HIGH_NUMBER_OF_BYTES_USED: 275808
```

1.1.1.2.9.2.1.50 Performance Schema metadata_locks Table

MariaDB starting with 10.5.2

The metadata_locks table was introduced in MariaDB 10.5.2.

The `metadata_locks` table contains metadata lock information.

To enable metadata lock instrumention, at runtime:

```
UPDATE performance_schema.setup_instruments SET enabled='YES', timed='YES'
  WHERE name LIKE 'wait/lock/metadata%';
```

or in the configuration file:

```
performance-schema-instrument='wait/lock/metadata/sql/mdl=ON'
```

The table is by default autosized, but the size can be configured with the performance_schema_max_metadata_locks system variabe.

The table is read-only, and TRUNCATE TABLE cannot be used to empty the table.

The table contains the following columns:

Field	Type	Null	Default	Description
OBJECT_TYPE	varchar(64)	NO	NULL	Object type. One of `BACKUP`, `COMMIT`, `EVENT`, `FUNCTION`, `GLOBAL`, `LOCKING SERVICE`, `PROCEDURE`, `SCHEMA`, `TABLE`, `TABLESPACE`, `TRIGGER` (unused) or `USER LEVEL LOCK`.
OBJECT_SCHEMA	varchar(64)	YES	NULL	Object schema.
OBJECT_NAME	varchar(64)	YES	NULL	Object name.
OBJECT_INSTANCE_BEGIN	bigint(20) unsigned	NO	NULL	Address in memory of the instrumented object.
LOCK_TYPE	varchar(32)	NO	NULL	Lock type. One of `BACKUP_FTWRL1`, `BACKUP_START`, `BACKUP_TRANS_DML`, `EXCLUSIVE`, `INTENTION_EXCLUSIVE`, `SHARED`, `SHARED_HIGH_PRIO`, `SHARED_NO_READ_WRITE`, `SHARED_NO_WRITE`, `SHARED_READ`, `SHARED_UPGRADABLE` or `SHARED_WRITE`.
LOCK_DURATION	varchar(32)	NO	NULL	Lock duration. One of `EXPLICIT` (locks released by explicit action, for example a global lock acquired with FLUSH TABLES WITH READ LOCK), `STATEMENT` (locks implicitly released at statement end) or `TRANSACTION` (locks implicitly released at transaction end).
LOCK_STATUS	varchar(32)	NO	NULL	Lock status. One of `GRANTED`, `KILLED`, `PENDING`, `POST_RELEASE_NOTIFY`, `PRE_ACQUIRE_NOTIFY`, `TIMEOUT` or `VICTIM`.
SOURCE	varchar(64)	YES	NULL	Source file containing the instrumented code that produced the event, as well as the line number where the instrumentation occurred. This allows one to examine the source code involved.
OWNER_THREAD_ID	bigint(20) unsigned	YES	NULL	Thread that requested the lock.
OWNER_EVENT_ID	bigint(20) unsigned	YES	NULL	Event that requested the lock.

1.1.1.2.9.2.1.51 Performance Schema mutex_instances Table

Description

The `mutex_instances` table lists all mutexes that the Performance Schema seeing while the server is executing.

A mutex is a code mechanism for ensuring that threads can only access resources one at a time. A second thread attempting to access a resource will find it protected by a mutex, and will wait for it to be unlocked.

The performance_schema_max_mutex_instances system variable specifies the maximum number of instrumented mutex instances.

Column	Description
`NAME`	Instrument name associated with the mutex.
`OBJECT_INSTANCE_BEGIN`	Memory address of the instrumented mutex.
`LOCKED_BY_THREAD_ID`	The `THREAD_ID` of the locking thread if a thread has a mutex locked, otherwise `NULL`.

1.1.1.2.9.2.1.52 Performance Schema objects_summary_global_by_type Table

It aggregates object wait events, and contains the following columns:

Column	Description
`OBJECT_TYPE`	Groups records together with `OBJECT_SCHEMA` and `OBJECT_NAME`.
`OBJECT_SCHEMA`	Groups records together with `OBJECT_TYPE` and `OBJECT_NAME`.
`OBJECT_NAME`	Groups records together with `OBJECT_SCHEMA` and `OBJECT_TYPE`.
`COUNT_STAR`	Number of summarized events
`SUM_TIMER_WAIT`	Total wait time of the summarized events that are timed.
`MIN_TIMER_WAIT`	Minimum wait time of the summarized events that are timed.
`AVG_TIMER_WAIT`	Average wait time of the summarized events that are timed.
`MAX_TIMER_WAIT`	Maximum wait time of the summarized events that are timed.

You can TRUNCATE the table, which will reset all counters to zero.

Example

```
SELECT * FROM objects_summary_global_by_type\G
...
*************************** 101. row ***************************
   OBJECT_TYPE: TABLE
 OBJECT_SCHEMA: test
   OBJECT_NAME: v
    COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
*************************** 102. row ***************************
   OBJECT_TYPE: TABLE
 OBJECT_SCHEMA: test
   OBJECT_NAME: xx2
    COUNT_STAR: 2
SUM_TIMER_WAIT: 1621920
MIN_TIMER_WAIT: 481344
AVG_TIMER_WAIT: 810960
MAX_TIMER_WAIT: 1140576
```

1.1.1.2.9.2.1.53 Performance Schema

performance_timers Table

Description

The `performance_timers` table lists available event timers.

It contains the following columns:

Column	Description
`TIMER_NAME`	Time name, used in the setup_timers table.
`TIMER_FREQUENCY`	Number of timer units per second. Dependent on the processor speed.
`TIMER_RESOLUTION`	Number of timer units by which timed values increase each time.
`TIMER_OVERHEAD`	Minimum timer overhead, determined during initialization by calling the timer 20 times and selecting the smallest value. Total overhead will be at least double this, as the timer is called at the beginning and end of each timed event.

Any `NULL` values indicate that that particular timer is not available on your platform, Any timer names with a non-NULL value can be used in the setup_timers table.

Example

```
SELECT * FROM performance_timers;
+-------------+-----------------+------------------+----------------------+
| TIMER_NAME  | TIMER_FREQUENCY | TIMER_RESOLUTION | TIMER_OVERHEAD       |
+-------------+-----------------+------------------+----------------------+
| CYCLE       |      2293651741 |                1 |                   28 |
| NANOSECOND  |      1000000000 |                1 |                   48 |
| MICROSECOND |         1000000 |                1 |                   52 |
| MILLISECOND |            1000 |             1000 | 9223372036854775807 |
| TICK        |             106 |                1 |                  496 |
+-------------+-----------------+------------------+----------------------+
```

1.1.1.2.9.2.1.54 Performance Schema prepared_statements_instances Table

MariaDB starting with 10.5.2

The prepared_statements_instances table was introduced in MariaDB 10.5.2.

The `prepared_statements_instances` table contains aggregated statistics of prepared statements.

The maximum number of rows in the table is determined by the performance_schema_max_prepared_statement_instances system variable, which is by default autosized on startup.

The table contains the following columns:

```
+----------------------------+-------------------------------------------------------------+-----
-+-----+---------+-------+
| Field                      | Type                                                        | Null
| Key | Default | Extra |
+----------------------------+-------------------------------------------------------------+-----
-+-----+---------+-------+
| OBJECT_INSTANCE_BEGIN      | bigint(20) unsigned                                         | NO
|     | NULL    |       |
| STATEMENT_ID               | bigint(20) unsigned                                         | NO
|     | NULL    |       |
| STATEMENT_NAME             | varchar(64)                                                 | YES
|     | NULL    |       |
| SQL_TEXT                   | longtext                                                    | NO
|     | NULL    |       |
| OWNER_THREAD_ID            | bigint(20) unsigned                                         | NO
|     | NULL    |       |
| OWNER_EVENT_ID             | bigint(20) unsigned                                         | NO
|     | NULL    |       |
```

```
| OWNER_OBJECT_TYPE           | enum('EVENT','FUNCTION','PROCEDURE','TABLE','TRIGGER') | YES  |
| NULL    |       |
| OWNER_OBJECT_SCHEMA         | varchar(64)                                            | YES
|     | NULL    |       |
| OWNER_OBJECT_NAME           | varchar(64)                                            | YES
|     | NULL    |       |
| TIMER_PREPARE               | bigint(20) unsigned                                    | NO
|     | NULL    |       |
| COUNT_REPREPARE             | bigint(20) unsigned                                    | NO
|     | NULL    |       |
| COUNT_EXECUTE               | bigint(20) unsigned                                    | NO
|     | NULL    |       |
| SUM_TIMER_EXECUTE           | bigint(20) unsigned                                    | NO
|     | NULL    |       |
| MIN_TIMER_EXECUTE           | bigint(20) unsigned                                    | NO
|     | NULL    |       |
| AVG_TIMER_EXECUTE           | bigint(20) unsigned                                    | NO
|     | NULL    |       |
| MAX_TIMER_EXECUTE           | bigint(20) unsigned                                    | NO
|     | NULL    |       |
| SUM_LOCK_TIME               | bigint(20) unsigned                                    | NO
|     | NULL    |       |
| SUM_ERRORS                  | bigint(20) unsigned                                    | NO
|     | NULL    |       |
| SUM_WARNINGS                | bigint(20) unsigned                                    | NO
|     | NULL    |       |
| SUM_ROWS_AFFECTED           | bigint(20) unsigned                                    | NO
|     | NULL    |       |
| SUM_ROWS_SENT               | bigint(20) unsigned                                    | NO
|     | NULL    |       |
| SUM_ROWS_EXAMINED           | bigint(20) unsigned                                    | NO
|     | NULL    |       |
| SUM_CREATED_TMP_DISK_TABLES | bigint(20) unsigned                                    | NO
|     | NULL    |       |
| SUM_CREATED_TMP_TABLES      | bigint(20) unsigned                                    | NO
|     | NULL    |       |
| SUM_SELECT_FULL_JOIN        | bigint(20) unsigned                                    | NO
|     | NULL    |       |
| SUM_SELECT_FULL_RANGE_JOIN  | bigint(20) unsigned                                    | NO
|     | NULL    |       |
| SUM_SELECT_RANGE            | bigint(20) unsigned                                    | NO
|     | NULL    |       |
| SUM_SELECT_RANGE_CHECK      | bigint(20) unsigned                                    | NO
|     | NULL    |       |
| SUM_SELECT_SCAN             | bigint(20) unsigned                                    | NO
|     | NULL    |       |
| SUM_SORT_MERGE_PASSES       | bigint(20) unsigned                                    | NO
|     | NULL    |       |
| SUM_SORT_RANGE              | bigint(20) unsigned                                    | NO
|     | NULL    |       |
| SUM_SORT_ROWS               | bigint(20) unsigned                                    | NO
|     | NULL    |       |
| SUM_SORT_SCAN               | bigint(20) unsigned                                    | NO
|     | NULL    |       |
| SUM_NO_INDEX_USED           | bigint(20) unsigned                                    | NO
|     | NULL    |       |
| SUM_NO_GOOD_INDEX_USED      | bigint(20) unsigned                                    | NO
|     | NULL    |       |
+-----------------------------+--------------------------------------------------------+-----
-+-----+---------+-------+
```

1.1.1.2.9.2.1.55 Performance Schema replication_applier_configuration Table

MariaDB starting with 10.5.2

The `replication_applier_configuration` table, along with many other new Performance Schema tables, was added in MariaDB 10.5.2.

The Performance Schema replication_applier_configuration table contains configuration settings affecting replica

transactions.

It contains the following fields.

Field	Type	Null	Description
CHANNEL_NAME	char(64)	NO	Replication channel name.
DESIRED_DELAY	int(11)	NO	Target number of seconds the replica should be delayed to the master.

1.1.1.2.9.2.1.56 Performance Schema replication_applier_status Table

MariaDB starting with 10.5.2

The `replication_applier_status` table, along with many other new Performance Schema tables, was added in MariaDB 10.5.2.

The Performance Schema replication_applier_status table contains information about the general transaction execution status on the replica.

It contains the following fields.

Field	Type	Null	Description
CHANNEL_NAME	char(64)	NO	The replication channel name.
SERVICE_STATE	enum('ON','OFF')	NO	Shows ON when the replication channel's applier threads are active or idle, OFF means that the applier threads are not active.
REMAINING_DELAY	int(10) unsigned	YES	Seconds the replica needs to wait to reach the desired delay from master.
COUNT_TRANSACTIONS_RETRIES	bigint(20) unsigned	NO	The number of retries that were made because the replication SQL thread failed to apply a transaction.

1.1.1.2.9.2.1.57 Performance Schema replication_applier_status_by_coordinator Table

MariaDB starting with 10.5.2

The `replication_applier_status_by_coordinator` table was added in MariaDB 10.5.2.

The Performance Schema replication_applier_status_by_coordinator table displays the status of the coordinator thread used in multi-threaded replicas to manage multiple worker threads.

It contains the following fields.

Column	Type	Null	Description
CHANNEL_NAME	varchar(256)	NO	Replication channel name.
THREAD_ID	bigint(20) unsigned	YES	The SQL/coordinator thread ID.
SERVICE_STATE	enum('ON','OFF')	NO	ON (thread exists and is active or idle) or OFF (thread no longer exists).
LAST_ERROR_NUMBER	int(11)	NO	Last error number that caused the SQL/coordinator thread to stop.
LAST_ERROR_MESSAGE	varchar(1024)	NO	Last error message that caused the SQL/coordinator thread to stop.
LAST_ERROR_TIMESTAMP	timestamp	NO	Timestamp that shows when the most recent SQL/coordinator error occured.
LAST_SEEN_TRANSACTION	char(57)	NO	The transaction the worker has last seen.
LAST_TRANS_RETRY_COUNT	int(11)	NO	Total number of retries attempted by last transaction.

1.1.1.2.9.2.1.58 Performance Schema replication_applier_status_by_worker Table

MariaDB starting with 10.6.0

The `replication_applier_status_by_worker` table was added in MariaDB 10.6.0.

The Performance Schema replication_applier_status_by_worker table displays replica worker thread specific information.

It contains the following fields.

Column	Description
CHANNEL_NAME	Name of replication channel through which the transaction is received.
THREAD_ID	Thread_Id as displayed in the performance_schema.threads table for thread with name 'thread/sql/rpl_parallel_thread'. THREAD_ID will be NULL when worker threads are stopped due to error/force stop.
SERVICE_STATE	Whether or not the thread is running.
LAST_SEEN_TRANSACTION	Last GTID executed by worker
LAST_ERROR_NUMBER	Last Error that occurred on a particular worker.
LAST_ERROR_MESSAGE	Last error specific message.
LAST_ERROR_TIMESTAMP	Time stamp of last error.
WORKER_IDLE_TIME	Total idle time in seconds that the worker thread has spent waiting for work from SQL thread.
LAST_TRANS_RETRY_COUNT	Total number of retries attempted by last transaction.

1.1.1.2.9.2.1.59 Performance Schema replication_connection_configuration Table

MariaDB starting with 10.5.2

The `replication_connection_configuration` table was added in MariaDB 10.6.0.

The Performance Schema replication_connection_configuration table displays replica's configuration settings used for connecting to the primary.

It contains the following fields.

Column	Type	Null	Description
CHANNEL_NAME	varchar(256)	NO	The replication channel used.
HOST	char(60)	NO	The host name of the source that the replica is connected to.
PORT	int(11)	NO	The port used to connect to the source.
USER	char(32)	NO	The user name of the replication user account used to connect to the source.
USING_GTID	enum('NO', 'CURRENT_POS', 'SLAVE_POS')	NO	Whether replication is using GTIDs or not.
SSL_ALLOWED	enum('YES', 'NO', 'IGNORED')	NO	Whether SSL is allowed for the replica connection.
SSL_CA_FILE	varchar(512)	NO	Path to the file that contains one or more certificates for trusted Certificate Authorities (CA) to use for TLS.

SSL_CA_PATH	varchar(512)	NO	Path to a directory that contains one or more PEM files that contain X509 certificates for a trusted Certificate Authority (CA) to use for TLS.
SSL_CERTIFICATE	varchar(512)	NO	Path to the certificate used to authenticate the master.
SSL_CIPHER	varchar(512)	NO	Which cipher is used for encription.
SSL_KEY	varchar(512)	NO	Path to the private key used for TLS.
SSL_VERIFY_SERVER_CERTIFICATE	enum('YES','NO')	NO	Whether the server certificate is verified as part of the SSL connection.
SSL_CRL_FILE	varchar(255)	NO	Path to the PEM file containing one or more revoked X.509 certificates.
SSL_CRL_PATH	varchar(255)	NO	PATH to a folder containing PEM files containing one or more revoked X.509 certificates.
CONNECTION_RETRY_INTERVAL	int(11)	NO	The number of seconds between connect retries.
CONNECTION_RETRY_COUNT	bigint(20) unsigned	NO	The number of times the replica can attempt to reconnect to the source in the event of a lost connection.
HEARTBEAT_INTERVAL	double(10,3) unsigned	NO	Number of seconds after which a heartbeat will be sent.
IGNORE_SERVER_IDS	longtext	NO	Binary log events from servers (ids) to ignore.
REPL_DO_DOMAIN_IDS	longtext	NO	Only apply binary logs from these domain ids.
REPL_IGNORE_DOMAIN_IDS	longtext	NO	Binary log events from domains to ignore.

1.1.1.2.9.2.1.60 Performance Schema rwlock_instances Table

The `rwlock_instances` table lists all read write lock (rwlock) instances that the Performance Schema sees while the server is executing. A read write is a mechanism for ensuring threads can either share access to common resources, or have exclusive access.

The performance_schema_max_rwlock_instances system variable specifies the maximum number of instrumented rwlock objects.

The `rwlock_instances` table contains the following columns:

Column	Description
`NAME`	Instrument name associated with the read write lock
`OBJECT_INSTANCE_BEGIN`	Address in memory of the instrumented lock
`WRITE_LOCKED_BY_THREAD_ID`	`THREAD_ID` of the locking thread if locked in write (exclusive) mode, otherwise `NULL` .
`READ_LOCKED_BY_COUNT`	Count of current read locks held

1.1.1.2.9.2.1.61 Performance Schema session_account_connect_attrs Table

Description

The `session_account_connect_attrs` table shows connection attributes for the current session.

Applications can pass key/value connection attributes to the server when a connection is made. The session_connect_attrs and `session_account_connect_attrs` tables provide access to this information, for all sessions and the current session respectively.

The C API functions mysql_options() and mysql_optionsv() are used for passing connection attributes to the server.

`session_account_connect_attrs` contains the following columns:

Column	Description
`PROCESSLIST_ID`	Session connection identifier.
`ATTR_NAME`	Attribute name.
`ATTR_VALUE`	Attribute value.
`ORDINAL_POSITION`	Order in which attribute was added to the connection attributes.

Example

```
SELECT * FROM performance_schema.session_account_connect_attrs;
+----------------+-----------------+------------------+------------------+
| PROCESSLIST_ID | ATTR_NAME       | ATTR_VALUE       | ORDINAL_POSITION |
+----------------+-----------------+------------------+------------------+
|             45 | _os             | debian-linux-gnu |                0 |
|             45 | _client_name    | libmysql         |                1 |
|             45 | _pid            | 7711             |                2 |
|             45 | _client_version | 10.0.5           |                3 |
|             45 | _platform       | x86_64           |                4 |
|             45 | program_name    | mysql            |                5 |
+----------------+-----------------+------------------+------------------+
```

1.1.1.2.9.2.1.62 Performance Schema session_connect_attrs Table

Contents

Description

`session_connect_attrs` is a Performance Schema table that shows connection attributes for all sessions. The Performance Schema needs to be enabled for the table to be populated.

Applications can pass key/value connection attributes to the server when a connection is made. The `session_connect_attrs` and session_account_connect_attrs tables provide access to this information, for all sessions and the current session respectively.

The C API functions mysql_options() and mysql_optionsv() are used for passing connection attributes to the server.

`session_connect_attrs` contains the following columns:

Column	Description
`PROCESSLIST_ID`	Session connection identifier.
`ATTR_NAME`	Attribute name.
`ATTR_VALUE`	Attribute value.
`ORDINAL_POSITION`	Order in which attribute was added to the connection attributes.

Example

Returning the current connection's attributes:

```
SELECT * FROM performance_schema.session_connect_attrs WHERE processlist_id=CONNECTION_ID();
+---------------+----------------+-----------------+------------------+
| PROCESSLIST_ID | ATTR_NAME      | ATTR_VALUE      | ORDINAL_POSITION |
+---------------+----------------+-----------------+------------------+
|             45 | _os            | debian-linux-gnu |                0 |
|             45 | _client_name   | libmysql        |                1 |
|             45 | _pid           | 7711            |                2 |
|             45 | _client_version | 10.0.5          |                3 |
|             45 | _platform      | x86_64          |                4 |
|             45 | program_name   | mysql           |                5 |
+---------------+----------------+-----------------+------------------+
```

Using Other Connectors

JDBC

Connection attributes values are set using the option connectionAttributes.

Example using connection string `jdbc:mariadb://localhost/?connectionAttributes=test:test1,test2:test2Val,test3`

```
SELECT * FROM performance_schema.session_connect_attrs WHERE processlist_id=17;
+---------------+----------------+--------------------+------------------+
| PROCESSLIST_ID | ATTR_NAME      | ATTR_VALUE         | ORDINAL_POSITION |
+---------------+----------------+--------------------+------------------+
|             17 | _client_name   | MariaDB Connector/J |                0 |
|             17 | _client_version | 3.1.3              |                1 |
|             17 | _server_host   | localhost          |                2 |
|             17 | _os            | Windows 11         |                3 |
|             17 | _thread        | 1                  |                4 |
|             17 | _java_vendor   | Oracle Corporation |                5 |
|             17 | _java_version  | 19.0.2             |                6 |
|             17 | test           | test1              |                7 |
|             17 | test2          | test2Val           |                8 |
|             17 | test3          | NULL               |                9 |
+---------------+----------------+--------------------+------------------+
```

Node.js

Connection attributes values are set using the option connectAttributes

Example using connection

```
const conn = await mariadb.createConnection({
 host: 'localhost',
 user: 'root',
 connectAttributes: { test: 'test1', test2: 'test2Val', test3: 'f' }
});
```

```
SELECT * FROM performance_schema.session_connect_attrs WHERE processlist_id=30;
+---------------+----------------+-----------------------+------------------+
| PROCESSLIST_ID | ATTR_NAME      | ATTR_VALUE            | ORDINAL_POSITION |
+---------------+----------------+-----------------------+------------------+
|             30 | _client_name   | MariaDB connector/Node |                0 |
|             30 | _client_version | 3.1.1                 |                1 |
|             30 | _server_host   | ::1                   |                2 |
|             30 | _os            | win32                 |                3 |
|             30 | _client_host   | NOSTROMO              |                4 |
|             30 | _node_version  | 18.15.0               |                5 |
|             30 | test           | test1                 |                6 |
|             30 | test2          | test2Val              |                7 |
|             30 | test3          | f                     |                8 |
+---------------+----------------+-----------------------+------------------+
```

R2DBC

Connection attributes values are set using the option connectionAttributes.

Example using connection string `jdbc:mariadb://localhost/?connectionAttributes=test:test1,test2:test2Val,test3`

```
    TreeMap<String, String> connectionAttributes = new TreeMap<>();
    connectionAttributes.put("entry1", "val1");
    connectionAttributes.put("entry2", "val2");

     MariadbConnectionConfiguration conf = MariadbConnectionConfiguration.builder()
            .host("localhost")
            .port(3306)
            .username("root")
            .connectionAttributes(connectionAttributes)
            .database("test")
            .build();
     MariadbConnectionFactory connFactory = new MariadbConnectionFactory(conf);
     MariadbConnection connection = connFactory.create().block();
```

```
SELECT * FROM performance_schema.session_connect_attrs WHERE processlist_id=34;
+----------------+-----------------+--------------------+------------------+
| PROCESSLIST_ID | ATTR_NAME       | ATTR_VALUE         | ORDINAL_POSITION |
+----------------+-----------------+--------------------+------------------+
|             34 | _client_name    | mariadb            |                0 |
|             34 | _client_version | 1.1.4              |                1 |
|             34 | _server_host    | localhost          |                2 |
|             34 | _os             | Windows 11         |                3 |
|             34 | _thread         | 49                 |                4 |
|             34 | _java_vendor    | Oracle Corporation |                5 |
|             34 | _java_version   | 19.0.2             |                6 |
|             34 | entry1          | val1               |                7 |
|             34 | entry2          | val2               |                8 |
+----------------+-----------------+--------------------+------------------+
```

1.1.1.2.9.2.1.63 Performance Schema session_status Table

MariaDB starting with 10.5.2

The `session_status` table was added in MariaDB 10.5.2.

The `session_status` table contains a list of status variables for the current session. The table only stores status variable statistics for threads which are instrumented, and does not collect statistics for `Com_xxx` variables.

The table contains the following columns:

Column	Description
`VARIABLE_NAME`	The session status variable name.
`VARIABLE_VALUE`	The session status variable value.

It is not possible to empty this table with a `TRUNCATE TABLE` statement.

1.1.1.2.9.2.1.64 Performance Schema setup_actors Table

The `setup_actors` table contains information for determining whether monitoring should be enabled for new client connection threads.

The default size is 100 rows, which can be changed by modifying the `performance_schema_setup_actors_size` system variable at server startup.

If a row in the table matches a new foreground thread's client and host, the matching `INSTRUMENTED` column in the threads table is set to either `YES` or `NO`, which allows selective application of instrumenting by host, by user, or combination thereof.

Column	Description

HOST	Host name, either a literal, or the % wildcard representing any host.
USER	User name, either a literal or the % wildcard representing any name.
ROLE	Unused

Initially, any user and host is matched:

```
SELECT * FROM performance_schema.setup_actors;
+------+------+------+
| HOST | USER | ROLE |
+------+------+------+
| %    | %    | %    |
+------+------+------+
```

1.1.1.2.9.2.1.65 Performance Schema setup_consumers Table

Lists the types of consumers for which event information is available.

The setup_consumers table contains the following columns:

Column	Description
NAME	Consumer name
ENABLED	YES or NO for whether or not the consumer is enabled. You can modify this column to ensure that event information is added, or is not added.

The table can be modified directly, or the server started with the option enabled, for example:

```
performance-schema-consumer-events-waits-history=ON
```

Example

```
SELECT * FROM performance_schema.setup_consumers;

+--------------------------------+---------+
| NAME                           | ENABLED |
+--------------------------------+---------+
| events_stages_current          | NO      |
| events_stages_history          | NO      |
| events_stages_history_long     | NO      |
| events_statements_current      | YES     |
| events_statements_history      | NO      |
| events_statements_history_long | NO      |
| events_waits_current           | NO      |
| events_waits_history           | NO      |
| events_waits_history_long      | NO      |
| global_instrumentation         | YES     |
| thread_instrumentation         | YES     |
| statements_digest              | YES     |
+--------------------------------+---------+
```

1.1.1.2.9.2.1.66 Performance Schema setup_instruments Table

The setup_instruments table contains a list of instrumented object classes for which it is possible to collect events. There is one row for each instrument in the source code. When an instrument is enabled and executed, instances are created which are then stored in the cond_instances, file_instances, mutex_instances, rwlock_instances or socket_instance tables.

It contains the following columns:

Column	Description
NAME	Instrument name
ENABLED	Whether or not the instrument is enabled. It can be disabled, and the instrument will produce no events.
TIMED	Whether or not the instrument is timed. It can be set, but if disabled, events produced by the instrument will have NULL values for the corresponding TIMER_START, TIMER_END, and TIMER_WAIT values.

Example

From MariaDB 10.5.7, default settings with the Performance Schema enabled:

```
SELECT * FROM setup_instruments ORDER BY name;
+--------------------------------------------------------------------------------+---------+-----
| NAME                                                                           | ENABLED | TIME
+--------------------------------------------------------------------------------+---------+-----
| idle                                                                           | YES     | YES
| memory/csv/blobroot                                                            | NO      | NO
| memory/csv/row                                                                 | NO      | NO
| memory/csv/tina_set                                                            | NO      | NO
| memory/csv/TINA_SHARE                                                          | NO      | NO
| memory/csv/Transparent_file                                                    | NO      | NO
| memory/innodb/adaptive hash index                                              | NO      | NO
| memory/innodb/btr0btr                                                          | NO      | NO
| memory/innodb/btr0buf                                                          | NO      | NO
| memory/innodb/btr0bulk                                                         | NO      | NO
| memory/innodb/btr0cur                                                          | NO      | NO
| memory/innodb/btr0pcur                                                         | NO      | NO
| memory/innodb/btr0sea                                                          | NO      | NO
| memory/innodb/buf0buf                                                          | NO      | NO
| memory/innodb/buf0dblwr                                                        | NO      | NO
| memory/innodb/buf0dump                                                         | NO      | NO
| memory/innodb/buf_buf_pool                                                     | NO      | NO
| memory/innodb/dict0dict                                                        | NO      | NO
| memory/innodb/dict0mem                                                         | NO      | NO
| memory/innodb/dict0stats                                                       | NO      | NO
| memory/innodb/dict_stats_bg_recalc_pool_t                                      | NO      | NO
| memory/innodb/dict_stats_index_map_t                                           | NO      | NO
| memory/innodb/dict_stats_n_diff_on_level                                       | NO      | NO
| memory/innodb/eval0eval                                                        | NO      | NO
| memory/innodb/fil0crypt                                                        | NO      | NO
| memory/innodb/fil0fil                                                          | NO      | NO
| memory/innodb/fsp0file                                                         | NO      | NO
| memory/innodb/fts0ast                                                          | NO      | NO
| memory/innodb/fts0blex                                                         | NO      | NO
| memory/innodb/fts0config                                                       | NO      | NO
| memory/innodb/fts0file                                                         | NO      | NO
| memory/innodb/fts0fts                                                          | NO      | NO
| memory/innodb/fts0opt                                                          | NO      | NO
| memory/innodb/fts0pars                                                         | NO      | NO
| memory/innodb/fts0que                                                          | NO      | NO
| memory/innodb/fts0sql                                                          | NO      | NO
| memory/innodb/fts0tlex                                                         | NO      | NO
| memory/innodb/gis0sea                                                          | NO      | NO
| memory/innodb/handler0alter                                                    | NO      | NO
| memory/innodb/hash0hash                                                        | NO      | NO
| memory/innodb/ha_innodb                                                        | NO      | NO
| memory/innodb/i_s                                                              | NO      | NO
| memory/innodb/lexyy                                                            | NO      | NO
| memory/innodb/lock0lock                                                        | NO      | NO
| memory/innodb/mem0mem                                                          | NO      | NO
| memory/innodb/os0event                                                         | NO      | NO
| memory/innodb/os0file                                                          | NO      | NO
| memory/innodb/other                                                            | NO      | NO
| memory/innodb/pars0lex                                                         | NO      | NO
| memory/innodb/rem0rec                                                          | NO      | NO
| memory/innodb/row0ftsort                                                       | NO      | NO
| memory/innodb/row0import                                                       | NO      | NO
| memory/innodb/row0log                                                          | NO      | NO
| memory/innodb/row0merge                                                        | NO      | NO
| memory/innodb/row0mysql                                                        | NO      | NO
| memory/innodb/row0sel                                                          | NO      | NO
```

memory/innodb/row_log_buf	NO	NO
memory/innodb/row_merge_sort	NO	NO
memory/innodb/srv0start	NO	NO
memory/innodb/std	NO	NO
memory/innodb/sync0arr	NO	NO
memory/innodb/sync0debug	NO	NO
memory/innodb/sync0rw	NO	NO
memory/innodb/sync0start	NO	NO
memory/innodb/sync0types	NO	NO
memory/innodb/trx0i_s	NO	NO
memory/innodb/trx0roll	NO	NO
memory/innodb/trx0rseg	NO	NO
memory/innodb/trx0seg	NO	NO
memory/innodb/trx0trx	NO	NO
memory/innodb/trx0undo	NO	NO
memory/innodb/ut0list	NO	NO
memory/innodb/ut0mem	NO	NO
memory/innodb/ut0new	NO	NO
memory/innodb/ut0pool	NO	NO
memory/innodb/ut0rbt	NO	NO
memory/innodb/ut0wqueue	NO	NO
memory/innodb/xtrabackup	NO	NO
memory/memory/HP_INFO	NO	NO
memory/memory/HP_KEYDEF	NO	NO
memory/memory/HP_PTRS	NO	NO
memory/memory/HP_SHARE	NO	NO
memory/myisam/filecopy	NO	NO
memory/myisam/FTB	NO	NO
memory/myisam/FTPARSER_PARAM	NO	NO
memory/myisam/FT_INFO	NO	NO
memory/myisam/ft_memroot	NO	NO
memory/myisam/ft_stopwords	NO	NO
memory/myisam/keycache_thread_var	NO	NO
memory/myisam/MI_DECODE_TREE	NO	NO
memory/myisam/MI_INFO	NO	NO
memory/myisam/MI_INFO::bulk_insert	NO	NO
memory/myisam/MI_INFO::ft1_to_ft2	NO	NO
memory/myisam/MI_SORT_PARAM	NO	NO
memory/myisam/MI_SORT_PARAM::wordroot	NO	NO
memory/myisam/MYISAM_SHARE	NO	NO
memory/myisam/MYISAM_SHARE::decode_tables	NO	NO
memory/myisam/preload_buffer	NO	NO
memory/myisam/record_buffer	NO	NO
memory/myisam/SORT_FT_BUF	NO	NO
memory/myisam/SORT_INFO::buffer	NO	NO
memory/myisam/SORT_KEY_BLOCKS	NO	NO
memory/myisam/stPageList::pages	NO	NO
memory/myisammrg/children	NO	NO
memory/myisammrg/MYRG_INFO	NO	NO
memory/partition/ha_partition::file	NO	NO
memory/partition/ha_partition::part_ids	NO	NO
memory/partition/Partition_admin	NO	NO
memory/partition/Partition_share	NO	NO
memory/partition/partition_sort_buffer	NO	NO
memory/performance_schema/accounts	YES	NO
memory/performance_schema/cond_class	YES	NO
memory/performance_schema/cond_instances	YES	NO
memory/performance_schema/events_stages_history	YES	NO
memory/performance_schema/events_stages_history_long	YES	NO
memory/performance_schema/events_stages_summary_by_account_by_event_name	YES	NO
memory/performance_schema/events_stages_summary_by_host_by_event_name	YES	NO
memory/performance_schema/events_stages_summary_by_thread_by_event_name	YES	NO
memory/performance_schema/events_stages_summary_by_user_by_event_name	YES	NO
memory/performance_schema/events_stages_summary_global_by_event_name	YES	NO
memory/performance_schema/events_statements_current	YES	NO
memory/performance_schema/events_statements_current.sqltext	YES	NO
memory/performance_schema/events_statements_current.tokens	YES	NO
memory/performance_schema/events_statements_history	YES	NO
memory/performance_schema/events_statements_history.sqltext	YES	NO
memory/performance_schema/events_statements_history.tokens	YES	NO
memory/performance_schema/events_statements_history_long	YES	NO
memory/performance_schema/events_statements_history_long.sqltext	YES	NO
memory/performance_schema/events_statements_history_long.tokens	YES	NO
memory/performance_schema/events_statements_summary_by_account_by_event_name	YES	NO
memory/performance_schema/events_statements_summary_by_digest	YES	NO

memory/performance_schema/events_statements_summary_by_digest.tokens	YES	NO
memory/performance_schema/events_statements_summary_by_host_by_event_name	YES	NO
memory/performance_schema/events_statements_summary_by_program	YES	NO
memory/performance_schema/events_statements_summary_by_thread_by_event_name	YES	NO
memory/performance_schema/events_statements_summary_by_user_by_event_name	YES	NO
memory/performance_schema/events_statements_summary_global_by_event_name	YES	NO
memory/performance_schema/events_transactions_history	YES	NO
memory/performance_schema/events_transactions_history_long	YES	NO
memory/performance_schema/events_transactions_summary_by_account_by_event_name	YES	NO
memory/performance_schema/events_transactions_summary_by_host_by_event_name	YES	NO
memory/performance_schema/events_transactions_summary_by_thread_by_event_name	YES	NO
memory/performance_schema/events_transactions_summary_by_user_by_event_name	YES	NO
memory/performance_schema/events_waits_history	YES	NO
memory/performance_schema/events_waits_history_long	YES	NO
memory/performance_schema/events_waits_summary_by_account_by_event_name	YES	NO
memory/performance_schema/events_waits_summary_by_host_by_event_name	YES	NO
memory/performance_schema/events_waits_summary_by_thread_by_event_name	YES	NO
memory/performance_schema/events_waits_summary_by_user_by_event_name	YES	NO
memory/performance_schema/file_class	YES	NO
memory/performance_schema/file_handle	YES	NO
memory/performance_schema/file_instances	YES	NO
memory/performance_schema/hosts	YES	NO
memory/performance_schema/memory_class	YES	NO
memory/performance_schema/memory_summary_by_account_by_event_name	YES	NO
memory/performance_schema/memory_summary_by_host_by_event_name	YES	NO
memory/performance_schema/memory_summary_by_thread_by_event_name	YES	NO
memory/performance_schema/memory_summary_by_user_by_event_name	YES	NO
memory/performance_schema/memory_summary_global_by_event_name	YES	NO
memory/performance_schema/metadata_locks	YES	NO
memory/performance_schema/mutex_class	YES	NO
memory/performance_schema/mutex_instances	YES	NO
memory/performance_schema/prepared_statements_instances	YES	NO
memory/performance_schema/rwlock_class	YES	NO
memory/performance_schema/rwlock_instances	YES	NO
memory/performance_schema/scalable_buffer	YES	NO
memory/performance_schema/session_connect_attrs	YES	NO
memory/performance_schema/setup_actors	YES	NO
memory/performance_schema/setup_objects	YES	NO
memory/performance_schema/socket_class	YES	NO
memory/performance_schema/socket_instances	YES	NO
memory/performance_schema/stage_class	YES	NO
memory/performance_schema/statement_class	YES	NO
memory/performance_schema/table_handles	YES	NO
memory/performance_schema/table_io_waits_summary_by_index_usage	YES	NO
memory/performance_schema/table_lock_waits_summary_by_table	YES	NO
memory/performance_schema/table_shares	YES	NO
memory/performance_schema/threads	YES	NO
memory/performance_schema/thread_class	YES	NO
memory/performance_schema/users	YES	NO
memory/sql/acl_cache	NO	NO
memory/sql/binlog_cache_mngr	NO	NO
memory/sql/binlog_pos	NO	NO
memory/sql/binlog_statement_buffer	NO	NO
memory/sql/binlog_ver_1_event	NO	NO
memory/sql/bison_stack	NO	NO
memory/sql/Blob_mem_storage::storage	NO	NO
memory/sql/DATE_TIME_FORMAT	NO	NO
memory/sql/dboptions_hash	NO	NO
memory/sql/DDL_LOG_MEMORY_ENTRY	NO	NO
memory/sql/display_table_locks	NO	NO
memory/sql/errmsgs	NO	NO
memory/sql/Event_basic::mem_root	NO	NO
memory/sql/Event_queue_element_for_exec::names	NO	NO
memory/sql/Event_scheduler::scheduler_param	NO	NO
memory/sql/Filesort_info::merge	NO	NO
memory/sql/Filesort_info::record_pointers	NO	NO
memory/sql/frm::string	NO	NO
memory/sql/gdl	NO	NO
memory/sql/Gis_read_stream::err_msg	NO	NO
memory/sql/global_system_variables	NO	NO
memory/sql/handler::errmsgs	NO	NO
memory/sql/handlerton	NO	NO
memory/sql/hash_index_key_buffer	NO	NO
memory/sql/host_cache::hostname	NO	NO
memory/sql/ignored_db	NO	NO

memory/sql/JOIN_CACHE	NO	NO
memory/sql/load_env_plugins	NO	NO
memory/sql/Locked_tables_list::m_locked_tables_root	NO	NO
memory/sql/MDL_context::acquire_locks	NO	NO
memory/sql/MPVIO_EXT::auth_info	NO	NO
memory/sql/MYSQL_BIN_LOG::basename	NO	NO
memory/sql/MYSQL_BIN_LOG::index	NO	NO
memory/sql/MYSQL_BIN_LOG::recover	NO	NO
memory/sql/MYSQL_LOCK	NO	NO
memory/sql/MYSQL_LOG::name	NO	NO
memory/sql/mysql_plugin	NO	NO
memory/sql/mysql_plugin_dl	NO	NO
memory/sql/MYSQL_RELAY_LOG::basename	NO	NO
memory/sql/MYSQL_RELAY_LOG::index	NO	NO
memory/sql/my_str_malloc	NO	NO
memory/sql/NAMED_ILINK::name	NO	NO
memory/sql/native_functions	NO	NO
memory/sql/plugin_bookmark	NO	NO
memory/sql/plugin_int_mem_root	NO	NO
memory/sql/plugin_mem_root	NO	NO
memory/sql/Prepared_statement::main_mem_root	NO	NO
memory/sql/Prepared_statement_map	NO	NO
memory/sql/PROFILE	NO	NO
memory/sql/Query_cache	NO	NO
memory/sql/Queue::queue_item	NO	NO
memory/sql/QUICK_RANGE_SELECT::alloc	NO	NO
memory/sql/QUICK_RANGE_SELECT::mrr_buf_desc	NO	NO
memory/sql/Relay_log_info::group_relay_log_name	NO	NO
memory/sql/root	NO	NO
memory/sql/Row_data_memory::memory	NO	NO
memory/sql/rpl_filter memory	NO	NO
memory/sql/Rpl_info_file::buffer	NO	NO
memory/sql/servers_cache	NO	NO
memory/sql/SLAVE_INFO	NO	NO
memory/sql/Sort_param::tmp_buffer	NO	NO
memory/sql/sp_head::call_mem_root	NO	NO
memory/sql/sp_head::execute_mem_root	NO	NO
memory/sql/sp_head::main_mem_root	NO	NO
memory/sql/sql_acl_mem	NO	NO
memory/sql/sql_acl_memex	NO	NO
memory/sql/String::value	NO	NO
memory/sql/ST_SCHEMA_TABLE	NO	NO
memory/sql/Sys_var_charptr::value	NO	NO
memory/sql/TABLE	NO	NO
memory/sql/table_mapping::m_mem_root	NO	NO
memory/sql/TABLE_RULE_ENT	NO	NO
memory/sql/TABLE_SHARE::mem_root	NO	NO
memory/sql/Table_triggers_list	NO	NO
memory/sql/Table_trigger_dispatcher::m_mem_root	NO	NO
memory/sql/TC_LOG_MMAP::pages	NO	NO
memory/sql/THD::db	NO	NO
memory/sql/THD::handler_tables_hash	NO	NO
memory/sql/thd::main_mem_root	NO	NO
memory/sql/THD::sp_cache	NO	NO
memory/sql/THD::transactions::mem_root	NO	NO
memory/sql/THD::variables	NO	NO
memory/sql/tz_storage	NO	NO
memory/sql/udf_mem	NO	NO
memory/sql/Unique::merge_buffer	NO	NO
memory/sql/Unique::sort_buffer	NO	NO
memory/sql/user_conn	NO	NO
memory/sql/User_level_lock	NO	NO
memory/sql/user_var_entry	NO	NO
memory/sql/user_var_entry::value	NO	NO
memory/sql/XID	NO	NO
stage/aria/Waiting for a resource	NO	NO
stage/innodb/alter table (end)	YES	YES
stage/innodb/alter table (insert)	YES	YES
stage/innodb/alter table (log apply index)	YES	YES
stage/innodb/alter table (log apply table)	YES	YES
stage/innodb/alter table (merge sort)	YES	YES
stage/innodb/alter table (read PK and internal sort)	YES	YES
stage/innodb/buffer pool load	YES	YES
stage/mysys/Waiting for table level lock	NO	NO
stage/sql/After apply log event	NO	NO

stage/sql/After create	NO	NO
stage/sql/After opening tables	NO	NO
stage/sql/After table lock	NO	NO
stage/sql/Allocating local table	NO	NO
stage/sql/altering table	NO	NO
stage/sql/Apply log event	NO	NO
stage/sql/Changing master	NO	NO
stage/sql/Checking master version	NO	NO
stage/sql/checking permissions	NO	NO
stage/sql/checking privileges on cached query	NO	NO
stage/sql/Checking query cache for query	NO	NO
stage/sql/closing tables	NO	NO
stage/sql/Commit	NO	NO
stage/sql/Commit implicit	NO	NO
stage/sql/Committing alter table to storage engine	NO	NO
stage/sql/Connecting to master	NO	NO
stage/sql/Converting HEAP to Aria	NO	NO
stage/sql/copy to tmp table	YES	YES
stage/sql/Copying to group table	NO	NO
stage/sql/Copying to tmp table	NO	NO
stage/sql/Creating delayed handler	NO	NO
stage/sql/Creating sort index	NO	NO
stage/sql/creating table	NO	NO
stage/sql/Creating tmp table	NO	NO
stage/sql/Deleting from main table	NO	NO
stage/sql/Deleting from reference tables	NO	NO
stage/sql/Discard_or_import_tablespace	NO	NO
stage/sql/Enabling keys	NO	NO
stage/sql/End of update loop	NO	NO
stage/sql/Executing	NO	NO
stage/sql/Execution of init_command	NO	NO
stage/sql/Explaining	NO	NO
stage/sql/Filling schema table	NO	NO
stage/sql/Finding key cache	NO	NO
stage/sql/Finished reading one binlog; switching to next binlog	NO	NO
stage/sql/Flushing relay log and master info repository.	NO	NO
stage/sql/Flushing relay-log info file.	NO	NO
stage/sql/Freeing items	NO	NO
stage/sql/Fulltext initialization	NO	NO
stage/sql/Got handler lock	NO	NO
stage/sql/Got old table	NO	NO
stage/sql/init	NO	NO
stage/sql/init for update	NO	NO
stage/sql/Insert	NO	NO
stage/sql/Invalidating query cache entries (table list)	NO	NO
stage/sql/Invalidating query cache entries (table)	NO	NO
stage/sql/Killing slave	NO	NO
stage/sql/Logging slow query	NO	NO
stage/sql/Making temporary file (append) before replaying LOAD DATA INFILE	NO	NO
stage/sql/Making temporary file (create) before replaying LOAD DATA INFILE	NO	NO
stage/sql/Manage keys	NO	NO
stage/sql/Master has sent all binlog to slave; waiting for more updates	NO	NO
stage/sql/Opening tables	NO	NO
stage/sql/Optimizing	NO	NO
stage/sql/Preparing	NO	NO
stage/sql/preparing for alter table	NO	NO
stage/sql/Processing binlog checkpoint notification	NO	NO
stage/sql/Processing requests	NO	NO
stage/sql/Purging old relay logs	NO	NO
stage/sql/Query end	NO	NO
stage/sql/Queueing master event to the relay log	NO	NO
stage/sql/Reading event from the relay log	NO	NO
stage/sql/Reading semi-sync ACK from slave	NO	NO
stage/sql/Recreating table	NO	NO
stage/sql/Registering slave on master	NO	NO
stage/sql/Removing duplicates	NO	NO
stage/sql/Removing tmp table	NO	NO
stage/sql/Rename	NO	NO
stage/sql/Rename result table	NO	NO
stage/sql/Requesting binlog dump	NO	NO
stage/sql/Reschedule	NO	NO
stage/sql/Reset for next command	NO	NO
stage/sql/Rollback	NO	NO
stage/sql/Rollback_implicit	NO	NO
stage/sql/Searching rows for update	NO	NO

stage/sql/Searching rows for update	NO	NO
stage/sql/Sending binlog event to slave	NO	NO
stage/sql/Sending cached result to client	NO	NO
stage/sql/Sending data	NO	NO
stage/sql/setup	NO	NO
stage/sql/Show explain	NO	NO
stage/sql/Slave has read all relay log; waiting for more updates	NO	NO
stage/sql/Sorting	NO	NO
stage/sql/Sorting for group	NO	NO
stage/sql/Sorting for order	NO	NO
stage/sql/Sorting result	NO	NO
stage/sql/starting	NO	NO
stage/sql/Starting cleanup	NO	NO
stage/sql/Statistics	NO	NO
stage/sql/Stopping binlog background thread	NO	NO
stage/sql/Storing result in query cache	NO	NO
stage/sql/Storing row into queue	NO	NO
stage/sql/System lock	NO	NO
stage/sql/table lock	NO	NO
stage/sql/Unlocking tables	NO	NO
stage/sql/Update	NO	NO
stage/sql/Updating	NO	NO
stage/sql/Updating main table	NO	NO
stage/sql/Updating reference tables	NO	NO
stage/sql/Upgrading lock	NO	NO
stage/sql/User lock	NO	NO
stage/sql/User sleep	NO	NO
stage/sql/Verifying table	NO	NO
stage/sql/Waiting for background binlog tasks	NO	NO
stage/sql/Waiting for backup lock	NO	NO
stage/sql/Waiting for delay_list	NO	NO
stage/sql/Waiting for event metadata lock	NO	NO
stage/sql/Waiting for GTID to be written to binary log	NO	NO
stage/sql/Waiting for handler insert	NO	NO
stage/sql/Waiting for handler lock	NO	NO
stage/sql/Waiting for handler open	NO	NO
stage/sql/Waiting for INSERT	NO	NO
stage/sql/Waiting for master to send event	NO	NO
stage/sql/Waiting for master update	NO	NO
stage/sql/Waiting for next activation	NO	NO
stage/sql/Waiting for other master connection to process the same GTID	NO	NO
stage/sql/Waiting for parallel replication deadlock handling to complete	NO	NO
stage/sql/Waiting for prior transaction to commit	NO	NO
stage/sql/Waiting for prior transaction to start commit	NO	NO
stage/sql/Waiting for query cache lock	NO	NO
stage/sql/Waiting for requests	NO	NO
stage/sql/Waiting for room in worker thread event queue	NO	NO
stage/sql/Waiting for schema metadata lock	NO	NO
stage/sql/Waiting for semi-sync ACK from slave	NO	NO
stage/sql/Waiting for semi-sync slave connection	NO	NO
stage/sql/Waiting for slave mutex on exit	NO	NO
stage/sql/Waiting for slave thread to start	NO	NO
stage/sql/Waiting for stored function metadata lock	NO	NO
stage/sql/Waiting for stored package body metadata lock	NO	NO
stage/sql/Waiting for stored procedure metadata lock	NO	NO
stage/sql/Waiting for table flush	NO	NO
stage/sql/Waiting for table metadata lock	NO	NO
stage/sql/Waiting for the next event in relay log	NO	NO
stage/sql/Waiting for the scheduler to stop	NO	NO
stage/sql/Waiting for the slave SQL thread to advance position	NO	NO
stage/sql/Waiting for the slave SQL thread to free enough relay log space	NO	NO
stage/sql/Waiting for trigger metadata lock	NO	NO
stage/sql/Waiting for work from SQL thread	NO	NO
stage/sql/Waiting in MASTER_GTID_WAIT()	NO	NO
stage/sql/Waiting in MASTER_GTID_WAIT() (primary waiter)	NO	NO
stage/sql/Waiting on empty queue	NO	NO
stage/sql/Waiting to finalize termination	NO	NO
stage/sql/Waiting until MASTER_DELAY seconds after master executed event	NO	NO
stage/sql/Writing to binlog	NO	NO
statement/abstract/new_packet	YES	YES
statement/abstract/Query	YES	YES
statement/abstract/relay_log	YES	YES
statement/com/Binlog Dump	YES	YES
statement/com/Bulk_execute	YES	YES
statement/com/Change user	YES	YES
statement/com/Clone data	YES	YES

statement/com/Close stmt	YES	YES
statement/com/Com_multi	YES	YES
statement/com/Connect	YES	YES
statement/com/Connect Out	YES	YES
statement/com/Create DB	YES	YES
statement/com/Daemon	YES	YES
statement/com/Debug	YES	YES
statement/com/Delayed insert	YES	YES
statement/com/Drop DB	YES	YES
statement/com/Error	YES	YES
statement/com/Execute	YES	YES
statement/com/Fetch	YES	YES
statement/com/Field List	YES	YES
statement/com/Init DB	YES	YES
statement/com/Kill	YES	YES
statement/com/Long Data	YES	YES
statement/com/Ping	YES	YES
statement/com/Prepare	YES	YES
statement/com/Processlist	YES	YES
statement/com/Quit	YES	YES
statement/com/Refresh	YES	YES
statement/com/Register Slave	YES	YES
statement/com/Reset connection	YES	YES
statement/com/Reset stmt	YES	YES
statement/com/Set option	YES	YES
statement/com/Shutdown	YES	YES
statement/com/Slave_IO	YES	YES
statement/com/Slave_SQL	YES	YES
statement/com/Slave_worker	YES	YES
statement/com/Sleep	YES	YES
statement/com/Statistics	YES	YES
statement/com/Table Dump	YES	YES
statement/com/Time	YES	YES
statement/com/Unimpl get tid	YES	YES
statement/scheduler/event	YES	YES
statement/sp/agg_cfetch	YES	YES
statement/sp/cclose	YES	YES
statement/sp/cfetch	YES	YES
statement/sp/copen	YES	YES
statement/sp/cpop	YES	YES
statement/sp/cpush	YES	YES
statement/sp/cursor_copy_struct	YES	YES
statement/sp/error	YES	YES
statement/sp/freturn	YES	YES
statement/sp/hpop	YES	YES
statement/sp/hpush_jump	YES	YES
statement/sp/hreturn	YES	YES
statement/sp/jump	YES	YES
statement/sp/jump_if_not	YES	YES
statement/sp/preturn	YES	YES
statement/sp/set	YES	YES
statement/sp/set_case_expr	YES	YES
statement/sp/set_trigger_field	YES	YES
statement/sp/stmt	YES	YES
statement/sql/	YES	YES
statement/sql/alter_db	YES	YES
statement/sql/alter_db_upgrade	YES	YES
statement/sql/alter_event	YES	YES
statement/sql/alter_function	YES	YES
statement/sql/alter_procedure	YES	YES
statement/sql/alter_sequence	YES	YES
statement/sql/alter_server	YES	YES
statement/sql/alter_table	YES	YES
statement/sql/alter_tablespace	YES	YES
statement/sql/alter_user	YES	YES
statement/sql/analyze	YES	YES
statement/sql/assign_to_keycache	YES	YES
statement/sql/backup	YES	YES
statement/sql/backup_lock	YES	YES
statement/sql/begin	YES	YES
statement/sql/binlog	YES	YES
statement/sql/call_procedure	YES	YES
statement/sql/change_db	YES	YES
statement/sql/change_master	YES	YES
statement/sql/check	YES	YES

statement/sql/checksum	YES	YES
statement/sql/commit	YES	YES
statement/sql/compound_sql	YES	YES
statement/sql/create_db	YES	YES
statement/sql/create_event	YES	YES
statement/sql/create_function	YES	YES
statement/sql/create_index	YES	YES
statement/sql/create_package	YES	YES
statement/sql/create_package_body	YES	YES
statement/sql/create_procedure	YES	YES
statement/sql/create_role	YES	YES
statement/sql/create_sequence	YES	YES
statement/sql/create_server	YES	YES
statement/sql/create_table	YES	YES
statement/sql/create_trigger	YES	YES
statement/sql/create_udf	YES	YES
statement/sql/create_user	YES	YES
statement/sql/create_view	YES	YES
statement/sql/dealloc_sql	YES	YES
statement/sql/delete	YES	YES
statement/sql/delete_multi	YES	YES
statement/sql/do	YES	YES
statement/sql/drop_db	YES	YES
statement/sql/drop_event	YES	YES
statement/sql/drop_function	YES	YES
statement/sql/drop_index	YES	YES
statement/sql/drop_package	YES	YES
statement/sql/drop_package_body	YES	YES
statement/sql/drop_procedure	YES	YES
statement/sql/drop_role	YES	YES
statement/sql/drop_sequence	YES	YES
statement/sql/drop_server	YES	YES
statement/sql/drop_table	YES	YES
statement/sql/drop_trigger	YES	YES
statement/sql/drop_user	YES	YES
statement/sql/drop_view	YES	YES
statement/sql/empty_query	YES	YES
statement/sql/error	YES	YES
statement/sql/execute_immediate	YES	YES
statement/sql/execute_sql	YES	YES
statement/sql/flush	YES	YES
statement/sql/get_diagnostics	YES	YES
statement/sql/grant	YES	YES
statement/sql/grant_role	YES	YES
statement/sql/ha_close	YES	YES
statement/sql/ha_open	YES	YES
statement/sql/ha_read	YES	YES
statement/sql/help	YES	YES
statement/sql/insert	YES	YES
statement/sql/insert_select	YES	YES
statement/sql/install_plugin	YES	YES
statement/sql/kill	YES	YES
statement/sql/load	YES	YES
statement/sql/lock_tables	YES	YES
statement/sql/optimize	YES	YES
statement/sql/preload_keys	YES	YES
statement/sql/prepare_sql	YES	YES
statement/sql/purge	YES	YES
statement/sql/purge_before_date	YES	YES
statement/sql/release_savepoint	YES	YES
statement/sql/rename_table	YES	YES
statement/sql/rename_user	YES	YES
statement/sql/repair	YES	YES
statement/sql/replace	YES	YES
statement/sql/replace_select	YES	YES
statement/sql/reset	YES	YES
statement/sql/resignal	YES	YES
statement/sql/revoke	YES	YES
statement/sql/revoke_all	YES	YES
statement/sql/revoke_role	YES	YES
statement/sql/rollback	YES	YES
statement/sql/rollback_to_savepoint	YES	YES
statement/sql/savepoint	YES	YES
statement/sql/select	YES	YES
statement/sql/set_option	YES	YES

statement/sql/show_authors	YES	YES
statement/sql/show_binlogs	YES	YES
statement/sql/show_binlog_events	YES	YES
statement/sql/show_binlog_status	YES	YES
statement/sql/show_charsets	YES	YES
statement/sql/show_collations	YES	YES
statement/sql/show_contributors	YES	YES
statement/sql/show_create_db	YES	YES
statement/sql/show_create_event	YES	YES
statement/sql/show_create_func	YES	YES
statement/sql/show_create_package	YES	YES
statement/sql/show_create_package_body	YES	YES
statement/sql/show_create_proc	YES	YES
statement/sql/show_create_table	YES	YES
statement/sql/show_create_trigger	YES	YES
statement/sql/show_create_user	YES	YES
statement/sql/show_databases	YES	YES
statement/sql/show_engine_logs	YES	YES
statement/sql/show_engine_mutex	YES	YES
statement/sql/show_engine_status	YES	YES
statement/sql/show_errors	YES	YES
statement/sql/show_events	YES	YES
statement/sql/show_explain	YES	YES
statement/sql/show_fields	YES	YES
statement/sql/show_function_status	YES	YES
statement/sql/show_generic	YES	YES
statement/sql/show_grants	YES	YES
statement/sql/show_keys	YES	YES
statement/sql/show_open_tables	YES	YES
statement/sql/show_package_body_status	YES	YES
statement/sql/show_package_status	YES	YES
statement/sql/show_plugins	YES	YES
statement/sql/show_privileges	YES	YES
statement/sql/show_procedure_status	YES	YES
statement/sql/show_processlist	YES	YES
statement/sql/show_profile	YES	YES
statement/sql/show_profiles	YES	YES
statement/sql/show_relaylog_events	YES	YES
statement/sql/show_slave_hosts	YES	YES
statement/sql/show_slave_status	YES	YES
statement/sql/show_status	YES	YES
statement/sql/show_storage_engines	YES	YES
statement/sql/show_tables	YES	YES
statement/sql/show_table_status	YES	YES
statement/sql/show_triggers	YES	YES
statement/sql/show_variables	YES	YES
statement/sql/show_warnings	YES	YES
statement/sql/shutdown	YES	YES
statement/sql/signal	YES	YES
statement/sql/start_all_slaves	YES	YES
statement/sql/start_slave	YES	YES
statement/sql/stop_all_slaves	YES	YES
statement/sql/stop_slave	YES	YES
statement/sql/truncate	YES	YES
statement/sql/uninstall_plugin	YES	YES
statement/sql/unlock_tables	YES	YES
statement/sql/update	YES	YES
statement/sql/update_multi	YES	YES
statement/sql/xa_commit	YES	YES
statement/sql/xa_end	YES	YES
statement/sql/xa_prepare	YES	YES
statement/sql/xa_recover	YES	YES
statement/sql/xa_rollback	YES	YES
statement/sql/xa_start	YES	YES
transaction	NO	NO
wait/io/file/aria/control	YES	YES
wait/io/file/aria/MAD	YES	YES
wait/io/file/aria/MAI	YES	YES
wait/io/file/aria/translog	YES	YES
wait/io/file/csv/data	YES	YES
wait/io/file/csv/metadata	YES	YES
wait/io/file/csv/update	YES	YES
wait/io/file/innodb/innodb_data_file	YES	YES
wait/io/file/innodb/innodb_log_file	YES	YES
wait/io/file/innodb/innodb_temp_file	YES	YES

wait/io/file/myisam/data_tmp	YES	YES
wait/io/file/myisam/dfile	YES	YES
wait/io/file/myisam/kfile	YES	YES
wait/io/file/myisam/log	YES	YES
wait/io/file/myisammrg/MRG	YES	YES
wait/io/file/mysys/charset	YES	YES
wait/io/file/mysys/cnf	YES	YES
wait/io/file/partition/ha_partition::parfile	YES	YES
wait/io/file/sql/binlog	YES	YES
wait/io/file/sql/binlog_cache	YES	YES
wait/io/file/sql/binlog_index	YES	YES
wait/io/file/sql/binlog_index_cache	YES	YES
wait/io/file/sql/binlog_state	YES	YES
wait/io/file/sql/casetest	YES	YES
wait/io/file/sql/dbopt	YES	YES
wait/io/file/sql/des_key_file	YES	YES
wait/io/file/sql/ERRMSG	YES	YES
wait/io/file/sql/file_parser	YES	YES
wait/io/file/sql/FRM	YES	YES
wait/io/file/sql/global_ddl_log	YES	YES
wait/io/file/sql/init	YES	YES
wait/io/file/sql/io_cache	YES	YES
wait/io/file/sql/load	YES	YES
wait/io/file/sql/LOAD_FILE	YES	YES
wait/io/file/sql/log_event_data	YES	YES
wait/io/file/sql/log_event_info	YES	YES
wait/io/file/sql/map	YES	YES
wait/io/file/sql/master_info	YES	YES
wait/io/file/sql/misc	YES	YES
wait/io/file/sql/partition_ddl_log	YES	YES
wait/io/file/sql/pid	YES	YES
wait/io/file/sql/query_log	YES	YES
wait/io/file/sql/relaylog	YES	YES
wait/io/file/sql/relaylog_cache	YES	YES
wait/io/file/sql/relaylog_index	YES	YES
wait/io/file/sql/relaylog_index_cache	YES	YES
wait/io/file/sql/relay_log_info	YES	YES
wait/io/file/sql/select_to_file	YES	YES
wait/io/file/sql/send_file	YES	YES
wait/io/file/sql/slow_log	YES	YES
wait/io/file/sql/tclog	YES	YES
wait/io/file/sql/trigger	YES	YES
wait/io/file/sql/trigger_name	YES	YES
wait/io/file/sql/wsrep_gra_log	YES	YES
wait/io/socket/sql/client_connection	NO	NO
wait/io/socket/sql/server_tcpip_socket	NO	NO
wait/io/socket/sql/server_unix_socket	NO	NO
wait/io/table/sql/handler	YES	YES
wait/lock/metadata/sql/mdl	NO	NO
wait/lock/table/sql/handler	YES	YES
wait/synch/cond/aria/BITMAP::bitmap_cond	NO	NO
wait/synch/cond/aria/COND_soft_sync	NO	NO
wait/synch/cond/aria/SERVICE_THREAD_CONTROL::COND_control	NO	NO
wait/synch/cond/aria/SHARE::key_del_cond	NO	NO
wait/synch/cond/aria/SORT_INFO::cond	NO	NO
wait/synch/cond/aria/TRANSLOG_BUFFER::prev_sent_to_disk_cond	NO	NO
wait/synch/cond/aria/TRANSLOG_BUFFER::waiting_filling_buffer	NO	NO
wait/synch/cond/aria/TRANSLOG_DESCRIPTOR::log_flush_cond	NO	NO
wait/synch/cond/aria/TRANSLOG_DESCRIPTOR::new_goal_cond	NO	NO
wait/synch/cond/innodb/commit_cond	NO	NO
wait/synch/cond/myisam/MI_SORT_INFO::cond	NO	NO
wait/synch/cond/mysys/COND_alarm	NO	NO
wait/synch/cond/mysys/COND_timer	NO	NO
wait/synch/cond/mysys/IO_CACHE_SHARE::cond	NO	NO
wait/synch/cond/mysys/IO_CACHE_SHARE::cond_writer	NO	NO
wait/synch/cond/mysys/my_thread_var::suspend	NO	NO
wait/synch/cond/mysys/THR_COND_threads	NO	NO
wait/synch/cond/mysys/WT_RESOURCE::cond	NO	NO
wait/synch/cond/sql/Ack_receiver::cond	NO	NO
wait/synch/cond/sql/COND_binlog_send	NO	NO
wait/synch/cond/sql/COND_flush_thread_cache	NO	NO
wait/synch/cond/sql/COND_group_commit_orderer	NO	NO
wait/synch/cond/sql/COND_gtid_ignore_duplicates	NO	NO
wait/synch/cond/sql/COND_manager	NO	NO
wait/synch/cond/sql/COND_parallel_entry	NO	NO

wait/synch/cond/sql/COND_prepare_ordered	NO	NO
wait/synch/cond/sql/COND_queue_state	NO	NO
wait/synch/cond/sql/COND_rpl_thread	NO	NO
wait/synch/cond/sql/COND_rpl_thread_pool	NO	NO
wait/synch/cond/sql/COND_rpl_thread_queue	NO	NO
wait/synch/cond/sql/COND_rpl_thread_stop	NO	NO
wait/synch/cond/sql/COND_server_started	NO	NO
wait/synch/cond/sql/COND_slave_background	NO	NO
wait/synch/cond/sql/COND_start_thread	NO	NO
wait/synch/cond/sql/COND_thread_cache	NO	NO
wait/synch/cond/sql/COND_wait_gtid	NO	NO
wait/synch/cond/sql/COND_wsrep_donor_monitor	NO	NO
wait/synch/cond/sql/COND_wsrep_gtid_wait_upto	NO	NO
wait/synch/cond/sql/COND_wsrep_joiner_monitor	NO	NO
wait/synch/cond/sql/COND_wsrep_ready	NO	NO
wait/synch/cond/sql/COND_wsrep_replaying	NO	NO
wait/synch/cond/sql/COND_wsrep_sst	NO	NO
wait/synch/cond/sql/COND_wsrep_sst_init	NO	NO
wait/synch/cond/sql/COND_wsrep_wsrep_slave_threads	NO	NO
wait/synch/cond/sql/Delayed_insert::cond	NO	NO
wait/synch/cond/sql/Delayed_insert::cond_client	NO	NO
wait/synch/cond/sql/Event_scheduler::COND_state	NO	NO
wait/synch/cond/sql/Item_func_sleep::cond	NO	NO
wait/synch/cond/sql/Master_info::data_cond	NO	NO
wait/synch/cond/sql/Master_info::sleep_cond	NO	NO
wait/synch/cond/sql/Master_info::start_cond	NO	NO
wait/synch/cond/sql/Master_info::stop_cond	NO	NO
wait/synch/cond/sql/MDL_context::COND_wait_status	NO	NO
wait/synch/cond/sql/MYSQL_BIN_LOG::COND_binlog_background_thread	NO	NO
wait/synch/cond/sql/MYSQL_BIN_LOG::COND_binlog_background_thread_end	NO	NO
wait/synch/cond/sql/MYSQL_BIN_LOG::COND_bin_log_updated	NO	NO
wait/synch/cond/sql/MYSQL_BIN_LOG::COND_queue_busy	NO	NO
wait/synch/cond/sql/MYSQL_BIN_LOG::COND_relay_log_updated	NO	NO
wait/synch/cond/sql/MYSQL_BIN_LOG::COND_xid_list	NO	NO
wait/synch/cond/sql/MYSQL_RELAY_LOG::COND_bin_log_updated	NO	NO
wait/synch/cond/sql/MYSQL_RELAY_LOG::COND_queue_busy	NO	NO
wait/synch/cond/sql/MYSQL_RELAY_LOG::COND_relay_log_updated	NO	NO
wait/synch/cond/sql/PAGE::cond	NO	NO
wait/synch/cond/sql/Query_cache::COND_cache_status_changed	NO	NO
wait/synch/cond/sql/Relay_log_info::data_cond	NO	NO
wait/synch/cond/sql/Relay_log_info::log_space_cond	NO	NO
wait/synch/cond/sql/Relay_log_info::start_cond	NO	NO
wait/synch/cond/sql/Relay_log_info::stop_cond	NO	NO
wait/synch/cond/sql/Rpl_group_info::sleep_cond	NO	NO
wait/synch/cond/sql/show_explain	NO	NO
wait/synch/cond/sql/TABLE_SHARE::cond	NO	NO
wait/synch/cond/sql/TABLE_SHARE::COND_rotation	NO	NO
wait/synch/cond/sql/TABLE_SHARE::tdc.COND_release	NO	NO
wait/synch/cond/sql/TC_LOG_MMAP::COND_active	NO	NO
wait/synch/cond/sql/TC_LOG_MMAP::COND_pool	NO	NO
wait/synch/cond/sql/TC_LOG_MMAP::COND_queue_busy	NO	NO
wait/synch/cond/sql/THD::COND_wakeup_ready	NO	NO
wait/synch/cond/sql/THD::COND_wsrep_thd	NO	NO
wait/synch/cond/sql/User_level_lock::cond	NO	NO
wait/synch/cond/sql/wait_for_commit::COND_wait_commit	NO	NO
wait/synch/cond/sql/wsrep_sst_thread	NO	NO
wait/synch/mutex/aria/LOCK_soft_sync	NO	NO
wait/synch/mutex/aria/LOCK_trn_list	NO	NO
wait/synch/mutex/aria/PAGECACHE::cache_lock	NO	NO
wait/synch/mutex/aria/SERVICE_THREAD_CONTROL::LOCK_control	NO	NO
wait/synch/mutex/aria/SHARE::bitmap::bitmap_lock	NO	NO
wait/synch/mutex/aria/SHARE::close_lock	NO	NO
wait/synch/mutex/aria/SHARE::intern_lock	NO	NO
wait/synch/mutex/aria/SHARE::key_del_lock	NO	NO
wait/synch/mutex/aria/SORT_INFO::mutex	NO	NO
wait/synch/mutex/aria/THR_LOCK_maria	NO	NO
wait/synch/mutex/aria/TRANSLOG_BUFFER::mutex	NO	NO
wait/synch/mutex/aria/TRANSLOG_DESCRIPTOR::dirty_buffer_mask_lock	NO	NO
wait/synch/mutex/aria/TRANSLOG_DESCRIPTOR::file_header_lock	NO	NO
wait/synch/mutex/aria/TRANSLOG_DESCRIPTOR::log_flush_lock	NO	NO
wait/synch/mutex/aria/TRANSLOG_DESCRIPTOR::purger_lock	NO	NO
wait/synch/mutex/aria/TRANSLOG_DESCRIPTOR::sent_to_disk_lock	NO	NO
wait/synch/mutex/aria/TRANSLOG_DESCRIPTOR::unfinished_files_lock	NO	NO
wait/synch/mutex/aria/TRN::state_lock	NO	NO
wait/synch/mutex/csv/tina	NO	NO

wait/synch/mutex/csv/TINA	NO	NO
wait/synch/mutex/csv/TINA_SHARE::mutex	NO	NO
wait/synch/mutex/innodb/buf_dblwr_mutex	NO	NO
wait/synch/mutex/innodb/buf_pool_mutex	NO	NO
wait/synch/mutex/innodb/commit_cond_mutex	NO	NO
wait/synch/mutex/innodb/dict_foreign_err_mutex	NO	NO
wait/synch/mutex/innodb/dict_sys_mutex	NO	NO
wait/synch/mutex/innodb/fil_system_mutex	NO	NO
wait/synch/mutex/innodb/flush_list_mutex	NO	NO
wait/synch/mutex/innodb/fts_delete_mutex	NO	NO
wait/synch/mutex/innodb/fts_doc_id_mutex	NO	NO
wait/synch/mutex/innodb/ibuf_bitmap_mutex	NO	NO
wait/synch/mutex/innodb/ibuf_mutex	NO	NO
wait/synch/mutex/innodb/ibuf_pessimistic_insert_mutex	NO	NO
wait/synch/mutex/innodb/lock_mutex	NO	NO
wait/synch/mutex/innodb/lock_wait_mutex	NO	NO
wait/synch/mutex/innodb/log_flush_order_mutex	NO	NO
wait/synch/mutex/innodb/log_sys_mutex	NO	NO
wait/synch/mutex/innodb/noredo_rseg_mutex	NO	NO
wait/synch/mutex/innodb/page_zip_stat_per_index_mutex	NO	NO
wait/synch/mutex/innodb/pending_checkpoint_mutex	NO	NO
wait/synch/mutex/innodb/purge_sys_pq_mutex	NO	NO
wait/synch/mutex/innodb/recalc_pool_mutex	NO	NO
wait/synch/mutex/innodb/recv_sys_mutex	NO	NO
wait/synch/mutex/innodb/redo_rseg_mutex	NO	NO
wait/synch/mutex/innodb/rtr_active_mutex	NO	NO
wait/synch/mutex/innodb/rtr_match_mutex	NO	NO
wait/synch/mutex/innodb/rtr_path_mutex	NO	NO
wait/synch/mutex/innodb/rw_lock_list_mutex	NO	NO
wait/synch/mutex/innodb/srv_innodb_monitor_mutex	NO	NO
wait/synch/mutex/innodb/srv_misc_tmpfile_mutex	NO	NO
wait/synch/mutex/innodb/srv_monitor_file_mutex	NO	NO
wait/synch/mutex/innodb/srv_threads_mutex	NO	NO
wait/synch/mutex/innodb/trx_mutex	NO	NO
wait/synch/mutex/innodb/trx_pool_manager_mutex	NO	NO
wait/synch/mutex/innodb/trx_pool_mutex	NO	NO
wait/synch/mutex/innodb/trx_sys_mutex	NO	NO
wait/synch/mutex/myisam/MI_CHECK::print_msg	NO	NO
wait/synch/mutex/myisam/MI_SORT_INFO::mutex	NO	NO
wait/synch/mutex/myisam/MYISAM_SHARE::intern_lock	NO	NO
wait/synch/mutex/myisammrg/MYRG_INFO::mutex	NO	NO
wait/synch/mutex/mysys/BITMAP::mutex	NO	NO
wait/synch/mutex/mysys/IO_CACHE::append_buffer_lock	NO	NO
wait/synch/mutex/mysys/IO_CACHE::SHARE_mutex	NO	NO
wait/synch/mutex/mysys/KEY_CACHE::cache_lock	NO	NO
wait/synch/mutex/mysys/LOCK_alarm	NO	NO
wait/synch/mutex/mysys/LOCK_timer	NO	NO
wait/synch/mutex/mysys/LOCK_uuid_generator	NO	NO
wait/synch/mutex/mysys/my_thread_var::mutex	NO	NO
wait/synch/mutex/mysys/THR_LOCK::mutex	NO	NO
wait/synch/mutex/mysys/THR_LOCK_charset	NO	NO
wait/synch/mutex/mysys/THR_LOCK_heap	NO	NO
wait/synch/mutex/mysys/THR_LOCK_lock	NO	NO
wait/synch/mutex/mysys/THR_LOCK_malloc	NO	NO
wait/synch/mutex/mysys/THR_LOCK_myisam	NO	NO
wait/synch/mutex/mysys/THR_LOCK_myisam_mmap	NO	NO
wait/synch/mutex/mysys/THR_LOCK_net	NO	NO
wait/synch/mutex/mysys/THR_LOCK_open	NO	NO
wait/synch/mutex/mysys/THR_LOCK_threads	NO	NO
wait/synch/mutex/mysys/TMPDIR_mutex	NO	NO
wait/synch/mutex/partition/Partition_share::auto_inc_mutex	NO	NO
wait/synch/mutex/sql/Ack_receiver::mutex	NO	NO
wait/synch/mutex/sql/Cversion_lock	NO	NO
wait/synch/mutex/sql/Delayed_insert::mutex	NO	NO
wait/synch/mutex/sql/Event_scheduler::LOCK_scheduler_state	NO	NO
wait/synch/mutex/sql/gtid_waiting::LOCK_gtid_waiting	NO	NO
wait/synch/mutex/sql/hash_filo::lock	NO	NO
wait/synch/mutex/sql/HA_DATA_PARTITION::LOCK_auto_inc	NO	NO
wait/synch/mutex/sql/LOCK_active_mi	NO	NO
wait/synch/mutex/sql/LOCK_after_binlog_sync	NO	NO
wait/synch/mutex/sql/LOCK_audit_mask	NO	NO
wait/synch/mutex/sql/LOCK_binlog	NO	NO
wait/synch/mutex/sql/LOCK_binlog_state	NO	NO
wait/synch/mutex/sql/LOCK_commit_ordered	NO	NO
wait/synch/mutex/sql/LOCK_crypt	NO	NO

wait/synch/mutex/sql/LOCK_delayed_create	NO	NO
wait/synch/mutex/sql/LOCK_delayed_insert	NO	NO
wait/synch/mutex/sql/LOCK_delayed_status	NO	NO
wait/synch/mutex/sql/LOCK_des_key_file	NO	NO
wait/synch/mutex/sql/LOCK_error_log	NO	NO
wait/synch/mutex/sql/LOCK_error_messages	NO	NO
wait/synch/mutex/sql/LOCK_event_queue	NO	NO
wait/synch/mutex/sql/LOCK_gdl	NO	NO
wait/synch/mutex/sql/LOCK_global_index_stats	NO	NO
wait/synch/mutex/sql/LOCK_global_system_variables	NO	NO
wait/synch/mutex/sql/LOCK_global_table_stats	NO	NO
wait/synch/mutex/sql/LOCK_global_user_client_stats	NO	NO
wait/synch/mutex/sql/LOCK_item_func_sleep	NO	NO
wait/synch/mutex/sql/LOCK_load_client_plugin	NO	NO
wait/synch/mutex/sql/LOCK_manager	NO	NO
wait/synch/mutex/sql/LOCK_parallel_entry	NO	NO
wait/synch/mutex/sql/LOCK_plugin	NO	NO
wait/synch/mutex/sql/LOCK_prepared_stmt_count	NO	NO
wait/synch/mutex/sql/LOCK_prepare_ordered	NO	NO
wait/synch/mutex/sql/LOCK_rpl_semi_sync_master_enabled	NO	NO
wait/synch/mutex/sql/LOCK_rpl_status	NO	NO
wait/synch/mutex/sql/LOCK_rpl_thread	NO	NO
wait/synch/mutex/sql/LOCK_rpl_thread_pool	NO	NO
wait/synch/mutex/sql/LOCK_server_started	NO	NO
wait/synch/mutex/sql/LOCK_slave_background	NO	NO
wait/synch/mutex/sql/LOCK_slave_state	NO	NO
wait/synch/mutex/sql/LOCK_start_thread	NO	NO
wait/synch/mutex/sql/LOCK_stats	NO	NO
wait/synch/mutex/sql/LOCK_status	NO	NO
wait/synch/mutex/sql/LOCK_system_variables_hash	NO	NO
wait/synch/mutex/sql/LOCK_table_cache	NO	NO
wait/synch/mutex/sql/LOCK_thread_cache	NO	NO
wait/synch/mutex/sql/LOCK_thread_id	NO	NO
wait/synch/mutex/sql/LOCK_unused_shares	NO	NO
wait/synch/mutex/sql/LOCK_user_conn	NO	NO
wait/synch/mutex/sql/LOCK_uuid_short_generator	NO	NO
wait/synch/mutex/sql/LOCK_wsrep_cluster_config	NO	NO
wait/synch/mutex/sql/LOCK_wsrep_config_state	NO	NO
wait/synch/mutex/sql/LOCK_wsrep_desync	NO	NO
wait/synch/mutex/sql/LOCK_wsrep_donor_monitor	NO	NO
wait/synch/mutex/sql/LOCK_wsrep_group_commit	NO	NO
wait/synch/mutex/sql/LOCK_wsrep_gtid_wait_upto	NO	NO
wait/synch/mutex/sql/LOCK_wsrep_joiner_monitor	NO	NO
wait/synch/mutex/sql/LOCK_wsrep_ready	NO	NO
wait/synch/mutex/sql/LOCK_wsrep_replaying	NO	NO
wait/synch/mutex/sql/LOCK_wsrep_slave_threads	NO	NO
wait/synch/mutex/sql/LOCK_wsrep_SR_pool	NO	NO
wait/synch/mutex/sql/LOCK_wsrep_SR_store	NO	NO
wait/synch/mutex/sql/LOCK_wsrep_sst	NO	NO
wait/synch/mutex/sql/LOCK_wsrep_sst_init	NO	NO
wait/synch/mutex/sql/LOG::LOCK_log	NO	NO
wait/synch/mutex/sql/Master_info::data_lock	NO	NO
wait/synch/mutex/sql/Master_info::run_lock	NO	NO
wait/synch/mutex/sql/Master_info::sleep_lock	NO	NO
wait/synch/mutex/sql/Master_info::start_stop_lock	NO	NO
wait/synch/mutex/sql/MDL_wait::LOCK_wait_status	NO	NO
wait/synch/mutex/sql/MYSQL_BIN_LOG::LOCK_binlog_background_thread	NO	NO
wait/synch/mutex/sql/MYSQL_BIN_LOG::LOCK_binlog_end_pos	NO	NO
wait/synch/mutex/sql/MYSQL_BIN_LOG::LOCK_index	NO	NO
wait/synch/mutex/sql/MYSQL_BIN_LOG::LOCK_xid_list	NO	NO
wait/synch/mutex/sql/MYSQL_RELAY_LOG::LOCK_binlog_end_pos	NO	NO
wait/synch/mutex/sql/MYSQL_RELAY_LOG::LOCK_index	NO	NO
wait/synch/mutex/sql/PAGE::lock	NO	NO
wait/synch/mutex/sql/Query_cache::structure_guard_mutex	NO	NO
wait/synch/mutex/sql/Relay_log_info::data_lock	NO	NO
wait/synch/mutex/sql/Relay_log_info::log_space_lock	NO	NO
wait/synch/mutex/sql/Relay_log_info::run_lock	NO	NO
wait/synch/mutex/sql/Rpl_group_info::sleep_lock	NO	NO
wait/synch/mutex/sql/Slave_reporting_capability::err_lock	NO	NO
wait/synch/mutex/sql/TABLE_SHARE::LOCK_ha_data	NO	NO
wait/synch/mutex/sql/TABLE_SHARE::LOCK_rotation	NO	NO
wait/synch/mutex/sql/TABLE_SHARE::LOCK_share	NO	NO
wait/synch/mutex/sql/TABLE_SHARE::tdc.LOCK_table_share	NO	NO
wait/synch/mutex/sql/TC_LOG_MMAP::LOCK_active	NO	NO
wait/synch/mutex/sql/TC_LOG_MMAP::LOCK_pending_checkpoint	NO	NO

```
| wait/synch/mutex/sql/TC_LOG_MMAP::LOCK_pool                                    | NO       | NO
| wait/synch/mutex/sql/TC_LOG_MMAP::LOCK_sync                                    | NO       | NO
| wait/synch/mutex/sql/THD::LOCK_thd_data                                        | NO       | NO
| wait/synch/mutex/sql/THD::LOCK_thd_kill                                        | NO       | NO
| wait/synch/mutex/sql/THD::LOCK_wakeup_ready                                    | NO       | NO
| wait/synch/mutex/sql/tz_LOCK                                                   | NO       | NO
| wait/synch/mutex/sql/wait_for_commit::LOCK_wait_commit                         | NO       | NO
| wait/synch/mutex/sql/wsrep_sst_thread                                          | NO       | NO
| wait/synch/rwlock/aria/KEYINFO::root_lock                                      | NO       | NO
| wait/synch/rwlock/aria/SHARE::mmap_lock                                        | NO       | NO
| wait/synch/rwlock/aria/TRANSLOG_DESCRIPTOR::open_files_lock                    | NO       | NO
| wait/synch/rwlock/myisam/MYISAM_SHARE::key_root_lock                           | NO       | NO
| wait/synch/rwlock/myisam/MYISAM_SHARE::mmap_lock                               | NO       | NO
| wait/synch/rwlock/mysys/SAFE_HASH::mutex                                       | NO       | NO
| wait/synch/rwlock/proxy_proto/rwlock                                           | NO       | NO
| wait/synch/rwlock/sql/CRYPTO_dynlock_value::lock                               | NO       | NO
| wait/synch/rwlock/sql/LOCK_all_status_vars                                     | NO       | NO
| wait/synch/rwlock/sql/LOCK_dboptions                                           | NO       | NO
| wait/synch/rwlock/sql/LOCK_grant                                               | NO       | NO
| wait/synch/rwlock/sql/LOCK_SEQUENCE                                            | NO       | NO
| wait/synch/rwlock/sql/LOCK_ssl_refresh                                         | NO       | NO
| wait/synch/rwlock/sql/LOCK_system_variables_hash                               | NO       | NO
| wait/synch/rwlock/sql/LOCK_sys_init_connect                                    | NO       | NO
| wait/synch/rwlock/sql/LOCK_sys_init_slave                                      | NO       | NO
| wait/synch/rwlock/sql/LOGGER::LOCK_logger                                      | NO       | NO
| wait/synch/rwlock/sql/MDL_context::LOCK_waiting_for                            | NO       | NO
| wait/synch/rwlock/sql/MDL_lock::rwlock                                         | NO       | NO
| wait/synch/rwlock/sql/Query_cache_query::lock                                  | NO       | NO
| wait/synch/rwlock/sql/TABLE_SHARE::LOCK_stat_serial                            | NO       | NO
| wait/synch/rwlock/sql/THD_list::lock                                           | NO       | NO
| wait/synch/rwlock/sql/THR_LOCK_servers                                         | NO       | NO
| wait/synch/rwlock/sql/THR_LOCK_udf                                             | NO       | NO
| wait/synch/rwlock/sql/Vers_field_stats::lock                                   | NO       | NO
| wait/synch/sxlock/innodb/btr_search_latch                                      | NO       | NO
| wait/synch/sxlock/innodb/dict_operation_lock                                   | NO       | NO
| wait/synch/sxlock/innodb/fil_space_latch                                       | NO       | NO
| wait/synch/sxlock/innodb/fts_cache_init_rw_lock                                | NO       | NO
| wait/synch/sxlock/innodb/fts_cache_rw_lock                                     | NO       | NO
| wait/synch/sxlock/innodb/index_online_log                                      | NO       | NO
| wait/synch/sxlock/innodb/index_tree_rw_lock                                    | NO       | NO
| wait/synch/sxlock/innodb/trx_i_s_cache_lock                                    | NO       | NO
| wait/synch/sxlock/innodb/trx_purge_latch                                       | NO       | NO
+--------------------------------------------------------------------------------+----------+-----
996 rows in set (0.005 sec)
```

1.1.1.2.9.2.1.67 Performance Schema setup_objects Table

Description

The `setup_objects` table determines whether objects are monitored by the performance schema or not. By default limited to 100 rows, this can be changed by setting the performance_schema_setup_objects_size system variable when the server starts.

It contains the following columns:

Column	Description
`OBJECT_TYPE`	Type of object to instrument, currently only . Currently, only `TABLE'` , for base table.
`OBJECT_SCHEMA`	Schema containing the object, either the literal or `%` for any schema.
`OBJECT_NAME`	Name of the instrumented object, either the literal or `%` for any object.
`ENABLED`	Whether the object's events are instrumented or not. Can be disabled, in which case monitoring is not enabled for those objects.
`TIMED`	Whether the object's events are timed or not. Can be modified.

When the Performance Schema looks for matches in the `setup_objects`, there may be more than one row matching, with different `ENABLED` and `TIMED` values. It looks for the most specific matches first, that is, it will first look for the specific database and table name combination, then the specific database, only then falling back to a wildcard for both.

Rows can be added or removed from the table, while for existing rows, only the `TIMED` and `ENABLED` columns can be updated. By default, all tables except those in the `performance_schema`, `information_schema` and `mysql` databases are instrumented.

1.1.1.2.9.2.1.68 Performance Schema setup_timers Table

Description

The `setup_timers` table shows the currently selected event timers.

It contains the following columns:

Column	Description
`NAME`	Type of instrument the timer is used for.
`TIMER_NAME`	Timer applying to the instrument type. Can be modified.

The `TIMER_NAME` value can be changed to choose a different timer, and can be any non-NULL value in the performance_timers.TIMER_NAME column.

If you modify the table, monitoring is immediately affected, and currently monitored events would use a combination of old and new timers, which is probably undesirable. It is best to reset the Performance Schema statistics if you make changes to this table.

Example

```
SELECT * FROM setup_timers;
+-----------+-------------+
| NAME      | TIMER_NAME  |
+-----------+-------------+
| idle      | MICROSECOND |
| wait      | CYCLE       |
| stage     | NANOSECOND  |
| statement | NANOSECOND  |
+-----------+-------------+
```

1.1.1.2.9.2.1.69 Performance Schema socket_instances Table

The `socket_instances` table lists active server connections, with each record being a Unix socket file or TCP/IP connection.

The `socket_instances` table contains the following columns:

Column	Description
`EVENT_NAME`	`NAME` from the `setup_instruments` table, and the name of the `wait/io/socket/*` instrument that produced the event.
`OBJECT_INSTANCE_BEGIN`	Memory address of the object.
`THREAD_ID`	Thread identifier that the server assigns to each socket.
`SOCKET_ID`	The socket's internal file handle.
`IP`	Client IP address. Blank for Unix socket file, otherwise an IPv4 or IPv6 address. Together with the PORT identifies the connection.
`PORT`	TCP/IP port number, from 0 to 65535. Together with the IP identifies the connection.

STATE	Socket status, either IDLE if waiting to receive a request from a client, or ACTIVE

1.1.1.2.9.2.1.70 Performance Schema socket_summary_by_event_name Table

It aggregates timer and byte count statistics for all socket I/O operations by socket instrument.

Column	Description
EVENT_NAME	Socket instrument.
COUNT_STAR	Number of summarized events
SUM_TIMER_WAIT	Total wait time of the summarized events that are timed.
MIN_TIMER_WAIT	Minimum wait time of the summarized events that are timed.
AVG_TIMER_WAIT	Average wait time of the summarized events that are timed.
MAX_TIMER_WAIT	Maximum wait time of the summarized events that are timed.
COUNT_READ	Number of all read operations, including RECV , RECVFROM , and RECVMSG .
SUM_TIMER_READ	Total wait time of all read operations that are timed.
MIN_TIMER_READ	Minimum wait time of all read operations that are timed.
AVG_TIMER_READ	Average wait time of all read operations that are timed.
MAX_TIMER_READ	Maximum wait time of all read operations that are timed.
SUM_NUMBER_OF_BYTES_READ	Bytes read by read operations.
COUNT_WRITE	Number of all write operations, including SEND , SENDTO , and SENDMSG .
SUM_TIMER_WRITE	Total wait time of all write operations that are timed.
MIN_TIMER_WRITE	Minimum wait time of all write operations that are timed.
AVG_TIMER_WRITE	Average wait time of all write operations that are timed.
MAX_TIMER_WRITE	Maximum wait time of all write operations that are timed.
SUM_NUMBER_OF_BYTES_WRITE	Bytes written by write operations.
COUNT_MISC	Number of all miscellaneous operations not counted above, including CONNECT , LISTEN , ACCEPT , CLOSE , and SHUTDOWN .
SUM_TIMER_MISC	Total wait time of all miscellaneous operations that are timed.
MIN_TIMER_MISC	Minimum wait time of all miscellaneous operations that are timed.
AVG_TIMER_MISC	Average wait time of all miscellaneous operations that are timed.
MAX_TIMER_MISC	Maximum wait time of all miscellaneous operations that are timed.

You can TRUNCATE the table, which will reset all counters to zero.

Example

```
SELECT * FROM socket_summary_by_event_name\G
*************************** 1. row ***************************
               EVENT_NAME: wait/io/socket/sql/server_tcpip_socket
               COUNT_STAR: 0
           SUM_TIMER_WAIT: 0
           MIN_TIMER_WAIT: 0
           AVG_TIMER_WAIT: 0
           MAX_TIMER_WAIT: 0
               COUNT_READ: 0
           SUM_TIMER_READ: 0
           MIN_TIMER_READ: 0
           AVG_TIMER_READ: 0
           MAX_TIMER_READ: 0
 SUM_NUMBER_OF_BYTES_READ: 0
              COUNT_WRITE: 0
          SUM_TIMER_WRITE: 0
          MIN_TIMER_WRITE: 0
          AVG_TIMER_WRITE: 0
          MAX_TIMER_WRITE: 0
SUM_NUMBER_OF_BYTES_WRITE: 0
               COUNT_MISC: 0
           SUM_TIMER_MISC: 0
           MIN_TIMER_MISC: 0
           AVG_TIMER_MISC: 0
           MAX_TIMER_MISC: 0
*************************** 2. row ***************************
               EVENT_NAME: wait/io/socket/sql/server_unix_socket
               COUNT_STAR: 0
           SUM_TIMER_WAIT: 0
           MIN_TIMER_WAIT: 0
           AVG_TIMER_WAIT: 0
           MAX_TIMER_WAIT: 0
               COUNT_READ: 0
           SUM_TIMER_READ: 0
           MIN_TIMER_READ: 0
           AVG_TIMER_READ: 0
           MAX_TIMER_READ: 0
 SUM_NUMBER_OF_BYTES_READ: 0
              COUNT_WRITE: 0
          SUM_TIMER_WRITE: 0
          MIN_TIMER_WRITE: 0
          AVG_TIMER_WRITE: 0
          MAX_TIMER_WRITE: 0
SUM_NUMBER_OF_BYTES_WRITE: 0
               COUNT_MISC: 0
           SUM_TIMER_MISC: 0
           MIN_TIMER_MISC: 0
           AVG_TIMER_MISC: 0
           MAX_TIMER_MISC: 0
...
```

1.1.1.2.9.2.1.71 Performance Schema socket_summary_by_instance Table

It aggregates timer and byte count statistics for all socket I/O operations by socket instance.

Column	Description
EVENT_NAME	Socket instrument.
OBJECT_INSTANCE_BEGIN	Address in memory.
COUNT_STAR	Number of summarized events
SUM_TIMER_WAIT	Total wait time of the summarized events that are timed.
MIN_TIMER_WAIT	Minimum wait time of the summarized events that are timed.
AVG_TIMER_WAIT	Average wait time of the summarized events that are timed.

MAX_TIMER_WAIT	Maximum wait time of the summarized events that are timed.
COUNT_READ	Number of all read operations, including RECV, RECVFROM, and RECVMSG.
SUM_TIMER_READ	Total wait time of all read operations that are timed.
MIN_TIMER_READ	Minimum wait time of all read operations that are timed.
AVG_TIMER_READ	Average wait time of all read operations that are timed.
MAX_TIMER_READ	Maximum wait time of all read operations that are timed.
SUM_NUMBER_OF_BYTES_READ	Bytes read by read operations.
COUNT_WRITE	Number of all write operations, including SEND, SENDTO, and SENDMSG.
SUM_TIMER_WRITE	Total wait time of all write operations that are timed.
MIN_TIMER_WRITE	Minimum wait time of all write operations that are timed.
AVG_TIMER_WRITE	Average wait time of all write operations that are timed.
MAX_TIMER_WRITE	Maximum wait time of all write operations that are timed.
SUM_NUMBER_OF_BYTES_WRITE	Bytes written by write operations.
COUNT_MISC	Number of all miscellaneous operations not counted above, including CONNECT, LISTEN, ACCEPT, CLOSE, and SHUTDOWN.
SUM_TIMER_MISC	Total wait time of all miscellaneous operations that are timed.
MIN_TIMER_MISC	Minimum wait time of all miscellaneous operations that are timed.
AVG_TIMER_MISC	Average wait time of all miscellaneous operations that are timed.
MAX_TIMER_MISC	Maximum wait time of all miscellaneous operations that are timed.

The corresponding row in the table is deleted when a connection terminates.

You can TRUNCATE the table, which will reset all counters to zero.

1.1.1.2.9.2.1.72 Performance Schema status_by_account Table

MariaDB starting with 10.5.2

The status_by_account table was added in MariaDB 10.5.2.

The status_by_account table contains status variable information by user/host account. The table does not collect statistics for Com_xxx variables.

The table contains the following columns:

Column	Description
USER	User for which the status variable is reported.
HOST	Host for which the status variable is reported.
VARIABLE_NAME	Status variable name.
VARIABLE_VALUE	Aggregated status variable value

If TRUNCATE TABLE is run, will aggregate the status from terminated sessions to user and host status, then reset the account status.

If FLUSH STATUS is run, session status from all active sessions are added to the global status variables, the status of all active sessions are reset, and values aggregated from disconnected sessions are reset.

1.1.1.2.9.2.1.73 Performance Schema status_by_host Table

MariaDB starting with 10.5.2

The `status_by_host` table was added in MariaDB 10.5.2.

The `status_by_host` table contains status variable information by host. The table does not collect statistics for `Com_xxx` variables.

The table contains the following columns:

Column	Description
HOST	Host for which the status variable is reported.
VARIABLE_NAME	Status variable name.
VARIABLE_VALUE	Aggregated status variable value

If TRUNCATE TABLE is run, will reset the aggregated host status from terminated sessions.

If FLUSH STATUS is run, session status from all active sessions are added to the global status variables, the status of all active sessions are reset, and values aggregated from disconnected sessions are reset.

1.1.1.2.9.2.1.74 Performance Schema status_by_thread Table

MariaDB starting with 10.5.2

The `session_status` table was added in MariaDB 10.5.2.

The `status_by_thread` table contains status variable information about active foreground threads. The table does not collect statistics for `Com_xxx` variables.

The table contains the following columns:

Column	Description
`THREAD_ID`	The thread identifier of the session in which the status variable is defined.
`VARIABLE_NAME`	Status variable name.
`VARIABLE_VALUE`	Aggregated status variable value.

If TRUNCATE TABLE is run, will aggregate the status for all threads to the global status and account status, then reset the thread status. If account statistics are not collected but host and user status are, the session status is added to host and user status.

1.1.1.2.9.2.1.75 Performance Schema status_by_user Table

MariaDB starting with 10.5.2

The `status_by_account` table was added in MariaDB 10.5.2.

The `status_by_account` table contains status variable information by user. The table does not collect statistics for `Com_xxx` variables.

The table contains the following columns:

Column	Description
USER	User for which the status variable is reported.
VARIABLE_NAME	Status variable name.
VARIABLE_VALUE	Aggregated status variable value

If TRUNCATE TABLE is run, will reset the aggregated user status from terminated sessions.

If FLUSH STATUS is run, session status from all active sessions are added to the global status variables, the status of all active sessions are reset, and values aggregated from disconnected sessions are reset.

1.1.1.2.9.2.1.76 Performance Schema table_handles Table

MariaDB starting with 10.5.2

The `table_handles` table was added in MariaDB 10.5.2.

The `table_handles` table contains table lock information. It uses the `wait/lock/table/sql/handler` instrument, which is enabled by default.

Information includes which table handles are open, which sessions are holding the locks, and how they are locked.

The table is read-only, and TRUNCATE TABLE cannot be performed on the table.

The maximum number of opened table objects is determined by the performance_schema_max_table_handles system variable.

The table contains the following columns:

Column	Description
OBJECT_TYPE	The table opened by a table handle.
OBJECT_SCHEMA	The schema that contains the object.
OBJECT_NAME	The name of the instrumented object.
OBJECT_INSTANCE_BEGIN	The table handle address in memory.
OWNER_THREAD_ID	The thread owning the table handle.
OWNER_EVENT_ID	The event which caused the table handle to be opened.
INTERNAL_LOCK	The table lock used at the SQL level.
EXTERNAL_LOCK	The table lock used at the storage engine level.

1.1.1.2.9.2.1.77 Performance Schema table_io_waits_summary_by_index_usage Table

The `table_io_waits_summary_by_index_usage` table records table I/O waits by index.

Column	Description
`OBJECT_TYPE`	`TABLE` in the case of all indexes.
`OBJECT_SCHEMA`	Schema name.
`OBJECT_NAME`	Table name.
`INDEX_NAME`	Index name, or `PRIMARY` for the primary index, `NULL` for no index (inserts are counted in this case).
`COUNT_STAR`	Number of summarized events and the sum of the `x_READ` and `x_WRITE` columns.
`SUM_TIMER_WAIT`	Total wait time of the summarized events that are timed.
`MIN_TIMER_WAIT`	Minimum wait time of the summarized events that are timed.
`AVG_TIMER_WAIT`	Average wait time of the summarized events that are timed.
`MAX_TIMER_WAIT`	Maximum wait time of the summarized events that are timed.
`COUNT_READ`	Number of all read operations, and the sum of the equivalent `x_FETCH` columns.
`SUM_TIMER_READ`	Total wait time of all read operations that are timed.
`MIN_TIMER_READ`	Minimum wait time of all read operations that are timed.
`AVG_TIMER_READ`	Average wait time of all read operations that are timed.
`MAX_TIMER_READ`	Maximum wait time of all read operations that are timed.

COUNT_WRITE	Number of all write operations, and the sum of the equivalent x_INSERT, x_UPDATE and x_DELETE columns.
SUM_TIMER_WRITE	Total wait time of all write operations that are timed.
MIN_TIMER_WRITE	Minimum wait time of all write operations that are timed.
AVG_TIMER_WRITE	Average wait time of all write operations that are timed.
MAX_TIMER_WRITE	Maximum wait time of all write operations that are timed.
COUNT_FETCH	Number of all fetch operations.
SUM_TIMER_FETCH	Total wait time of all fetch operations that are timed.
MIN_TIMER_FETCH	Minimum wait time of all fetch operations that are timed.
AVG_TIMER_FETCH	Average wait time of all fetch operations that are timed.
MAX_TIMER_FETCH	Maximum wait time of all fetch operations that are timed.
COUNT_INSERT	Number of all insert operations.
SUM_TIMER_INSERT	Total wait time of all insert operations that are timed.
MIN_TIMER_INSERT	Minimum wait time of all insert operations that are timed.
AVG_TIMER_INSERT	Average wait time of all insert operations that are timed.
MAX_TIMER_INSERT	Maximum wait time of all insert operations that are timed.
COUNT_UPDATE	Number of all update operations.
SUM_TIMER_UPDATE	Total wait time of all update operations that are timed.
MIN_TIMER_UPDATE	Minimum wait time of all update operations that are timed.
AVG_TIMER_UPDATE	Average wait time of all update operations that are timed.
MAX_TIMER_UPDATE	Maximum wait time of all update operations that are timed.
COUNT_DELETE	Number of all delete operations.
SUM_TIMER_DELETE	Total wait time of all delete operations that are timed.
MIN_TIMER_DELETE	Minimum wait time of all delete operations that are timed.
AVG_TIMER_DELETE	Average wait time of all delete operations that are timed.
MAX_TIMER_DELETE	Maximum wait time of all delete operations that are timed.

You can TRUNCATE the table, which will reset all counters to zero. The table is also truncated if the table_io_waits_summary_by_table table is truncated.

If a table's index structure is changed, index statistics recorded in this table may also be reset.

1.1.1.2.9.2.1.78 Performance Schema table_io_waits_summary_by_table Table

The table_io_waits_summary_by_table table records table I/O waits by table.

Column	Description
OBJECT_TYPE	Since this table records waits by table, always set to TABLE.
OBJECT_SCHEMA	Schema name.
OBJECT_NAME	Table name.
COUNT_STAR	Number of summarized events and the sum of the x_READ and x_WRITE columns.
SUM_TIMER_WAIT	Total wait time of the summarized events that are timed.
MIN_TIMER_WAIT	Minimum wait time of the summarized events that are timed.
AVG_TIMER_WAIT	Average wait time of the summarized events that are timed.
MAX_TIMER_WAIT	Maximum wait time of the summarized events that are timed.

COUNT_READ	Number of all read operations, and the sum of the equivalent x_FETCH columns.
SUM_TIMER_READ	Total wait time of all read operations that are timed.
MIN_TIMER_READ	Minimum wait time of all read operations that are timed.
AVG_TIMER_READ	Average wait time of all read operations that are timed.
MAX_TIMER_READ	Maximum wait time of all read operations that are timed.
COUNT_WRITE	Number of all write operations, and the sum of the equivalent x_INSERT, x_UPDATE and x_DELETE columns.
SUM_TIMER_WRITE	Total wait time of all write operations that are timed.
MIN_TIMER_WRITE	Minimum wait time of all write operations that are timed.
AVG_TIMER_WRITE	Average wait time of all write operations that are timed.
MAX_TIMER_WRITE	Maximum wait time of all write operations that are timed.
COUNT_FETCH	Number of all fetch operations.
SUM_TIMER_FETCH	Total wait time of all fetch operations that are timed.
MIN_TIMER_FETCH	Minimum wait time of all fetch operations that are timed.
AVG_TIMER_FETCH	Average wait time of all fetch operations that are timed.
MAX_TIMER_FETCH	Maximum wait time of all fetch operations that are timed.
COUNT_INSERT	Number of all insert operations.
SUM_TIMER_INSERT	Total wait time of all insert operations that are timed.
MIN_TIMER_INSERT	Minimum wait time of all insert operations that are timed.
AVG_TIMER_INSERT	Average wait time of all insert operations that are timed.
MAX_TIMER_INSERT	Maximum wait time of all insert operations that are timed.
COUNT_UPDATE	Number of all update operations.
SUM_TIMER_UPDATE	Total wait time of all update operations that are timed.
MIN_TIMER_UPDATE	Minimum wait time of all update operations that are timed.
AVG_TIMER_UPDATE	Average wait time of all update operations that are timed.
MAX_TIMER_UPDATE	Maximum wait time of all update operations that are timed.
COUNT_DELETE	Number of all delete operations.
SUM_TIMER_DELETE	Total wait time of all delete operations that are timed.
MIN_TIMER_DELETE	Minimum wait time of all delete operations that are timed.
AVG_TIMER_DELETE	Average wait time of all delete operations that are timed.
MAX_TIMER_DELETE	Maximum wait time of all delete operations that are timed.

You can TRUNCATE the table, which will reset all counters to zero. Truncating this table will also truncate the table_io_waits_summary_by_index_usage table.

1.1.1.2.9.2.1.79 Performance Schema table_lock_waits_summary_by_table Table

The table_lock_waits_summary_by_table table records table lock waits by table.

Column	Description
OBJECT_TYPE	Since this table records waits by table, always set to TABLE.
OBJECT_SCHEMA	Schema name.

OBJECT_NAME	Table name.
COUNT_STAR	Number of summarized events and the sum of the x_READ and x_WRITE columns.
SUM_TIMER_WAIT	Total wait time of the summarized events that are timed.
MIN_TIMER_WAIT	Minimum wait time of the summarized events that are timed.
AVG_TIMER_WAIT	Average wait time of the summarized events that are timed.
MAX_TIMER_WAIT	Maximum wait time of the summarized events that are timed.
COUNT_READ	Number of all read operations, and the sum of the equivalent x_READ_NORMAL, x_READ_WITH_SHARED_LOCKS, x_READ_HIGH_PRIORITY and x_READ_NO_INSERT columns.
SUM_TIMER_READ	Total wait time of all read operations that are timed.
MIN_TIMER_READ	Minimum wait time of all read operations that are timed.
AVG_TIMER_READ	Average wait time of all read operations that are timed.
MAX_TIMER_READ	Maximum wait time of all read operations that are timed.
COUNT_WRITE	Number of all write operations, and the sum of the equivalent x_WRITE_ALLOW_WRITE, x_WRITE_CONCURRENT_INSERT, x_WRITE_DELAYED, x_WRITE_LOW_PRIORITY and x_WRITE_NORMAL columns.
SUM_TIMER_WRITE	Total wait time of all write operations that are timed.
MIN_TIMER_WRITE	Minimum wait time of all write operations that are timed.
AVG_TIMER_WRITE	Average wait time of all write operations that are timed.
MAX_TIMER_WRITE	Maximum wait time of all write operations that are timed.
COUNT_READ_NORMAL	Number of all internal read normal locks.
SUM_TIMER_READ_NORMAL	Total wait time of all internal read normal locks that are timed.
MIN_TIMER_READ_NORMAL	Minimum wait time of all internal read normal locks that are timed.
AVG_TIMER_READ_NORMAL	Average wait time of all internal read normal locks that are timed.
MAX_TIMER_READ_NORMAL	Maximum wait time of all internal read normal locks that are timed.
COUNT_READ_WITH_SHARED_LOCKS	Number of all internal read with shared locks.
SUM_TIMER_READ_WITH_SHARED_LOCKS	Total wait time of all internal read with shared locks that are timed.
MIN_TIMER_READ_WITH_SHARED_LOCKS	Minimum wait time of all internal read with shared locks that are timed.
AVG_TIMER_READ_WITH_SHARED_LOCKS	Average wait time of all internal read with shared locks that are timed.
MAX_TIMER_READ_WITH_SHARED_LOCKS	Maximum wait time of all internal read with shared locks that are timed.
COUNT_READ_HIGH_PRIORITY	Number of all internal read high priority locks.
SUM_TIMER_READ_HIGH_PRIORITY	Total wait time of all internal read high priority locks that are timed.
MIN_TIMER_READ_HIGH_PRIORITY	Minimum wait time of all internal read high priority locks that are timed.
AVG_TIMER_READ_HIGH_PRIORITY	Average wait time of all internal read high priority locks that are timed.
MAX_TIMER_READ_HIGH_PRIORITY	Maximum wait time of all internal read high priority locks that are timed.
COUNT_READ_NO_INSERT	Number of all internal read no insert locks.
SUM_TIMER_READ_NO_INSERT	Total wait time of all internal read no insert locks that are timed.
MIN_TIMER_READ_NO_INSERT	Minimum wait time of all internal read no insert locks that are timed.
AVG_TIMER_READ_NO_INSERT	Average wait time of all internal read no insert locks that are timed.
MAX_TIMER_READ_NO_INSERT	Maximum wait time of all internal read no insert locks that are timed.
COUNT_READ_EXTERNAL	Number of all external read locks.

SUM_TIMER_READ_EXTERNAL	Total wait time of all external read locks that are timed.
MIN_TIMER_READ_EXTERNAL	Minimum wait time of all external read locks that are timed.
AVG_TIMER_READ_EXTERNAL	Average wait time of all external read locks that are timed.
MAX_TIMER_READ_EXTERNAL	Maximum wait time of all external read locks that are timed.
COUNT_WRITE_ALLOW_WRITE	Number of all internal read normal locks.
SUM_TIMER_WRITE_ALLOW_WRITE	Total wait time of all internal write allow write locks that are timed.
MIN_TIMER_WRITE_ALLOW_WRITE	Minimum wait time of all internal write allow write locks that are timed.
AVG_TIMER_WRITE_ALLOW_WRITE	Average wait time of all internal write allow write locks that are timed.
MAX_TIMER_WRITE_ALLOW_WRITE	Maximum wait time of all internal write allow write locks that are timed.
COUNT_WRITE_CONCURRENT_INSERT	Number of all internal concurrent insert write locks.
SUM_TIMER_WRITE_CONCURRENT_INSERT	Total wait time of all internal concurrent insert write locks that are timed.
MIN_TIMER_WRITE_CONCURRENT_INSERT	Minimum wait time of all internal concurrent insert write locks that are timed.
AVG_TIMER_WRITE_CONCURRENT_INSERT	Average wait time of all internal concurrent insert write locks that are timed.
MAX_TIMER_WRITE_CONCURRENT_INSERT	Maximum wait time of all internal concurrent insert write locks that are timed.
COUNT_WRITE_DELAYED	Number of all internal write delayed locks.
SUM_TIMER_WRITE_DELAYED	Total wait time of all internal write delayed locks that are timed.
MIN_TIMER_WRITE_DELAYED	Minimum wait time of all internal write delayed locks that are timed.
AVG_TIMER_WRITE_DELAYED	Average wait time of all internal write delayed locks that are timed.
MAX_TIMER_WRITE_DELAYED	Maximum wait time of all internal write delayed locks that are timed.
COUNT_WRITE_LOW_PRIORITY	Number of all internal write low priority locks.
SUM_TIMER_WRITE_LOW_PRIORITY	Total wait time of all internal write low priority locks that are timed.
MIN_TIMER_WRITE_LOW_PRIORITY	Minimum wait time of all internal write low priority locks that are timed.
AVG_TIMER_WRITE_LOW_PRIORITY	Average wait time of all internal write low priority locks that are timed.
MAX_TIMER_WRITE_LOW_PRIORITY	Maximum wait time of all internal write low priority locks that are timed.
COUNT_WRITE_NORMAL	Number of all internal write normal locks.
SUM_TIMER_WRITE_NORMAL	Total wait time of all internal write normal locks that are timed.
MIN_TIMER_WRITE_NORMAL	Minimum wait time of all internal write normal locks that are timed.
AVG_TIMER_WRITE_NORMAL	Average wait time of all internal write normal locks that are timed.
MAX_TIMER_WRITE_NORMAL	Maximum wait time of all internal write normal locks that are timed.
COUNT_WRITE_EXTERNAL	Number of all external write locks.
SUM_TIMER_WRITE_EXTERNAL	Total wait time of all external write locks that are timed.
MIN_TIMER_WRITE_EXTERNAL	Minimum wait time of all external write locks that are timed.
AVG_TIMER_WRITE_EXTERNAL	Average wait time of all external write locks that are timed.
MAX_TIMER_WRITE_EXTERNAL	Maximum wait time of all external write locks that are timed.

You can TRUNCATE the table, which will reset all counters to zero.

1.1.1.2.9.2.1.80 Performance Schema threads Table

Each server thread is represented as a row in the `threads` table.

The `threads` table contains the following columns:

Column	Description

THREAD_ID	A unique thread identifier.
NAME	Name associated with the server's thread instrumentation code, for example thread/sql/main for the server's main() function, and thread/sql/one_connection for a user connection.
TYPE	FOREGROUND or BACKGROUND, depending on the thread type. User connection threads are FOREGROUND, internal server threads are BACKGROUND.
PROCESSLIST_ID	The PROCESSLIST.ID value for threads displayed in the INFORMATION_SCHEMA.PROCESSLIST table, or 0 for background threads. Also corresponds with the CONNECTION_ID() return value for the thread.
PROCESSLIST_USER	Foreground thread user, or NULL for a background thread.
PROCESSLIST_HOST	Foreground thread host, or NULL for a background thread.
PROCESSLIST_DB	Thread's default database, or NULL if none exists.
PROCESSLIST_COMMAND	Type of command executed by the thread. These correspond to the the COM_xxx client/server protocol commands, and the Com_xxx status variables. See Thread Command Values.
PROCESSLIST_TIME	Time in seconds the thread has been in its current state.
PROCESSLIST_STATE	Action, event or state indicating what the thread is doing.
PROCESSLIST_INFO	Statement being executed by the thread, or NULL if a statement is not being executed. If a statement results in calling other statements, such as for a stored procedure, the innermost statement from the stored procedure is shown here.
PARENT_THREAD_ID	THREAD_ID of the parent thread, if any. Subthreads can for example be spawned as a result of INSERT DELAYED statements.
ROLE	Unused.
INSTRUMENTED	YES or NO for Whether the thread is instrumented or not. For foreground threads, the initial value is determined by whether there's a user/host match in the setup_actors table. Subthreads are again matched, while for background threads, this will be set to YES by default. To monitor events that the thread executes, INSTRUMENTED must be YES and the thread_instrumentation consumer in the setup_consumers table must also be YES.
HISTORY	YES or NO for Whether to log historical events for the thread. For foreground threads, the initial value is determined by whether there's a user/host match in the setup_actors table. Subthreads are again matched, while for background threads, this will be set to YES by default. To monitor events that the thread executes, INSTRUMENTED must be YES and the thread_instrumentation consumer in the setup_consumers table must also be YES. Added in MariaDB 10.5.
CONNECTION_TYPE	The protocol used to establish the connection. One of TCP/IP, SSL/TLS, Socket, Named Pipe, Shared Memory, or NULL for background threads. Added in MariaDB 10.5.
THREAD_OS_ID	The thread or task identifier as defined by the underlying operating system, if there is one. Added in MariaDB 10.5

Example

```
SELECT * FROM performance_schema.threads\G;
*************************** 1. row ***************************
          THREAD_ID: 1
               NAME: thread/sql/main
               TYPE: BACKGROUND
     PROCESSLIST_ID: NULL
   PROCESSLIST_USER: NULL
   PROCESSLIST_HOST: NULL
     PROCESSLIST_DB: NULL
PROCESSLIST_COMMAND: NULL
   PROCESSLIST_TIME: 215859
  PROCESSLIST_STATE: Table lock
   PROCESSLIST_INFO: INTERNAL DDL LOG RECOVER IN PROGRESS
   PARENT_THREAD_ID: NULL
               ROLE: NULL
       INSTRUMENTED: YES
...
*************************** 21. row ***************************
          THREAD_ID: 64
               NAME: thread/sql/one_connection
               TYPE: FOREGROUND
     PROCESSLIST_ID: 44
   PROCESSLIST_USER: root
   PROCESSLIST_HOST: localhost
     PROCESSLIST_DB: NULL
PROCESSLIST_COMMAND: Query
   PROCESSLIST_TIME: 0
  PROCESSLIST_STATE: Sending data
   PROCESSLIST_INFO: SELECT * FROM performance_schema.threads
   PARENT_THREAD_ID: NULL
               ROLE: NULL
       INSTRUMENTED: YES
```

1.1.1.2.9.2.1.81 Performance Schema user_variables_by_thread Table

MariaDB starting with 10.5.2

The `user_variables_by_thread` table was added in MariaDB 10.5.2.

The `user_variables_by_thread` table contains information about user-defined variables and the threads that defined them.

TRUNCATE TABLE cannot be performed on the table.

The table contains the following columns:

Column	Description
THREAD_ID	The thread identifier of the session in which the variable is defined.
VARIABLE_NAME	The variable name, without the leading @ character.
VARIABLE_VALUE	The variable value

Example

```
SET @var = 0;

SELECT * FROM user_variables_by_thread;
+-----------+---------------+----------------+
| THREAD_ID | VARIABLE_NAME | VARIABLE_VALUE |
+-----------+---------------+----------------+
|        11 | var           | 0              |
+-----------+---------------+----------------+
```

1.1.1.2.9.2.1.82 Performance Schema users Table

Description

Each user that connects to the server is stored as a row in the `users` table, along with current and total connections.

The table size is determined at startup by the value of the performance_schema_users_size system variable. If this is set to `0`, user statistics will be disabled.

Column	Description
`USER`	The connection's client user name for the connection, or `NULL` if an internal thread.
`CURRENT_CONNECTIONS`	Current connections for the user.
`TOTAL_CONNECTIONS`	Total connections for the user.

Example

```
SELECT * FROM performance_schema.users;
+------------------+---------------------+-------------------+
| USER             | CURRENT_CONNECTIONS | TOTAL_CONNECTIONS |
+------------------+---------------------+-------------------+
| debian-sys-maint |                   0 |                35 |
| NULL             |                  20 |                23 |
| root             |                   1 |                 2 |
+------------------+---------------------+-------------------+
```

1.1.1.2.9.2.2 Performance Schema Overview

Contents

The Performance Schema is a feature for monitoring server performance.

Introduction

It is implemented as a storage engine, and so will appear in the list of storage engines available.

```
SHOW ENGINES;
+--------------------+---------+----------------------------------+--------------+------+--------
| Engine             | Support | Comment                          | Transactions | XA   | Savepoi
+--------------------+---------+----------------------------------+--------------+------+--------
| ...                |         |                                  |              |      |
| PERFORMANCE_SCHEMA | YES     | Performance Schema               | NO           | NO   | NO
| ...                |         |                                  |              |      |
+--------------------+---------+----------------------------------+--------------+------+--------
```

However, `performance_schema` is not a regular storage engine for storing data, it's a mechanism for implementing the Performance Schema feature.

The storage engine contains a database called `performance_schema`, which in turn consists of a number of tables that can be queried with regular SQL statements, returning specific performance information.

```
USE performance_schema
```

```
SHOW TABLES;
+-----------------------------------------------+
| Tables_in_performance_schema                  |
+-----------------------------------------------+
| accounts                                      |
...
| users                                         |
+-----------------------------------------------+
80 rows in set (0.00 sec)
```

See List of Performance Schema Tables for a full list and links to detailed descriptions of each table. From MariaDB 10.5, there are 80 Performance Schema tables, while until MariaDB 10.4, there are 52.

Activating the Performance Schema

The performance schema is disabled by default for performance reasons. You can check its current status by looking at the value of the performance_schema system variable.

```
SHOW VARIABLES LIKE 'performance_schema';
+--------------------+-------+
| Variable_name      | Value |
+--------------------+-------+
| performance_schema | ON    |
+--------------------+-------+
```

The performance schema cannot be activated at runtime - it must be set when the server starts by adding the following line in your `my.cnf` configuration file.

```
performance_schema=ON
```

Until MariaDB 10.4, all memory used by the Performance Schema is allocated at startup. From MariaDB 10.5, some memory is allocated dynamically, depending on load, number of connections, number of tables open etc.

Enabling the Performance Schema

You need to set up all consumers (starting collection of data) and instrumentations (what to collect):

```
UPDATE performance_schema.setup_consumers SET ENABLED = 'YES';
UPDATE performance_schema.setup_instruments SET ENABLED = 'YES', TIMED = 'YES';
```

You can decide what to enable/disable with `WHERE NAME like "%what_to_enable"` ; You can disable instrumentations by setting `ENABLED` to `"NO"` .

You can also do this in your my.cnf file. The following enables all instrumentation of all stages (computation units) in MariaDB:

```
[mysqld]
performance_schema=ON
performance-schema-instrument='stage/%=ON'
performance-schema-consumer-events-stages-current=ON
performance-schema-consumer-events-stages-history=ON
performance-schema-consumer-events-stages-history-long=ON
```

Listing Performance Schema Variables

```
SHOW VARIABLES LIKE "perf%";
+----------------------------------------------------+-------+
| Variable_name                                      | Value |
+----------------------------------------------------+-------+
| performance_schema                                 | ON    |
...
| performance_schema_users_size                      | 100   |
+----------------------------------------------------+-------+
```

See Performance Schema System Variables for a full list of available system variables.

Note that the "consumer" events are not shown on this list, as they are only available as options, not as system variables, and they can only be enabled at startup.

Column Comments

MariaDB starting with 10.7.1

From MariaDB 10.7.1, comments have been added to table columns in the Performance Schema. These can be viewed with, for example:

```
SELECT column_name, column_comment FROM information_schema.columns
  WHERE table_schema='performance_schema' AND table_name='file_instances';
...
*************************** 2. row ***************************
   column_name: EVENT_NAME
column_comment: Instrument name associated with the file.
*************************** 3. row ***************************
   column_name: OPEN_COUNT
column_comment: Open handles on the file. A value of greater than zero means
                that the file is currently open.
...
```

1.1.1.2.9.2.3 Performance Schema Status Variables

Contents

This page documents status variables related to the Performance Schema. See Server Status Variables for a complete list of status variables that can be viewed with SHOW STATUS.

See also the Full list of MariaDB options, system and status variables.

`Performance_schema_accounts_lost`

- **Description:** Number of times a row could not be added to the performance schema accounts table due to it being full. The global value can be flushed by FLUSH STATUS.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Performance_schema_cond_classes_lost

- **Description:** Number of condition instruments that could not be loaded.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Performance_schema_cond_instances_lost

- **Description:** Number of instances a condition object could not be created. The global value can be flushed by FLUSH STATUS.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Performance_schema_digest_lost

- **Description:** The global value can be flushed by FLUSH STATUS.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Performance_schema_file_classes_lost

- **Description:** Number of file instruments that could not be loaded.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Performance_schema_file_handles_lost

- **Description:** Number of instances a file object could not be opened. The global value can be flushed by FLUSH STATUS.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Performance_schema_file_instances_lost

- **Description:** Number of instances a file object could not be created. The global value can be flushed by FLUSH STATUS.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Performance_schema_hosts_lost

- **Description:** Number of times a row could not be added to the performance schema hosts table due to it being full. The global value can be flushed by FLUSH STATUS.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Performance_schema_index_stat_lost

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.5.2

Performance_schema_locker_lost

- **Description:** Number of events not recorded, due to either being recursive, or having a deeper nested events stack than the implementation limit. The global value can be flushed by FLUSH STATUS.

- **Scope:** Global, Session
- **Data Type:** `numeric`

`Performance_schema_memory_classes_lost`

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.5.2

`Performance_schema_metadata_lock_lost`

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.5.2

`Performance_schema_mutex_classes_lost`

- **Description:** Number of mutual exclusion instruments that could not be loaded.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Performance_schema_mutex_instances_lost`

- **Description:** Number of instances a mutual exclusion object could not be created. The global value can be flushed by FLUSH STATUS.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Performance_schema_nested_statement_lost`

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.5.2

`Performance_schema_prepared_statements_lost`

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.5.2

`Performance_schema_program_lost`

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.5.2

`Performance_schema_rwlock_classes_lost`

- **Description:** Number of read/write lock instruments that could not be loaded.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Performance_schema_rwlock_instances_lost

- **Description:** Number of instances a read/write lock object could not be created. The global value can be flushed by FLUSH STATUS.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Performance_schema_session_connect_attrs_lost

- **Description:** Number of connections for which connection attribute truncation has occurred. The global value can be flushed by FLUSH STATUS.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Performance_schema_socket_classes_lost

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`

Performance_schema_socket_instances_lost

- **Description:** Number of instances a socket object could not be created. The global value can be flushed by FLUSH STATUS.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Performance_schema_stage_classes_lost

- **Description:** Number of stage event instruments that could not be loaded. The global value can be flushed by FLUSH STATUS.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Performance_schema_statement_classes_lost

- **Description:** Number of statement instruments that could not be loaded. The global value can be flushed by FLUSH STATUS.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Performance_schema_table_handles_lost

- **Description:** Number of instances a table object could not be opened. The global value can be flushed by FLUSH STATUS.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Performance_schema_table_instances_lost

- **Description:** Number of instances a table object could not be created. The global value can be flushed by FLUSH STATUS.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Performance_schema_table_lock_stat_lost

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.5.2

`Performance_schema_thread_classes_lost`

- **Description:** Number of thread instruments that could not be loaded.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Performance_schema_thread_instances_lost`

- **Description:** Number of instances thread object could not be created. The global value can be flushed by FLUSH STATUS.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Performance_schema_users_lost`

- **Description:** Number of times a row could not be added to the performance schema users table due to it being full. The global value can be flushed by FLUSH STATUS.
- **Scope:** Global, Session
- **Data Type:** `numeric`

1.1.1.2.9.2.4 Performance Schema System Variables

Contents

The following variables are used with MariaDB's Performance Schema. See Performance Schema Options for Performance Schema options that are not system variables. See Server System Variables for a complete list of system variables and instructions on setting them.

See also the Full list of MariaDB options, system and status variables.

performance_schema

- **Description:** If set to `1` (`0` is default), enables the Performance Schema
- **Commandline:** `--performance-schema=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

performance_schema_accounts_size

- **Description:** Maximum number of rows in the performance_schema.accounts table. If set to 0, the Performance Schema will not store statistics in the accounts table. Use `-1` (the default) for automated sizing.
- **Commandline:** `--performance-schema-accounts-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `-1`
- **Range:** `-1` to `1048576`

---

performance_schema_digests_size

- **Description:** Maximum number of rows that can be stored in the events_statements_summary_by_digest table. `0` for disabling, `-1` (the default) for automated sizing.
- **Commandline:** `--performance-schema-digests-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `-1`
- **Range:** `-1` to `1048576`

---

performance_schema_events_stages_history_long_size

- **Description:** Number of rows in the events_stages_history_long table. `0` for disabling, `-1` (the default) for automated sizing.
- **Commandline:** `--performance-schema-events-stages-history-long-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `-1`
- **Range:** `-1` to `1048576`

---

performance_schema_events_stages_history_size

- **Description:** Number of rows per thread in the events_stages_history table. `0` for disabling, `-1` (the default) for automated sizing.
- **Commandline:** `--performance-schema-events-stages-history-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `-1`
- **Range:** `-1` to `1024`

---

performance_schema_events_statements_history_long_size

- **Description:** Number of rows in the events_statements_history_long table. `0` for disabling, `-1` (the default) for automated sizing.
- **Commandline:** `--performance-schema-events-statements-history-long-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `-1`
- **Range:** `-1` to `1048576`

---

performance_schema_events_statements_history_size

- **Description:** Number of rows per thread in the events_statements_history table. `0` for disabling, `-1` (the default) for automated sizing.
- **Commandline:** `--performance-schema-events-statements-history-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `-1`
- **Range:** `-1` to `1024`

---

performance_schema_events_transactions_history_long_size

- **Description:** Number of rows in events_transactions_history_long table. Use `0` to disable, `-1` for automated sizing.
- **Commandline:** `--performance-schema-events-transactions-history-long-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `-1`
- **Range:** `-1` to `1048576`
- **Introduced:** MariaDB 10.5.2

---

performance_schema_events_transactions_history_size

- **Description:**Number of rows per thread in events_transactions_history. Use 0 to disable, -1 for automated sizing.
- **Commandline:** `--performance-schema-events-transactions-history-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `-1`
- **Range:** `-1` to `1024`
- **Introduced:** MariaDB 10.5.2

---

performance_schema_events_waits_history_long_size

- **Description:** Number of rows contained in the events_waits_history_long table. `0` for disabling, `-1` (the default) for automated sizing.
- **Commandline:** `--performance-schema-events-waits-history-long-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `-1`
- **Range:** `-1` to `1048576`

---

performance_schema_events_waits_history_size

- **Description:** Number of rows per thread contained in the events_waits_history table. `0` for disabling, `-1` (the default) for automated sizing.
- **Commandline:** `--performance-schema-events-waits-history-size=#`
- **Scope:** Global

- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `-1`
- **Range:** `-1` to `1024`

---

performance_schema_hosts_size

- **Description:** Number of rows stored in the hosts table. If set to zero, no connection statistics are kept for the hosts table. `-1` (the default) for automated sizing.
- **Commandline:** `--performance-schema-hosts-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `-1`
- **Range:** `-1` to `1048576`

---

performance_schema_max_cond_classes

- **Description:** Specifies the maximum number of condition instruments.
- **Commandline:** `--performance-schema-max-cond-classes=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `90` (>= MariaDB 10.5.1), `80` (<= MariaDB 10.5.0)
- **Range:** `0` to `256`

---

performance_schema_max_cond_instances

- **Description:** Specifies the maximum number of instrumented condition objects. `0` for disabling, `-1` (the default) for automated sizing.
- **Commandline:** `--performance-schema-max-cond-instances=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `-1`
- **Range:** `-1` to `1048576`

---

performance_schema_max_digest_length

- **Description:** Maximum length considered for digest text, when stored in performance_schema tables.
- **Commandline:** `--performance-schema-max-digest-length=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `1024`
- **Range:** `0` to `1048576`

---

performance_schema_max_file_classes

- **Description:** Specifies the maximum number of file instruments.
- **Commandline:** `--performance-schema-max-file-classes=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `80` (>= MariaDB 10.5.2), `50` (<= MariaDB 10.5.1)
- **Range:** `0` to `256`

---

performance_schema_max_file_handles

- **Description:** Specifies the maximum number of opened file objects. Should always be higher than open_files_limit.
- **Commandline:** `--performance-schema-max-file-handles=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `32768`
- **Range:** `-1` to `32768`

---

performance_schema_max_file_instances

- **Description:** Specifies the maximum number of instrumented file objects. `0` for disabling, `-1` (the default) for automated sizing.
- **Commandline:** `--performance-schema-max-file-instances=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `-1`
- **Range:** `-1` to `1048576`

---

performance_schema_max_index_stat

- **Description:** Maximum number of index statistics for instrumented tables. Use 0 to disable, -1 for automated scaling.
- **Commandline:** `--performance-schema-max-index-stat=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `-1`
- **Range:** `-1` to `1048576`
- **Introduced:** MariaDB 10.5.2

---

performance_schema_max_memory_classes

- **Description:** Maximum number of memory pool instruments.
- **Commandline:** `--performance-schema-max-memory-classes=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `320`
- **Range:** `0` to `1024`
- **Introduced:** MariaDB 10.5.2

---

performance_schema_max_metadata_locks

- **Description:** Maximum number of Performance Schema metadata locks. Use 0 to disable, -1 for automated scaling.
- **Commandline:** `--performance-schema-max-metadata-locks=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `-1`
- **Range:** `-1` to `104857600`
- **Introduced:** MariaDB 10.5.2

---

performance_schema_max_mutex_classes

- **Description:** Specifies the maximum number of mutex instruments.
- **Commandline:** `--performance-schema-max-mutex-classes=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `210` (>= MariaDB 10.5.2), `200` (<= MariaDB 10.5.1)

- **Range:** `0` to `256`

---

performance_schema_max_mutex_instances

- **Description:** Specifies the maximum number of instrumented mutex instances. `0` for disabling, `-1` (the default) for automated sizing.
- **Commandline:** `--performance-schema-max-mutex-instances=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `-1`
- **Range:** `-1` to `104857600`

---

performance_schema_max_prepared_statement_instances

- **Description:** Maximum number of instrumented prepared statements. Use 0 to disable, -1 for automated scaling.
- **Commandline:** `--performance-schema-max-prepared-statement-instances=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `-1`
- **Range:** `-1` to `1048576`
- **Introduced:** MariaDB 10.5.2

---

performance_schema_max_program_instances

- **Description:** Maximum number of instrumented programs. Use 0 to disable, -1 for automated scaling.
- **Commandline:** `--performance-schema-max-program-instances=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `-1`
- **Range:** `-1` to `1048576`
- **Introduced:** MariaDB 10.5.2

---

performance_schema_max_rwlock_classes

- **Description:** Specifies the maximum number of rwlock instruments.
- **Commandline:** `--performance-schema-max-rwlock-classes=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `50` (>= MariaDB 10.5.2), `40` (<= MariaDB 10.5.1)
- **Range:** `0` to `256`

---

performance_schema_max_rwlock_instances

- **Description:** Specifies the maximum number of instrumented rwlock objects. `0` for disabling, `-1` (the default) for automated sizing.
- **Commandline:** `--performance-schema-max-rwlock-instances=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `-1`
- **Range:** `-1` to `104857600`

---

performance_schema_max_socket_classes

- **Description:** Specifies the maximum number of socket instruments.

- **Commandline:** `--performance-schema-max-socket-classes=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `10`
- **Range:** `0` to `256`

---

performance_schema_max_socket_instances

- **Description:** Specifies the maximum number of instrumented socket objects. `0` for disabling, `-1` (the default) for automated sizing.
- **Commandline:** `--performance-schema-max-socket-instances=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `-1`
- **Range:** `-1` to `1048576`

---

performance_schema_max_sql_text_length

- **Description:** Maximum length of displayed sql text.
- **Commandline:** `--performance-schema-max-sql-text-length=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `1024`
- **Range:** `0` to `1048576`
- **Introduced:** MariaDB 10.5.2

---

performance_schema_max_stage_classes

- **Description:** Specifies the maximum number of stage instruments.
- **Commandline:** `--performance-schema-max-stage-classes=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `160`
- **Range:** `0` to `256`

---

performance_schema_max_statement_classes

- **Description:** Specifies the maximum number of statement instruments. Automatically calculated at server build based on the number of available statements. Should be left as either autosized or disabled, as changing to any positive value has no benefit and will most likely allocate unnecessary memory. Setting to zero disables all statement instrumentation, and no memory will be allocated for this purpose.
- **Commandline:** `--performance-schema-max-statement-classes=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** Autosized (see description)
- **Range:** `0` to `256`

---

performance_schema_max_statement_stack

- **Description:** Number of rows per thread in EVENTS_STATEMENTS_CURRENT.
- **Commandline:** `--performance-schema-max-statement-stack=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `10`

- **Range:** `1` to `256`
- **Introduced:** MariaDB 10.5.2

---

performance_schema_max_table_handles

- **Description:** Specifies the maximum number of opened table objects. `0` for disabling, `-1` (the default) for automated sizing. See also the Performance Schema table_handles table.
- **Commandline:** `--performance-schema-max-table-handles=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `-1`
- **Range:** `-1` to `1048576`

---

performance_schema_max_table_instances

- **Description:** Specifies the maximum number of instrumented table objects. `0` for disabling, `-1` (the default) for automated sizing.
- **Commandline:** `--performance-schema-max-table-instances=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `-1`
- **Range:** `-1` to `1048576`

---

performance_schema_max_table_lock_stat

- **Description:** Maximum number of lock statistics for instrumented tables. Use 0 to disable, -1 for automated scaling.
- **Commandline:** `--performance-schema-max-table-lock-stat=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `-1`
- **Range:** `-1` to `1048576`
- **Introduced:** MariaDB 10.5.2

---

performance_schema_max_thread_classes

- **Description:** Specifies the maximum number of thread instruments.
- **Commandline:** `--performance-schema-max-thread-classes=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `50`
- **Range:** `0` to `256`

---

performance_schema_max_thread_instances

- **Description:** Specifies how many of the running server threads (see max_connections and max_delayed_threads) can be instrumented. Should be greater than the sum of max_connections and max_delayed_threads. `0` for disabling, `-1` (the default) for automated sizing.
- **Commandline:** `--performance-schema-max-thread-instances=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `-1`
- **Range:** `-1` to `1048576`

---

performance_schema_session_connect_attrs_size

- **Description:** Per thread preallocated memory for holding connection attribute strings. Incremented if the strings are larger than the reserved space. `0` for disabling, `-1` (the default) for automated sizing.
- **Commandline:** `--performance-schema-session-connect-attrs-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `-1`
- **Range:** `-1` to `1048576`

---

performance_schema_setup_actors_size

- **Description:** The maximum number of rows to store in the performance schema setup_actors table. `-1` (from MariaDB 10.5.2) denotes automated sizing.
- **Commandline:** `--performance-schema-setup-actors-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `-1` (>= MariaDB 10.5.2), `100` (<= MariaDB 10.5.1)
- **Range:** `-1` to `1024` (>= MariaDB 10.5.2), `0` to `1024` (<= MariaDB 10.5.1)

---

performance_schema_setup_objects_size

- **Description:** The maximum number of rows that can be stored in the performance schema setup_objects table. `-1` (from MariaDB 10.5.2) denotes automated sizing.
- **Commandline:** `--performance-schema-setup-objects-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `-1` (>= MariaDB 10.5.2), `100` (<= MariaDB 10.5.1)
- **Range:** `-1` to `1048576` (>= MariaDB 10.5.2), `0` to `1048576` (<= MariaDB 10.5.1)

---

performance_schema_users_size

- **Description:** Number of rows in the performance_schema.users table. If set to 0, the Performance Schema will not store connection statistics in the users table. `-1` (the default) for automated sizing.
- **Commandline:** `--performance-schema-users-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `-1`
- **Range:** `-1` to `1048576`

1.1.1.2.9.2.5 Performance Schema Digests

The Performance Schema digest is a normalized form of a statement, with the specific data values removed. It allows statistics to be gathered for similar kinds of statements.

For example:

```
SELECT * FROM customer WHERE age < 20
SELECT * FROM customer WHERE age < 30
```

With the data values removed, both of these statements normalize to:

```
SELECT * FROM customer WHERE age < ?
```

which is the digest text. The digest text is then MD5 hashed, resulting in a digest. For example:

```
DIGEST_TEXT: SELECT * FROM `performance_schema` . `users`
DIGEST: 0f70cec4015f2a346df4ac0e9475d9f1
```

By contrast, the following two statements would not have the same digest as, while the data values are the same, they call upon different tables.

```
SELECT * FROM customer1 WHERE age < 20
SELECT * FROM customer2 WHERE age < 20
```

The digest text is limited to 1024 bytes. Queries exceeding this limit are truncated with '...', meaning that long queries that would otherwise have different digests may share the same digest.

Digest information is used in a number of performance scheme tables. These include

- events_statements_current
- events_statements_history
- events_statements_history_long
- events_statements_summary_by_digest (a summary table by schema and digest)

1.1.1.2.9.2.6 PERFORMANCE_SCHEMA Storage Engine

If you run SHOW ENGINES, you'll see the following storage engine listed.

```
SHOW ENGINES\G
...
      Engine: PERFORMANCE_SCHEMA
     Support: YES
     Comment: Performance Schema
Transactions: NO
          XA: NO
  Savepoints: NO
...
```

The PERFORMANCE_SCHEMA is not a regular storage engine for storing data, it's a mechanism for implementing the Performance Schema feature.

The SHOW ENGINE PERFORMANCE_SCHEMA STATUS statement is also available, which shows how much memory is used by the tables and internal buffers.

See Performance Schema for more details.

1.1.1.2.9.3 The mysql Database Tables

MariaDB comes pre-installed with a system database called `mysql` containing many important tables storing, in particular, grant and privilege information. Before MariaDB 10.4, system tables used the MyISAM storage engine. From MariaDB 10.4, they use Aria.

mysql.column_stats Table

Column stats for engine-independent statistics.

mysql.columns_priv Table

Column-level privileges

mysql.db Table

Database-level access and privileges.

mysql.event Table

Information about MariaDB events.

mysql.func Table

User-defined function information

mysql.general_log Table

Contents of the general query log if written to table

mysql.global_priv Table

Global privileges.

mysql.gtid_slave_pos Table

For replicas to keep track of the GTID.

mysql.help_category Table

Help categories.

mysql.help_keyword Table

Help keywords.

mysql.help_relation Table

HELP command relations

mysql.help_topic Table

Help topics.

mysql.index_stats Table

Index stats for engine-independent statistics.

mysql.innodb_index_stats

Data related to particular persistent index statistics, multiple rows for each index.

mysql.innodb_table_stats

Data related to persistent indexes, one row per table.

mysql.password_reuse_check_history Table

Contains old passwords for purposes of preventing password reuse.

mysql.plugin Table

Plugins loaded with INSTALL SONAME, INSTALL PLUGIN or the mariadb-plugin utility.

mysql.proc Table

Information about stored routines.

mysql.procs_priv Table

Stored procedure and stored function privileges

mysql.proxies_priv Table

Proxy privileges.

mysql.roles_mapping Table

MariaDB roles information.

mysql.servers Table

MariaDB servers.

mysql.slow_log Table

Contents of the slow query log if written to table.

mysql.tables_priv Table

Table-level privileges

mysql.table_stats Table

Table stats for engine-independent statistics.

mysql.time_zone Table

Time zone table in the mysql database.

mysql.time_zone_leap_second Table

Time zone leap second.

mysql.time_zone_name Table

Time zone name.

mysql.time_zone_transition Table

Time zone transition table.

mysql.time_zone_transition_type Table

Time zone transition type table.

mysql.transaction_registry Table

Used for transaction-precise versioning.

mysql.user Table

User access and global privileges.

Obsolete mysql Database Tables

Tables no longer present in the mysql system database.

Spider mysql Database Tables

System tables related to the Spider storage engine.

There are 2 related questions.

1.1.1.2.9.3.1 mysql.column_stats Table

The `mysql.column_stats` table is one of three tables storing data used for Engine-independent table statistics. The others are mysql.table_stats and mysql.index_stats.

Note that statistics for blob and text columns are not collected. If explicitly specified, a warning is returned.

MariaDB starting with 10.4

In MariaDB 10.4 and later, this table uses the Aria storage engine.

MariaDB until 10.3

In MariaDB 10.3 and before, this table uses the MyISAM storage engine.

The `mysql.column_stats` table contains the following fields:

Field	Type	Null	Key	Default	Description
`db_name`	`varchar(64)`	NO	PRI	`NULL`	Database the table is in.
`table_name`	`varchar(64)`	NO	PRI	`NULL`	Table name.
`column_name`	`varchar(64)`	NO	PRI	`NULL`	Name of the column.
`min_value`	`varchar(255)`	YES		`NULL`	Minimum value in the table (in text form).
`max_value`	`varchar(255)`	YES		`NULL`	Maximum value in the table (in text form).
`nulls_ratio`	`decimal(12,4)`	YES		`NULL`	Fraction of `NULL` values (0- no `NULL`s, 0.5 - half values are `NULL`s, 1 - all values are `NULL`s).
`avg_length`	`decimal(12,4)`	YES		`NULL`	Average length of column value, in bytes. Counted as if one ran `SELECT AVG(LENGTH(col))`. This doesn't count `NULL` bytes, assumes endspace removal for `CHAR(n)`, etc.
`avg_frequency`	`decimal(12,4)`	YES		`NULL`	Average number of records with the same value

hist_size	tinyint(3) unsigned	YES		NULL	Histogram size in bytes, from 0-255, or, from MariaDB 10.7, number of buckets if the histogram type is JSON_HB.
hist_type	enum('SINGLE_PREC_HB', 'DOUBLE_PREC_HB') (>= MariaDB 10.7) enum('SINGLE_PREC_HB', 'DOUBLE_PREC_HB','JSON_HB') (<= MariaDB 10.6)	YES		NULL	Histogram type. See the histogram_type system variable.
histogram	blob (>= MariaDB 10.7) varbinary(255) (<=MariaDB 10.6)	YES		NULL	

It is possible to manually update the table. See Manual updates to statistics tables for details.

1.1.1.2.9.3.2 mysql.columns_priv Table

The mysql.columns_priv table contains information about column-level privileges. The table can be queried and although it is possible to directly update it, it is best to use GRANT for setting privileges.

Note that the MariaDB privileges occur at many levels. A user may be granted a privilege at the column level, but may still not have permission on a table level, for example. See privileges for a more complete view of the MariaDB privilege system.

The INFORMATION_SCHEMA.COLUMN_PRIVILEGES table derives its contents from mysql.columns_priv.

MariaDB starting with 10.4

In MariaDB 10.4 and later, this table uses the Aria storage engine.

MariaDB until 10.3

In MariaDB 10.3 and before, this table uses the MyISAM storage engine.

The mysql.columns_priv table contains the following fields:

Field	Type	Null	Key	Default	Description
Host	char(60)	NO	PRI		Host (together with User, Db, Table_name and Column_name makes up the unique identifier for this record.
Db	char(64)	NO	PRI		Database name (together with User, Host, Table_name and Column_name makes up the unique identifier for this record.
User	char(80)	NO	PRI		User (together with Host, Db, Table_name and Column_name makes up the unique identifier for this record.
Table_name	char(64)	NO	PRI		Table name (together with User, Db, Host and Column_name makes up the unique identifier for this record.
Column_name	char(64)	NO	PRI		Column name (together with User, Db, Table_name and Host makes up the unique identifier for this record.
Timestamp	timestamp	NO		CURRENT_TIMESTAMP	
Column_priv	set('Select', 'Insert', 'Update', 'References')	NO			The privilege type. See Column Privileges for details.

The Acl_column_grants status variable, added in MariaDB 10.1.4, indicates how many rows the mysql.columns_priv table contains.

1.1.1.2.9.3.3 mysql.db Table

The `mysql.db` table contains information about database-level privileges. The table can be queried and although it is possible to directly update it, it is best to use GRANT for setting privileges.

Note that the MariaDB privileges occur at many levels. A user may not be granted a privilege at the database level, but may still have permission on a table level, for example. See privileges for a more complete view of the MariaDB privilege system.

MariaDB starting with 10.4

In MariaDB 10.4 and later, this table uses the Aria storage engine.

MariaDB until 10.3

In MariaDB 10.3 and before, this table uses the MyISAM storage engine.

The `mysql.db` table contains the following fields:

Field	Type	Null	Key	Default	Description	Introduced
`Host`	`char(60)`	NO	PRI		Host (together with `User` and `Db` makes up the unique identifier for this record. Until MariaDB 5.5, if the host field was blank, the corresponding record in the mysql.host table would be examined. From MariaDB 10.0, a blank host field is the same as the `%` wildcard.	
`Db`	`char(64)`	NO	PRI		Database (together with `User` and `Host` makes up the unique identifier for this record.	
`User`	`char(80)`	NO	PRI		User (together with `Host` and `Db` makes up the unique identifier for this record.	
`Select_priv`	`enum('N','Y')`	NO		N	Can perform SELECT statements.	
`Insert_priv`	`enum('N','Y')`	NO		N	Can perform INSERT statements.	
`Update_priv`	`enum('N','Y')`	NO		N	Can perform UPDATE statements.	
`Delete_priv`	`enum('N','Y')`	NO		N	Can perform DELETE statements.	
`Create_priv`	`enum('N','Y')`	NO		N	Can CREATE TABLE's.	
`Drop_priv`	`enum('N','Y')`	NO		N	Can DROP DATABASE's or DROP TABLE's.	
`Grant_priv`	`enum('N','Y')`	NO		N	User can grant privileges they possess.	
`References_priv`	`enum('N','Y')`	NO		N	Unused	
`Index_priv`	`enum('N','Y')`	NO		N	Can create an index on a table using the CREATE INDEX statement. Without the `INDEX` privilege, user can still create indexes when creating a table using the CREATE TABLE statement if the user has have the `CREATE` privilege, and user can create indexes using the ALTER TABLE statement if they have the `ALTER` privilege.	
`Alter_priv`	`enum('N','Y')`	NO		N	Can perform ALTER TABLE statements.	
`Create_tmp_table_priv`	`enum('N','Y')`	NO		N	Can create temporary tables with the CREATE TEMPORARY TABLE statement.	

Lock_tables_priv	enum('N','Y')	NO		N	Acquire explicit locks using the LOCK TABLES statement; user also needs to have the SELECT privilege on a table in order to lock it.	
Create_view_priv	enum('N','Y')	NO		N	Can create a view using the CREATE_VIEW statement.	
Show_view_priv	enum('N','Y')	NO		N	Can show the CREATE VIEW statement to create a view using the SHOW CREATE VIEW statement.	
Create_routine_priv	enum('N','Y')	NO		N	Can create stored programs using the CREATE PROCEDURE and CREATE FUNCTION statements.	
Alter_routine_priv	enum('N','Y')	NO		N	Can change the characteristics of a stored function using the ALTER FUNCTION statement.	
Execute_priv	enum('N','Y')	NO		N	Can execute stored procedure or functions.	
Event_priv	enum('N','Y')	NO		N	Create, drop and alter events.	
Trigger_priv	enum('N','Y')	NO		N	Can execute triggers associated with tables the user updates, execute the CREATE TRIGGER and DROP TRIGGER statements.	
Delete_history_priv	enum('N','Y')	NO		N	Can delete rows created through system versioning.	MariaDB 10.3.5

The Acl_database_grants status variable, added in MariaDB 10.1.4, indicates how many rows the mysql.db table contains.

1.1.1.2.9.3.4 mysql.event Table

The mysql.event table contains information about MariaDB events. Similar information can be obtained by viewing the INFORMATION_SCHEMA.EVENTS table, or with the SHOW EVENTS and SHOW CREATE EVENT statements.

The table is upgraded live, and there is no need to restart the server if the table has changed.

MariaDB starting with 10.4

In MariaDB 10.4 and later, this table uses the Aria storage engine.

MariaDB until 10.3

In MariaDB 10.3 and before, this table uses the MyISAM storage engine.

The mysql.event table contains the following fields:

Field	Type	Null	Key	Default	Description
db	char(64)	NO	PRI		
name	char(64)	NO	PRI		
body	longblob	NO		NULL	
definer	char(141)	NO			
execute_at	datetime	YES		NULL	
interval_value	int(11)	YES		NULL	

interval_field	enum('YEAR', 'QUARTER', 'MONTH', 'DAY', 'HOUR', 'MINUTE', 'WEEK', 'SECOND', 'MICROSECOND', 'YEAR_MONTH', 'DAY_HOUR', 'DAY_MINUTE', 'DAY_SECOND', 'HOUR_MINUTE', 'HOUR_SECOND', 'MINUTE_SECOND', 'DAY_MICROSECOND', 'HOUR_MICROSECOND', 'MINUTE_MICROSECOND', 'SECOND_MICROSECOND')	YES		NULL	
created	timestamp	NO		CURRENT_TIMESTAMP	
modified	timestamp	NO		0000-00-00 00:00:00	
last_executed	datetime	YES		NULL	
starts	datetime	YES		NULL	
ends	datetime	YES		NULL	
status	enum('ENABLED', 'DISABLED', 'SLAVESIDE_DISABLED')	NO		ENABLED	Current status of the event, one of enabled, disabled, or disabled on the slaveside.
on_completion	enum('DROP','PRESERVE')	NO		DROP	
sql_mode	set('REAL_AS_FLOAT', 'PIPES_AS_CONCAT', 'ANSI_QUOTES', 'IGNORE_SPACE', 'IGNORE_BAD_TABLE_OPTIONS', 'ONLY_FULL_GROUP_BY', 'NO_UNSIGNED_SUBTRACTION', 'NO_DIR_IN_CREATE', 'POSTGRESQL', 'ORACLE', 'MSSQL', 'DB2', 'MAXDB', 'NO_KEY_OPTIONS', 'NO_TABLE_OPTIONS', 'NO_FIELD_OPTIONS', 'MYSQL323', 'MYSQL40', 'ANSI', 'NO_AUTO_VALUE_ON_ZERO', 'NO_BACKSLASH_ESCAPES', 'STRICT_TRANS_TABLES', 'STRICT_ALL_TABLES', 'NO_ZERO_IN_DATE', 'NO_ZERO_DATE', 'INVALID_DATES', 'ERROR_FOR_DIVISION_BY_ZERO', 'TRADITIONAL', 'NO_AUTO_CREATE_USER', 'HIGH_NOT_PRECEDENCE', 'NO_ENGINE_SUBSTITUTION', 'PAD_CHAR_TO_FULL_LENGTH')	NO			The SQL_MODE at the time the event was created.
comment	char(64)	NO			
originator	int(10) unsigned	NO		NULL	
time_zone	char(64)	NO		SYSTEM	
character_set_client	char(32)	YES		NULL	
collation_connection	char(32)	YES		NULL	
db_collation	char(32)	YES		NULL	
body_utf8	longblob	YES		NULL	

1.1.1.2.9.3.5 mysql.func Table

The `mysql.func` table stores information about user-defined functions (UDFs) created with the CREATE FUNCTION UDF statement.

MariaDB starting with 10.4
In MariaDB 10.4 and later, this table uses the Aria storage engine.

MariaDB until 10.3
In MariaDB 10.3 and before, this table uses the MyISAM storage engine.

The `mysql.func` table contains the following fields:

Field	Type	Null	Key	Default	Description
`name`	`char(64)`	NO	PRI		UDF name
`ret`	`tinyint(1)`	NO		0	
`dl`	`char(128)`	NO			Shared library name
`type`	`enum('function','aggregate')`	NO		`NULL`	Type, either `function` or `aggregate`. Aggregate functions are summary functions such as SUM() and AVG().

Example

```
SELECT * FROM mysql.func;
+------------------------------+-----+--------------+-----------+
| name                         | ret | dl           | type      |
+------------------------------+-----+--------------+-----------+
| spider_direct_sql            |   2 | ha_spider.so | function  |
| spider_bg_direct_sql         |   2 | ha_spider.so | aggregate |
| spider_ping_table            |   2 | ha_spider.so | function  |
| spider_copy_tables           |   2 | ha_spider.so | function  |
| spider_flush_table_mon_cache |   2 | ha_spider.so | function  |
+------------------------------+-----+--------------+-----------+
```

1.1.1.2.9.3.6 mysql.general_log Table

The `mysql.general_log` table stores the contents of the General Query Log if general logging is active and the output is being written to table (see Writing logs into tables).

It contains the following fields:

Field	Type	Null	Key	Default	Description
`event_time`	`timestamp(6)`	NO		`CURRENT_TIMESTAMP(6)`	Time the query was executed.
`user_host`	`mediumtext`	NO		`NULL`	User and host combination.
`thread_id`	`int(11)`	NO		`NULL`	Thread id.
`server_id`	`int(10) unsigned`	NO		`NULL`	Server id.
`command_type`	`varchar(64)`	NO		`NULL`	Type of command.
`argument`	`mediumtext`	NO		`NULL`	Full query.

Example

```
SELECT * FROM mysql.general_log\G
*************************** 1. row ***************************
  event_time: 2014-11-11 08:40:04.117177
   user_host: root[root] @ localhost []
   thread_id: 74
   server_id: 1
command_type: Query
    argument: SELECT * FROM test.s
*************************** 2. row ***************************
  event_time: 2014-11-11 08:40:10.501131
   user_host: root[root] @ localhost []
   thread_id: 74
   server_id: 1
command_type: Query
    argument: SELECT * FROM mysql.general_log
...
```

1.1.1.2.9.3.7 mysql.global_priv Table

MariaDB starting with 10.4.1

The `mysql.global_priv` table was introduced in MariaDB 10.4.1 to replace the mysql.user table.

The `mysql.global_priv` table contains information about users that have permission to access the MariaDB server, and their global privileges.

Note that the MariaDB privileges occur at many levels. A user may not be granted `create` privilege at the user level, but may still have `create` permission on certain tables or databases, for example. See privileges for a more complete view of the MariaDB privilege system.

The `mysql.global_priv` table contains the following fields:

Field	Type	Null	Key	Default	Description
`Host`	`char(60)`	NO	PRI		Host (together with `User` makes up the unique identifier for this account).
`User`	`char(80)`	NO	PRI		User (together with `Host` makes up the unique identifier for this account).
`Priv`	`longtext`	NO			Global privileges, granted to the account and other account properties

From MariaDB 10.5.2, in order to help the server understand which version a privilege record was written by, the `priv` field contains a new JSON field, `version_id` (MDEV-21704).

Examples

```
select * from mysql.global_priv;
+-----------+-------------+-------------------------------------------------------------------------
-------------------------------------------------------------------+
| Host      | User        | Priv
|
+-----------+-------------+-------------------------------------------------------------------------
-------------------------------------------------------------------+
| localhost | root        | {"access":
18446744073709551615,"plugin":"mysql_native_password","authentication_string":"*6C387FC3893DBA1
E3BA155E74754DA6682D04747"} |
| 127.%     | msandbox    |
{"access":1073740799,"plugin":"mysql_native_password","authentication_string":"*6C387FC3893DBA1
E3BA155E74754DA6682D04747"}            |
| localhost | msandbox    |
{"access":1073740799,"plugin":"mysql_native_password","authentication_string":"*6C387FC3893DBA1
E3BA155E74754DA6682D04747"}            |
| localhost | msandbox_rw |
{"access":487487,"plugin":"mysql_native_password","authentication_string":"*6C387FC3893DBA1E3BA
155E74754DA6682D04747"}                |
| 127.%     | msandbox_rw |
{"access":487487,"plugin":"mysql_native_password","authentication_string":"*6C387FC3893DBA1E3BA
155E74754DA6682D04747"}                |
| 127.%     | msandbox_ro |
{"access":262145,"plugin":"mysql_native_password","authentication_string":"*6C387FC3893DBA1E3BA
155E74754DA6682D04747"}                |
| localhost | msandbox_ro |
{"access":262145,"plugin":"mysql_native_password","authentication_string":"*6C387FC3893DBA1E3BA
155E74754DA6682D04747"}                |
| 127.%     | rsandbox    |
{"access":524288,"plugin":"mysql_native_password","authentication_string":"*B07EB15A2E7BD9620DA
E47B194D5B9DBA14377AD"}                |
+-----------+-------------+-------------------------------------------------------------------------
-------------------------------------------------------------------+
```

Readable format:

```
SELECT CONCAT(user, '@', host, ' => ', JSON_DETAILED(priv)) FROM mysql.global_priv;

+-----------------------------------------------------------------------------------------------+
| CONCAT(user, '@', host, ' => ', JSON_DETAILED(priv))                                          |
+-----------------------------------------------------------------------------------------------+
| root@localhost => {
    "access": 18446744073709551615,
    "plugin": "mysql_native_password",
    "authentication_string": "*6C387FC3893DBA1E3BA155E74754DA6682D04747"
} |
| msandbox@127.% => {
    "access": 1073740799,
    "plugin": "mysql_native_password",
    "authentication_string": "*6C387FC3893DBA1E3BA155E74754DA6682D04747"
}           |
+-----------------------------------------------------------------------------------------------+
```

A particular user:

```
SELECT CONCAT(user, '@', host, ' => ', JSON_DETAILED(priv)) FROM mysql.global_priv
  WHERE user='marijn';
+-----------------------------------------------------------------------------------------------+
| CONCAT(user, '@', host, ' => ', JSON_DETAILED(priv))                                          |
+-----------------------------------------------------------------------------------------------+
| marijn@localhost => {
    "access": 0,
    "plugin": "mysql_native_password",
    "authentication_string": "",
    "account_locked": true,
    "password_last_changed": 1558017158
} |
+-----------------------------------------------------------------------------------------------+
```

From MariaDB 10.5.2:

```
GRANT FILE ON *.* TO user1@localhost;
SELECT Host, User, JSON_DETAILED(Priv) FROM mysql.global_priv WHERE user='user1'\G

*************************** 1. row ***************************
               Host: localhost
               User: user1
JSON_DETAILED(Priv): {
    "access": 512,
    "plugin": "mysql_native_password",
    "authentication_string": "",
    "password_last_changed": 1581070979,
    "version_id": 100502
}
```

1.1.1.2.9.3.8 mysql.gtid_slave_pos Table

The `mysql.gtid_slave_pos` table is used in replication by replica servers to keep track of their current position (the global transaction ID of the last transaction applied). Using the table allows the replica to maintain a consistent value for the gtid_slave_pos system variable across server restarts. See Global Transaction ID.

You should never attempt to modify the table directly. If you do need to change the global gtid_slave_pos value, use `SET GLOBAL gtid_slave_pos = ...` instead.

The table is updated with the new position as part of each transaction committed during replication. This makes it preferable that the table is using the same storage engine as the tables otherwise being modified in the transaction, since otherwise a multi-engine transaction is needed that can reduce performance.

Starting from MariaDB 10.3.1, multiple versions of this table are supported, each using a different storage engine. This is selected with the gtid_pos_auto_engines option, by giving a comma-separated list of engine names. The server will then on-demand create an extra version of the table using the appropriate storage engine, and select the table version using the same engine as the rest of the transaction, avoiding multi-engine transactions.

For example, when `gtid_pos_auto_engines=innodb,rocksdb` , tables `mysql.gtid_slave_pos_InnoDB` and `mysql.gtid_slave_pos_RocksDB` will be created and used, if needed. If there is no match to the storage engine, the default `mysql.gtid_slave_pos` table will be used; this also happens if non-transactional updates (like MyISAM) are replicated, since there is then no active transaction at the time of the `mysql.gtid_slave_pos` table update.

Prior to MariaDB 10.3.1, only the default `mysql.gtid_slave_pos` table is available. In these versions, the table should preferably be using the storage engine that is used for most replicated transactions.

The default `mysql.gtid_slave_pos` table will be initially created using the default storage engine set for the server (which itself defaults to InnoDB). If the application load is primarily non-transactional MyISAM or Aria tables, it can be beneficial to change the storage engine to avoid including an InnoDB update with every operation:

```
ALTER TABLE mysql.gtid_slave_pos ENGINE=MyISAM;
```

The `mysql.gtid_slave_pos` table should not be changed manually in any other way. From MariaDB 10.3.1, it is preferable to use the `gtid_pos_auto_engines` server variable to get the GTID position updates to use the TokuDB or RocksDB storage engine.

Note that for scalability reasons, the automatic creation of a new `mysql.gtid_slave_posXXX` table happens asynchronously when the first transaction with the new storage engine is committed. So the very first few transactions will update the old version of the table, until the new version is created and available.

The table `mysql.gtid_slave_pos` contains the following fields

Field	Type	Null	Key	Default	Description
`domain_id`	`int(10) unsigned`	NO	PRI	`NULL`	Domain id (see Global Transaction ID domain ID.
`sub_id`	`bigint(20) unsigned`	NO	PRI	`NULL`	This field enables multiple parallel transactions within same `domain_id` to update this table without contention. At any instant, the replication state corresponds to records with largest `sub_id` for each `domain_id` .
`server_id`	`int(10) unsigned`	NO		`NULL`	Server id.

seq_no	bigint(20) unsigned	NO		NULL	Sequence number, an integer that is monotonically increasing for each new event group logged into the binlog.

From MariaDB 10.3.1, some status variables are available to monitor the use of the different gtid_slave_pos table versions:

Transactions_gtid_foreign_engine

Number of replicated transactions where the update of the gtid_slave_pos table had to choose a storage engine that did not otherwise participate in the transaction. This can indicate that setting gtid_pos_auto_engines might be useful.

Rpl_transactions_multi_engine

Number of replicated transactions that involved changes in multiple (transactional) storage engines, before considering the update of gtid_slave_pos . These are transactions that were already cross-engine, independent of the GTID position update introduced by replication

Transactions_multi_engine

Number of transactions that changed data in multiple (transactional) storage engines. If this is significantly larger than Rpl_transactions_multi_engine, it indicates that setting gtid_pos_auto_engines could reduce the need for cross-engine transactions.

1.1.1.2.9.3.9 mysql.help_category Table

mysql.help_category is one of the four tables used by the HELP command. It is populated when the server is installed by the fill_help_tables.sql script. The other help tables are help_relation, help_topic and help_keyword.

This table uses the Aria storage engine. Prior to MariaDB 10.4 it used the MyISAM engine.

The mysql.help_category table contains the following fields:

Field	Type	Null	Key	Default	Description
help_category_id	smallint(5) unsigned	NO	PRI	NULL	
name	char(64)	NO	UNI	NULL	
parent_category_id	smallint(5) unsigned	YES		NULL	
url	char(128)	NO		NULL	

Example

```
SELECT * FROM help_category;
+------------------+----------------------------------------------+--------------------+-----+
| help_category_id | name                                         | parent_category_id | url |
+------------------+----------------------------------------------+--------------------+-----+
|                1 | Geographic                                   |                  0 |     |
|                2 | Polygon properties                           |                 34 |     |
|                3 | WKT                                          |                 34 |     |
|                4 | Numeric Functions                            |                 38 |     |
|                5 | Plugins                                      |                 35 |     |
|                6 | MBR                                          |                 34 |     |
|                7 | Control flow functions                       |                 38 |     |
|                8 | Transactions                                 |                 35 |     |
|                9 | Help Metadata                                |                 35 |     |
|               10 | Account Management                           |                 35 |     |
|               11 | Point properties                             |                 34 |     |
|               12 | Encryption Functions                         |                 38 |     |
|               13 | LineString properties                        |                 34 |     |
|               14 | Miscellaneous Functions                      |                 38 |     |
|               15 | Logical operators                            |                 38 |     |
|               16 | Functions and Modifiers for Use with GROUP BY |                 35 |     |
|               17 | Information Functions                        |                 38 |     |
|               18 | Comparison operators                         |                 38 |     |
|               19 | Bit Functions                                |                 38 |     |
|               20 | Table Maintenance                            |                 35 |     |
|               21 | User-Defined Functions                       |                 35 |     |
|               22 | Data Types                                   |                 35 |     |
|               23 | Compound Statements                          |                 35 |     |
|               24 | Geometry constructors                        |                 34 |     |
|               25 | GeometryCollection properties                |                  1 |     |
|               26 | Administration                               |                 35 |     |
|               27 | Data Manipulation                            |                 35 |     |
|               28 | Utility                                      |                 35 |     |
|               29 | Language Structure                           |                 35 |     |
|               30 | Geometry relations                           |                 34 |     |
|               31 | Date and Time Functions                      |                 38 |     |
|               32 | WKB                                          |                 34 |     |
|               33 | Procedures                                   |                 35 |     |
|               34 | Geographic Features                          |                 35 |     |
|               35 | Contents                                     |                  0 |     |
|               36 | Geometry properties                          |                 34 |     |
|               37 | String Functions                             |                 38 |     |
|               38 | Functions                                    |                 35 |     |
|               39 | Data Definition                              |                 35 |     |
+------------------+----------------------------------------------+--------------------+-----+
```

1.1.1.2.9.3.10 mysql.help_keyword Table

`mysql.help_keyword` is one of the four tables used by the HELP command. It is populated when the server is installed by the `fill_help_tables.sql` script. The other help tables are help_relation, help_category and help_topic.

This table uses the Aria storage engine. Prior to MariaDB 10.4 it used the MyISAM engine.

The `mysql.help_keyword` table contains the following fields:

Field	Type	Null	Key	Default	Description
`help_keyword_id`	`int(10) unsigned`	NO	PRI	`NULL`	
`name`	`char(64)`	NO	UNI	`NULL`	

Example

```
SELECT * FROM help_keyword;
+-----------------+---------------------------------+
| help_keyword_id | name                            |
+-----------------+---------------------------------+
|               0 | JOIN                            |
|               1 | HOST                            |
|               2 | REPEAT                          |
|               3 | SERIALIZABLE                    |
|               4 | REPLACE                         |
|               5 | AT                              |
|               6 | SCHEDULE                        |
|               7 | RETURNS                         |
|               8 | STARTS                          |
|               9 | MASTER_SSL_CA                   |
|              10 | NCHAR                           |
|              11 | COLUMNS                         |
|              12 | COMPLETION                      |
...
```

1.1.1.2.9.3.11 mysql.help_relation Table

`mysql.help_relation` is one of the four tables used by the HELP command. It is populated when the server is installed by the `fill_help_tables.sql` script. The other help tables are help_topic, help_category and help_keyword.

This table uses the Aria storage engine. Prior to MariaDB 10.4 it used the MyISAM engine.

The `mysql.help_relation` table contains the following fields:

Field	Type	Null	Key	Default	Description
help_topic_id	int(10) unsigned	NO	PRI	NULL	
help_keyword_id	int(10) unsigned	NO	PRI	NULL	

Example

```
...
|           106 |             456 |
|           463 |             456 |
|           468 |             456 |
|           463 |             457 |
|           194 |             458 |
|           478 |             458 |
|           374 |             459 |
|           459 |             459 |
|            39 |             460 |
|            58 |             460 |
|           185 |             460 |
|           264 |             460 |
|           269 |             460 |
|           209 |             461 |
|           468 |             461 |
|           201 |             462 |
|           468 |             463 |
+---------------+-----------------+
```

1.1.1.2.9.3.12 mysql.help_topic Table

`mysql.help_topic` is one of the four tables used by the HELP command. It is populated when the server is installed by the `fill_help_tables.sql` script. The other help tables are help_relation, help_category and help_keyword.

This table uses the Aria storage engine. Prior to MariaDB 10.4 it used the MyISAM engine.

The `mysql.help_topic` table contains the following fields:

Field	Type	Null	Key	Default	Description

help_topic_id	int(10) unsigned	NO	PRI	NULL	
name	char(64)	NO	UNI	NULL	
help_category_id	smallint(5) unsigned	NO		NULL	
description	text	NO		NULL	
example	text	NO		NULL	
url	char(128)	NO		NULL	

Example

```
SELECT * FROM help_topic\G;
...
*************************** 704. row ***************************
   help_topic_id: 692
           name: JSON_DEPTH
help_category_id: 41
     description: JSON functions were added in MariaDB 10.2.3.

Syntax
------
JSON_DEPTH(json_doc)

Description
-----------
Returns the maximum depth of the given JSON document, or
NULL if the argument is null. An error will occur if the
argument is an invalid JSON document.
Scalar values or empty arrays or objects have a depth of 1.
Arrays or objects that are not empty but contain only
elements or member values of depth 1 will have a depth of 2.
In other cases, the depth will be greater than 2.

Examples
--------
SELECT JSON_DEPTH('[]'), JSON_DEPTH('true'),
JSON_DEPTH('{}');
+------------------+--------------------+------------------+
| JSON_DEPTH('[]') | JSON_DEPTH('true') |
JSON_DEPTH('{}') |
+------------------+--------------------+------------------+
| 1 | 1 | 1 |
+------------------+--------------------+------------------+

SELECT JSON_DEPTH('[1, 2, 3]'), JSON_DEPTH('[[], {},
[]]');
+-------------------------+-----------------------------+
| JSON_DEPTH('[1, 2, 3]') | JSON_DEPTH('[[], {}, []]') |
+-------------------------+-----------------------------+
| 2 | 2 |
+-------------------------+-----------------------------+

SELECT JSON_DEPTH('[1, 2, [3, 4, 5, 6], 7]');
+---------------------------------------+
| JSON_DEPTH('[1, 2, [3, 4, 5, 6], 7]') |
+---------------------------------------+
| 3 |
+---------------------------------------+

URL: https://mariadb.com/kb/en/json_depth/
         example:
             url: https://mariadb.com/kb/en/json_depth/
```

1.1.1.2.9.3.13 mysql.index_stats Table

The `mysql.index_stats` table is one of three tables storing data used for Engine-independent table statistics. The others are mysql.column_stats and mysql.table_stats.

MariaDB starting with 10.4

In MariaDB 10.4 and later, this table uses the Aria storage engine.

MariaDB until 10.3

In MariaDB 10.3 and before, this table uses the MyISAM storage engine.

The `mysql.index_stats` table contains the following fields:

Field	Type	Null	Key	Default	Description
`db_name`	`varchar(64)`	NO	PRI	`NULL`	Database the table is in.
`table_name`	`varchar(64)`	NO	PRI	`NULL`	Table name
`index_name`	`varchar(64)`	NO	PRI	`NULL`	Name of the index
`prefix_arity`	`int(11) unsigned`	NO	PRI	`NULL`	Index prefix length. 1 for the first keypart, 2 for the first two, and so on. InnoDB's extended keys are supported.
`avg_frequency`	`decimal(12,4)`	YES		`NULL`	Average number of records one will find for given values of (keypart1, keypart2, ..), provided the values will be found in the table.

It is possible to manually update the table. See Manual updates to statistics tables for details.

1.1.1.2.9.3.14 mysql.innodb_index_stats

Contents

1. Example

The `mysql.innodb_index_stats` table stores data related to particular InnoDB Persistent Statistics, and contains multiple rows for each index.

This table, along with the related mysql.innodb_table_stats table, can be manually updated in order to force or test differing query optimization plans. After updating, `FLUSH TABLE innodb_index_stats` is required to load the changes.

`mysql.innodb_index_stats` is not replicated, although any ANALYZE TABLE statements on the table will be by default..

It contains the following fields:

Field	Type	Null	Key	Default	Description
`database_name`	`varchar(64)`	NO	PRI	`NULL`	Database name.
`table_name`	`varchar(64)`	NO	PRI	`NULL`	Table, partition or subpartition name.
`index_name`	`varchar(64)`	NO	PRI	`NULL`	Index name.
`last_update`	`timestamp`	NO		`current_timestamp()`	Time that this row was last updated.
`stat_name`	`varchar(64)`	NO	PRI	`NULL`	Statistic name.
`stat_value`	`bigint(20) unsigned`	NO		`NULL`	Estimated statistic value.
`sample_size`	`bigint(20) unsigned`	YES		`NULL`	Number of pages sampled for the estimated statistic value.
`stat_description`	`varchar(1024)`	NO		`NULL`	Statistic description.

Example

```
SELECT * FROM mysql.innodb_index_stats\G
*************************** 1. row ***************************
   database_name: mysql
      table_name: gtid_slave_pos
      index_name: PRIMARY
     last_update: 2017-08-19 20:38:34
       stat_name: n_diff_pfx01
      stat_value: 0
     sample_size: 1
stat_description: domain_id
*************************** 2. row ***************************
   database_name: mysql
      table_name: gtid_slave_pos
      index_name: PRIMARY
     last_update: 2017-08-19 20:38:34
       stat_name: n_diff_pfx02
      stat_value: 0
     sample_size: 1
stat_description: domain_id,sub_id
*************************** 3. row ***************************
   database_name: mysql
      table_name: gtid_slave_pos
      index_name: PRIMARY
     last_update: 2017-08-19 20:38:34
       stat_name: n_leaf_pages
      stat_value: 1
     sample_size: NULL
stat_description: Number of leaf pages in the index
*************************** 4. row ***************************
   database_name: mysql
      table_name: gtid_slave_pos
      index_name: PRIMARY
     last_update: 2017-08-19 20:38:34
       stat_name: size
      stat_value: 1
     sample_size: NULL
stat_description: Number of pages in the index
*************************** 5. row ***************************
   database_name: test
      table_name: ft
      index_name: FTS_DOC_ID_INDEX
     last_update: 2017-09-15 12:58:39
       stat_name: n_diff_pfx01
      stat_value: 0
     sample_size: 1
stat_description: FTS_DOC_ID
*************************** 6. row ***************************
   database_name: test
      table_name: ft
      index_name: FTS_DOC_ID_INDEX
     last_update: 2017-09-15 12:58:39
       stat_name: n_leaf_pages
      stat_value: 1
     sample_size: NULL
stat_description: Number of leaf pages in the index
...
```

1.1.1.2.9.3.15 mysql.innodb_table_stats

Contents

The `mysql.innodb_table_stats` table stores data related to InnoDB Persistent Statistics, and contains one row per table.

This table, along with the related mysql.innodb_index_stats table, can be manually updated in order to force or test differing query optimization plans. After updating, `FLUSH TABLE innodb_table_stats` is required to load the changes.

`mysql.innodb_table_stats` is not replicated, although any ANALYZE TABLE statements on the table will be by default..

It contains the following fields:

Field	Type	Null	Key	Default	Description
database_name	varchar(64)	NO	PRI	NULL	Database name.
table_name	varchar(64)	NO	PRI	NULL	Table, partition or subpartition name.
last_update	timestamp	NO		current_timestamp()	Time that this row was last updated.
n_rows	bigint(20) unsigned	NO		NULL	Number of rows in the table.
clustered_index_size	bigint(20) unsigned	NO		NULL	Size, in pages, of the primary index.
sum_of_other_index_sizes	bigint(20) unsigned	NO		NULL	Size, in pages, of non-primary indexes.

Example

```
SELECT * FROM mysql.innodb_table_stats\G
*************************** 1. row ***************************
           database_name: mysql
              table_name: gtid_slave_pos
             last_update: 2017-08-19 20:38:34
                 n_rows: 0
    clustered_index_size: 1
sum_of_other_index_sizes: 0
*************************** 2. row ***************************
           database_name: test
              table_name: ft
             last_update: 2017-09-15 12:58:39
                 n_rows: 0
    clustered_index_size: 1
sum_of_other_index_sizes: 2
...
```

1.1.1.2.9.3.16 mysql.password_reuse_check_history Table

MariaDB starting with 10.7.0

The mysql.password_reuse_check_history Table is installed as part of the password_reuse_check plugin, available from MariaDB 10.7.0.

The `mysql.password_reuse_check_history` table stores old passwords, so that when a user sets a new password, it can be checked for purposes of preventing password reuse.

It contains the following fields:

Field	Type	Null	Key	Default	Description
hash	binary(64)	NO	PRI	NULL	
time	timestamp	NO	MUL	current_timestamp()	

1.1.1.2.9.3.17 mysql.plugin Table

The `mysql.plugin` table can be queried to get information about installed plugins.

This table only contains information about plugins that have been installed via the following methods:

- The INSTALL SONAME statement.
- The INSTALL PLUGIN statement.
- The mariadb-plugin utility.

This table does not contain information about:

- Built-in plugins.
- Plugins loaded with the --plugin-load-add option.
- Plugins loaded with the --plugin-load option.

This table only contains enough information to reload the plugin when the server is restarted, which means it only contains the plugin name and the plugin library.

This table uses the Aria storage engine.

The `mysql.plugin` table contains the following fields:

Field	Type	Null	Key	Default	Description
`name`	`varchar(64)`	NO	PRI		Plugin name.
`dl`	`varchar(128)`	NO			Name of the plugin library.

Example

```
SELECT * FROM mysql.plugin;
+---------------------------+------------------------+
| name                      | dl                     |
+---------------------------+------------------------+
| spider                    | ha_spider.so           |
| spider_alloc_mem          | ha_spider.so           |
| METADATA_LOCK_INFO        | metadata_lock_info.so  |
| OQGRAPH                   | ha_oqgraph.so          |
| cassandra                 | ha_cassandra.so        |
| QUERY_RESPONSE_TIME       | query_response_time.so |
| QUERY_RESPONSE_TIME_AUDIT | query_response_time.so |
| LOCALES                   | locales.so             |
| sequence                  | ha_sequence.so         |
+---------------------------+------------------------+
```

1.1.1.2.9.3.18 mysql.proc Table

The `mysql.proc` table contains information about stored procedures and stored functions. It contains similar information to that stored in the INFORMATION SCHEMA.ROUTINES table.

MariaDB starting with 10.4

In MariaDB 10.4 and later, this table uses the Aria storage engine.

MariaDB until 10.3

In MariaDB 10.3 and before, this table uses the MyISAM storage engine.

The `mysql.proc` table contains the following fields:

Field	Type	Null	Key	Default	Description
`db`	`char(64)`	NO	PRI		Database name.
`name`	`char(64)`	NO	PRI		Routine name.
`type`	`enum('FUNCTION','PROCEDURE','PACKAGE', 'PACKAGE BODY')`	NO	PRI	`NULL`	Whether stored procedure, stored function or, from MariaDB 10.3.5, a package or package body.
`specific_name`	`char(64)`	NO			
`language`	`enum('SQL')`	NO		SQL	Always `SQL`.
`sql_data_access`	`enum('CONTAINS_SQL', 'NO_SQL', 'READS_SQL_DATA', 'MODIFIES_SQL_DATA')`	NO		`CONTAINS_SQL`	
`is_deterministic`	`enum('YES','NO')`	NO		NO	Whether the routine is deterministic (can produce only one result for a given list of parameters) or not.

security_type	enum('INVOKER','DEFINER')	NO		DEFINER	INVOKER or DEFINER. Indicates which user's privileges apply to this routine.
param_list	blob	NO		NULL	List of parameters.
returns	longblob	NO		NULL	What the routine returns.
body	longblob	NO		NULL	Definition of the routine.
definer	char(141)	NO			If the security_type is DEFINER, this value indicates which user defined this routine.
created	timestamp	NO		CURRENT_TIMESTAMP	Date and time the routine was created.
modified	timestamp	NO		0000-00-00 00:00:00	Date and time the routine was modified.
sql_mode	set('REAL_AS_FLOAT', 'PIPES_AS_CONCAT', 'ANSI_QUOTES', 'IGNORE_SPACE', 'IGNORE_BAD_TABLE_OPTIONS', 'ONLY_FULL_GROUP_BY', 'NO_UNSIGNED_SUBTRACTION', 'NO_DIR_IN_CREATE', 'POSTGRESQL', 'ORACLE', 'MSSQL', 'DB2', 'MAXDB', 'NO_KEY_OPTIONS', 'NO_TABLE_OPTIONS', 'NO_FIELD_OPTIONS', 'MYSQL323', 'MYSQL40', 'ANSI', 'NO_AUTO_VALUE_ON_ZERO', 'NO_BACKSLASH_ESCAPES', 'STRICT_TRANS_TABLES', 'STRICT_ALL_TABLES', 'NO_ZERO_IN_DATE', 'NO_ZERO_DATE', 'INVALID_DATES', 'ERROR_FOR_DIVISION_BY_ZERO', 'TRADITIONAL', 'NO_AUTO_CREATE_USER', 'HIGH_NOT_PRECEDENCE', 'NO_ENGINE_SUBSTITUTION', 'PAD_CHAR_TO_FULL_LENGTH', 'EMPTY_STRING_IS_NULL', 'SIMULTANEOUS_ASSIGNMENT')	NO			The SQL_MODE at the time the routine was created.
comment	text	NO		NULL	Comment associated with the routine.
character_set_client	char(32)	YES		NULL	The character set used by the client that created the routine.
collation_connection	char(32)	YES		NULL	The collation (and character set) used by the connection that created the routine.
db_collation	char(32)	YES		NULL	The default collation (and character set) for the database, at the time the routine was created.
body_utf8	longblob	YES		NULL	Definition of the routine in utf8.
aggregate	enum('NONE', 'GROUP')	NO		NONE	From MariaDB 10.3.3
Field	**Type**	**Null**	**Key**	**Default**	**Description**

1.1.1.2.9.3.19 mysql.procs_priv Table

The `mysql.procs_priv` table contains information about stored procedure and stored function privileges. See CREATE PROCEDURE and CREATE FUNCTION on creating these.

The INFORMATION_SCHEMA.ROUTINES table derives its contents from `mysql.procs_priv`.

MariaDB starting with 10.4

In MariaDB 10.4 and later, this table uses the Aria storage engine.

MariaDB until 10.3

In MariaDB 10.3 and before, this table uses the MyISAM storage engine.

The `mysql.procs_priv` table contains the following fields:

Field	Type	Null	Key	Default	Description
`Host`	`char(60)`	NO	PRI		Host (together with `Db`, `User`, `Routine_name` and `Routine_type` makes up the unique identifier for this record).
`Db`	`char(64)`	NO	PRI		Database (together with `Host`, `User`, `Routine_name` and `Routine_type` makes up the unique identifier for this record).
`User`	`char(80)`	NO	PRI		User (together with `Host`, `Db`, `Routine_name` and `Routine_type` makes up the unique identifier for this record).
`Routine_name`	`char(64)`	NO	PRI		Routine_name (together with `Host`, `Db` `User` and `Routine_type` makes up the unique identifier for this record).
`Routine_type`	`enum('FUNCTION','PROCEDURE', 'PACKAGE', 'PACKAGE BODY')`	NO	PRI	`NULL`	Whether the routine is a stored procedure, stored function, or, from MariaDB 10.3.5, a package or package body.
`Grantor`	`char(141)`	NO	MUL		
`Proc_priv`	`set('Execute','Alter Routine','Grant')`	NO			The routine privilege. See Function Privileges and Procedure Privileges for details.
`Timestamp`	`timestamp`	NO		`CURRENT_TIMESTAMP`	

The Acl_function_grants status variable, added in MariaDB 10.1.4, indicates how many rows the `mysql.columns_priv` table contains with the `FUNCTION` routine type.

The Acl_procedure_grants status variable, added in MariaDB 10.1.4, indicates how many rows the `mysql.columns_priv` table contains with the `PROCEDURE` routine type.

1.1.1.2.9.3.20 mysql.roles_mapping Table

The `mysql.roles_mapping` table contains information about mariaDB roles.

MariaDB starting with 10.4

In MariaDB 10.4 and later, this table uses the Aria storage engine.

MariaDB until 10.3

In MariaDB 10.3 and before, this table uses the MyISAM storage engine.

The `mysql.roles_mapping` table contains the following fields:

Field	Type	Null	Key	Default	Description
`Host`	`char(60)`	NO	PRI		Host (together with `User` and `Role` makes up the unique identifier for this record.

`User`	`char(80)`	NO	PRI		User (together with `Host` and `Role` makes up the unique identifier for this record.
`Role`	`char(80)`	NO	PRI		Role (together with `Host` and `User` makes up the unique identifier for this record.
`Admin_option`	`enum('N','Y')`	NO		N	Whether the role can be granted (see the CREATE ROLE `WITH ADMIN` clause).

The Acl_role_grants status variable, added in MariaDB 10.1.4, indicates how many rows the `mysql.roles_mapping` table contains.

1.1.1.2.9.3.21 mysql.servers Table

The `mysql.servers` table contains information about servers as used by the Spider, FEDERATED or FederatedX, Connect storage engines (see CREATE SERVER).

This table uses the Aria storage engine (until MariaDB 10.3, it used the MyISAM storage engine).

The `mysql.servers` table contains the following fields:

Field	Type	Null	Key	Default	Description
`Server_name`	`char(64)`	NO	PRI		
`Host`	`char(64)`	NO			
`Db`	`char(64)`	NO			
`Username`	`char(80)`	NO			
`Password`	`char(64)`	NO			
`Port`	`int(4)`	NO		0	
`Socket`	`char(64)`	NO			
`Wrapper`	`char(64)`	NO			`mysql` or `mariadb`
`Owner`	`char(64)`	NO			

Example

```
SELECT * FROM mysql.servers\G
*************************** 1. row ***************************
Server_name: s
       Host: 192.168.1.106
         Db: test
   Username: Remote
   Password:
       Port: 0
     Socket:
    Wrapper: mariadb
      Owner:
```

1.1.1.2.9.3.22 mysql.slow_log Table

The `mysql.slow_log` table stores the contents of the Slow Query Log if slow logging is active and the output is being written to table (see Writing logs into tables).

It contains the following fields:

Field	Type	Null	Key	Default	Description
`start_time`	`timestamp(6)`	NO		`CURRENT_TIMESTAMP(6)`	Time the query began.
`user_host`	`mediumtext`	NO		`NULL`	User and host combination.
`query_time`	`time(6)`	NO		`NULL`	Total time the query took to execute.
`lock_time`	`time(6)`	NO		`NULL`	Total time the query was locked.

rows_sent	int(11)	NO		NULL	Number of rows sent.
rows_examined	int(11)	NO		NULL	Number of rows examined.
db	varchar(512)	NO		NULL	Default database.
last_insert_id	int(11)	NO		NULL	last_insert_id.
insert_id	int(11)	NO		NULL	Insert id.
server_id	int(10) unsigned	NO		NULL	The server's id.
sql_text	mediumtext	NO		NULL	Full query.
thread_id	bigint(21) unsigned	NO		NULL	Thread id.
rows_affected	int(11)	NO		NULL	Number of rows affected by an UPDATE or DELETE (from MariaDB 10.1.2)

Example

```
SELECT * FROM mysql.slow_log\G
...
*************************** 2. row ***************************
    start_time: 2014-11-11 07:56:28.721519
     user_host: root[root] @ localhost []
    query_time: 00:00:12.000215
     lock_time: 00:00:00.000000
     rows_sent: 1
 rows_examined: 0
            db: test
last_insert_id: 0
     insert_id: 0
     server_id: 1
      sql_text: SELECT SLEEP(12)
     thread_id: 74
...
```

1.1.1.2.9.3.23 mysql.tables_priv Table

The `mysql.tables_priv` table contains information about table-level privileges. The table can be queried and although it is possible to directly update it, it is best to use GRANT for setting privileges.

Note that the MariaDB privileges occur at many levels. A user may be granted a privilege at the table level, but may still not have permission on a database level, for example. See privileges for a more complete view of the MariaDB privilege system.

The INFORMATION_SCHEMA.TABLE_PRIVILEGES table derives its contents from `mysql.tables_priv`.

MariaDB starting with 10.4

In MariaDB 10.4 and later, this table uses the Aria storage engine.

MariaDB until 10.3

In MariaDB 10.3 and before, this table uses the MyISAM storage engine.

The `mysql.tables_priv` table contains the following fields:

Field	Type	Null	Key	Default	Description
Host	char(60)	NO	PRI		Host (together with `User`, `Db` and `Table_name` makes up the unique identifier for this record.

`Db`	`char(64)`	NO	PRI		Database (together with `User`, `Host` and `Table_name` makes up the unique identifier for this record.
`User`	`char(80)`	NO	PRI		User (together with `Host`, `Db` and `Table_name` makes up the unique identifier for this record.
`Table_name`	`char(64)`	NO	PRI		Table name (together with `User`, `Db` and `Table` makes up the unique identifier for this record.
`Grantor`	`char(141)`	NO	MUL		
`Timestamp`	`timestamp`	NO		`CURRENT_TIMESTAMP`	
`Table_priv`	`set('Select', 'Insert', 'Update', 'Delete', 'Create', 'Drop', 'Grant', 'References', 'Index', 'Alter', 'Create View', 'Show view', 'Trigger', 'Delete versioning rows')`	NO			The table privilege type. See Table Privileges for details.
`Column_priv`	`set('Select', 'Insert', 'Update', 'References')`	NO			The column privilege type. See Column Privileges for details.

The Acl_table_grants status variable, added in MariaDB 10.1.4, indicates how many rows the `mysql.tables_priv` table contains.

1.1.1.2.9.3.24 mysql.table_stats Table

The `mysql.table_stats` table is one of three tables storing data used for Engine-independent table statistics. The others are mysql.column_stats and mysql.index_stats.

MariaDB starting with 10.4

In MariaDB 10.4 and later, this table uses the Aria storage engine.

MariaDB until 10.3

In MariaDB 10.3 and before, this table uses the MyISAM storage engine.

The `mysql.table_stats` table contains the following fields:

Field	Type	Null	Key	Default	Description
`db_name`	`varchar(64)`	NO	PRI	`NULL`	Database the table is in .
`table_name`	`varchar(64)`	NO	PRI	`NULL`	Table name.
`cardinality`	`bigint(21) unsigned`	YES		`NULL`	Number of records in the table.

It is possible to manually update the table. See Manual updates to statistics tables for details.

1.1.1.2.9.3.25 mysql.time_zone Table

The `mysql.time_zone` table is one of the mysql system tables that can contain time zone information. It is usually

preferable for the system to handle the time zone, in which case the table will be empty (the default), but you can populate the mysql time zone tables using the mariadb-tzinfo-to-sql utility. See Time Zones for details.

This table uses the Aria storage engine.

The `mysql.time_zone` table contains the following fields:

Field	Type	Null	Key	Default	Description
`Time_zone_id`	`int(10) unsigned`	NO	PRI	`NULL`	ID field, auto_increments.
`Use_leap_seconds`	`enum('Y','N')`	NO		N	Whether or not leap seconds are used.

Example

```
SELECT * FROM mysql.time_zone;
+--------------+------------------+
| Time_zone_id | Use_leap_seconds |
+--------------+------------------+
|            1 | N                |
|            2 | N                |
|            3 | N                |
|            4 | N                |
|            5 | N                |
|            6 | N                |
|            7 | N                |
|            8 | N                |
|            9 | N                |
|           10 | N                |
...
+--------------+------------------+
```

1.1.1.2.9.3.26 mysql.time_zone_leap_second Table

The `mysql.time_zone_leap_second` table is one of the mysql system tables that can contain time zone information. It is usually preferable for the system to handle the time zone, in which case the table will be empty (the default), but you can populate the mysql time zone tables using the mariadb-tzinfo-to-sql utility. See Time Zones for details.

MariaDB starting with 10.4

In MariaDB 10.4 and later, this table uses the Aria storage engine.

MariaDB until 10.3

In MariaDB 10.3 and before, this table uses the MyISAM storage engine.

The `mysql.time_zone_leap_second` table contains the following fields:

Field	Type	Null	Key	Default	Description
`Transition_time`	`bigint(20)`	NO	PRI	`NULL`	
`Correction`	`int(11)`	NO		`NULL`	

1.1.1.2.9.3.27 mysql.time_zone_name Table

The `mysql.time_zone_name` table is one of the mysql system tables that can contain time zone information. It is usually preferable for the system to handle the time zone, in which case the table will be empty (the default), but you can populate the mysql time zone tables using the mariadb-tzinfo-to-sql utility. See Time Zones for details.

This table uses the Aria storage engine.

The `mysql.time_zone_name` table contains the following fields:

Field	Type	Null	Key	Default	Description
`Name`	`char(64)`	NO	PRI	`NULL`	Name of the time zone.

Time_zone_id	int(10) unsigned	NO	PRI	NULL	ID field, auto_increments.

Example

```
SELECT * FROM mysql.time_zone_name;
+--------------------+--------------+
| Name               | Time_zone_id |
+--------------------+--------------+
| Africa/Abidjan     |            1 |
| Africa/Accra       |            2 |
| Africa/Addis_Ababa |            3 |
| Africa/Algiers     |            4 |
| Africa/Asmara      |            5 |
| Africa/Asmera      |            6 |
| Africa/Bamako      |            7 |
| Africa/Bangui      |            8 |
| Africa/Banjul      |            9 |
| Africa/Bissau      |           10 |
...
+--------------------+--------------+
```

1.1.1.2.9.3.28 mysql.time_zone_transition Table

The `mysql.time_zone_transition` table is one of the `mysql` system tables that can contain time zone information. It is usually preferable for the system to handle the time zone, in which case the table will be empty (the default), but you can populate the `mysql` time zone tables using the mariadb-tzinfo-to-sql utility. See Time Zones for details.

This table uses the Aria storage engine.

The `mysql.time_zone_transition` table contains the following fields:

Field	Type	Null	Key	Default	Description
Time_zone_id	int(10) unsigned	NO	PRI	NULL	
Transition_time	bigint(20)	NO	PRI	NULL	
Transition_type_id	int(10) unsigned	NO		NULL	

Example

```
SELECT * FROM mysql.time_zone_transition;
+--------------+-----------------+--------------------+
| Time_zone_id | Transition_time | Transition_type_id |
+--------------+-----------------+--------------------+
|            1 |     -1830383032 |                  1 |
|            2 |     -1640995148 |                  2 |
|            2 |     -1556841600 |                  1 |
|            2 |     -1546388400 |                  2 |
|            2 |     -1525305600 |                  1 |
|            2 |     -1514852400 |                  2 |
|            2 |     -1493769600 |                  1 |
|            2 |     -1483316400 |                  2 |
|            2 |     -1462233600 |                  1 |
|            2 |     -1451780400 |                  2 |
...
+--------------+-----------------+--------------------+
```

1.1.1.2.9.3.29 mysql.time_zone_transition_type Table

The `mysql.time_zone_transition_type` table is one of the `mysql` system tables that can contain time zone information. It is usually preferable for the system to handle the time zone, in which case the table will be empty (the default), but you can populate the `mysql` time zone tables using the mariadb-tzinfo-to-sql utility. See Time Zones for details.

This table uses the Aria storage engine.

The `mysql.time_zone_transition_type` table contains the following fields:

Field	Type	Null	Key	Default	Description
`Time_zone_id`	`int(10) unsigned`	NO	PRI	`NULL`	
`Transition_type_id`	`int(10) unsigned`	NO	PRI	`NULL`	
`Offset`	`int(11)`	NO		0	
`Is_DST`	`tinyint(3) unsigned`	NO		0	
`Abbreviation`	`char(8)`	NO			

Example

```
SELECT * FROM mysql.time_zone_transition_type;
+--------------+--------------------+--------+--------+--------------+
| Time_zone_id | Transition_type_id | Offset | Is_DST | Abbreviation |
+--------------+--------------------+--------+--------+--------------+
|            1 |                  0 |   -968 |      0 | LMT          |
|            1 |                  1 |      0 |      0 | GMT          |
|            2 |                  0 |    -52 |      0 | LMT          |
|            2 |                  1 |   1200 |      1 | GHST         |
|            2 |                  2 |      0 |      0 | GMT          |
|            3 |                  0 |   8836 |      0 | LMT          |
|            3 |                  1 |  10800 |      0 | EAT          |
|            3 |                  2 |   9000 |      0 | BEAT         |
|            3 |                  3 |   9900 |      0 | BEAUT        |
|            3 |                  4 |  10800 |      0 | EAT          |
...
+--------------+--------------------+--------+--------+--------------+
```

1.1.1.2.9.3.30 mysql.transaction_registry Table

The `mysql.transaction_registry` table was introduced in MariaDB 10.3.4 as part of system-versioned tables. It is used for transaction-precise versioning, and contains the following fields:

Field	Type	Null	Key	Default	Description
transaction_id	bigint(20) unsigned	NO	Primary	NULL	
commit_id	bigint(20) unsigned	NO	Unique	NULL	
begin_timestamp	timestamp(6)	NO	Multiple	0000-00-00 00:00:00.000000	Timestamp when the transaction began (BEGIN statement), however see MDEV-16024.
commit	timestamp(6)	NO	Multiple	0000-00-00 00:00:00.000000	Timestamp when the transaction was committed.
isolation_level	enum('READ-UNCOMMITTED','READ-COMMITTED','REPEATABLE-READ','SERIALIZABLE')	NO		NULL	Transaction isolation level.

1.1.1.2.9.3.31 mysql.user Table

MariaDB starting with 10.4

In MariaDB 10.4 and later, the mysql.global_priv table has replaced the `mysql.user` table, and `mysql.user` should be considered obsolete. It is now a view into `mysql.global_priv` created for compatibility with older applications and monitoring scripts. New tools are supposed to use `INFORMATION_SCHEMA` tables. From MariaDB 10.4.13, the dedicated `mariadb.sys` user is created as the definer of the view. Previously, `root` was the definer, which resulted in privilege problems when this username was changed (MDEV-19650).

The `mysql.user` table contains information about users that have permission to access the MariaDB server, and their global privileges. The table can be queried and although it is possible to directly update it, it is best to use GRANT and CREATE USER for adding users and privileges.

Note that the MariaDB privileges occur at many levels. A user may not be granted `create` privilege at the user level, but may still have `create` permission on certain tables or databases, for example. See privileges for a more complete view of the MariaDB privilege system.

The `mysql.user` table contains the following fields:

Field	Type	Null	Key	Default	Description
`Host`	`char(60)`	NO	PRI		Host (together with `User` makes up the unique identifier for this account.
`User`	`char(80)`	NO	PRI		User (together with `Host` makes up the unique identifier for this account.
`Password`	`longtext` (>= MariaDB 10.4.1), `char(41)` (<= MariaDB 10.4.0)	NO			Hashed password, generated by the PASSWORD() function.
`Select_priv`	`enum('N','Y')`	NO		N	Can perform SELECT statements.
`Insert_priv`	`enum('N','Y')`	NO		N	Can perform INSERT statements.
`Update_priv`	`enum('N','Y')`	NO		N	Can perform UPDATE statements.
`Delete_priv`	`enum('N','Y')`	NO		N	Can perform DELETE statements.
`Create_priv`	`enum('N','Y')`	NO		N	Can CREATE DATABASE's or CREATE TABLE's.
`Drop_priv`	`enum('N','Y')`	NO		N	Can DROP DATABASE's or DROP TABLE's.
`Reload_priv`	`enum('N','Y')`	NO		N	Can execute FLUSH statements or equivalent mariadb-admin commands.
`Shutdown_priv`	`enum('N','Y')`	NO		N	Can shut down the server with SHUTDOWN or mariadb-admin shutdown.
`Process_priv`	`enum('N','Y')`	NO		N	Can show information about active processes, via SHOW PROCESSLIST or mariadb-admin processlist.
`File_priv`	`enum('N','Y')`	NO		N	Read and write files on the server, using statements like LOAD DATA INFILE or functions like LOAD_FILE(). Also needed to create CONNECT outward tables. MariaDB server must have permission to access those files.
`Grant_priv`	`enum('N','Y')`	NO		N	User can grant privileges they possess.
`References_priv`	`enum('N','Y')`	NO		N	Unused
`Index_priv`	`enum('N','Y')`	NO		N	Can create an index on a table using the CREATE INDEX statement. Without the `INDEX` privilege, user can still create indexes when creating a table using the CREATE TABLE statement if the user has have the `CREATE` privilege, and user can create indexes using the ALTER TABLE statement if they have the `ALTER` privilege.
`Alter_priv`	`enum('N','Y')`	NO		N	Can perform ALTER TABLE statements.
`Show_db_priv`	`enum('N','Y')`	NO		N	Can list all databases using the SHOW DATABASES statement. Without the `SHOW DATABASES` privilege, user can still issue the `SHOW DATABASES` statement, but it will only list databases containing tables on which they have privileges.

Super_priv	enum('N','Y')	NO		N	Can execute superuser statements: CHANGE MASTER TO, KILL (users who do not have this privilege can only KILL their own threads), PURGE LOGS, SET global system variables, or the mariadb-admin debug command. Also, this permission allows the user to write data even if the read_only startup option is set, enable or disable logging, enable or disable replication on slaves, specify a DEFINER for statements that support that clause, connect once after reaching the MAX_CONNECTIONS. If a statement has been specified for the init-connect mysqld option, that command will not be executed when a user with SUPER privileges connects to the server.
Create_tmp_table_priv	enum('N','Y')	NO		N	Can create temporary tables with the CREATE TEMPORARY TABLE statement.
Lock_tables_priv	enum('N','Y')	NO		N	Acquire explicit locks using the LOCK TABLES statement; user also needs to have the SELECT privilege on a table in order to lock it.
Execute_priv	enum('N','Y')	NO		N	Can execute stored procedure or functions.
Repl_slave_priv	enum('N','Y')	NO		N	Accounts used by slave servers on the master need this privilege. This is needed to get the updates made on the master.
Repl_client_priv	enum('N','Y')	NO		N	Can execute SHOW MASTER STATUS and SHOW SLAVE STATUS statements.
Create_view_priv	enum('N','Y')	NO		N	Can create a view using the CREATE_VIEW statement.
Show_view_priv	enum('N','Y')	NO		N	Can show the CREATE VIEW statement to create a view using the SHOW CREATE VIEW statement.
Create_routine_priv	enum('N','Y')	NO		N	Can create stored programs using the CREATE PROCEDURE and CREATE FUNCTION statements.
Alter_routine_priv	enum('N','Y')	NO		N	Can change the characteristics of a stored function using the ALTER FUNCTION statement.
Create_user_priv	enum('N','Y')	NO		N	Can create a user using the CREATE USER statement, or implicitly create a user with the GRANT statement.
Event_priv	enum('N','Y')	NO		N	Create, drop and alter events.
Trigger_priv	enum('N','Y')	NO		N	Can execute triggers associated with tables the user updates, execute the CREATE TRIGGER and DROP TRIGGER statements.
Create_tablespace_priv	enum('N','Y')	NO		N	
Delete_history_priv	enum('N','Y')	NO		N	Can delete rows created through system versioning.
ssl_type	enum('', 'ANY', 'X509', 'SPECIFIED')	NO			TLS type - see TLS options.
ssl_cipher	blob	NO		NULL	TLS cipher - see TLS options.
x509_issuer	blob	NO		NULL	X509 cipher - see TLS options.
x509_subject	blob	NO		NULL	SSL subject - see TLS options.
max_questions	int(11) unsigned	NO		0	Number of queries the user can perform per hour. Zero is unlimited. See per-account resource limits.

max_updates	int(11) unsigned	NO		0	Number of updates the user can perform per hour. Zero is unlimited. See per-account resource limits.
max_connections	int(11) unsigned	NO		0	Number of connections the account can start per hour. Zero is unlimited. See per-account resource limits.
max_user_connections	int(11)	NO		0	Number of simultaneous connections the account can have. Zero is unlimited. See per-account resource limits.
plugin	char(64)	NO			Authentication plugin used on connection. If empty, uses the default.
authentication_string	text	NO		NULL	Authentication string for the authentication plugin.
password_expired	enum('N','Y')	NO		N	MySQL-compatibility option, not implemented in MariaDB.
is_role	enum('N','Y')	NO		N	Whether the user is a role.
default_role	char(80)	NO		N	Role which will be enabled on user login automatically.
max_statement_time	decimal(12,6)	NO		0.000000	If non-zero, how long queries can run before being killed automatically.
Field	**Type**	**Null**	**Key**	**Default**	**Description**

The Acl_roles status variable indicates how many rows the `mysql.user` table contains where `is_role='Y'`.

The Acl_users status variable, indicates how many rows the `mysql.user` table contains where `is_role='N'`.

Authentication Plugin

When the `plugin` column is empty, MariaDB defaults to authenticating accounts with either the `mysql_native_password` or the `mysql_old_password` plugins. It decides which based on the hash used in the value for the `Password` column. When there's no password set or when the 4.1 password hash is used, (which is 41 characters long), MariaDB uses the `mysql_native_password` plugin. The `mysql_old_password` plugin is used with pre-4.1 password hashes, (which are 16 characters long).

MariaDB also supports the use of alternative authentication plugins. When the `plugin` column is not empty for the given account, MariaDB uses it to authenticate connection attempts. The specific plugin then uses the value of either the `Password` column or the `authentication_string` column to authenticate the user.

A specific authentication plugin can be used for an account by providing the `IDENTIFIED VIA authentication_plugin` clause with the CREATE USER, ALTER USER, or GRANT statements.

For example, the following statement would create an account that authenticates with the PAM authentication plugin:

```
CREATE USER foo2@test IDENTIFIED VIA pam;
```

If the specific authentication plugin uses the `authentication_string` column, then this value for the account can be specified after a `USING` or `AS` keyword. For example, the PAM authentication plugin accepts a service name that would go into the `authentication_string` column for the account:

```
CREATE USER foo2@test IDENTIFIED VIA pam USING 'mariadb';
```

1.1.1.2.9.3.32 Spider mysql Database Tables

The Spider storage engine installs the following system tables in the `mysql` database.

mysql.spider_link_failed_log Table

The mysql.spider_link_failed_log table.

mysql.spider_link_mon_servers Table

The mysql.spider_link_mon_servers table.

mysql.spider_tables Table

The mysql.spider_tables table.

mysql.spider_table_crd Table

The mysql.spider_table_crd table.

mysql.spider_table_position_for_recovery Table

The mysql.spider_table_position_for_recovery table.

mysql.spider_table_sts Table

The mysql.spider_table_sts table.

mysql.spider_xa Table

The mysql.spider_xa table.

mysql.spider_xa_failed_log Table

The mysql.spider_xa_failed_log table.

mysql.spider_xa_member Table

The mysql.spider_xa_member table.

1.1.1.2.9.3.32.1 mysql.spider_link_failed_log Table

The `mysql.spider_link_failed_log` table is installed by the Spider storage engine.

MariaDB starting with 10.4

In MariaDB 10.4 and later, this table uses the Aria storage engine.

MariaDB until 10.3

In MariaDB 10.3 and before, this table uses the MyISAM storage engine.

It contains the following fields:

Field	Type	Null	Key	Default	Description
db_name	char(64)	NO			
table_name	char(199)	NO			
link_id	char(64)	NO			
failed_time	timestamp	NO		current_timestamp()	

1.1.1.2.9.3.32.2 mysql.spider_link_mon_servers Table

The `mysql.spider_link_mon_servers` table is installed by the Spider storage engine.

MariaDB starting with 10.4

In MariaDB 10.4 and later, this table uses the Aria storage engine.

MariaDB until 10.3

In MariaDB 10.3 and before, this table uses the MyISAM storage engine.

It contains the following fields:

Field	Type	Null	Key	Default	Description

db_name	char(64)	NO	PRI		
table_name	char(199)	NO	PRI		
link_id	char(64)	NO	PRI		
sid	int(10) unsigned	NO	PRI	0	
server	char(64)	YES		NULL	
scheme	char(64)	YES		NULL	
host	char(64)	YES		NULL	
port	char(5)	YES		NULL	
socket	text	YES		NULL	
username	char(64)	YES		NULL	
password	char(64)	YES		NULL	
ssl_ca	text	YES		NULL	
ssl_capath	text	YES		NULL	
ssl_cert	text	YES		NULL	
ssl_cipher	char(64)	YES		NULL	
ssl_key	text	YES		NULL	
ssl_verify_server_cert	tinyint(4)	NO		0	
default_file	text	YES		NULL	
default_group	char(64)	YES		NULL	
dsn	char(64)	YES		NULL	
filedsn	text	YES		NULL	
driver	char(64)	YES		NULL	

1.1.1.2.9.3.32.3 mysql.spider_tables Table

The `mysql.spider_tables` table is installed by the Spider storage engine.

MariaDB starting with 10.4

In MariaDB 10.4 and later, this table uses the Aria storage engine.

MariaDB until 10.3

In MariaDB 10.3 and before, this table uses the MyISAM storage engine.

It contains the following fields:

Field	Type	Null	Key	Default	Description
db_name	char(64)	NO	PRI		
table_name	char(199)	NO	PRI		
link_id	int(11)	NO	PRI	0	
priority	bigint(20)	NO	MUL	0	
server	char(64)	YES		NULL	
scheme	char(64)	YES		NULL	
host	char(64)	YES		NULL	
port	char(5)	YES		NULL	
socket	text	YES		NULL	
username	char(64)	YES		NULL	
password	char(64)	YES		NULL	

ssl_ca	text	YES		NULL	
ssl_capath	text	YES		NULL	
ssl_cert	text	YES		NULL	
ssl_cipher	char(64)	YES		NULL	
ssl_key	text	YES		NULL	
ssl_verify_server_cert	tinyint(4)	NO		0	
monitoring_binlog_pos_at_failing	tinyint(4)	NO		0	
default_file	text	YES		NULL	
default_group	char(64)	YES		NULL	
dsn	char(64)	YES		NULL	
filedsn	text	YES		NULL	
driver	char(64)	YES		NULL	
tgt_db_name	char(64)	YES		NULL	
tgt_table_name	char(64)	YES		NULL	
link_status	tinyint(4)	NO		1	
block_status	tinyint(4)	NO		0	
static_link_id	char(64)	YES		NULL	

1.1.1.2.9.3.32.4 mysql.spider_table_crd Table

The `mysql.spider_table_crd` table is installed by the Spider storage engine.

MariaDB starting with 10.4
In MariaDB 10.4 and later, this table uses the Aria storage engine.

MariaDB until 10.3
In MariaDB 10.3 and before, this table uses the MyISAM storage engine.

It contains the following fields:

Field	Type	Null	Key	Default	Description
db_name	char(64)	NO	PRI		
table_name	char(199)	NO	PRI		
key_seq	int(10) unsigned	NO	PRI	0	
cardinality	bigint(20)	NO		0	

1.1.1.2.9.3.32.5 mysql.spider_table_position_for_recovery Table

The `mysql.spider_table_position_for_recovery` table is installed by the Spider storage engine.

MariaDB starting with 10.4
In MariaDB 10.4 and later, this table uses the Aria storage engine.

MariaDB until 10.3
In MariaDB 10.3 and before, this table uses the MyISAM storage engine.

It contains the following fields:

Field	Type	Null	Key	Default	Description

db_name	char(64)	NO	PRI		
table_name	char(199)	NO	PRI		
failed_link_id	int(11)	NO	PRI	0	
source_link_id	int(11)	NO	PRI	0	
file	text	YES		NULL	
position	text	YES		NULL	
gtid	text	YES		NULL	

1.1.1.2.9.3.32.6 mysql.spider_table_sts Table

The `mysql.spider_table_sts` table is installed by the Spider storage engine.

MariaDB starting with 10.4

In MariaDB 10.4 and later, this table uses the Aria storage engine.

MariaDB until 10.3

In MariaDB 10.3 and before, this table uses the MyISAM storage engine.

It contains the following fields:

Field	Type	Null	Key	Default	Description
db_name	char(64)	NO	PRI		
table_name	char(199)	NO	PRI		
data_file_length	bigint(20) unsigned	NO		0	
max_data_file_length	bigint(20) unsigned	NO		0	
index_file_length	bigint(20) unsigned	NO		0	
records	bigint(20) unsigned	NO		0	
mean_rec_length	bigint(20) unsigned	NO		0	
check_time	datetime	NO		0000-00-00 00:00:00	
create_time	datetime	NO		0000-00-00 00:00:00	
update_time	datetime	NO		0000-00-00 00:00:00	
checksum	bigint(20) unsigned	YES		NULL	

1.1.1.2.9.3.32.7 mysql.spider_xa Table

The `mysql.spider_xa` table is installed by the Spider storage engine.

MariaDB starting with 10.4

In MariaDB 10.4 and later, this table uses the Aria storage engine.

MariaDB until 10.3

In MariaDB 10.3 and before, this table uses the MyISAM storage engine.

It contains the following fields:

Field	Type	Null	Key	Default	Description
format_id	int(11)	NO	PRI	0	
gtrid_length	int(11)	NO	PRI	0	
bqual_length	int(11)	NO		0	
data	binary(128)	NO	PRI		

status	char(8)	NO	MUL		

1.1.1.2.9.3.32.8 mysql.spider_xa_failed_log Table

The `mysql.spider_xa_failed_log` table is installed by the Spider storage engine.

MariaDB starting with 10.4

In MariaDB 10.4 and later, this table uses the Aria storage engine.

MariaDB until 10.3

In MariaDB 10.3 and before, this table uses the MyISAM storage engine.

It contains the following fields:

Field	Type	Null	Key	Default	Description
format_id	int(11)	NO		0	
gtrid_length	int(11)	NO		0	
bqual_length	int(11)	NO		0	
data	binary(128)	NO	MUL		
scheme	char(64)	NO			
host	char(64)	NO			
port	char(5)	NO			
socket	text	NO		NULL	
username	char(64)	NO			
password	char(64)	NO			
ssl_ca	text	YES		NULL	
ssl_capath	text	YES		NULL	
ssl_cert	text	YES		NULL	
ssl_cipher	char(64)	YES		NULL	
ssl_key	text	YES		NULL	
ssl_verify_server_cert	tinyint(4)	NO		0	
default_file	text	YES		NULL	
default_group	char(64)	YES		NULL	
dsn	char(64)	YES		NULL	
filedsn	text	YES		NULL	
driver	char(64)	YES		NULL	
thread_id	int(11)	YES		NULL	
status	char(8)	NO			
failed_time	timestamp	NO		current_timestamp()	

1.1.1.2.9.3.32.9 mysql.spider_xa_member Table

The `mysql.spider_xa_member` table is installed by the Spider storage engine.

MariaDB starting with 10.4

In MariaDB 10.4 and later, this table uses the Aria storage engine.

MariaDB until 10.3

In MariaDB 10.3 and before, this table uses the MyISAM storage engine.

It contains the following fields:

Field	Type	Null	Key	Default	Description
format_id	int(11)	NO		0	
gtrid_length	int(11)	NO		0	
bqual_length	int(11)	NO		0	
data	binary(128)	NO	MUL		
scheme	char(64)	NO			
host	char(64)	NO			
port	char(5)	NO			
socket	text	NO		NULL	
username	char(64)	NO			
password	char(64)	NO			
ssl_ca	text	YES		NULL	
ssl_capath	text	YES		NULL	
ssl_cert	text	YES		NULL	
ssl_cipher	char(64)	YES		NULL	
ssl_key	text	YES		NULL	
ssl_verify_server_cert	tinyint(4)	NO		0	
default_file	text	YES		NULL	
default_group	char(64)	YES		NULL	
dsn	char(64)	YES		NULL	
filedsn	text	YES		NULL	
driver	char(64)	YES		NULL	

1.1.1.2.9.4 Sys Schema

MariaDB starting with 10.6.0

The sys_schema is a collection of views, functions and procedures to help administrators get insight into database usage.

This article is currently incomplete.

Sys Schema sys_config Table
Configuration options for the Sys Schema.

Sys Schema Stored Functions
Stored functions available in the Sys Schema.

Sys Schema Stored Procedures
Stored procedures available in the Sys Schema.

1.1.1.2.9.4.1 Sys Schema sys_config Table

MariaDB starting with 10.6.0

The Sys Schema *sys_config* table was added in MariaDB 10.6.0. The *sys_config* table is also backported to MariaDB-10.5-enterprise.

The *sys.sys_config* table holds configuration options for the Sys Schema.

This is a persistent table (using the Aria storage engine), with the configuration persisting across upgrades (new options are added with INSERT IGNORE.

The table also has two related triggers, which maintain the user that INSERTs or UPDATEs the configuration - sys_config_insert_set_user and sys_config_update_set_user respectively.

Its structure is as follows:

```
+----------+--------------+------+-----+-------------------+-----------------------------+
| Field    | Type         | Null | Key | Default           | Extra                       |
+----------+--------------+------+-----+-------------------+-----------------------------+
| variable | varchar(128) | NO   | PRI | NULL              |                             |
| value    | varchar(128) | YES  |     | NULL              |                             |
| set_time | timestamp    | NO   |     | CURRENT_TIMESTAMP | on update CURRENT_TIMESTAMP |
| set_by   | varchar(128) | YES  |     | NULL              |                             |
+----------+--------------+------+-----+-------------------+-----------------------------+
```

Note, when functions check for configuration options, they first check whether a similar named user variable exists with a value, and if this is not set then pull the configuration option from this table in to that named user variable. This is done for performance reasons (to not continually SELECT from the table), however this comes with the side effect that once inited, the values last with the session, somewhat like how session variables are inited from global variables. If the values within this table are changed, they will not take effect until the user logs in again.

Options Included

Variable	Default Value	Description
statement_truncate_len	64	Sets the size to truncate statements to, for the format_statement function.
statement_performance_analyzer.limit	100	The maximum number of rows to include for the views that does not have a built-in limit (e.g. the 95th percentile view). If not set the limit is 100.
statement_performance_analyzer.view	NULL	Used together with the 'custom' view. If the value contains a space, it is considered a query, otherwise it must be an existing view querying the performance_schema.events_statements_summary_by_digest table.
diagnostics.allow_i_s_tables	OFF	Specifies whether it is allowed to do table scan queries on information_schema.TABLES for the diagnostics procedure.
diagnostics.include_raw	OFF	Set to 'ON' to include the raw data (e.g. the original output of "SELECT * FROM sys.metrics") for the diagnostics procedure.
ps_thread_trx_info.max_length	65535	Sets the maximum output length for JSON object output by the ps_thread_trx_info() function.

Notes

Some early versions of *sys_config* were stored in InnoDB format.

1.1.1.2.9.4.2 Sys Schema Stored Functions

The following stored functions are available in the Sys Schema.

extract_schema_from_file_name

Returns the schema (database) name.

extract_table_from_file_name

Returns the table name from the provided path.

format_bytes

Returns a string consisting of a value and the units in a human-readable format.

format_path

Returns a modified path, replacing subpaths matching the values of various system variables.

format_statement

Returns a reduced length string.

format_time

Given a time in picoseconds, returns a human-readable time and unit.

list_add

Takes a list to be be modified and a value to be added to the list, returning the resulting value.

list_drop

Takes a list to be be modified and a value to be dropped, returning the resulting value.

ps_is_account_enabled

Whether or not Performance Schema instrumentation for a given account is enabled.

ps_is_consumer_enabled

Whether or not Performance Schema instrumentation for a given consumer is enabled.

ps_is_instrument_default_enabled

Whether or not a Performance Schema instrument is enabled by default.

ps_is_instrument_default_timed

Whether or not a Performance Schema instrument is timed by default.

ps_is_thread_instrumented

Whether or not instrumentation for a given connection_id is enabled.

ps_thread_account

Returns the account associated with the given thread_id.

ps_thread_id

Returns the thread_id associated with the given connection_id.

ps_thread_stack

Returns statements, stages, events within the Performance Schema for a given thread_id.

ps_thread_trx_info

Returns a JSON object with information about the thread specified by the given thread_id.

quote_identifier

Returns quoted, properly escaped identifier.

sys_get_config

Returns a configuration option value from the sys_config table.

version_major

Returns the MariaDB Server major release version.

version_minor

Returns the MariaDB Server minor release version.

version_patch

MariaDB Server patch release version.

1.1.1.2.9.4.2.1 extract_schema_from_file_name

Syntax

```
sys.extract_schema_from_file_name(path)
```

Description

`extract_schema_from_file_name` is a stored function available with the Sys Schema.

Given a file path, it returns the schema (database) name. The file name is assumed to be within the schema directory, and therefore the function will not return the expected result with partitions, or when tables are defined using the DATA_DIRECTORY table option.

The function does not examine anything on disk. The return value, a VARCHAR(64), is determined solely from the provided path.

Examples

```
SELECT sys.extract_schema_from_file_name('/usr/local/mysql/data/db/t1.ibd');
+-----------------------------------------------------------------------+
| sys.extract_schema_from_file_name('/usr/local/mysql/data/db/t1.ibd') |
+-----------------------------------------------------------------------+
| db                                                                    |
+-----------------------------------------------------------------------+
```

1.1.1.2.9.4.2.2 extract_table_from_file_name

Syntax

```
sys.extract_table_from_file_name(path)
```

Description

`extract_table_from_file_name` is a stored function available with the Sys Schema.

Given a file path, it returns the table name.

The function does not examine anything on disk. The return value, a VARCHAR(64), is determined solely from the provided path.

Examples

```
SELECT sys.extract_table_from_file_name('/usr/local/mysql/data/db/t1.ibd');
+-----------------------------------------------------------------------+
| sys.extract_table_from_file_name('/usr/local/mysql/data/db/t1.ibd') |
+-----------------------------------------------------------------------+
| t1                                                                    |
+-----------------------------------------------------------------------+
```

1.1.1.2.9.4.2.3 format_bytes

Syntax

```
sys.format_bytes(double)
```

Description

`format_bytes` is a stored function available with the Sys Schema.

Given a byte count, returns a string consisting of a value and the units in a human-readable format. The units will be in

bytes, KiB (kibibytes), MiB (mebibytes), GiB (gibibytes), TiB (tebibytes), or PiB (pebibytes).

The binary prefixes (kibi, mebi, gibi, tebi and pebi) were created in December 1998 by the International Electrotechnical Commission to avoid possible ambiguity, as the widely-used prefixes kilo, mega, giga, tera and peta can be used to refer to both the power-of-10 decimal system multipliers and the power-of-two binary system multipliers.

Examples

```
SELECT sys.format_bytes(1000),sys.format_bytes(1024);
+-----------------------+-----------------------+
| sys.format_bytes(1000) | sys.format_bytes(1024) |
+-----------------------+-----------------------+
| 1000 bytes             | 1.00 KiB              |
+-----------------------+-----------------------+

SELECT sys.format_bytes(1000000),sys.format_bytes(1048576);
+--------------------------+--------------------------+
| sys.format_bytes(1000000) | sys.format_bytes(1048576) |
+--------------------------+--------------------------+
| 976.56 KiB                | 1.00 MiB                 |
+--------------------------+--------------------------+

SELECT sys.format_bytes(1000000000),sys.format_bytes(1073741874);
+-----------------------------+-----------------------------+
| sys.format_bytes(1000000000) | sys.format_bytes(1073741874) |
+-----------------------------+-----------------------------+
| 953.67 MiB                  | 1.00 GiB                    |
+-----------------------------+-----------------------------+

SELECT sys.format_bytes(1000000000000),sys.format_bytes(1099511627776);
+--------------------------------+--------------------------------+
| sys.format_bytes(1000000000000) | sys.format_bytes(1099511627776) |
+--------------------------------+--------------------------------+
| 931.32 GiB                     | 1.00 TiB                       |
+--------------------------------+--------------------------------+

SELECT sys.format_bytes(1000000000000000),sys.format_bytes(1125899906842624);
+-----------------------------------+-----------------------------------+
| sys.format_bytes(1000000000000000) | sys.format_bytes(1125899906842624) |
+-----------------------------------+-----------------------------------+
| 909.49 TiB                        | 1.00 PiB                          |
+-----------------------------------+-----------------------------------+
```

1.1.1.2.9.4.2.4 format_path

Syntax

```
sys.format_path(path)
```

Description

`format_path` is a stored function available with the Sys Schema that, given a path, returns a modified path after replacing subpaths matching the values of various system variables with the variable name.

The system variables that are matched are, in order:

- datadir
- tmpdir
- slave_load_tmpdir
- innodb_data_home_dir
- innodb_log_group_home_dir
- innodb_undo_directory
- basedir

Examples

```
SELECT @@tmpdir;
+-----------------------------------+
| @@tmpdir                          |
+-----------------------------------+
| /home/ian/sandboxes/msb_10_8_2/tmp |
+-----------------------------------+

SELECT sys.format_path('/home/ian/sandboxes/msb_10_8_2/tmp/testdb.ibd');
+-----------------------------------------------------------------+
| sys.format_path('/home/ian/sandboxes/msb_10_8_2/tmp/testdb.ibd') |
+-----------------------------------------------------------------+
| @@tmpdir/testdb.ibd                                             |
+-----------------------------------------------------------------+
```

1.1.1.2.9.4.2.5 format_statement

Syntax

```
sys.format_statement(statement)
```

Description

Returns a reduced length string. The length is specified by the statement_truncate_len configuration option (default 64), and the removed part of the string (if any) is replaced with an ellipsis (three dots).

The function is intended for use in formatting lengthy SQL statements to a fixed length.

Examples

Default truncation length 64:

```
SELECT sys.format_statement(
  'SELECT field1, field2, field3, field4, field5, field6 FROM table1'
  ) AS formatted_statement;
+-----------------------------------------------------------------+
| formatted_statement                                             |
+-----------------------------------------------------------------+
| SELECT field1, field2, field3, ... d4, field5, field6 FROM table1 |
+-----------------------------------------------------------------+
```

Reducing the truncation length to 48:

```
SET @sys.statement_truncate_len = 48;

SELECT sys.format_statement(
  'SELECT field1, field2, field3, field4, field5, field6 FROM table1'
  ) AS formatted_statement;
+-------------------------------------------------+
| formatted_statement                             |
+-------------------------------------------------+
| SELECT field1, field2, ... d5, field6 FROM table1 |
+-------------------------------------------------+
```

1.1.1.2.9.4.2.6 format_time

Syntax

```
sys.format_time(picoseconds)
```

Description

`format_time` is a stored function available with the Sys Schema. Given a time in picoseconds, returns a human-readable time value and unit indicator. Unit can be:

- ps - picoseconds
- ns - nanoseconds
- us - microseconds
- ms - milliseconds
- s - seconds
- m - minutes
- h - hours
- d - days
- w - weeks

This function is very similar to the FORMAT_PICO_TIME function introduced in MariaDB 11.0.2, but with the following differences:

- Represents minutes as `m` rather than `min`.
- Represent weeks.

Examples

```
SELECT
    sys.format_time(43) AS ps,
    sys.format_time(4321) AS ns,
    sys.format_time(43211234) AS us,
    sys.format_time(432112344321) AS ms,
    sys.format_time(43211234432123) AS s,
    sys.format_time(432112344321234) AS m,
    sys.format_time(4321123443212345) AS h,
    sys.format_time(43211234432123545) AS d,
    sys.format_time(432112344321234543) AS w;
+-------+---------+----------+-----------+---------+--------+--------+--------+---------+
| ps    | ns      | us       | ms        | s       | m      | h      | d      | w       |
+-------+---------+----------+-----------+---------+--------+--------+--------+---------+
| 43 ps | 4.32 ns | 43.21 us | 432.11 ms | 43.21 s | 7.20 m | 1.20 h | 5.00 d | 71.45 w |
+-------+---------+----------+-----------+---------+--------+--------+--------+---------+
```

1.1.1.2.9.4.2.7 list_add

Syntax

```
sys.list_add(list,value)
```

Description

`list_add` is a stored function available with the Sys Schema.

It takes a *list* to be be modified and a *value* to be added to the list, returning the resulting value. This can be used, for example, to add a value to a system variable taking a comma-delimited list of options, such as sql_mode.

The related function list_drop can be used to drop a value from a list.

Examples

```
SELECT @@sql_mode;
+--------------------------------------------------------------------+
| @@sql_mode                                                         |
+--------------------------------------------------------------------+
| STRICT_TRANS_TABLES,ERROR_FOR_DIVISION_BY_ZERO,
NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION |
+--------------------------------------------------------------------+

SET @@sql_mode = sys.list_add(@@sql_mode, 'NO_ZERO_DATE');

SELECT @@sql_mode;
+--------------------------------------------------------------------+
| @@sql_mode                                                         |
+--------------------------------------------------------------------+
| STRICT_TRANS_TABLES,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,
NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION |
+--------------------------------------------------------------------+
```

1.1.1.2.9.4.2.8 list_drop

Syntax

```
sys.list_drop(list,value)
```

Description

`list_drop` is a stored function available with the Sys Schema.

It takes a *list* to be be modified and a *value* to be dropped from the list, returning the resulting value. This can be used, for example, to remove a value from a system variable taking a comma-delimited list of options, such as sql_mode.

The related function list_add can be used to add a value to a list.

Examples

```
SELECT @@sql_mode;
+--------------------------------------------------------------------+
| @@sql_mode                                                         |
+--------------------------------------------------------------------+
| STRICT_TRANS_TABLES,ERROR_FOR_DIVISION_BY_ZERO,
NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION |
+--------------------------------------------------------------------+

SET @@sql_mode = sys.list_drop(@@sql_mode, 'NO_ENGINE_SUBSTITUTION');

SELECT @@sql_mode;
+--------------------------------------------------------------------+
| @@sql_mode                                                         |
+--------------------------------------------------------------------+
| STRICT_TRANS_TABLES,ERROR_FOR_DIVISION_BY_ZERO,
NO_AUTO_CREATE_USER |
+--------------------------------------------------------------------+
```

1.1.1.2.9.4.2.9 ps_is_account_enabled

Syntax

```
sys.ps_is_account_enabled(host,user)
```

Description

`ps_is_account_enabled` is a stored function available with the Sys Schema.

It takes *host* and *user* arguments, and returns an ENUM('YES','NO') depending on whether Performance Schema instrumentation for the given account is enabled.

Examples

```
SELECT sys.ps_is_account_enabled('localhost', 'root');
+-----------------------------------------------+
| sys.ps_is_account_enabled('localhost', 'root') |
+-----------------------------------------------+
| YES                                           |
+-----------------------------------------------+
```

1.1.1.2.9.4.2.10 ps_is_consumer_enabled

Syntax

```
sys.ps_is_consumer_enabled(consumer)
```

Description

`ps_is_consumer_enabled` is a stored function available with the Sys Schema.

It returns an ENUM('YES','NO') depending on whether Performance Schema instrumentation for the given consumer is enabled, and NULL if not given a valid consumer name.

Examples

```
SELECT sys.ps_is_consumer_enabled('global_instrumentation');
+-------------------------------------------------------+
| sys.ps_is_consumer_enabled('global_instrumentation') |
+-------------------------------------------------------+
| YES                                                   |
+-------------------------------------------------------+

SELECT sys.ps_is_consumer_enabled('events_stages_current');
+-----------------------------------------------------+
| sys.ps_is_consumer_enabled('events_stages_current') |
+-----------------------------------------------------+
| NO                                                  |
+-----------------------------------------------------+

SELECT sys.ps_is_consumer_enabled('nonexistent_consumer');
+-------------------------------------------------------+
| sys.ps_is_consumer_enabled('nonexistent_consumer') |
+-------------------------------------------------------+
| NULL                                                  |
+-------------------------------------------------------+
```

1.1.1.2.9.4.2.11 ps_is_instrument_default_enabled

Syntax

```
sys.ps_is_instrument_default_enabled(instrument)
```

Description

`ps_is_instrument_default_enabled` is a stored function available with the Sys Schema.

It returns `YES` if the given Performance Schema *instrument* is enabled by default, and `NO` if it is not, does not exist, or is a NULL value.

Examples

```
SELECT sys.ps_is_instrument_default_enabled('statement/sql/select');
+-----------------------------------------------------------+
| sys.ps_is_instrument_default_enabled('statement/sql/select') |
+-----------------------------------------------------------+
| YES                                                       |
+-----------------------------------------------------------+

SELECT sys.ps_is_instrument_default_enabled('memory/sql/udf_mem');
+-----------------------------------------------------------+
| sys.ps_is_instrument_default_enabled('memory/sql/udf_mem') |
+-----------------------------------------------------------+
| NO                                                        |
+-----------------------------------------------------------+

SELECT sys.ps_is_instrument_default_enabled('memory/sql/nonexistent');
+-----------------------------------------------------------------+
| sys.ps_is_instrument_default_enabled('memory/sql/nonexistent') |
+-----------------------------------------------------------------+
| NO                                                              |
+-----------------------------------------------------------------+

SELECT sys.ps_is_instrument_default_enabled(NULL);
+-------------------------------------------+
| sys.ps_is_instrument_default_enabled(NULL) |
+-------------------------------------------+
| NO                                        |
+-------------------------------------------+
```

1.1.1.2.9.4.2.12 ps_is_instrument_default_timed

Syntax

```
sys.ps_is_instrument_default_timed(instrument)
```

Description

`ps_is_instrument_default_timed` is a stored function available with the Sys Schema.

It returns `YES` if the given Performance Schema *instrument* is timed by default, and `NO` if it is not, does not exist, or is a NULL value.

Examples

```
SELECT sys.ps_is_instrument_default_timed('statement/sql/select');
+-----------------------------------------------------------+
| sys.ps_is_instrument_default_timed('statement/sql/select') |
+-----------------------------------------------------------+
| YES                                                       |
+-----------------------------------------------------------+

SELECT sys.ps_is_instrument_default_timed('memory/sql/udf_mem');
+---------------------------------------------------------+
| sys.ps_is_instrument_default_timed('memory/sql/udf_mem') |
+---------------------------------------------------------+
| NO                                                      |
+---------------------------------------------------------+

SELECT sys.ps_is_instrument_default_timed('memory/sql/nonexistent');
+-------------------------------------------------------------+
| sys.ps_is_instrument_default_timed('memory/sql/udf_memsds') |
+-------------------------------------------------------------+
| NO                                                          |
+-------------------------------------------------------------+

SELECT sys.ps_is_instrument_default_timed(NULL);
+-----------------------------------------+
| sys.ps_is_instrument_default_timed(NULL) |
+-----------------------------------------+
| NO                                      |
+-----------------------------------------+
```

1.1.1.2.9.4.2.13 ps_is_thread_instrumented

Syntax

```
sys.ps_is_thread_instrumented(connection_id)
```

Description

`ps_is_thread_instrumented` is a stored function available with the Sys Schema that returns whether or not Performance Schema instrumentation for the given *connection_id* is enabled.

- `YES` - instrumentation is enabled
- `NO` - instrumentation is not enabled
- `UNKNOWN` - the connection ID is unknown
- `NULL` - NULL value

Examples

```
SELECT sys.ps_is_thread_instrumented(CONNECTION_ID());
+-----------------------------------------------+
| sys.ps_is_thread_instrumented(CONNECTION_ID()) |
+-----------------------------------------------+
| YES                                           |
+-----------------------------------------------+

SELECT sys.ps_is_thread_instrumented(2042);
+-----------------------------------+
| sys.ps_is_thread_instrumented(2042) |
+-----------------------------------+
| UNKNOWN                           |
+-----------------------------------+

SELECT sys.ps_is_thread_instrumented(NULL);
+-----------------------------------+
| sys.ps_is_thread_instrumented(NULL) |
+-----------------------------------+
| NULL                              |
+-----------------------------------+
```

1.1.1.2.9.4.2.14 ps_thread_account

Syntax

```
sys.ps_thread_account(thread_id)
```

Description

`ps_thread_account` is a stored function available with the Sys Schema that returns the account (username@hostname) associated with the given *thread_id*.

Returns `NULL` if the thread_id is not found.

Examples

```
SELECT sys.ps_thread_account(sys.ps_thread_id(CONNECTION_ID()));
+-----------------------------------------------------------+
| sys.ps_thread_account(sys.ps_thread_id(CONNECTION_ID())) |
+-----------------------------------------------------------+
| msandbox@localhost                                        |
+-----------------------------------------------------------+

SELECT sys.ps_thread_account(sys.ps_thread_id(2042));
+-----------------------------------------------+
| sys.ps_thread_account(sys.ps_thread_id(2042)) |
+-----------------------------------------------+
| NULL                                          |
+-----------------------------------------------+

SELECT sys.ps_thread_account(sys.ps_thread_id(NULL));
+-----------------------------------------------+
| sys.ps_thread_account(sys.ps_thread_id(NULL)) |
+-----------------------------------------------+
| msandbox@localhost                            |
+-----------------------------------------------+
```

1.1.1.2.9.4.2.15 ps_thread_id

Syntax

```
sys.ps_thread_id(connection_id)
```

Description

`ps_thread_id` is a stored function available with the Sys Schema that returns the thread_id associated with the given *connection_id*. If the *connection_id* is NULL, returns the thread_id for the current connection.

Examples

```
SELECT * FROM performance_schema.threads\G
*************************** 13. row ***************************
          THREAD_ID: 13
               NAME: thread/sql/one_connection
               TYPE: FOREGROUND
     PROCESSLIST_ID: 3
   PROCESSLIST_USER: msandbox
   PROCESSLIST_HOST: localhost
     PROCESSLIST_DB: test
PROCESSLIST_COMMAND: Query
   PROCESSLIST_TIME: 0
  PROCESSLIST_STATE: Sending data
   PROCESSLIST_INFO: SELECT * FROM performance_schema.threads
   PARENT_THREAD_ID: 1
               ROLE: NULL
       INSTRUMENTED: YES
            HISTORY: YES
    CONNECTION_TYPE: Socket
       THREAD_OS_ID: 24379


SELECT sys.ps_thread_id(3);
+----------------------+
| sys.ps_thread_id(3) |
+----------------------+
|                   13 |
+----------------------+

SELECT sys.ps_thread_id(NULL);
+------------------------+
| sys.ps_thread_id(NULL) |
+------------------------+
|                     13 |
+------------------------+
```

1.1.1.2.9.4.2.16 ps_thread_stack

Syntax

```
sys.ps_thread_stack(thread_id, verbose)
```

Description

`ps_thread_stack` is a stored function available with the Sys Schema that, for a given *thread_id*, returns all statements, stages, and events within the Performance Schema, as a JSON formatted stack.

The boolean *verbose* argument specifies whether or not to include `file:lineno` information in the events.

Examples

```
SELECT sys.ps_thread_stack(13, FALSE) AS thread_stack\G
*************************** 1. row ***************************
thread_stack: {"rankdir": "LR","nodesep": "0.10",
  "stack_created": "2022-03-28 16:01:06",
  "mysql_version": "10.8.2-MariaDB",
  "mysql_user": "msandbox@localhost",
  "events": []}
```

1.1.1.2.9.4.2.17 ps_thread_trx_info

Syntax

```
sys.ps_thread_trx_info(thread_id)
```

Contents

Description

`ps_thread_trx_info` is a stored function available with the Sys Schema.

It returns a JSON object with information about the thread specified by the given *thread_id*. This information includes:

- the current transaction
- executed statements (derived from the Performance Schema events_transactions_current Table and the Performance Schema events_statements_history Table (full data will only returned if the consumers for those tables are enabled).

The maximum length of the returned JSON object is determined by the value of the ps_thread_trx_info.max_length sys_config option (by default 65535). If the returned value exceeds this length, a JSON object error is returned.

Examples

```
```

1.1.1.2.9.4.2.18 quote_identifier

Syntax

```
sys.quote_identifier(str)
```

Description

`quote_identifier` is a stored function available with the Sys Schema.

It quotes a string to produce a result that can be used as an identifier in an SQL statement. The string is returned enclosed by backticks (" ` ") and with each instance of backtick (" ` ") doubled. If the argument is `NULL`, the return value is the word "`NULL`" without enclosing backticks.

Examples

```
SELECT sys.quote_identifier("Identifier with spaces");
+-----------------------------------------------+
| sys.quote_identifier("Identifier with spaces") |
+-----------------------------------------------+
| `Identifier with spaces`                      |
+-----------------------------------------------+

SELECT sys.quote_identifier("Identifier` containing `backticks");
+-------------------------------------------------------------+
| sys.quote_identifier("Identifier` containing `backticks") |
+-------------------------------------------------------------+
| `Identifier`` containing ``backticks`                      |
+-------------------------------------------------------------+
```

1.1.1.2.9.4.2.19 sys_get_config

Syntax

```
sys.sys_get_config(name,default)
```

Contents

Description

`sys_get_config` is a stored function available with the Sys Schema.

The function returns a configuration option value from the sys_config table. It takes two arguments; *name*, a configuration option name, and *default*, which is returned if the given option does not exist in the table.

Both arguments are VARCHAR(128) and can be NULL. Returns NULL if *name* is NULL, or if the given option is not found and *default* is NULL.

Examples

```
SELECT sys.sys_get_config('ps_thread_trx_info.max_length',NULL);
+-----------------------------------------------------------+
| sys.sys_get_config('ps_thread_trx_info.max_length',NULL) |
+-----------------------------------------------------------+
| 65535                                                     |
+-----------------------------------------------------------+
```

1.1.1.2.9.4.2.20 version_major

Syntax

```
sys.version_major()
```

Description

`version_major` is a stored function available with the Sys Schema.

It returns the MariaDB Server major release version.

Examples

```
SELECT VERSION(),
 sys.version_major() AS major,
 sys.version_minor() AS minor,
 sys.version_patch() AS patch;
+----------------+-------+-------+-------+
| VERSION()      | major | minor | patch |
+----------------+-------+-------+-------+
| 10.8.2-MariaDB |    10 |     8 |     2 |
+----------------+-------+-------+-------+
```

1.1.1.2.9.4.2.21 version_minor

Syntax

```
sys.version_minor()
```

Description

`version_minor` is a stored function available with the Sys Schema.

It returns the MariaDB Server minor release version.

Examples

```
SELECT VERSION(),
 sys.version_major() AS major,
 sys.version_minor() AS minor,
 sys.version_patch() AS patch;
+----------------+-------+-------+-------+
| VERSION()      | major | minor | patch |
+----------------+-------+-------+-------+
| 10.8.2-MariaDB |    10 |     8 |     2 |
+----------------+-------+-------+-------+
```

1.1.1.2.9.4.2.22 version_patch

Syntax

```
sys.version_patch()
```

Description

`version_patch` is a stored function available with the Sys Schema.

It returns the MariaDB Server patch release version.

Examples

```
SELECT VERSION(),
 sys.version_major() AS major,
 sys.version_minor() AS minor,
 sys.version_patch() AS patch;
+----------------+-------+-------+-------+
| VERSION()      | major | minor | patch |
+----------------+-------+-------+-------+
| 10.8.2-MariaDB |    10 |     8 |     2 |
+----------------+-------+-------+-------+
```

1.1.1.2.9.4.3 Sys Schema Stored Procedures

This article is currently incomplete.

The following stored procedures are available in the Sys Schema.

create_synonym_db

Takes a source db and create a synonym db with views that point to all of t...

optimizer_switch Helper Functions

Syntax optimizer_switch_on() optimizer_switch_off() optimizer_switch_choice...

ps_trace_thread

Dumps all Performance Schema data for an instrumented thread to a .dot formatted graph file.

ps_truncate_all_tables

Resets all aggregated instrumentation.

statement_performance_analyzer

Returns a report on running statements.

table_exists

Given a database and table name, returns the table type.

1.1.1.2.9.4.3.1 create_synonym_db

Syntax

```
create_synonym_db(db_name,synonym)

# db_name (VARCHAR(64))
# synonym (VARCHAR(64))
```

Description

`create_synonym_db` is a stored procedure available with the Sys Schema.

Takes a source database name *db_name* and *synonym* name and creates a synonym database with views that point to all of the tables within the source database. Useful for example for creating a synonym for the performance_schema or information_schema databases.

Returns an error if the source database doesn't exist, or the synonym already exists.

Example

```
SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| test               |
+--------------------+

CALL sys.create_synonym_db('performance_schema', 'perf');
+----------------------------------------+
| summary                                |
+----------------------------------------+
| Created 81 views in the `perf` database |
+----------------------------------------+

SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| perf               |
| performance_schema |
| sys                |
| test               |
+--------------------+

SHOW FULL TABLES FROM perf;
+-----------------------------------------------------+------------+
| Tables_in_perf                                      | Table_type |
+-----------------------------------------------------+------------+
| accounts                                            | VIEW       |
| cond_instances                                      | VIEW       |
| events_stages_current                               | VIEW       |
| events_stages_history                               | VIEW       |
| events_stages_history_long                          | VIEW       |
...
```

1.1.1.2.9.4.3.2 optimizer_switch Helper Functions

Syntax

```
optimizer_switch_on()
optimizer_switch_off()
optimizer_switch_choice("on" | "off")
```

Contents

Description

The above procedures can be used to check which optimizer_switch options are `on` or `off`. The result set is sorted according to optimizer_switch option name.

Example

```
select @@optimizer_switch\G
*************************** 1. row ***************************
index_merge=on,index_merge_union=on,index_merge_sort_union=on,index_merge_intersection=on,
index_merge_sort_intersection=off,engine_condition_pushdown=off,index_condition_pushdown=on,
derived_merge=on,derived_with_keys=on,firstmatch=on,loosescan=on,materialization=on,
in_to_exists=on,semijoin=on,partial_match_rowid_merge=on,partial_match_table_scan=on,
subquery_cache=on,mrr=off,mrr_cost_based=off,mrr_sort_keys=off,outer_join_with_cache=on,
semijoin_with_cache=on,join_cache_incremental=on,join_cache_hashed=on,join_cache_bka=on,
optimize_join_buffer_size=on,table_elimination=on,extended_keys=on,exists_to_in=on,
orderby_uses_equalities=on,condition_pushdown_for_derived=on,split_materialized=on,
condition_pushdown_for_subquery=on,rowid_filter=on,condition_pushdown_from_having=on,
not_null_range_scan=off

call sys.optimizer_switch_on();
+---------------------------------+------+
| option                          | opt  |
+---------------------------------+------+
| condition_pushdown_for_derived  | on   |
| condition_pushdown_for_subquery | on   |
| condition_pushdown_from_having  | on   |
| derived_merge                   | on   |
| derived_with_keys               | on   |
| exists_to_in                    | on   |
| extended_keys                   | on   |
| firstmatch                      | on   |
| index_condition_pushdown        | on   |
| index_merge                     | on   |
| index_merge_intersection        | on   |
| index_merge_sort_union          | on   |
| index_merge_union               | on   |
| in_to_exists                    | on   |
| join_cache_bka                  | on   |
| join_cache_hashed               | on   |
| join_cache_incremental          | on   |
| loosescan                       | on   |
| materialization                 | on   |
| optimize_join_buffer_size       | on   |
| orderby_uses_equalities         | on   |
| outer_join_with_cache           | on   |
| partial_match_rowid_merge       | on   |
| partial_match_table_scan        | on   |
| rowid_filter                    | on   |
| semijoin                        | on   |
| semijoin_with_cache             | on   |
| split_materialized              | on   |
| subquery_cache                  | on   |
| table_elimination               | on   |
+---------------------------------+------+


call sys.optimizer_switch_off();
+-------------------------------+------+
| option                        | opt  |
+-------------------------------+------+
| engine_condition_pushdown     | off  |
| index_merge_sort_intersection | off  |
| mrr                           | off  |
| mrr_cost_based                | off  |
| mrr_sort_keys                 | off  |
| not_null_range_scan           | off  |
+-------------------------------+------+
```

Notes

`sys.optimizer_switch_on()` is a shortcut for `sys.optimizer_switch_choice("on");`
`sys.optimizer_switch_off()` is a shortcut for `sys.optimizer_switch_choice("off");`

1.1.1.2.9.4.3.3 ps_trace_thread

Syntax

```
ps_trace_thread(thread_id, outfile, max_runtime, interval, start_fresh, auto_setup, debug)
```

Description

`ps_trace_thread` is a stored procedure available with the Sys Schema.

Parameters:

- thread_id INT: The thread to trace.
- outfile VARCHAR(255): Name of the .dot file to be create.
- max_runtime DECIMAL(20,2): Maximum time in seconds to collect data. Fractional seconds can be used, and NULL results in data being collected for the default sixty seconds.
- interval DECIMAL(20,2): Time in seconds to sleep between data collection. Fractional seconds can be used, and NULL results in the sleep being the default one second.
- start_fresh BOOLEAN: Whether to reset all Performance Schema data before tracing.
- auto_setup BOOLEAN: Whether to disable all other threads, enable all instruments and consumers, and reset the settings at the end of the run.
- debug BOOLEAN: Whether to include file:lineno information in the graph.

Dumps all Performance Schema data for an instrumented thread to a .dot formatted graph file (for use with the DOT graph description language). All returned result sets should be used for a complete graph.

Session binary logging is disabled during execution, by adjusting the sql_log_bin session value (note the permissions required).

Examples

```
CALL sys.ps_trace_thread(25, CONCAT('/tmp/stack-', REPLACE(NOW(), ' ', '-'), '.dot'),
  NULL, NULL, TRUE, TRUE, TRUE);

--------------------+
| summary            |
+--------------------+
| Disabled 0 threads |
+--------------------+

+-------------------------------------------+
| Info                                      |
+-------------------------------------------+
| Data collection starting for THREAD_ID = 25 |
+-------------------------------------------+

+------------------------------------------------------------+
| Info                                                       |
+------------------------------------------------------------+
| Stack trace written to /tmp/stack-2023-04-05-19:06:29.dot |
+------------------------------------------------------------+

+-------------------------------------------------------------------------+
| Convert to PDF                                                          |
+-------------------------------------------------------------------------+
| dot -Tpdf -o /tmp/stack_25.pdf /tmp/stack-2023-04-05-19:06:29.dot |
+-------------------------------------------------------------------------+

+-------------------------------------------------------------------------+
| Convert to PNG                                                          |
+-------------------------------------------------------------------------+
| dot -Tpng -o /tmp/stack_25.png /tmp/stack-2023-04-05-19:06:29.dot |
+-------------------------------------------------------------------------+
```

1.1.1.2.9.4.3.4 ps_truncate_all_tables

Syntax

```
ps_truncate_all_tables(bool display)
```

Description

`ps_truncate_all_tables` is a stored procedure available with the Sys Schema.

The procedure resets all aggregated instrumentation as a snapshot, producing a result set indicating the number of truncated tables. The boolean parameter *display* specifies whether to display each TRUNCATE TABLE statement before execution.

Examples

```
CALL sys.ps_truncate_all_tables(false);
+---------------------+
| summary             |
+---------------------+
| Truncated 44 tables |
+---------------------+
```

```
CALL sys.ps_truncate_all_tables(true);
+-----------------------------------------------------------------+
| status                                                          |
+-----------------------------------------------------------------+
| Running: TRUNCATE TABLE performance_schema.events_stages_history |
+-----------------------------------------------------------------+

...

+-------------------------------------------------------------------------------+
| status                                                                        |
+-------------------------------------------------------------------------------+
| Running: TRUNCATE TABLE performance_schema.table_lock_waits_summary_by_table |
+-------------------------------------------------------------------------------+

+---------------------+
| summary             |
+---------------------+
| Truncated 44 tables |
+---------------------+
```

1.1.1.2.9.4.3.5 statement_performance_analyzer

Syntax

```
statement_performance_analyzer(in_action,in_table, in_views)

# in_action ENUM('snapshot', 'overall', 'delta', 'create_tmp',
                 'create_table', 'save', 'cleanup')
# in_table VARCHAR(129)
# in_views SET ('with_runtimes_in_95th_percentile', 'analysis',
                'with_errors_or_warnings', 'with_full_table_scans',
                'with_sorting', 'with_temp_tables', 'custom')
```

Description

`statement_performance_analyzer` is a stored procedure available with the Sys Schema which returns a report on running statements.

The following options from the sys_config table impact the output:

- statement_performance_analyzer.limit - maximum number of rows (default 100) returned for views that have no built-in limit.
- statement_performance_analyzer.view - custom query/view to be used (default NULL). If the statement_performance_analyzer.limit configuration option is greater than 0, there can't be a LIMIT clause in the query/view definition

If the debug option is set (default OFF), the procedure will also produce debugging output.

1.1.1.2.9.4.3.6 table_exists

Syntax

```
table_exists(in_db_name,in_table_name, out_table_type)

# in_db_name VARCHAR(64)
# in_table_name VARCHAR(64)
# out_table_type ENUM('', 'BASE TABLE', 'VIEW', 'TEMPORARY')
```

Description

`table_exists` is a stored procedure available with the Sys Schema.

Given a database *in_db_name* and table name *in_table_name*, returns the table type in the OUT parameter *out_table_type*. The return value is an ENUM field containing one of:

- '' - the table does not exist
- 'BASE TABLE' - a regular table
- 'VIEW' - a view
- 'TEMPORARY' - a temporary table

Examples

```
CALL sys.table_exists('mysql', 'time_zone', @table_type); SELECT @table_type;
+-------------+
| @table_type |
+-------------+
| BASE TABLE  |
+-------------+

CALL sys.table_exists('mysql', 'user', @table_type); SELECT @table_type;
+-------------+
| @table_type |
+-------------+
| VIEW        |
+-------------+
```

1.1.1.2.9.5 mariadb_schema

Contents

1. History

`mariadb_schema` is a data type qualifier that allows one to create MariaDB native date types in an SQL_MODE that has conflicting data type translations.

`mariadb_schema` was introduced in MariaDB 10.3.24, MariaDB 10.4.14 and MariaDB 10.5.5.

For example, in SQL_MODE=ORACLE, if one creates a table with the DATE type, it will actually create a DATETIME column to match what an Oracle user is expecting. To be able to create a MariaDB DATE in Oracle mode one would have to use `mariadb_schema`:

```
CREATE TABLE t1 (d mariadb_schema.DATE);
```

`mariadb_schema` is also shown if one creates a table with `DATE` in MariaDB native mode and then does a SHOW CREATE TABLE in `ORACLE` mode:

```
SET sql_mode=DEFAULT;
CREATE OR REPLACE TABLE t1 (
  d DATE
);
SET SQL_mode=ORACLE;
SHOW CREATE TABLE t1;
+-------+-----------------------------------------------------------------+
| Table | Create Table                                                    |
+-------+-----------------------------------------------------------------+
| t1    | CREATE TABLE "t1" (
  "d" mariadb_schema.date DEFAULT NULL
) |
+-------+-----------------------------------------------------------------+
```

When the server sees the `mariadb_schema` qualifier, it disables sql_mode-specific data type translation and interprets the data type literally, so for example `mariadb_schema.DATE` is interpreted as the traditional MariaDB `DATE` data type, no matter what the current sql_mode is.

The `mariadb_schema` prefix is displayed only when the data type name would be ambiguous otherwise. The prefix is displayed together with MariaDB `DATE` when SHOW CREATE TABLE is executed in SQL_MODE=ORACLE. The prefix is not displayed when SHOW CREATE TABLE is executed in SQL_MODE=DEFAULT, or when a non-ambiguous data type is displayed.

Note, the `mariadb_schema` prefix can be used with any data type, including non-ambiguous ones:

```
CREATE OR REPLACE TABLE t1 (a mariadb_schema.INT);
SHOW CREATE TABLE t1;
+-------+-----------------------------------------------------+
| Table | Create Table                                        |
+-------+-----------------------------------------------------+
| t1    | CREATE TABLE "t1" (
  "a" int(11) DEFAULT NULL
) |
+-------+-----------------------------------------------------+
```

Currently the `mariadb_schema` prefix is only used in the following case:

- For a MariaDB native DATE type when running SHOW CREATE TABLE in Oracle mode.

History

When running with SQL_MODE=ORACLE, MariaDB server translates the data type `DATE` to `DATETIME`, for better Oracle compatibility:

```
SET SQL_mode=ORACLE;
CREATE OR REPLACE TABLE t1 (
  d DATE
);
SHOW CREATE TABLE t1;
+-------+-----------------------------------------------------+
| Table | Create Table                                        |
+-------+-----------------------------------------------------+
| t1    | CREATE TABLE "t1" (
  "d" datetime DEFAULT NULL
) |
+-------+-----------------------------------------------------+
```

Notice, `DATE` was translated to `DATETIME`.

This translation may cause some ambiguity. Suppose a user creates a table with a column of the traditional MariaDB `DATE` data type using the default sql_mode, but then switches to SQL_MODE=ORACLE and runs a SHOW CREATE TABLE statement:

```
SET sql_mode=DEFAULT;
CREATE OR REPLACE TABLE t1 (
  d DATE
);
SET SQL_mode=ORACLE;
SHOW CREATE TABLE t1;
```

Before `mariadb_schema` was introduced, the above script displayed:

```
CREATE TABLE "t1" (
  "d" date DEFAULT NULL
);
```

which had two problems:

- It was confusing for the reader: its not clear if it is the traditional MariaDB `DATE`, or is it Oracle-alike date (which is actually `DATETIME`);
- It broke replication and caused data type mismatch on the master and on the slave (see MDEV-19632).

To address this problem, starting from the mentioned versions, MariaDB uses the idea of qualified data types:

```
SET sql_mode=DEFAULT;
CREATE OR REPLACE TABLE t1 (
  d DATE
);
SET SQL_mode=ORACLE;
SHOW CREATE TABLE t1;
+-------+-----------------------------------------------------------------+
| Table | Create Table                                                    |
+-------+-----------------------------------------------------------------+
| t1    | CREATE TABLE "t1" (
  "d" mariadb_schema.date DEFAULT NULL
) |
+-------+-----------------------------------------------------------------+
```

1.1.1.2.9.6 Writing Logs Into Tables

By default, all logs are disabled or written into files. The general query log and the slow query log can also be written to special tables in the `mysql` database. During the startup, entries will always be written into files.

Note that EXPLAIN output will only be recorded if the slow query log is written to a file and not to a table.

To write logs into tables, the log_output server system variable is used. Allowed values are `FILE`, `TABLE` and `NONE`. It is possible to specify multiple values, separated with commas, to write the logs into both tables and files. `NONE` disables logging and has precedence over the other values.

So, to write logs into tables, one of the following settings can be used:

```
SET GLOBAL log_output = 'TABLE';
SET GLOBAL log_output = 'FILE,TABLE';
```

The general log will be written into the general_log table, and the slow query log will be written into the slow_log table. Only a limited set of operations are supported for those special tables. For example, direct DML statements (like `INSERT`) on those tables will fail with an error similar to the following:

```
ERROR 1556 (HY000): You can't use locks with log tables.
```

To flush data to the tables, use FLUSH TABLES instead of FLUSH LOGS.

To empty the contents of the log tables, TRUNCATE TABLE can be used.

The log tables use the CSV storage engine by default. This allows an external program to read the files if needed: normal CSV files are stored in the `mysql` subdirectory, in the data dir. However that engine is slow because it does not support indexes, so you can convert the tables to MyISAM (but not other storage engines). To do so, first temporarily disable logging:

```
SET GLOBAL general_log = 'OFF';
ALTER TABLE mysql.general_log ENGINE = MyISAM;
ALTER TABLE mysql.slow_log ENGINE = MyISAM;
SET GLOBAL general_log = @old_log_state;
```

CHECK TABLE and CHECKSUM TABLE are supported.

CREATE TABLE is supported. ALTER TABLE, RENAME TABLE and DROP TABLE are supported when logging is disabled, but log tables cannot be partitioned.

The contents of the log tables is not logged in the binary log thus cannot be replicated.

1.1.1.2.10 BINLOG

Syntax

```
BINLOG 'str'
```

Description

`BINLOG` is an internal-use statement. It is generated by the mariadb-binlog program as the printable representation of certain events in binary log files. The `'str'` value is a base 64-encoded string that the server decodes to determine the data change indicated by the corresponding event. This statement requires the SUPER privilege (<= MariaDB 10.5.1) or the BINLOG REPLAY privilege (>= MariaDB 10.5.2).

1.1.1.2.11 PURGE BINARY LOGS

Syntax

```
PURGE { BINARY | MASTER } LOGS
    { TO 'log_name' | BEFORE datetime_expr }
```

Description

The `PURGE BINARY LOGS` statement deletes all the binary log files listed in the log index file prior to the specified log file name or date. `BINARY` and `MASTER` are synonyms. Deleted log files also are removed from the list recorded in the index file, so that the given log file becomes the first in the list.

The datetime expression is in the format 'YYYY-MM-DD hh:mm:ss'.

> If a replica is active but has yet to read from a binary log file you attempt to delete, the statement will fail with an error. However, if the replica is not connected and has yet to read from a log file you delete, the file will be deleted, but the replica will be unable to continue replicating once it connects again.

This statement has no effect if the server was not started with the --log-bin option to enable binary logging.

To list the binary log files on the server, use SHOW BINARY LOGS. To see which files they are reading, use SHOW SLAVE STATUS (or SHOW REPLICA STATUS from MariaDB 10.5.1). You can only delete the files that are older than the oldest file that is used by the slaves.

To delete all binary log files, use RESET MASTER. To move to a new log file (for example if you want to remove the current log file), use FLUSH LOGS before you execute `PURGE LOGS`.

If the expire_logs_days server system variable is not set to 0, the server automatically deletes binary log files after the given number of days. From MariaDB 10.6, the binlog_expire_logs_seconds variable allows more precise control over binlog deletion, and takes precedence if both are non-zero.

Requires the SUPER privilege or, from MariaDB 10.5.2, the BINLOG ADMIN privilege, to run.

Examples

```
PURGE BINARY LOGS TO 'mariadb-bin.000063';
```

```
PURGE BINARY LOGS BEFORE '2013-04-21';
```

```
PURGE BINARY LOGS BEFORE '2013-04-22 09:55:22';
```

1.1.1.2.12 CACHE INDEX

Syntax

```
CACHE INDEX
  tbl_index_list [, tbl_index_list] ...
  IN key_cache_name

tbl_index_list:
  tbl_name [[INDEX|KEY] (index_name[, index_name] ...)]
```

Description

The `CACHE INDEX` statement assigns table indexes to a specific key cache. It is used only for MyISAM tables.

A default key cache exists and cannot be destroyed. To create more key caches, the key_buffer_size server system variable.

The associations between tables indexes and key caches are lost on server restart. To recreate them automatically, it is necessary to configure caches in a configuration file and include some `CACHE INDEX` (and optionally `LOAD INDEX`) statements in the init file.

Examples

The following statement assigns indexes from the tables t1, t2, and t3 to the key cache named hot_cache:

```
CACHE INDEX t1, t2, t3 IN hot_cache;
+---------+--------------------+----------+----------+
| Table   | Op                 | Msg_type | Msg_text |
+---------+--------------------+----------+----------+
| test.t1 | assign_to_keycache | status   | OK       |
| test.t2 | assign_to_keycache | status   | OK       |
| test.t3 | assign_to_keycache | status   | OK       |
+---------+--------------------+----------+----------+
```

Implementation (for MyISAM)

Normally CACHE INDEX should not take a long time to execute. Internally it's implemented the following way:

- Find the right key cache (under LOCK_global_system_variables)
- Open the table with a TL_READ_NO_INSERT lock.
- Flush the original key cache for the given file (under key cache lock)
- Flush the new key cache for the given file (safety)
- Move the file to the new key cache (under file share lock)

The only possible long operations are getting the locks for the table and flushing the original key cache, if there were many key blocks for the file in it.

We plan to also add CACHE INDEX for Aria tables if there is a need for this.

1.1.1.2.13 DESCRIBE

Syntax

```
{DESCRIBE | DESC} tbl_name [col_name | wild]
```

Contents

Description

`DESCRIBE` provides information about the columns in a table. It is a shortcut for `SHOW COLUMNS FROM`. These statements also display information for views.

`col_name` can be a column name, or a string containing the SQL " `%` " and " `_` " wildcard characters to obtain output only for the columns with names matching the string. There is no need to enclose the string within quotes unless it contains spaces or other special characters.

```
DESCRIBE city;
+------------+----------+------+-----+---------+----------------+
| Field      | Type     | Null | Key | Default | Extra          |
+------------+----------+------+-----+---------+----------------+
| Id         | int(11)  | NO   | PRI | NULL    | auto_increment |
| Name       | char(35) | YES  |     | NULL    |                |
| Country    | char(3)  | NO   | UNI |         |                |
| District   | char(20) | YES  | MUL |         |                |
| Population | int(11)  | YES  |     | NULL    |                |
+------------+----------+------+-----+---------+----------------+
```

The description for `SHOW COLUMNS` provides more information about the output columns.

1.1.1.2.14 EXECUTE Statement

Syntax

```
EXECUTE stmt_name
    [USING expression[, expression] ...]
```

Contents

MariaDB starting with 10.2.3

`EXECUTE` with expression as parameters was introduced in MariaDB 10.2.3. Before that one could only use variables (@var_name) as parameters.

Description

After preparing a statement with `PREPARE`, you execute it with an `EXECUTE` statement that refers to the prepared statement name. If the prepared statement contains any parameter markers, you must supply a `USING` clause that lists user variables containing the values to be bound to the parameters. Parameter values can be supplied only by user variables, and the `USING` clause must name exactly as many variables as the number of parameter markers in the statement.

You can execute a given prepared statement multiple times, passing different variables to it or setting the variables to different values before each execution.

If the specified statement has not been PREPAREd, an error similar to the following is produced:

```
ERROR 1243 (HY000): Unknown prepared statement handler (stmt_name) given to EXECUTE
```

Example

See example in PREPARE.

1.1.1.2.15 HELP Command

Syntax

```
HELP search_string
```

Description

The `HELP` command can be used in any MariaDB client, such as the mariadb command-line client, to get basic syntax help and a short description for most commands and functions.

If you provide an argument to the `HELP` command, the mariadb client uses it as a search string to access server-side help. The proper operation of this command requires that the help tables in the `mysql` database be initialized with help topic information.

If there is no match for the search string, the search fails. Use `HELP contents` to see a list of the help categories:

```
HELP contents
You asked for help about help category: "Contents"
For more information, type 'help <item>', where <item> is one of the following
categories:
   Account Management
   Administration
   Compound Statements
   Data Definition
   Data Manipulation
   Data Types
   Functions
   Functions and Modifiers for Use with GROUP BY
   Geographic Features
   Help Metadata
   Language Structure
   Plugins
   Procedures
   Sequences
   Table Maintenance
   Transactions
   User-Defined Functions
   Utility
```

If a search string matches multiple items, MariaDB shows a list of matching topics:

```
HELP drop
Many help items for your request exist.
To make a more specific request, please type 'help <item>',
where <item> is one of the following
topics:
   ALTER TABLE
   DROP DATABASE
   DROP EVENT
   DROP FUNCTION
   DROP FUNCTION UDF
   DROP INDEX
   DROP PACKAGE
   DROP PACKAGE BODY
   DROP PROCEDURE
   DROP ROLE
   DROP SEQUENCE
   DROP SERVER
   DROP TABLE
   DROP TRIGGER
   DROP USER
   DROP VIEW
```

Then you can enter a topic as the search string to see the help entry for that topic.

The help is provided with the MariaDB server and makes use of four help tables found in the `mysql` database: help_relation, help_topic, help_category and help_keyword. These tables are populated by the mariadb-install-db or `fill_help_table.sql` scripts.

1.1.1.2.16 KILL [CONNECTION | QUERY]

Syntax

```
KILL [HARD | SOFT] { {CONNECTION|QUERY} thread_id | QUERY ID query_id | USER user_name }
```

Contents

Description

Each connection to mysqld runs in a separate thread. You can see which threads are running with the `SHOW PROCESSLIST` statement and kill a thread with the `KILL thread_id` statement. `KILL` allows the optional `CONNECTION` or `QUERY` modifier:

- `KILL CONNECTION` is the same as `KILL` with no modifier: It terminates the connection associated with the given thread or query id.
- `KILL QUERY` terminates the statement that the connection thread_id is currently executing, but leaves the connection itself intact.
- `KILL QUERY ID` terminates the query by query_id, leaving the connection intact.

If a connection is terminated that has an active transaction, the transaction will be rolled back. If only a query is killed, the current transaction will stay active. See also idle_transaction_timeout.

If you have the PROCESS privilege, you can see all threads. If you have the SUPER privilege, or, from MariaDB 10.5.2, the CONNECTION ADMIN privilege, you can kill all threads and statements. Otherwise, you can see and kill only your own threads and statements.

> Killing queries that repair or create indexes on MyISAM and Aria tables may result in corrupted tables. Use the `SOFT` option to avoid this!

The `HARD` option (default) kills a command as soon as possible. If you use `SOFT`, then critical operations that may leave a table in an inconsistent state will not be interrupted. Such operations include `REPAIR` and `INDEX` creation for MyISAM and Aria tables (REPAIR TABLE, OPTIMIZE TABLE).

`KILL ... USER username` will kill all connections/queries for a given user. `USER` can be specified one of the following ways:

- username (Kill without regard to hostname)
- username@hostname
- CURRENT_USER or CURRENT_USER()

If you specify a thread id and that thread does not exist, you get the following error:

```
ERROR 1094 (HY000): Unknown thread id: <thread_id>
```

If you specify a query id that doesn't exist, you get the following error:

```
ERROR 1957 (HY000): Unknown query id: <query_id>
```

However, if you specify a user name, no error is issued for non-connected (or even non-existing) users. To check if the connection/query has been killed, you can use the ROW_COUNT() function.

A client whose connection is killed receives the following error:

```
ERROR 1317 (70100): Query execution was interrupted
```

To obtain a list of existing sessions, use the SHOW PROCESSLIST statement or query the Information Schema

PROCESSLIST table.

Note: You cannot use `KILL` with the Embedded MariaDB Server library because the embedded server merely runs inside the threads of the host application. It does not create any connection threads of its own.

Note: You can also use `mariadb-admin kill thread_id [,thread_id...]` to kill connections. To get a list of running queries, use `mariadb-admin processlist`. See mariadb-admin.

Percona Toolkit contains a program, pt-kill that can be used to automatically kill connections that match certain criteria. For example, it can be used to terminate idle connections, or connections that have been busy for more than 60 seconds.

1.1.1.2.17 LOAD INDEX

Syntax

```
LOAD INDEX INTO CACHE
  tbl_index_list [, tbl_index_list] ...

tbl_index_list:
  tbl_name
    [[INDEX|KEY] (index_name[, index_name] ...)]
    [IGNORE LEAVES]
```

Description

The `LOAD INDEX INTO CACHE` statement preloads a table index into the key cache to which it has been assigned by an explicit `CACHE INDEX` statement, or into the default key cache otherwise. `LOAD INDEX INTO CACHE` is used only for MyISAM or Aria tables.

The `IGNORE LEAVES` modifier causes only blocks for the nonleaf nodes of the index to be preloaded.

1.1.1.2.18 RESET

Syntax

```
RESET reset_option [, reset_option] ...
```

Description

The `RESET` statement is used to clear the state of various server operations. You must have the `RELOAD privilege` to execute `RESET`.

`RESET` acts as a stronger version of the FLUSH statement.

The different `RESET` options are:

Option	Description
SLAVE ["connection_name"] [ALL]	Deletes all relay logs from the slave and reset the replication position in the master binary log.
MASTER	Deletes all old binary logs, makes the binary index file (--log-bin-index) empty and creates a new binary log file. This is useful when you want to reset the master to an initial state. If you want to just delete old, not used binary logs, you should use the PURGE BINARY LOGS command.
QUERY CACHE	Removes all queries from the query cache. See also FLUSH QUERY CACHE.

1.1.1.2.19 SHUTDOWN

Contents

Syntax

```
SHUTDOWN [WAIT FOR ALL { SLAVES | REPLICAS } ]
```

Description

The `SHUTDOWN` command shuts the server down.

WAIT FOR ALL REPLICAS / SLAVES

MariaDB starting with 10.4.4

The `WAIT FOR ALL SLAVES` option was first added in MariaDB 10.4.4. `WAIT FOR ALL REPLICAS` has been a synonym since MariaDB 10.5.1.

When a primary server is shutdown and it goes through the normal shutdown process, the primary kills client threads in random order. By default, the primary also considers its binary log dump threads to be regular client threads. As a consequence, the binary log dump threads can be killed while client threads still exist, and this means that data can be written on the primary during a normal shutdown that won't be replicated. This is true even if semi-synchronous replication is being used.

In MariaDB 10.4 and later, this problem can be solved by shutting down the server with the SHUTDOWN command and by providing the `WAIT FOR ALL REPLICAS` / `WAIT FOR ALL SLAVES` option to the command. For example:

```
SHUTDOWN WAIT FOR ALL REPLICAS;
```

When the `WAIT FOR ALL REPLICAS` option is provided, the server only kills its binary log dump threads after all client threads have been killed, and it only completes the shutdown after the last binary log has been sent to all connected replicas.

See Replication Threads: Binary Log Dump Threads and the Shutdown Process for more information.

Required Permissions

One must have a `SHUTDOWN` privilege (see GRANT) to use this command. It is the same privilege one needs to use the mariadb-admin shutdown command.

Shutdown for Upgrades

If you are doing a shutdown to migrate to another major version of MariaDB, please ensure that the innodb_fast_shutdown variable is not 2 (fast crash shutdown). The default of this variable is 1.

Example

The following example shows how to create an event which turns off the server at a certain time:

```
CREATE EVENT `test`.`shutd`
    ON SCHEDULE
        EVERY 1 DAY
        STARTS '2014-01-01 20:00:00'
    COMMENT 'Shutdown Maria when the office is closed'
DO BEGIN
    SHUTDOWN;
END;
```

Other Ways to Stop mariadbd

You can use the mariadb-admin shutdown command to take down mariadbd cleanly.

You can also use the system kill command on Unix with signal SIGTERM (15)

```
kill -SIGTERM pid-of-mariadbd-process
```

You can find the process number of the server process in the file that ends with `.pid` in your data directory.

The above is identical to `mariadb-admin shutdown` .

On windows you should use:

```
NET STOP MariaDB
```

1.1.1.2.20 USE [DATABASE]

Syntax

```
USE db_name
```

In MariaDB 11.3 one can also use

```
USE DATABASE db_name;
```

Description

The `'USE db_name'` statement tells MariaDB to use the `db_name` database as the default (current) database for subsequent statements. The database remains the default until the end of the session or another `USE` statement is issued:

```
USE db1;
SELECT COUNT(*) FROM mytable;   # selects from db1.mytable
USE db2;
SELECT COUNT(*) FROM mytable;   # selects from db2.mytable
```

The DATABASE() function (SCHEMA() is a synonym) returns the default database.

Another way to set the default database is specifying its name at mariadb command line client startup.

One cannot use `USE DATABASE` to a database one has no privileges to. The reason is that a user with no privileges to a database should not be able to know if a database exists or not.

1.1.1.3 Data Definition

SQL Commands for defining data, such as ALTER, CREATE, DROP, RENAME etc.

CREATE

Articles on the various CREATE statements.

ALTER

The various ALTER statements in MariaDB.

DROP
Articles on various DROP commands.

Atomic DDL
Making DDL Atomic/Crash-safe.

CONSTRAINT
Define a CHECK or FOREIGN KEY constraint.

MERGE
Allows you to access a collection of identical MyISAM tables as one.

RENAME TABLE
Change a table's name.

TRUNCATE TABLE
DROP and re-CREATE a table.

There are 3 related questions.

1.1.1.3.1 CREATE

Articles on the various CREATE statements.

CREATE DATABASE
Create a database.

CREATE EVENT
Create and schedule a new event.

CREATE FUNCTION
Creates a stored function.

CREATE FUNCTION UDF
Create a user-defined function.

CREATE INDEX
Create an index on one or more columns.

CREATE LOGFILE GROUP
The CREATE LOGFILE GROUP statement is not supported by MariaDB. It was orig...

CREATE PACKAGE
Create a stored package, in Oracle-mode.

CREATE PACKAGE BODY
Creates the package body for a stored package.

CREATE PROCEDURE
Creates a stored procedure.

CREATE ROLE
Add new roles.

CREATE SEQUENCE
Creates a sequence that generates new values when called with NEXT VALUE FOR.

CREATE SERVER
Define a server.

CREATE TABLE
Creates a new table.

CREATE TABLESPACE
CREATE TABLESPACE is not available in MariaDB.

CREATE TRIGGER
Create a new trigger.

CREATE USER
Create new MariaDB accounts.

CREATE VIEW
Create or replace a view.

Silent Column Changes
MariaDB silently changes column specifications in certain situations.

Generated (Virtual and Persistent/Stored) Columns
Generated (virtual and persistent/stored) columns.

Invisible Columns
Invisible columns are hidden in certain contexts.

There are 1 related questions.

1.1.1.3.1.1 CREATE DATABASE

Syntax

```
CREATE [OR REPLACE] {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
    [create_specification] ...

create_specification:
    [DEFAULT] CHARACTER SET [=] charset_name
  | [DEFAULT] COLLATE [=] collation_name
  | COMMENT [=] 'comment'
```

Contents

Description

`CREATE DATABASE` creates a database with the given name. To use this statement, you need the CREATE privilege for the database. `CREATE SCHEMA` is a synonym for `CREATE DATABASE`.

For valid identifiers to use as database names, see Identifier Names.

OR REPLACE

If the optional `OR REPLACE` clause is used, it acts as a shortcut for:

```
DROP DATABASE IF EXISTS db_name;
CREATE DATABASE db_name ...;
```

IF NOT EXISTS

When the `IF NOT EXISTS` clause is used, MariaDB will return a warning instead of an error if the specified database already exists.

COMMENT

MariaDB starting with 10.5.0

From MariaDB 10.5.0, it is possible to add a comment of a maximum of 1024 bytes. If the comment length exceeds this length, a error/warning code 4144 is thrown. The database comment is also added to the db.opt file, as well as to the information_schema.schemata table.

Examples

```
CREATE DATABASE db1;
Query OK, 1 row affected (0.18 sec)

CREATE DATABASE db1;
ERROR 1007 (HY000): Can't create database 'db1'; database exists

CREATE OR REPLACE DATABASE db1;
Query OK, 2 rows affected (0.00 sec)

CREATE DATABASE IF NOT EXISTS db1;
Query OK, 1 row affected, 1 warning (0.01 sec)

SHOW WARNINGS;
+-------+------+----------------------------------------------+
| Level | Code | Message                                      |
+-------+------+----------------------------------------------+
| Note  | 1007 | Can't create database 'db1'; database exists |
+-------+------+----------------------------------------------+
```

Setting the character sets and collation. See Setting Character Sets and Collations for more details.

```
CREATE DATABASE czech_slovak_names
  CHARACTER SET = 'keybcs2'
  COLLATE = 'keybcs2_bin';
```

Comments, from MariaDB 10.5.0:

```
CREATE DATABASE presentations COMMENT 'Presentations for conferences';
```

1.1.1.3.1.2 CREATE EVENT

Syntax

```
CREATE [OR REPLACE]
    [DEFINER = { user | CURRENT_USER | role | CURRENT_ROLE }]
    EVENT
    [IF NOT EXISTS]
    event_name
    ON SCHEDULE schedule
    [ON COMPLETION [NOT] PRESERVE]
    [ENABLE | DISABLE | DISABLE ON SLAVE]
    [COMMENT 'comment']
    DO sql_statement;

schedule:
    AT timestamp [+ INTERVAL interval] ...
  | EVERY interval
    [STARTS timestamp [+ INTERVAL interval] ...]
    [ENDS timestamp [+ INTERVAL interval] ...]

interval:
    quantity {YEAR | QUARTER | MONTH | DAY | HOUR | MINUTE |
              WEEK | SECOND | YEAR_MONTH | DAY_HOUR | DAY_MINUTE |
              DAY_SECOND | HOUR_MINUTE | HOUR_SECOND | MINUTE_SECOND}
```

Contents

Description

This statement creates and schedules a new event. It requires the `EVENT` privilege for the schema in which the event is to be created.

The minimum requirements for a valid CREATE EVENT statement are as follows:

- The keywords `CREATE EVENT` plus an event name, which uniquely identifies the event in the current schema. (Prior to MySQL 5.1.12, the event name needed to be unique only among events created by the same user on a given database.)
- An `ON SCHEDULE` clause, which determines when and how often the event executes.
- A `DO` clause, which contains the SQL statement to be executed by an event.

Here is an example of a minimal `CREATE EVENT` statement:

```
CREATE EVENT myevent
    ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 1 HOUR
    DO
      UPDATE myschema.mytable SET mycol = mycol + 1;
```

The previous statement creates an event named myevent. This event executes once — one hour following its creation — by running an SQL statement that increments the value of the myschema.mytable table's mycol column by 1.

The event_name must be a valid MariaDB identifier with a maximum length of 64 characters. It may be delimited using back ticks, and may be qualified with the name of a database schema. An event is associated with both a MariaDB user (the definer) and a schema, and its name must be unique among names of events within that schema. In general, the rules governing event names are the same as those for names of stored routines. See Identifier Names.

If no schema is indicated as part of event_name, the default (current) schema is assumed.

For valid identifiers to use as event names, see Identifier Names.

OR REPLACE

The `OR REPLACE` clause was included in MariaDB 10.1.4. If used and the event already exists, instead of an error being returned, the existing event will be dropped and replaced by the newly defined event.

IF NOT EXISTS

If the `IF NOT EXISTS` clause is used, MariaDB will return a warning instead of an error if the event already exists. Cannot be used together with OR REPLACE.

ON SCHEDULE

The `ON SCHEDULE` clause can be used to specify when the event must be triggered.

AT

If you want to execute the event only once (one time event), you can use the `AT` keyword, followed by a timestamp. If you use `CURRENT_TIMESTAMP`, the event acts as soon as it is created. As a convenience, you can add one or more intervals to that timestamp. You can also specify a timestamp in the past, so that the event is stored but not triggered, until you modify it via ALTER EVENT.

The following example shows how to create an event that will be triggered tomorrow at a certain time:

```
CREATE EVENT example
ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 1 DAY + INTERVAL 3 HOUR
DO something;
```

You can also specify that an event must be triggered at a regular interval (recurring event). In such cases, use the `EVERY` clause followed by the interval.

If an event is recurring, you can specify when the first execution must happen via the `STARTS` clause and a maximum time for the last execution via the `ENDS` clause. `STARTS` and `ENDS` clauses are followed by a timestamp and, optionally, one or more intervals. The `ENDS` clause can specify a timestamp in the past, so that the event is stored but not executed until you modify it via ALTER EVENT.

In the following example, next month a recurring event will be triggered hourly for a week:

```
CREATE EVENT example
ON SCHEDULE EVERY 1 HOUR
STARTS CURRENT_TIMESTAMP + INTERVAL 1 MONTH
ENDS CURRENT_TIMESTAMP + INTERVAL 1 MONTH + INTERVAL 1 WEEK
DO some_task;
```

Intervals consist of a quantity and a time unit. The time units are the same used for other staments and time functions, except that you can't use microseconds for events. For simple time units, like `HOUR` or `MINUTE`, the quantity is an integer number, for example '10 MINUTE'. For composite time units, like `HOUR_MINUTE` or `HOUR_SECOND`, the quantity must be a string with all involved simple values and their separators, for example '2:30' or '2:30:30'.

ON COMPLETION [NOT] PRESERVE

The `ON COMPLETION` clause can be used to specify if the event must be deleted after its last execution (that is, after its `AT` or `ENDS` timestamp is past). By default, events are dropped when they are expired. To explicitly state that this is the desired behaviour, you can use `ON COMPLETION NOT PRESERVE`. Instead, if you want the event to be preserved, you can use `ON COMPLETION PRESERVE`.

In you specify `ON COMPLETION NOT PRESERVE`, and you specify a timestamp in the past for `AT` or `ENDS` clause, the event will be immediatly dropped. In such cases, you will get a Note 1558: "Event execution time is in the past and ON COMPLETION NOT PRESERVE is set. The event was dropped immediately after creation".

ENABLE/DISABLE/DISABLE ON SLAVE

Events are `ENABLE`d by default. If you want to stop MariaDB from executing an event, you may specify `DISABLE`. When it is ready to be activated, you may enable it using `ALTER EVENT`. Another option is `DISABLE ON SLAVE`, which indicates that an event was created on a master and has been replicated to the slave, which is prevented from executing the event. If `DISABLE ON SLAVE` is specifically set, the event will be disabled everywhere. It will not be executed on the mater or the slaves.

COMMENT

The `COMMENT` clause may be used to set a comment for the event. Maximum length for comments is 64 characters. The comment is a string, so it must be quoted. To see events comments, you can query the INFORMATION_SCHEMA.EVENTS table (the column is named `EVENT_COMMENT`).

Examples

Minimal `CREATE EVENT` statement:

```
CREATE EVENT myevent
    ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 1 HOUR
    DO
      UPDATE myschema.mytable SET mycol = mycol + 1;
```

An event that will be triggered tomorrow at a certain time:

```
CREATE EVENT example
ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 1 DAY + INTERVAL 3 HOUR
DO something;
```

Next month a recurring event will be triggered hourly for a week:

```
CREATE EVENT example
ON SCHEDULE EVERY 1 HOUR
STARTS CURRENT_TIMESTAMP + INTERVAL 1 MONTH
ENDS CURRENT_TIMESTAMP + INTERVAL 1 MONTH + INTERVAL 1 WEEK
DO some_task;
```

OR REPLACE and IF NOT EXISTS:

```
CREATE EVENT myevent
    ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 1 HOUR
    DO
      UPDATE myschema.mytable SET mycol = mycol + 1;
ERROR 1537 (HY000): Event 'myevent' already exists

CREATE OR REPLACE EVENT myevent
    ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 1 HOUR
    DO
      UPDATE myschema.mytable SET mycol = mycol + 1;;
Query OK, 0 rows affected (0.00 sec)

CREATE EVENT IF NOT EXISTS myevent
    ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 1 HOUR
    DO
      UPDATE myschema.mytable SET mycol = mycol + 1;
Query OK, 0 rows affected, 1 warning (0.00 sec)

 SHOW WARNINGS;
+-------+------+---------------------------------+
| Level | Code | Message                         |
+-------+------+---------------------------------+
| Note  | 1537 | Event 'myevent' already exists |
+-------+------+---------------------------------+
```

1.1.1.3.1.3 CREATE FUNCTION

Syntax

```
CREATE [OR REPLACE]
    [DEFINER = {user | CURRENT_USER | role | CURRENT_ROLE }]
    [AGGREGATE] FUNCTION [IF NOT EXISTS] func_name ([func_parameter[,...]])
    RETURNS type
    [characteristic ...]
    RETURN func_body

func_parameter:
    [ IN | OUT | INOUT | IN OUT ]  param_name type

type:
    Any valid MariaDB data type

characteristic:
    LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
  | COMMENT 'string'

func_body:
    Valid SQL procedure statement
```

Contents

Description

Use the `CREATE FUNCTION` statement to create a new stored function. You must have the CREATE ROUTINE database privilege to use `CREATE FUNCTION`. A function takes any number of arguments and returns a value from the function body. The function body can be any valid SQL expression as you would use, for example, in any select expression. If you have the appropriate privileges, you can call the function exactly as you would any built-in function. See Security below for details on privileges.

You can also use a variant of the `CREATE FUNCTION` statement to install a user-defined function (UDF) defined by a plugin. See CREATE FUNCTION (UDF) for details.

You can use a SELECT statement for the function body by enclosing it in parentheses, exactly as you would to use a subselect for any other expression. The `SELECT` statement must return a single value. If more than one column is returned when the function is called, error 1241 results. If more than one row is returned when the function is called, error 1242 results. Use a `LIMIT` clause to ensure only one row is returned.

You can also replace the `RETURN` clause with a BEGIN...END compound statement. The compound statement must contain a `RETURN` statement. When the function is called, the `RETURN` statement immediately returns its result, and any statements after `RETURN` are effectively ignored.

By default, a function is associated with the current database. To associate the function explicitly with a given database, specify the fully-qualified name as *`db_name.func_name`* when you create it. If the function name is the same as the name of a built-in function, you must use the fully qualified name when you call it.

The parameter list enclosed within parentheses must always be present. If there are no parameters, an empty parameter list

of () should be used. Parameter names are not case sensitive.

Each parameter can be declared to use any valid data type, except that the COLLATE attribute cannot be used.

For valid identifiers to use as function names, see Identifier Names.

IN | OUT | INOUT | IN OUT

MariaDB starting with 10.8.0

The function parameter qualifiers for `IN`, `OUT`, `INOUT`, and `IN OUT` were added in a 10.8.0 preview release. Prior to 10.8.0 quantifiers were supported only in procedures.

`OUT`, `INOUT` and its equivalent `IN OUT`, are only valid if called from `SET` and not `SELECT`. These quantifiers are especially useful for creating functions with more than one return value. This allows functions to be more complex and nested.

```
DELIMITER $$
CREATE FUNCTION add_func3(IN a INT, IN b INT, OUT c INT) RETURNS INT
BEGIN
  SET c = 100;
  RETURN a + b;
END;
$$
DELIMITER ;

SET @a = 2;
SET @b = 3;
SET @c = 0;
SET @res= add_func3(@a, @b, @c);

SELECT add_func3(@a, @b, @c);
ERROR 4186 (HY000): OUT or INOUT argument 3 for function add_func3 is not allowed here

DELIMITER $$
CREATE FUNCTION add_func4(IN a INT, IN b INT, d INT) RETURNS INT
BEGIN
  DECLARE c, res INT;
  SET res = add_func3(a, b, c) + d;
  if (c > 99) then
    return  3;
  else
    return res;
  end if;
END;
$$

DELIMITER ;

SELECT add_func4(1,2,3);
+------------------+
| add_func4(1,2,3) |
+------------------+
|                3 |
+------------------+
```

AGGREGATE

It is possible to create stored aggregate functions as well. See Stored Aggregate Functions for details.

RETURNS

The `RETURNS` clause specifies the return type of the function. `NULL` values are permitted with all return types.

What happens if the `RETURN` clause returns a value of a different type? It depends on the SQL_MODE in effect at the moment of the function creation.

If the SQL_MODE is strict (STRICT_ALL_TABLES or STRICT_TRANS_TABLES flags are specified), a 1366 error will be produced.

Otherwise, the value is coerced to the proper type. For example, if a function specifies an `ENUM` or `SET` value in the `RETURNS` clause, but the `RETURN` clause returns an integer, the value returned from the function is the string for the

corresponding `ENUM` member of set of `SET` members.

MariaDB stores the SQL_MODE system variable setting that is in effect at the time a routine is created, and always executes the routine with this setting in force, regardless of the server SQL mode in effect when the routine is invoked.

LANGUAGE SQL

`LANGUAGE SQL` is a standard SQL clause, and it can be used in MariaDB for portability. However that clause has no meaning, because SQL is the only supported language for stored functions.

A function is deterministic if it can produce only one result for a given list of parameters. If the result may be affected by stored data, server variables, random numbers or any value that is not explicitly passed, then the function is not deterministic. Also, a function is non-deterministic if it uses non-deterministic functions like NOW() or CURRENT_TIMESTAMP(). The optimizer may choose a faster execution plan if it known that the function is deterministic. In such cases, you should declare the routine using the `DETERMINISTIC` keyword. If you want to explicitly state that the function is not deterministic (which is the default) you can use the `NOT DETERMINISTIC` keywords.

If you declare a non-deterministic function as `DETERMINISTIC`, you may get incorrect results. If you declare a deterministic function as `NOT DETERMINISTIC`, in some cases the queries will be slower.

OR REPLACE

If the optional `OR REPLACE` clause is used, it acts as a shortcut for:

```
DROP FUNCTION IF EXISTS function_name;
CREATE FUNCTION function_name ...;
```

with the exception that any existing privileges for the function are not dropped.

IF NOT EXISTS

If the IF NOT EXISTS clause is used, MariaDB will return a warning instead of an error if the function already exists. Cannot be used together with OR REPLACE.

[NOT] DETERMINISTIC

The `[NOT] DETERMINISTIC` clause also affects binary logging, because the `STATEMENT` format can not be used to store or replicate non-deterministic statements.

`CONTAINS SQL`, `NO SQL`, `READS SQL DATA`, and `MODIFIES SQL DATA` are informative clauses that tell the server what the function does. MariaDB does not check in any way whether the specified clause is correct. If none of these clauses are specified, `CONTAINS SQL` is used by default.

MODIFIES SQL DATA

`MODIFIES SQL DATA` means that the function contains statements that may modify data stored in databases. This happens if the function contains statements like DELETE, UPDATE, INSERT, REPLACE or DDL.

READS SQL DATA

`READS SQL DATA` means that the function reads data stored in databases, but does not modify any data. This happens if SELECT statements are used, but there no write operations are executed.

CONTAINS SQL

`CONTAINS SQL` means that the function contains at least one SQL statement, but it does not read or write any data stored in a database. Examples include SET or DO.

NO SQL

`NO SQL` means nothing, because MariaDB does not currently support any language other than SQL.

Oracle Mode

A subset of Oracle's PL/SQL language is supported in addition to the traditional SQL/PSM-based MariaDB syntax. See Oracle mode for details on changes when running Oracle mode.

Security

You must have the EXECUTE privilege on a function to call it. MariaDB automatically grants the `EXECUTE` and `ALTER ROUTINE` privileges to the account that called `CREATE FUNCTION`, even if the `DEFINER` clause was used.

Each function has an account associated as the definer. By default, the definer is the account that created the function. Use the `DEFINER` clause to specify a different account as the definer. You must have the SUPER privilege, or, from MariaDB 10.5.2, the SET USER privilege, to use the `DEFINER` clause. See Account Names for details on specifying accounts.

The `SQL SECURITY` clause specifies what privileges are used when a function is called. If `SQL SECURITY` is `INVOKER`, the function body will be evaluated using the privileges of the user calling the function. If `SQL SECURITY` is `DEFINER`, the function body is always evaluated using the privileges of the definer account. `DEFINER` is the default.

This allows you to create functions that grant limited access to certain data. For example, say you have a table that stores some employee information, and that you've granted `SELECT` privileges only on certain columns to the user account `roger`.

```
CREATE TABLE employees (name TINYTEXT, dept TINYTEXT, salary INT);
GRANT SELECT (name, dept) ON employees TO roger;
```

To allow the user the get the maximum salary for a department, define a function and grant the `EXECUTE` privilege:

```
CREATE FUNCTION max_salary (dept TINYTEXT) RETURNS INT RETURN
  (SELECT MAX(salary) FROM employees WHERE employees.dept = dept);
GRANT EXECUTE ON FUNCTION max_salary TO roger;
```

Since `SQL SECURITY` defaults to `DEFINER`, whenever the user `roger` calls this function, the subselect will execute with your privileges. As long as you have privileges to select the salary of each employee, the caller of the function will be able to get the maximum salary for each department without being able to see individual salaries.

Character sets and collations

Function return types can be declared to use any valid character set and collation. If used, the COLLATE attribute needs to be preceded by a CHARACTER SET attribute.

If the character set and collation are not specifically set in the statement, the database defaults at the time of creation will be used. If the database defaults change at a later stage, the stored function character set/collation will not be changed at the same time; the stored function needs to be dropped and recreated to ensure the same character set/collation as the database is used.

Examples

The following example function takes a parameter, performs an operation using an SQL function, and returns the result.

```
CREATE FUNCTION hello (s CHAR(20))
    RETURNS CHAR(50) DETERMINISTIC
    RETURN CONCAT('Hello, ',s,'!');

SELECT hello('world');
+----------------+
| hello('world') |
+----------------+
| Hello, world!  |
+----------------+
```

You can use a compound statement in a function to manipulate data with statements like `INSERT` and `UPDATE`. The following example creates a counter function that uses a temporary table to store the current value. Because the compound statement contains statements terminated with semicolons, you have to first change the statement delimiter with the `DELIMITER` statement to allow the semicolon to be used in the function body. See Delimiters in the mariadb client for more.

```
CREATE TEMPORARY TABLE counter (c INT);
INSERT INTO counter VALUES (0);
DELIMITER //
CREATE FUNCTION counter () RETURNS INT
  BEGIN
    UPDATE counter SET c = c + 1;
    RETURN (SELECT c FROM counter LIMIT 1);
  END //
DELIMITER ;
```

Character set and collation:

```
CREATE FUNCTION hello2 (s CHAR(20))
  RETURNS CHAR(50) CHARACTER SET 'utf8' COLLATE 'utf8_bin' DETERMINISTIC
  RETURN CONCAT('Hello, ',s,'!');
```

1.1.1.3.1.4 CREATE FUNCTION UDF

Syntax

```
CREATE [OR REPLACE] [AGGREGATE] FUNCTION [IF NOT EXISTS] function_name
    RETURNS {STRING|INTEGER|REAL|DECIMAL}
    SONAME shared_library_name
```

Contents

Description

A user-defined function (UDF) is a way to extend MariaDB with a new function that works like a native (built-in) MariaDB function such as ABS() or CONCAT().

`function_name` is the name that should be used in SQL statements to invoke the function.

To create a function, you must have the INSERT privilege for the mysql database. This is necessary because `CREATE FUNCTION` adds a row to the mysql.func system table that records the function's name, type, and shared library name. If you do not have this table, you should run the mariadb-upgrade command to create it.

UDFs need to be written in C, C++ or another language that uses C calling conventions, MariaDB needs to have been dynamically compiled, and your operating system must support dynamic loading.

For an example, see `sql/udf_example.cc` in the source tree. For a collection of existing UDFs see http://www.mysqludf.org/.

Statements making use of user-defined functions are not safe for replication.

For creating a stored function as opposed to a user-defined function, see CREATE FUNCTION.

For valid identifiers to use as function names, see Identifier Names.

RETURNS

The `RETURNS` clause indicates the type of the function's return value, and can be one of STRING, INTEGER, REAL or DECIMAL. `DECIMAL` functions currently return string values and should be written like STRING functions.

shared_library_name

`shared_library_name` is the basename of the shared object file that contains the code that implements the function. The file must be located in the plugin directory. This directory is given by the value of the plugin_dir system variable. Note that before MariaDB/MySQL 5.1, the shared object could be located in any directory that was searched by your system's dynamic linker.

AGGREGATE

Aggregate functions are summary functions such as SUM() and AVG().

> **MariaDB starting with 10.4**
> Aggregate UDF functions can be used as window functions.

OR REPLACE

> **MariaDB starting with 10.1.3**
> The `OR REPLACE` clause was added in MariaDB 10.1.3

If the optional `OR REPLACE` clause is used, it acts as a shortcut for:

```
DROP FUNCTION IF EXISTS function_name;
CREATE FUNCTION name ...;
```

IF NOT EXISTS

> **MariaDB starting with 10.1.3**
> The `IF NOT EXISTS` clause was added in MariaDB 10.1.3

When the IF NOT EXISTS clause is used, MariaDB will return a warning instead of an error if the specified function already exists. Cannot be used together with OR REPLACE.

Upgrading a UDF

To upgrade the UDF's shared library, first run a DROP FUNCTION statement, then upgrade the shared library and finally run the CREATE FUNCTION statement. If you upgrade without following this process, you may crash the server.

Examples

```
CREATE FUNCTION jsoncontains_path RETURNS integer SONAME 'ha_connect.so';
Query OK, 0 rows affected (0.00 sec)
```

OR REPLACE and IF NOT EXISTS:

```
CREATE FUNCTION jsoncontains_path RETURNS integer SONAME 'ha_connect.so';
ERROR 1125 (HY000): Function 'jsoncontains_path' already exists

CREATE OR REPLACE FUNCTION jsoncontains_path RETURNS integer SONAME 'ha_connect.so';
Query OK, 0 rows affected (0.00 sec)

CREATE FUNCTION IF NOT EXISTS jsoncontains_path RETURNS integer SONAME 'ha_connect.so';
Query OK, 0 rows affected, 1 warning (0.00 sec)

SHOW WARNINGS;
+-------+------+----------------------------------------------+
| Level | Code | Message                                      |
+-------+------+----------------------------------------------+
| Note  | 1125 | Function 'jsoncontains_path' already exists |
+-------+------+----------------------------------------------+
```

1.1.1.3.1.5 CREATE INDEX

Syntax

```
CREATE [OR REPLACE] [UNIQUE|FULLTEXT|SPATIAL] INDEX
  [IF NOT EXISTS] index_name
    [index_type]
    ON tbl_name (index_col_name,...)
    [WAIT n | NOWAIT]
    [index_option]
    [algorithm_option | lock_option] ...

index_col_name:
    col_name [(length)] [ASC | DESC]

index_type:
    USING {BTREE | HASH | RTREE}

index_option:
    [ KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'
  | CLUSTERING={YES| NO} ]
  [ IGNORED | NOT IGNORED ]

algorithm_option:
    ALGORITHM [=] {DEFAULT|INPLACE|COPY|NOCOPY|INSTANT}

lock_option:
    LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}
```

Contents

Description

The *CREATE INDEX* statement is used to add indexes to a table. Indexes can be created at the same as the table, with the [[|create-table|CREATE TABLE]] statement. In some cases, such as for InnoDB primary keys, doing so during creation is preferable, as adding a primary key will involve rebuilding the table.

The statement is mapped to an ALTER TABLE statement to create indexes. See ALTER TABLE. CREATE INDEX cannot be used to create a PRIMARY KEY; use ALTER TABLE instead.

If another connection is using the table, a metadata lock is active, and this statement will wait until the lock is released. This is also true for non-transactional tables.

Another shortcut, DROP INDEX, allows the removal of an index.

For valid identifiers to use as index names, see Identifier Names.

Note that KEY_BLOCK_SIZE is currently ignored in CREATE INDEX, although it is included in the output of SHOW CREATE TABLE.

Privileges

Executing the `CREATE INDEX` statement requires the `INDEX` privilege for the table or the database.

Online DDL

Online DDL is supported with the ALGORITHM and LOCK clauses.

See InnoDB Online DDL Overview for more information on online DDL with InnoDB.

CREATE OR REPLACE INDEX

If the `OR REPLACE` clause is used and if the index already exists, then instead of returning an error, the server will drop the existing index and replace it with the newly defined index.

CREATE INDEX IF NOT EXISTS

If the `IF NOT EXISTS` clause is used, then the index will only be created if an index with the same name does not already exist. If the index already exists, then a warning will be triggered by default.

Index Definitions

See CREATE TABLE: Index Definitions for information about index definitions.

WAIT/NOWAIT

Set the lock wait timeout. See WAIT and NOWAIT.

ALGORITHM

See ALTER TABLE: ALGORITHM for more information.

LOCK

See ALTER TABLE: LOCK for more information.

Progress Reporting

MariaDB provides progress reporting for `CREATE INDEX` statement for clients that support the new progress reporting protocol. For example, if you were using the mariadb client, then the progress report might look like this::

```
CREATE INDEX i ON tab (num);
Stage: 1 of 2 'copy to tmp table'    46% of stage
```

The progress report is also shown in the output of the SHOW PROCESSLIST statement and in the contents of the information_schema.PROCESSLIST table.

See Progress Reporting for more information.

WITHOUT OVERLAPS

MariaDB starting with 10.5.3

The WITHOUT OVERLAPS clause allows one to constrain a primary or unique index such that application-time periods cannot overlap.

Examples

Creating a unique index:

```
CREATE UNIQUE INDEX HomePhone ON Employees(Home_Phone);
```

OR REPLACE and IF NOT EXISTS:

```
CREATE INDEX xi ON xx5 (x);
Query OK, 0 rows affected (0.03 sec)

CREATE INDEX xi ON xx5 (x);
ERROR 1061 (42000): Duplicate key name 'xi'

CREATE OR REPLACE INDEX xi ON xx5 (x);
Query OK, 0 rows affected (0.03 sec)

CREATE INDEX IF NOT EXISTS xi ON xx5 (x);
Query OK, 0 rows affected, 1 warning (0.00 sec)

SHOW WARNINGS;
+-------+------+-------------------------+
| Level | Code | Message                 |
+-------+------+-------------------------+
| Note  | 1061 | Duplicate key name 'xi' |
+-------+------+-------------------------+
```

From MariaDB 10.5.3, creating a unique index for an application-time period table with a WITHOUT OVERLAPS constraint:

```
CREATE UNIQUE INDEX u ON rooms (room_number, p WITHOUT OVERLAPS);
```

1.1.1.3.1.6 CREATE LOGFILE GROUP

The `CREATE LOGFILE GROUP` statement is not supported by MariaDB. It was originally inherited from MySQL NDB Cluster. See MDEV-19295 for more information.

1.1.1.3.1.7 CREATE PACKAGE

MariaDB starting with 10.3.5

Oracle-style packages were introduced in MariaDB 10.3.5.

Syntax

```
CREATE
    [ OR REPLACE]
    [DEFINER = { user | CURRENT_USER | role | CURRENT_ROLE }]
    PACKAGE [ IF NOT EXISTS ]
    [ db_name . ] package_name
    [ package_characteristic ... ]
{ AS | IS }
    [ package_specification_element ... ]
END [ package_name ]


package_characteristic:
    COMMENT 'string'
  | SQL SECURITY { DEFINER | INVOKER }


package_specification_element:
    FUNCTION_SYM package_specification_function ;
  | PROCEDURE_SYM package_specification_procedure ;


package_specification_function:
    func_name [ ( func_param [, func_param]... ) ]
    RETURNS func_return_type
    [ package_routine_characteristic... ]

package_specification_procedure:
    proc_name [ ( proc_param [, proc_param]... ) ]
    [ package_routine_characteristic... ]

func_return_type:
    type

func_param:
    param_name [ IN | OUT | INOUT | IN OUT ] type

proc_param:
    param_name [ IN | OUT | INOUT | IN OUT ] type

type:
    Any valid MariaDB explicit or anchored data type


package_routine_characteristic:
      COMMENT  'string'
    | LANGUAGE SQL
    | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
    | SQL SECURITY { DEFINER | INVOKER }
```

Contents

Description

The `CREATE PACKAGE` statement can be used when Oracle SQL_MODE is set.

The `CREATE PACKAGE` creates the specification for a stored package (a collection of logically related stored objects). A stored package specification declares public routines (procedures and functions) of the package, but does not implement these routines.

A package whose specification was created by the `CREATE PACKAGE` statement, should later be implemented using the CREATE PACKAGE BODY statement.

Function parameter quantifiers IN | OUT | INOUT | IN OUT

MariaDB starting with 10.8.0

The function parameter quantifiers for `IN`, `OUT`, `INOUT`, and `IN OUT` where added in a 10.8.0 preview release. Prior to 10.8.0 quantifiers were supported only in procedures.

`OUT`, `INOUT` and its equivalent `IN OUT`, are only valid if called from `SET` and not `SELECT`. These quantifiers are especially useful for creating functions and procedures with more than one return value. This allows functions and procedures to be more complex and nested.

Examples

```
SET sql_mode=ORACLE;
DELIMITER $$
CREATE OR REPLACE PACKAGE employee_tools AS
  FUNCTION getSalary(eid INT) RETURN DECIMAL(10,2);
  PROCEDURE raiseSalary(eid INT, amount DECIMAL(10,2));
  PROCEDURE raiseSalaryStd(eid INT);
  PROCEDURE hire(ename TEXT, esalary DECIMAL(10,2));
END;
$$
DELIMITER ;
```

1.1.1.3.1.8 CREATE PACKAGE BODY

MariaDB starting with 10.3.5

Oracle-style packages were introduced in MariaDB 10.3.5.

Syntax

```
CREATE [ OR REPLACE ]
    [DEFINER = { user | CURRENT_USER | role | CURRENT_ROLE }]
    PACKAGE BODY
    [ IF NOT EXISTS ]
    [ db_name . ] package_name
    [ package_characteristic... ]
{ AS | IS }
    package_implementation_declare_section
    package_implementation_executable_section
END [ package_name]


package_implementation_declare_section:
    package_implementation_item_declaration
      [ package_implementation_item_declaration... ]
      [ package_implementation_routine_definition... ]
  | package_implementation_routine_definition
      [ package_implementation_routine_definition...]

package_implementation_item_declaration:
    variable_declaration ;

variable_declaration:
    variable_name[,...] type [:= expr ]

package_implementation_routine_definition:
    FUNCTION package_specification_function
       [ package_implementation_function_body ] ;
  | PROCEDURE package_specification_procedure
       [ package_implementation_procedure_body ] ;


package_implementation_function_body:
    { AS | IS } package_routine_body [func_name]

package_implementation_procedure_body:
    { AS | IS } package_routine_body [proc_name]

package_routine_body:
```

```
    [ package_routine_declarations ]
    BEGIN
      statements [ EXCEPTION exception_handlers ]
    END


package_routine_declarations:
    package_routine_declaration ';' [package_routine_declaration ';']...


package_routine_declaration:
          variable_declaration
        | condition_name CONDITION FOR condition_value
        | user_exception_name EXCEPTION
        | CURSOR_SYM cursor_name
          [ ( cursor_formal_parameters ) ]
          IS select_statement
        ;


package_implementation_executable_section:
          END
        | BEGIN
            statement ; [statement ; ]...
          [EXCEPTION exception_handlers]
          END

exception_handlers:
           exception_handler [exception_handler...]

exception_handler:
          WHEN_SYM condition_value [, condition_value]...
            THEN_SYM statement ; [statement ;]...

condition_value:
          condition_name
        | user_exception_name
        | SQLWARNING
        | SQLEXCEPTION
        | NOT FOUND
        | OTHERS_SYM
        | SQLSTATE [VALUE] sqlstate_value
        | mariadb_error_code
```

Contents

Description

The `CREATE PACKAGE BODY` statement can be used when Oracle SQL_MODE is set.

The `CREATE PACKAGE BODY` statement creates the package body for a stored package. The package specification must be previously created using the CREATE PACKAGE statement.

A package body provides implementations of the package public routines and can optionally have:

- package-wide private variables
- package private routines
- forward declarations for private routines
- an executable initialization section

Examples

```
SET sql_mode=ORACLE;
DELIMITER $$
CREATE OR REPLACE PACKAGE employee_tools AS
  FUNCTION getSalary(eid INT) RETURN DECIMAL(10,2);
  PROCEDURE raiseSalary(eid INT, amount DECIMAL(10,2));
  PROCEDURE raiseSalaryStd(eid INT);
  PROCEDURE hire(ename TEXT, esalary DECIMAL(10,2));
END;
$$
CREATE PACKAGE BODY employee_tools AS
  -- package body variables
  stdRaiseAmount DECIMAL(10,2):=500;

  -- private routines
  PROCEDURE log (eid INT, ecmnt TEXT) AS
  BEGIN
    INSERT INTO employee_log (id, cmnt) VALUES (eid, ecmnt);
  END;

  -- public routines
  PROCEDURE hire(ename TEXT, esalary DECIMAL(10,2)) AS
    eid INT;
  BEGIN
    INSERT INTO employee (name, salary) VALUES (ename, esalary);
    eid:= last_insert_id();
    log(eid, 'hire ' || ename);
  END;

  FUNCTION getSalary(eid INT) RETURN DECIMAL(10,2) AS
    nSalary DECIMAL(10,2);
  BEGIN
    SELECT salary INTO nSalary FROM employee WHERE id=eid;
    log(eid, 'getSalary id=' || eid || ' salary=' || nSalary);
    RETURN nSalary;
  END;

  PROCEDURE raiseSalary(eid INT, amount DECIMAL(10,2)) AS
  BEGIN
    UPDATE employee SET salary=salary+amount WHERE id=eid;
    log(eid, 'raiseSalary id=' || eid || ' amount=' || amount);
  END;

  PROCEDURE raiseSalaryStd(eid INT) AS
  BEGIN
    raiseSalary(eid, stdRaiseAmount);
    log(eid, 'raiseSalaryStd id=' || eid);
  END;

BEGIN
  -- This code is executed when the current session
  -- accesses any of the package routines for the first time
  log(0, 'Session ' || connection_id() || ' ' || current_user || ' started');
END;
$$

DELIMITER ;
```

1.1.1.3.1.9 CREATE PROCEDURE

Syntax

```
CREATE
    [OR REPLACE]
    [DEFINER = { user | CURRENT_USER | role | CURRENT_ROLE }]
    PROCEDURE [IF NOT EXISTS] sp_name ([proc_parameter[,...]])
    [characteristic ...] routine_body

proc_parameter:
    [ IN | OUT | INOUT ] param_name type

type:
    Any valid MariaDB data type

characteristic:
    LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
  | COMMENT 'string'

routine_body:
    Valid SQL procedure statement
```

Contents

Description

Creates a stored procedure. By default, a routine is associated with the default database. To associate the routine explicitly with a given database, specify the name as db_name.sp_name when you create it.

When the routine is invoked, an implicit USE db_name is performed (and undone when the routine terminates). The causes the routine to have the given default database while it executes. USE statements within stored routines are disallowed.

When a stored procedure has been created, you invoke it by using the `CALL` statement (see CALL).

To execute the `CREATE PROCEDURE` statement, it is necessary to have the `CREATE ROUTINE` privilege. By default, MariaDB automatically grants the `ALTER ROUTINE` and `EXECUTE` privileges to the routine creator. See also Stored Routine Privileges.

The `DEFINER` and SQL SECURITY clauses specify the security context to be used when checking access privileges at routine execution time, as described here. Requires the SUPER privilege, or, from MariaDB 10.5.2, the SET USER privilege.

If the routine name is the same as the name of a built-in SQL function, you must use a space between the name and the following parenthesis when defining the routine, or a syntax error occurs. This is also true when you invoke the routine later. For this reason, we suggest that it is better to avoid re-using the names of existing SQL functions for your own stored routines.

The IGNORE_SPACE SQL mode applies to built-in functions, not to stored routines. It is always allowable to have spaces after a routine name, regardless of whether IGNORE_SPACE is enabled.

The parameter list enclosed within parentheses must always be present. If there are no parameters, an empty parameter list of () should be used. Parameter names are not case sensitive.

Each parameter can be declared to use any valid data type, except that the COLLATE attribute cannot be used.

For valid identifiers to use as procedure names, see Identifier Names.

Things to be Aware of With CREATE OR REPLACE

- One can't use `OR REPLACE` together with `IF EXISTS`.

CREATE PROCEDURE IF NOT EXISTS

If the `IF NOT EXISTS` clause is used, then the procedure will only be created if a procedure with the same name does not already exist. If the procedure already exists, then a warning will be triggered by default.

IN/OUT/INOUT

Each parameter is an `IN` parameter by default. To specify otherwise for a parameter, use the keyword `OUT` or `INOUT` before the parameter name.

An `IN` parameter passes a value into a procedure. The procedure might modify the value, but the modification is not visible to the caller when the procedure returns. An `OUT` parameter passes a value from the procedure back to the caller. Its initial value is NULL within the procedure, and its value is visible to the caller when the procedure returns. An `INOUT` parameter is initialized by the caller, can be modified by the procedure, and any change made by the procedure is visible to the caller when the procedure returns.

For each `OUT` or `INOUT` parameter, pass a user-defined variable in the `CALL` statement that invokes the procedure so that you can obtain its value when the procedure returns. If you are calling the procedure from within another stored procedure or function, you can also pass a routine parameter or local routine variable as an `IN` or `INOUT` parameter.

DETERMINISTIC/NOT DETERMINISTIC

`DETERMINISTIC` and `NOT DETERMINISTIC` apply only to functions. Specifying `DETERMINISTC` or `NON-DETERMINISTIC` in procedures has no effect. The default value is `NOT DETERMINISTIC`. Functions are `DETERMINISTIC` when they always return the same value for the same input. For example, a truncate or substring function. Any function involving data, therefore, is always `NOT DETERMINISTIC`.

CONTAINS SQL/NO SQL/READS SQL DATA/MODIFIES SQL DATA

`CONTAINS SQL`, `NO SQL`, `READS SQL DATA`, and `MODIFIES SQL DATA` are informative clauses that tell the server what the function does. MariaDB does not check in any way whether the specified clause is correct. If none of these clauses are specified, `CONTAINS SQL` is used by default.

`MODIFIES SQL DATA` means that the function contains statements that may modify data stored in databases. This happens if the function contains statements like DELETE, UPDATE, INSERT, REPLACE or DDL.

`READS SQL DATA` means that the function reads data stored in databases, but does not modify any data. This happens if SELECT statements are used, but there no write operations are executed.

`CONTAINS SQL` means that the function contains at least one SQL statement, but it does not read or write any data stored in a database. Examples include SET or DO.

`NO SQL` means nothing, because MariaDB does not currently support any language other than SQL.

The routine_body consists of a valid SQL procedure statement. This can be a simple statement such as SELECT or INSERT, or it can be a compound statement written using BEGIN and END. Compound statements can contain declarations, loops, and other control structure statements. See Programmatic and Compound Statements for syntax details.

MariaDB allows routines to contain DDL statements, such as `CREATE` and DROP. MariaDB also allows stored procedures (but not stored functions) to contain SQL transaction statements such as `COMMIT`.

For additional information about statements that are not allowed in stored routines, see Stored Routine Limitations.

Invoking stored procedure from within programs

For information about invoking stored procedures from within programs written in a language that has a MariaDB/MySQL interface, see CALL.

OR REPLACE

If the optional `OR REPLACE` clause is used, it acts as a shortcut for:

```
DROP PROCEDURE IF EXISTS name;
CREATE PROCEDURE name ...;
```

with the exception that any existing privileges for the procedure are not dropped.

sql_mode

MariaDB stores the sql_mode system variable setting that is in effect at the time a routine is created, and always executes the routine with this setting in force, regardless of the server SQL mode in effect when the routine is invoked.

Character Sets and Collations

Procedure parameters can be declared with any character set/collation. If the character set and collation are not specifically set, the database defaults at the time of creation will be used. If the database defaults change at a later stage, the stored procedure character set/collation will not be changed at the same time; the stored procedure needs to be dropped and recreated to ensure the same character set/collation as the database is used.

Oracle Mode

A subset of Oracle's PL/SQL language is supported in addition to the traditional SQL/PSM-based MariaDB syntax. See Oracle mode for details on changes when running Oracle mode.

Examples

The following example shows a simple stored procedure that uses an `OUT` parameter. It uses the DELIMITER command to set a new delimiter for the duration of the process — see Delimiters in the mariadb client.

```
DELIMITER //

CREATE PROCEDURE simpleproc (OUT param1 INT)
 BEGIN
  SELECT COUNT(*) INTO param1 FROM t;
 END;
//

DELIMITER ;

CALL simpleproc(@a);

SELECT @a;
+------+
| @a   |
+------+
|    1 |
+------+
```

Character set and collation:

```
DELIMITER //

CREATE PROCEDURE simpleproc2 (
  OUT param1 CHAR(10) CHARACTER SET 'utf8' COLLATE 'utf8_bin'
)
 BEGIN
  SELECT CONCAT('a'),f1 INTO param1 FROM t;
 END;
//

DELIMITER ;
```

CREATE OR REPLACE:

```
DELIMITER //

CREATE PROCEDURE simpleproc2 (
  OUT param1 CHAR(10) CHARACTER SET 'utf8' COLLATE 'utf8_bin'
)
 BEGIN
  SELECT CONCAT('a'),f1 INTO param1 FROM t;
 END;
//
ERROR 1304 (42000): PROCEDURE simpleproc2 already exists

DELIMITER ;

DELIMITER //

CREATE OR REPLACE PROCEDURE simpleproc2 (
  OUT param1 CHAR(10) CHARACTER SET 'utf8' COLLATE 'utf8_bin'
)
 BEGIN
  SELECT CONCAT('a'),f1 INTO param1 FROM t;
 END;
//
ERROR 1304 (42000): PROCEDURE simpleproc2 already exists

DELIMITER ;
Query OK, 0 rows affected (0.03 sec)
```

1.1.1.1.8 CREATE ROLE

1.1.6.2 CREATE SEQUENCE

1.1.1.3.1.12 CREATE SERVER

Syntax

```
CREATE [OR REPLACE] SERVER [IF NOT EXISTS] server_name
    FOREIGN DATA WRAPPER wrapper_name
    OPTIONS (option [, option] ...)

option:
  { HOST character-literal
  | DATABASE character-literal
  | USER character-literal
  | PASSWORD character-literal
  | SOCKET character-literal
  | OWNER character-literal
  | PORT numeric-literal }
```

Contents

Description

This statement creates the definition of a server for use with the Spider, Connect, FEDERATED or FederatedX storage engine. The CREATE SERVER statement creates a new row within the servers table within the mysql database. This statement requires the SUPER privilege or, from MariaDB 10.5.2, the FEDERATED ADMIN privilege.

The server_name should be a unique reference to the server. Server definitions are global within the scope of the server, it is not possible to qualify the server definition to a specific database. server_name has a maximum length of 64 characters (names longer than 64 characters are silently truncated), and is case insensitive. You may specify the name as a quoted

string.

The wrapper_name may be quoted with single quotes. Supported values are:

- `mysql`
- `mariadb` (in MariaDB 10.3 and later)

For each option you must specify either a character literal or numeric literal. Character literals are UTF-8, support a maximum length of 64 characters and default to a blank (empty) string. String literals are silently truncated to 64 characters. Numeric literals must be a number between 0 and 9999, default value is 0.

Note: The `OWNER` option is currently not applied, and has no effect on the ownership or operation of the server connection that is created.

The CREATE SERVER statement creates an entry in the mysql.servers table that can later be used with the CREATE TABLE statement when creating a Spider, Connect, FederatedX or FEDERATED table. The options that you specify will be used to populate the columns in the mysql.servers table. The table columns are Server_name, Host, Db, Username, Password, Port and Socket.

DROP SERVER removes a previously created server definition.

CREATE SERVER is not written to the binary log, irrespective of the binary log format being used and therefore will not replicate. From MariaDB 10.1.13, Galera replicates the CREATE SERVER, ALTER SERVER and DROP SERVER statements.

For valid identifiers to use as server names, see Identifier Names.

OR REPLACE

If the optional `OR REPLACE` clause is used, it acts as a shortcut for:

```
DROP SERVER IF EXISTS name;
CREATE SERVER server_name ...;
```

IF NOT EXISTS

If the IF NOT EXISTS clause is used, MariaDB will return a warning instead of an error if the server already exists. Cannot be used together with OR REPLACE.

Examples

```
CREATE SERVER s
FOREIGN DATA WRAPPER mariadb
OPTIONS (USER 'Remote', HOST '192.168.1.106', DATABASE 'test');
```

OR REPLACE and IF NOT EXISTS:

```
CREATE SERVER s
FOREIGN DATA WRAPPER mariadb
OPTIONS (USER 'Remote', HOST '192.168.1.106', DATABASE 'test');
ERROR 1476 (HY000): The foreign server, s, you are trying to create already exists

CREATE OR REPLACE SERVER s
FOREIGN DATA WRAPPER mariadb
OPTIONS (USER 'Remote', HOST '192.168.1.106', DATABASE 'test');
Query OK, 0 rows affected (0.00 sec)

CREATE SERVER IF NOT EXISTS s
FOREIGN DATA WRAPPER mariadb
OPTIONS (USER 'Remote', HOST '192.168.1.106', DATABASE 'test');
Query OK, 0 rows affected, 1 warning (0.00 sec)

SHOW WARNINGS;
+-------+------+-----------------------------------------------------------------+
| Level | Code | Message                                                         |
+-------+------+-----------------------------------------------------------------+
| Note  | 1476 | The foreign server, s, you are trying to create already exists  |
+-------+------+-----------------------------------------------------------------+
```

1.1.1.2.1.6 CREATE TABLE

1.1.1.3.1.14 CREATE TABLESPACE

The `CREATE TABLESPACE` statement is not supported by MariaDB. It was originally inherited from MySQL NDB Cluster. In MySQL 5.7 and later, the statement is also supported for InnoDB. However, MariaDB has chosen not to include that specific feature. See MDEV-19294 for more information.

1.1.1.3.1.15 CREATE TRIGGER

Syntax

```
CREATE [OR REPLACE]
    [DEFINER = { user | CURRENT_USER | role | CURRENT_ROLE }]
    TRIGGER [IF NOT EXISTS] trigger_name trigger_time trigger_event
    ON tbl_name FOR EACH ROW
   [{ FOLLOWS | PRECEDES } other_trigger_name ]
   trigger_stmt;
```

Contents

Description

This statement creates a new trigger. A trigger is a named database object that is associated with a table, and that activates when a particular event occurs for the table. The trigger becomes associated with the table named `tbl_name`, which must refer to a permanent table. You cannot associate a trigger with a `TEMPORARY` table or a view.

`CREATE TRIGGER` requires the TRIGGER privilege for the table associated with the trigger.

You can have multiple triggers for the same `trigger_time` and `trigger_event`.

For valid identifiers to use as trigger names, see Identifier Names.

OR REPLACE

If used and the trigger already exists, instead of an error being returned, the existing trigger will be dropped and replaced by the newly defined trigger.

DEFINER

The `DEFINER` clause determines the security context to be used when checking access privileges at trigger activation time. Usage requires the SUPER privilege, or, from MariaDB 10.5.2, the SET USER privilege.

IF NOT EXISTS

If the `IF NOT EXISTS` clause is used, the trigger will only be created if a trigger of the same name does not exist. If the trigger already exists, by default a warning will be returned.

trigger_time

`trigger_time` is the trigger action time. It can be `BEFORE` or `AFTER` to indicate that the trigger activates before or after each row to be modified.

trigger_event

`trigger_event` indicates the kind of statement that activates the trigger. The `trigger_event` can be one of the following:

- `INSERT` : The trigger is activated whenever a new row is inserted into the table; for example, through INSERT, LOAD DATA, and REPLACE statements.
- `UPDATE` : The trigger is activated whenever a row is modified; for example, through UPDATE statements.
- `DELETE` : The trigger is activated whenever a row is deleted from the table; for example, through DELETE and REPLACE statements. However, `DROP TABLE` and `TRUNCATE` statements on the table do not activate this trigger, because they do not use `DELETE` . Dropping a partition does not activate `DELETE` triggers, either.

FOLLOWS/PRECEDES other_trigger_name

The `FOLLOWS other_trigger_name` and `PRECEDES other_trigger_name` options were added in MariaDB 10.2.3 as part of supporting multiple triggers per action time. This is the same syntax used by MySQL 5.7, although MySQL 5.7 does not have multi-trigger support.

`FOLLOWS` adds the new trigger after another trigger while `PRECEDES` adds the new trigger before another trigger. If neither option is used, the new trigger is added last for the given action and time.

`FOLLOWS` and `PRECEDES` are not stored in the trigger definition. However the trigger order is guaranteed to not change over time. mariadb-dump and other backup methods will not change trigger order. You can verify the trigger order from the `ACTION_ORDER` column in INFORMATION_SCHEMA.TRIGGERS table.

```
SELECT trigger_name, action_order FROM information_schema.triggers
  WHERE event_object_table='t1';
```

Atomic DDL

MariaDB starting with 10.6.1

MariaDB 10.6.1 supports Atomic DDL and `CREATE TRIGGER` is atomic.

Examples

```
CREATE DEFINER=`root`@`localhost` TRIGGER increment_animal
  AFTER INSERT ON animals FOR EACH ROW
   UPDATE animal_count SET animal_count.animals = animal_count.animals+1;
```

OR REPLACE and IF NOT EXISTS

```
CREATE DEFINER=`root`@`localhost` TRIGGER increment_animal
  AFTER INSERT ON animals FOR EACH ROW
    UPDATE animal_count SET animal_count.animals = animal_count.animals+1;
ERROR 1359 (HY000): Trigger already exists

CREATE OR REPLACE DEFINER=`root`@`localhost` TRIGGER increment_animal
  AFTER INSERT ON animals  FOR EACH ROW
    UPDATE animal_count SET animal_count.animals = animal_count.animals+1;
Query OK, 0 rows affected (0.12 sec)

CREATE DEFINER=`root`@`localhost` TRIGGER IF NOT EXISTS increment_animal
  AFTER INSERT ON animals FOR EACH ROW
    UPDATE animal_count SET animal_count.animals = animal_count.animals+1;
Query OK, 0 rows affected, 1 warning (0.00 sec)

SHOW WARNINGS;
+-------+------+-----------------------+
| Level | Code | Message               |
+-------+------+-----------------------+
| Note  | 1359 | Trigger already exists |
+-------+------+-----------------------+
1 row in set (0.00 sec)
```

1.1.1.1.1 CREATE USER

1.1.1.3.1.17 CREATE VIEW

Syntax

```
CREATE
    [OR REPLACE]
    [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
    [DEFINER = { user | CURRENT_USER | role | CURRENT_ROLE }]
    [SQL SECURITY { DEFINER | INVOKER }]
    VIEW [IF NOT EXISTS] view_name [(column_list)]
    AS select_statement
    [WITH [CASCADED | LOCAL] CHECK OPTION]
```

Contents

Description

The CREATE VIEW statement creates a new view, or replaces an existing one if the OR REPLACE clause is given. If the view does not exist, CREATE OR REPLACE VIEW is the same as CREATE VIEW. If the view does exist, CREATE OR REPLACE VIEW is the same as ALTER VIEW.

The select_statement is a SELECT statement that provides the definition of the view. (When you select from the view, you select in effect using the SELECT statement.) select_statement can select from base tables or other views.

The view definition is "frozen" at creation time, so changes to the underlying tables afterwards do not affect the view definition. For example, if a view is defined as SELECT * on a table, new columns added to the table later do not become part of the view. A SHOW CREATE VIEW shows that such queries are rewritten and column names are included in the view definition.

The view definition must be a query that does not return errors at view creation times. However, the base tables used by the views might be altered later and the query may not be valid anymore. In this case, querying the view will result in an error. CHECK TABLE helps in finding this kind of problems.

The ALGORITHM clause affects how MariaDB processes the view. The DEFINER and SQL SECURITY clauses specify the security context to be used when checking access privileges at view invocation time. The WITH CHECK OPTION clause can be given to constrain inserts or updates to rows in tables referenced by the view. These clauses are described later in this section.

The CREATE VIEW statement requires the CREATE VIEW privilege for the view, and some privilege for each column selected by the SELECT statement. For columns used elsewhere in the SELECT statement you must have the SELECT privilege. If the OR REPLACE clause is present, you must also have the DROP privilege for the view.

A view belongs to a database. By default, a new view is created in the default database. To create the view explicitly in a given database, specify the name as db_name.view_name when you create it.

```
CREATE VIEW test.v AS SELECT * FROM t;
```

Base tables and views share the same namespace within a database, so a database cannot contain a base table and a view that have the same name.

Views must have unique column names with no duplicates, just like base tables. By default, the names of the columns retrieved by the SELECT statement are used for the view column names. To define explicit names for the view columns, the optional column_list clause can be given as a list of comma-separated identifiers. The number of names in column_list must be the same as the number of columns retrieved by the SELECT statement.

MySQL until 5.1.28

Prior to MySQL 5.1.29, When you modify an existing view, the current view definition is backed up and saved. It is stored in that table's database directory, in a subdirectory named arc. The backup file for a view v is named v.frm-00001. If you alter the view again, the next backup is named v.frm-00002. The three latest view backup definitions are stored. Backed

up view definitions are not preserved by mysqldump, or any other such programs, but you can retain them using a file copy operation. However, they are not needed for anything but to provide you with a backup of your previous view definition. It is safe to remove these backup definitions, but only while mysqld is not running. If you delete the arc subdirectory or its files while mysqld is running, you will receive an error the next time you try to alter the view:

```
MariaDB [test]> ALTER VIEW v AS SELECT * FROM t;
ERROR 6 (HY000): Error on delete of '.\test\arc/v.frm-0004' (Errcode: 2)
```

Columns retrieved by the SELECT statement can be simple references to table columns. They can also be expressions that use functions, constant values, operators, and so forth.

Unqualified table or view names in the SELECT statement are interpreted with respect to the default database. A view can refer to tables or views in other databases by qualifying the table or view name with the proper database name.

A view can be created from many kinds of SELECT statements. It can refer to base tables or other views. It can use joins, UNION, and subqueries. The SELECT need not even refer to any tables. The following example defines a view that selects two columns from another table, as well as an expression calculated from those columns:

```
CREATE TABLE t (qty INT, price INT);

INSERT INTO t VALUES(3, 50);

CREATE VIEW v AS SELECT qty, price, qty*price AS value FROM t;

SELECT * FROM v;
+------+-------+-------+
| qty  | price | value |
+------+-------+-------+
|    3 |    50 |   150 |
+------+-------+-------+
```

A view definition is subject to the following restrictions:

- The SELECT statement cannot contain a subquery in the FROM clause.
- The SELECT statement cannot refer to system or user variables.
- Within a stored program, the definition cannot refer to program parameters or local variables.
- The SELECT statement cannot refer to prepared statement parameters.
- Any table or view referred to in the definition must exist. However, after a view has been created, it is possible to drop a table or view that the definition refers to. In this case, use of the view results in an error. To check a view definition for problems of this kind, use the CHECK TABLE statement.
- The definition cannot refer to a TEMPORARY table, and you cannot create a TEMPORARY view.
- Any tables named in the view definition must exist at definition time.
- You cannot associate a trigger with a view.
- For valid identifiers to use as view names, see Identifier Names.

ORDER BY is allowed in a view definition, but it is ignored if you select from a view using a statement that has its own ORDER BY.

For other options or clauses in the definition, they are added to the options or clauses of the statement that references the view, but the effect is undefined. For example, if a view definition includes a LIMIT clause, and you select from the view using a statement that has its own LIMIT clause, it is undefined which limit applies. This same principle applies to options such as ALL, DISTINCT, or SQL_SMALL_RESULT that follow the SELECT keyword, and to clauses such as INTO, FOR UPDATE, and LOCK IN SHARE MODE.

The PROCEDURE clause cannot be used in a view definition, and it cannot be used if a view is referenced in the FROM clause.

If you create a view and then change the query processing environment by changing system variables, that may affect the results that you get from the view:

```
CREATE VIEW v (mycol) AS SELECT 'abc';

SET sql_mode = '';

SELECT "mycol" FROM v;
+-------+
| mycol |
+-------+
| mycol |
+-------+

SET sql_mode = 'ANSI_QUOTES';

SELECT "mycol" FROM v;
+-------+
| mycol |
+-------+
| abc   |
+-------+
```

The DEFINER and SQL SECURITY clauses determine which MariaDB account to use when checking access privileges for the view when a statement is executed that references the view. They were added in MySQL 5.1.2. The legal SQL SECURITY characteristic values are DEFINER and INVOKER. These indicate that the required privileges must be held by the user who defined or invoked the view, respectively. The default SQL SECURITY value is DEFINER.

If a user value is given for the DEFINER clause, it should be a MariaDB account in 'user_name'@'host_name' format (the same format used in the GRANT statement). The user_name and host_name values both are required. The definer can also be given as CURRENT_USER or CURRENT_USER(). The default DEFINER value is the user who executes the CREATE VIEW statement. This is the same as specifying DEFINER = CURRENT_USER explicitly.

If you specify the DEFINER clause, these rules determine the legal DEFINER user values:

- If you do not have the SUPER privilege, or, from MariaDB 10.5.2, the SET USER privilege, the only legal user value is your own account, either specified literally or by using CURRENT_USER. You cannot set the definer to some other account.
- If you have the SUPER privilege, or, from MariaDB 10.5.2, the SET USER privilege, you can specify any syntactically legal account name. If the account does not actually exist, a warning is generated.
- If the SQL SECURITY value is DEFINER but the definer account does not exist when the view is referenced, an error occurs.

Within a view definition, CURRENT_USER returns the view's DEFINER value by default. For views defined with the SQL SECURITY INVOKER characteristic, CURRENT_USER returns the account for the view's invoker. For information about user auditing within views, see http://dev.mysql.com/doc/refman/5.1/en/account-activity-auditing.html.

Within a stored routine that is defined with the SQL SECURITY DEFINER characteristic, CURRENT_USER returns the routine's DEFINER value. This also affects a view defined within such a program, if the view definition contains a DEFINER value of CURRENT_USER.

View privileges are checked like this:

- At view definition time, the view creator must have the privileges needed to use the top-level objects accessed by the view. For example, if the view definition refers to table columns, the creator must have privileges for the columns, as described previously. If the definition refers to a stored function, only the privileges needed to invoke the function can be checked. The privileges required when the function runs can be checked only as it executes: For different invocations of the function, different execution paths within the function might be taken.
- When a view is referenced, privileges for objects accessed by the view are checked against the privileges held by the view creator or invoker, depending on whether the SQL SECURITY characteristic is DEFINER or INVOKER, respectively.
- If reference to a view causes execution of a stored function, privilege checking for statements executed within the function depend on whether the function is defined with a SQL SECURITY characteristic of DEFINER or INVOKER. If the security characteristic is DEFINER, the function runs with the privileges of its creator. If the characteristic is INVOKER, the function runs with the privileges determined by the view's SQL SECURITY characteristic.

Example: A view might depend on a stored function, and that function might invoke other stored routines. For example, the following view invokes a stored function f():

```
CREATE VIEW v AS SELECT * FROM t WHERE t.id = f(t.name);

Suppose that f() contains a statement such as this:

IF name IS NULL then
  CALL p1();
ELSE
  CALL p2();
END IF;
```

The privileges required for executing statements within f() need to be checked when f() executes. This might mean that privileges are needed for p1() or p2(), depending on the execution path within f(). Those privileges must be checked at runtime, and the user who must possess the privileges is determined by the SQL SECURITY values of the view v and the function f().

The DEFINER and SQL SECURITY clauses for views are extensions to standard SQL. In standard SQL, views are handled using the rules for SQL SECURITY INVOKER.

If you invoke a view that was created before MySQL 5.1.2, it is treated as though it was created with a SQL SECURITY DEFINER clause and with a DEFINER value that is the same as your account. However, because the actual definer is unknown, MySQL issues a warning. To make the warning go away, it is sufficient to re-create the view so that the view definition includes a DEFINER clause.

The optional ALGORITHM clause is an extension to standard SQL. It affects how MariaDB processes the view. ALGORITHM takes three values: MERGE, TEMPTABLE, or UNDEFINED. The default algorithm is UNDEFINED if no ALGORITHM clause is present. See View Algorithms for more information.

Some views are updatable. That is, you can use them in statements such as UPDATE, DELETE, or INSERT to update the contents of the underlying table. For a view to be updatable, there must be a one-to-one relationship between the rows in the view and the rows in the underlying table. There are also certain other constructs that make a view non-updatable. See Inserting and Updating with Views.

WITH CHECK OPTION

The WITH CHECK OPTION clause can be given for an updatable view to prevent inserts or updates to rows except those for which the WHERE clause in the select_statement is true.

In a WITH CHECK OPTION clause for an updatable view, the LOCAL and CASCADED keywords determine the scope of check testing when the view is defined in terms of another view. The LOCAL keyword restricts the CHECK OPTION only to the view being defined. CASCADED causes the checks for underlying views to be evaluated as well. When neither keyword is given, the default is CASCADED.

For more information about updatable views and the WITH CHECK OPTION clause, see Inserting and Updating with Views.

IF NOT EXISTS

MariaDB starting with 10.1.3

The `IF NOT EXISTS` clause was added in MariaDB 10.1.3

When the IF NOT EXISTS clause is used, MariaDB will return a warning instead of an error if the specified view already exists. Cannot be used together with the `OR REPLACE` clause.

Atomic DDL

MariaDB starting with 10.6.1

MariaDB 10.6.1 supports Atomic DDL and `CREATE VIEW` is atomic.

Examples

```
CREATE TABLE t (a INT, b INT) ENGINE = InnoDB;

INSERT INTO t VALUES (1,1), (2,2), (3,3);

CREATE VIEW v AS SELECT a, a*2 AS a2 FROM t;

SELECT * FROM v;
+------+------+
| a    | a2   |
+------+------+
|    1 |    2 |
|    2 |    4 |
|    3 |    6 |
+------+------+
```

OR REPLACE and IF NOT EXISTS:

```
CREATE VIEW v AS SELECT a, a*2 AS a2 FROM t;
ERROR 1050 (42S01): Table 'v' already exists

CREATE OR REPLACE VIEW v AS SELECT a, a*2 AS a2 FROM t;
Query OK, 0 rows affected (0.04 sec)

CREATE VIEW IF NOT EXISTS v AS SELECT a, a*2 AS a2 FROM t;
Query OK, 0 rows affected, 1 warning (0.01 sec)

SHOW WARNINGS;
+-------+------+--------------------------+
| Level | Code | Message                  |
+-------+------+--------------------------+
| Note  | 1050 | Table 'v' already exists |
+-------+------+--------------------------+
```

1.1.1.3.1.18 Silent Column Changes

When a CREATE TABLE or ALTER TABLE command is issued, MariaDB will silently change a column specification in the following cases:

- PRIMARY KEY columns are always NOT NULL.
- Any trailing spaces from SET and ENUM values are discarded.
- TIMESTAMP columns are always NOT NULL, and display sizes are discarded
- A row-size limit of 65535 bytes applies
- If strict SQL mode is not enabled (it is enabled by default from MariaDB 10.2), a VARCHAR column longer than 65535 become TEXT, and a VARBINARY columns longer than 65535 becomes a BLOB. If strict mode is enabled the silent changes will not be made, and an error will occur.
- If a USING clause specifies an index that's not permitted by the storage engine, the engine will instead use another available index type that can be applied without affecting results.
- If the CHARACTER SET binary attribute is specified, the column is created as the matching binary data type. A TEXT becomes a BLOB, CHAR a BINARY and VARCHAR a VARBINARY. ENUMs and SETs are created as defined.

To ease imports from other RDBMSs, MariaDB will also silently map the following data types:

Other Vendor Type	MariaDB Type
BOOL	TINYINT
BOOLEAN	TINYINT
CHARACTER VARYING(M)	VARCHAR(M)
FIXED	DECIMAL
FLOAT4	FLOAT
FLOAT8	DOUBLE
INT1	TINYINT
INT2	SMALLINT
INT3	MEDIUMINT

INT4	INT
INT8	BIGINT
LONG VARBINARY	MEDIUMBLOB
LONG VARCHAR	MEDIUMTEXT
LONG	MEDIUMTEXT
MIDDLEINT	MEDIUMINT
NUMERIC	DECIMAL

Currently, all MySQL types are supported in MariaDB.

For type mapping between Cassandra and MariaDB, see Cassandra storage engine.

Example

Silent changes in action:

```
CREATE TABLE SilenceIsGolden
   (
    f1 TEXT CHARACTER SET binary,
    f2 VARCHAR(15) CHARACTER SET binary,
    f3 CHAR CHARACTER SET binary,
    f4 ENUM('x','y','z') CHARACTER SET binary,
    f5 VARCHAR (65536),
    f6 VARBINARY (65536),
    f7 INT1
   );
Query OK, 0 rows affected, 2 warnings (0.31 sec)

SHOW WARNINGS;
+-------+------+-----------------------------------------------+
| Level | Code | Message                                       |
+-------+------+-----------------------------------------------+
| Note  | 1246 | Converting column 'f5' from VARCHAR to TEXT   |
| Note  | 1246 | Converting column 'f6' from VARBINARY to BLOB |
+-------+------+-----------------------------------------------+

DESCRIBE SilenceIsGolden;
+-------+-------------------+------+-----+---------+-------+
| Field | Type              | Null | Key | Default | Extra |
+-------+-------------------+------+-----+---------+-------+
| f1    | blob              | YES  |     | NULL    |       |
| f2    | varbinary(15)     | YES  |     | NULL    |       |
| f3    | binary(1)         | YES  |     | NULL    |       |
| f4    | enum('x','y','z') | YES  |     | NULL    |       |
| f5    | mediumtext        | YES  |     | NULL    |       |
| f6    | mediumblob        | YES  |     | NULL    |       |
| f7    | tinyint(4)        | YES  |     | NULL    |       |
+-------+-------------------+------+-----+---------+-------+
```

1.1.1.3.1.19 Generated (Virtual and Persistent/Stored) Columns

Syntax

```
<type>  [GENERATED ALWAYS]  AS   ( <expression> )
[VIRTUAL | PERSISTENT | STORED]  [UNIQUE] [UNIQUE KEY] [COMMENT <text>]
```

Contents

MariaDB's generated columns syntax is designed to be similar to the syntax for Microsoft SQL Server's computed columns and Oracle Database's virtual columns. In MariaDB 10.2 and later, the syntax is also compatible with the syntax for MySQL's generated columns.

Description

A generated column is a column in a table that cannot explicitly be set to a specific value in a DML query. Instead, its value is automatically generated based on an expression. This expression might generate the value based on the values of other columns in the table, or it might generate the value by calling built-in functions or user-defined functions (UDFs).

There are two types of generated columns:

- `PERSISTENT` (a.k.a. `STORED`): This type's value is actually stored in the table.
- `VIRTUAL`: This type's value is not stored at all. Instead, the value is generated dynamically when the table is queried. This type is the default.

Generated columns are also sometimes called computed columns or virtual columns.

Supported Features

Storage Engine Support

- Generated columns can only be used with storage engines which support them. If you try to use a storage engine that does not support them, then you will see an error similar to the following:

```
ERROR 1910 (HY000): TokuDB storage engine does not support computed columns
```

- InnoDB, Aria, MyISAM and CONNECT support generated columns.
- A column in a MERGE table can be built on a `PERSISTENT` generated column.
  - However, a column in a MERGE table can **not** be defined as a `VIRTUAL` and `PERSISTENT` generated column.

Data Type Support

- All data types are supported when defining generated columns.
- Using the ZEROFILL column option is supported when defining generated columns.
- Using the AUTO_INCREMENT column option is **not** supported when defining generated columns. Until MariaDB 10.2.25, it was supported, but this support was removed, because it would not work correctly. See MDEV-11117.

Index Support

- Using a generated column as a table's primary key is **not** supported. See MDEV-5590 for more information. If you try to use one as a primary key, then you will see an error similar to the following:

```
ERROR 1903 (HY000): Primary key cannot be defined upon a computed column
```

- Using `PERSISTENT` generated columns as part of a foreign key is supported.

- Referencing `PERSISTENT` generated columns as part of a foreign key is also supported.
  - However, using the `ON UPDATE CASCADE`, `ON UPDATE SET NULL`, or `ON DELETE SET NULL` clauses is **not** supported. If you try to use an unsupported clause, then you will see an error similar to the following:

```
ERROR 1905 (HY000): Cannot define foreign key with ON UPDATE SET NULL clause on a computed
column
```

- Defining indexes on both `VIRTUAL` and `PERSISTENT` generated columns is supported.
  - If an index is defined on a generated column, then the optimizer considers using it in the same way as indexes based on "real" columns.

Statement Support

- Generated columns are used in DML queries just as if they were "real" columns.
  - However, `VIRTUAL` and `PERSISTENT` generated columns differ in how their data is stored.
    - Values for `PERSISTENT` generated columns are generated whenever a DML queries inserts or updates the row with the special `DEFAULT` value. This generates the columns value, and it is stored in the table like the other "real" columns. This value can be read by other DML queries just like the other "real" columns.
    - Values for `VIRTUAL` generated columns are not stored in the table. Instead, the value is generated dynamically whenever the column is queried. If other columns in a row are queried, but the `VIRTUAL` generated column is not one of the queried columns, then the column's value is not generated.
- The SELECT statement supports generated columns.
- Generated columns can be referenced in the INSERT, UPDATE, and DELETE statements.
  - However, `VIRTUAL` or `PERSISTENT` generated columns cannot be explicitly set to any other values than `NULL` or DEFAULT. If a generated column is explicitly set to any other value, then the outcome depends on whether strict mode is enabled in sql_mode. If it is not enabled, then a warning will be raised and the default generated value will be used instead. If it is enabled, then an error will be raised instead.
- The CREATE TABLE statement has limited support for generated columns.
  - It supports defining generated columns in a new table.
  - It supports using generated columns to partition tables.
  - It does **not** support using the versioning clauses with generated columns.
- The ALTER TABLE statement has limited support for generated columns.
  - It supports the `MODIFY` and `CHANGE` clauses for `PERSISTENT` generated columns.
  - It does **not** support the `MODIFY` clause for `VIRTUAL` generated columns if ALGORITHM is not set to `COPY`. See MDEV-15476 for more information.
  - It does **not** support the `CHANGE` clause for `VIRTUAL` generated columns if ALGORITHM is not set to `COPY`. See MDEV-17035 for more information.
  - It does **not** support altering a table if ALGORITHM is not set to `COPY` if the table has a `VIRTUAL` generated column that is indexed. See MDEV-14046 for more information.
  - It does **not** support adding a `VIRTUAL` generated column with the `ADD` clause if the same statement is also adding other columns if ALGORITHM is not set to `COPY`. See MDEV-17468 for more information.
  - It also does **not** support altering an existing column into a `VIRTUAL` generated column.
  - It supports using generated columns to partition tables.
  - It does **not** support using the versioning clauses with generated columns.
- The SHOW CREATE TABLE statement supports generated columns.
- The DESCRIBE statement can be used to check whether a table has generated columns.
  - You can tell which columns are generated by looking for the ones where the `Extra` column is set to either `VIRTUAL` or `PERSISTENT`. For example:

```
DESCRIBE table1;
+-------+-------------+------+-----+---------+------------+
| Field | Type        | Null | Key | Default | Extra      |
+-------+-------------+------+-----+---------+------------+
| a     | int(11)     | NO   |     | NULL    |            |
| b     | varchar(32) | YES  |     | NULL    |            |
| c     | int(11)     | YES  |     | NULL    | VIRTUAL    |
| d     | varchar(5)  | YES  |     | NULL    | PERSISTENT |
+-------+-------------+------+-----+---------+------------+
```

- Generated columns can be properly referenced in the `NEW` and `OLD` rows in triggers.
- Stored procedures support generated columns.

- The HANDLER statement supports generated columns.

Expression Support

- Most legal, deterministic expressions which can be calculated are supported in expressions for generated columns.
- Most built-in functions are supported in expressions for generated columns.
  - However, some built-in functions can't be supported for technical reasons. For example, If you try to use an unsupported function in an expression, an error is generated similar to the following:

```
ERROR 1901 (HY000): Function or expression 'dayname()' cannot be used in the GENERATED ALWAYS AS clause of `v`
```

- Subqueries are **not** supported in expressions for generated columns because the underlying data can change.
- Using anything that depends on data outside the row is **not** supported in expressions for generated columns.
- Stored functions are **not** supported in expressions for generated columns. See MDEV-17587 for more information.
- Non-deterministic built-in functions are supported in expressions for not indexed `VIRTUAL` generated columns.
- Non-deterministic built-in functions are **not** supported in expressions for `PERSISTENT` or indexed `VIRTUAL` generated columns.
- User-defined functions (UDFs) are supported in expressions for generated columns.
  - However, MariaDB can't check whether a UDF is deterministic, so it is up to the user to be sure that they do not use non-deterministic UDFs with `VIRTUAL` generated columns.
- Defining a generated column based on other generated columns defined before it in the table definition is supported. For example:

```
CREATE TABLE t1 (a int as (1), b int as (a));
```

- However, defining a generated column based on other generated columns defined after in the table definition is **not** supported in expressions for generation columns because generated columns are calculated in the order they are defined.
- Using an expression that exceeds 255 characters in length is supported in expressions for generated columns. The new limit for the entire table definition, including all expressions for generated columns, is 65,535 bytes.
- Using constant expressions is supported in expressions for generated columns. For example:

```
CREATE TABLE t1 (a int as (1));
```

Making Stored Values Consistent

When a generated column is `PERSISTENT` or indexed, the value of the expression needs to be consistent regardless of the SQL Mode flags in the current session. If it is not, then the table will be seen as corrupted when the value that should actually be returned by the computed expression and the value that was previously stored and/or indexed using a different sql_mode setting disagree.

There are currently two affected classes of inconsistencies: character padding and unsigned subtraction:

- For a `VARCHAR` or `TEXT` generated column the length of the value returned can vary depending on the PAD_CHAR_TO_FULL_LENGTH sql_mode flag. To make the value consistent, create the generated column using an RTRIM() or RPAD() function. Alternately, create the generated column as a `CHAR` column so that its data is always fully padded.
- If a `SIGNED` generated column is based on the subtraction of an `UNSIGNED` value, the resulting value can vary depending on how large the value is and the NO_UNSIGNED_SUBTRACTION sql_mode flag. To make the value consistent, use CAST() to ensure that each `UNSIGNED` operand is `SIGNED` before the subtraction.

MariaDB starting with 10.5

Beginning in MariaDB 10.5, there is a fatal error generated when trying to create a generated column whose value can change depending on the SQL Mode when its data is `PERSISTENT` or indexed.

For an existing generated column that has a potentially inconsistent value, a warning about a bad expression is generated the first time it is used (if warnings are enabled).

Beginning in MariaDB 10.4.8, MariaDB 10.3.18, and MariaDB 10.2.27 a potentially inconsistent generated column outputs a warning when created or first used (without restricting their creation).

Here is an example of two tables that would be rejected in MariaDB 10.5 and warned about in the other listed versions:

```
CREATE TABLE bad_pad (
  txt CHAR(5),
  -- CHAR -> VARCHAR or CHAR -> TEXT can't be persistent or indexed:
  vtxt VARCHAR(5) AS (txt) PERSISTENT
);

CREATE TABLE bad_sub (
  num1 BIGINT UNSIGNED,
  num2 BIGINT UNSIGNED,
  -- The resulting value can vary for some large values
  vnum BIGINT AS (num1 - num2) VIRTUAL,
  KEY(vnum)
);
```

The warnings for the above tables look like this:

```
Warning (Code 1901): Function or expression '`txt`' cannot be used in the GENERATED ALWAYS AS
clause of `vtxt`
Warning (Code 1105): Expression depends on the @@sql_mode value PAD_CHAR_TO_FULL_LENGTH

Warning (Code 1901): Function or expression '`num1` - `num2`' cannot be used in the GENERATED
ALWAYS AS clause of `vnum`
Warning (Code 1105): Expression depends on the @@sql_mode value NO_UNSIGNED_SUBTRACTION
```

To work around the issue, force the padding or type to make the generated column's expression return a consistent value. For example:

```
CREATE TABLE good_pad (
  txt CHAR(5),
  -- Using RTRIM() or RPAD() makes the value consistent:
  vtxt VARCHAR(5) AS (RTRIM(txt)) PERSISTENT,
  -- When not persistent or indexed, it is OK for the value to vary by mode:
  vtxt2 VARCHAR(5) AS (txt) VIRTUAL,
  -- CHAR -> CHAR is always OK:
  txt2 CHAR(5) AS (txt) PERSISTENT
);

CREATE TABLE good_sub (
  num1 BIGINT UNSIGNED,
  num2 BIGINT UNSIGNED,
  -- The indexed value will always be consistent in this expression:
  vnum BIGINT AS (CAST(num1 AS SIGNED) - CAST(num2 AS SIGNED)) VIRTUAL,
  KEY(vnum)
);
```

MySQL Compatibility Support

- The `STORED` keyword is supported as an alias for the `PERSISTENT` keyword.
- Tables created with MySQL 5.7 or later that contain MySQL's generated columns can be imported into MariaDB without a dump and restore.

Implementation Differences

Generated columns are subject to various constraints in other DBMSs that are not present in MariaDB's implementation. Generated columns may also be called computed columns or virtual columns in different implementations. The various details for a specific implementation can be found in the documentation for each specific DBMS.

Implementation Differences Compared to Microsoft SQL Server

MariaDB's generated columns implementation does not enforce the following restrictions that are present in Microsoft SQL Server's computed columns implementation:

- MariaDB allows server variables in generated column expressions, including those that change dynamically, such as warning_count.
- MariaDB allows the CONVERT_TZ() function to be called with a named time zone as an argument, even though time zone names and time offsets are configurable.

- MariaDB allows the CAST() function to be used with non-unicode character sets, even though character sets are configurable and differ between binaries/versions.
- MariaDB allows FLOAT expressions to be used in generated columns. Microsoft SQL Server considers these expressions to be "imprecise" due to potential cross-platform differences in floating-point implementations and precision.
- Microsoft SQL Server requires the ARITHABORT mode to be set, so that division by zero returns an error, and not a NULL.
- Microsoft SQL Server requires `QUOTED_IDENTIFIER` to be set in sql_mode. In MariaDB, if data is inserted without `ANSI_QUOTES` set in sql_mode, then it will be processed and stored differently in a generated column that contains quoted identifiers.

Microsoft SQL Server enforces the above restrictions by doing one of the following things:

- Refusing to create computed columns.
- Refusing to allow updates to a table containing them.
- Refusing to use an index over such a column if it can not be guaranteed that the expression is fully deterministic.

In MariaDB, as long as the sql_mode, language, and other settings that were in effect during the CREATE TABLE remain unchanged, the generated column expression will always be evaluated the same. If any of these things change, then please be aware that the generated column expression might not be evaluated the same way as it previously was.

If you try to update a virtual column, you will get an error if the default strict mode is enabled in sql_mode, or a warning otherwise.

Development History

Generated columns was originally developed by Andrey Zhakov. It was then modified by Sanja Byelkin and Igor Babaev at Monty Program for inclusion in MariaDB. Monty did the work on MariaDB 10.2 to lift a some of the old limitations.

Examples

Here is an example table that uses both `VIRTUAL` and `PERSISTENT` virtual columns:

```
USE TEST;

CREATE TABLE table1 (
     a INT NOT NULL,
     b VARCHAR(32),
     c INT AS (a mod 10) VIRTUAL,
     d VARCHAR(5) AS (left(b,5)) PERSISTENT);
```

If you describe the table, you can easily see which columns are virtual by looking in the "Extra" column:

```
DESCRIBE table1;
+-------+-------------+------+-----+---------+------------+
| Field | Type        | Null | Key | Default | Extra      |
+-------+-------------+------+-----+---------+------------+
| a     | int(11)     | NO   |     | NULL    |            |
| b     | varchar(32) | YES  |     | NULL    |            |
| c     | int(11)     | YES  |     | NULL    | VIRTUAL    |
| d     | varchar(5)  | YES  |     | NULL    | PERSISTENT |
+-------+-------------+------+-----+---------+------------+
```

To find out what function(s) generate the value of the virtual column you can use `SHOW CREATE TABLE`:

```
SHOW CREATE TABLE table1;

| table1 | CREATE TABLE `table1` (
  `a` int(11) NOT NULL,
  `b` varchar(32) DEFAULT NULL,
  `c` int(11) AS (a mod 10) VIRTUAL,
  `d` varchar(5) AS (left(b,5)) PERSISTENT
) ENGINE=MyISAM DEFAULT CHARSET=latin1 |
```

If you try to insert non-default values into a virtual column, you will receive a warning and what you tried to insert will be ignored and the derived value inserted instead:

```
WARNINGS;
Show warnings enabled.

INSERT INTO table1 VALUES (1, 'some text',default,default);
Query OK, 1 row affected (0.00 sec)

INSERT INTO table1 VALUES (2, 'more text',5,default);
Query OK, 1 row affected, 1 warning (0.00 sec)

Warning (Code 1645): The value specified for computed column 'c' in table 'table1' has been ignor

INSERT INTO table1 VALUES (123, 'even more text',default,'something');
Query OK, 1 row affected, 2 warnings (0.00 sec)

Warning (Code 1645): The value specified for computed column 'd' in table 'table1' has been ignor
Warning (Code 1265): Data truncated for column 'd' at row 1

SELECT * FROM table1;
+-----+----------------+------+-------+
| a   | b              | c    | d     |
+-----+----------------+------+-------+
|   1 | some text      |    1 | some  |
|   2 | more text      |    2 | more  |
| 123 | even more text |    3 | even  |
+-----+----------------+------+-------+
3 rows in set (0.00 sec)
```

If the `ZEROFILL` clause is specified, it should be placed directly after the type definition, before the `AS (<expression>)`:

```
CREATE TABLE table2 (a INT, b INT ZEROFILL AS (a*2) VIRTUAL);
INSERT INTO table2 (a) VALUES (1);

SELECT * FROM table2;
+------+------------+
| a    | b          |
+------+------------+
|    1 | 0000000002 |
+------+------------+
1 row in set (0.00 sec)
```

You can also use virtual columns to implement a "poor man's partial index". See example at the end of Unique Index.

1.1.1.3.1.20 Invisible Columns

MariaDB starting with 10.3.3

Invisible columns (sometimes also called hidden columns) first appeared in MariaDB 10.3.3.

Columns can be given an `INVISIBLE` attribute in a CREATE TABLE or ALTER TABLE statement. These columns will then not be listed in the results of a SELECT * statement, nor do they need to be assigned a value in an INSERT statement, unless INSERT explicitly mentions them by name.

Since `SELECT *` does not return the invisible columns, new tables or views created in this manner will have no trace of the invisible columns. If specifically referenced in the SELECT statement, the columns will be brought into the view/new table, but the INVISIBLE attribute will not.

Invisible columns can be declared as `NOT NULL`, but then require a `DEFAULT` value.

It is not possible for all columns in a table to be invisible.

Examples

```
CREATE TABLE t (x INT INVISIBLE);
ERROR 1113 (42000): A table must have at least 1 column

CREATE TABLE t (x INT, y INT INVISIBLE, z INT INVISIBLE NOT NULL);
ERROR 4106 (HY000): Invisible column `z` must have a default value

CREATE TABLE t (x INT, y INT INVISIBLE, z INT INVISIBLE NOT NULL DEFAULT 4);

INSERT INTO t VALUES (1),(2);

INSERT INTO t (x,y) VALUES (3,33);

SELECT * FROM t;
+------+
| x    |
+------+
|    1 |
|    2 |
|    3 |
+------+

SELECT x,y,z FROM t;
+------+------+---+
| x    | y    | z |
+------+------+---+
|    1 | NULL | 4 |
|    2 | NULL | 4 |
|    3 |   33 | 4 |
+------+------+---+

DESC t;
+-------+---------+------+-----+---------+-----------+
| Field | Type    | Null | Key | Default | Extra     |
+-------+---------+------+-----+---------+-----------+
| x     | int(11) | YES  |     | NULL    |           |
| y     | int(11) | YES  |     | NULL    | INVISIBLE |
| z     | int(11) | NO   |     | 4       | INVISIBLE |
+-------+---------+------+-----+---------+-----------+

ALTER TABLE t MODIFY x INT INVISIBLE, MODIFY y INT, MODIFY z INT NOT NULL DEFAULT 4;

DESC t;
+-------+---------+------+-----+---------+-----------+
| Field | Type    | Null | Key | Default | Extra     |
+-------+---------+------+-----+---------+-----------+
| x     | int(11) | YES  |     | NULL    | INVISIBLE |
| y     | int(11) | YES  |     | NULL    |           |
| z     | int(11) | NO   |     | 4       |           |
+-------+---------+------+-----+---------+-----------+
```

Creating a view from a table with hidden columns:

```
CREATE VIEW v1 AS SELECT * FROM t;

DESC v1;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| y     | int(11) | YES  |     | NULL    |       |
| z     | int(11) | NO   |     | 4       |       |
+-------+---------+------+-----+---------+-------+

CREATE VIEW v2 AS SELECT x,y,z FROM t;

DESC v2;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| x     | int(11) | YES  |     | NULL    |       |
| y     | int(11) | YES  |     | NULL    |       |
| z     | int(11) | NO   |     | 4       |       |
+-------+---------+------+-----+---------+-------+
```

Adding a Surrogate Primary Key:

```
create table t1 (x bigint unsigned not null, y varchar(16), z text);

insert into t1 values (123, 'qq11', 'ipsum');

insert into t1 values (123, 'qq22', 'lorem');

alter table t1 add pkid serial primary key invisible first;

insert into t1 values (123, 'qq33', 'amet');

select * from t1;
+-----+------+-------+
| x   | y    | z     |
+-----+------+-------+
| 123 | qq11 | ipsum |
| 123 | qq22 | lorem |
| 123 | qq33 | amet  |
+-----+------+-------+

select pkid, z from t1;
+------+-------+
| pkid | z     |
+------+-------+
|    1 | ipsum |
|    2 | lorem |
|    3 | amet  |
+------+-------+
```

1.1.1.2.1.1 ALTER

1.1.1.2.1.1.1 ALTER TABLE

1.1.1.2.1.1.2 ALTER DATABASE

1.1.1.2.1.1.3 ALTER EVENT

1.1.1.2.1.1.4 ALTER FUNCTION

1.1.1.2.1.1.5 ALTER LOGFILE GROUP

1.1.1.2.1.1.6 ALTER PROCEDURE

1.1.6.4 ALTER SEQUENCE

1.1.1.2.1.1.8 ALTER SERVER

1.1.1.2.1.1.9 ALTER TABLESPACE

1.1.1.1.2 ALTER USER

1.1.1.2.1.1.11 ALTER VIEW

1.1.1.3.3 DROP

Articles on various DROP commands.

DROP DATABASE
Drop all tables and delete database.

DROP EVENT
Removes an existing event.

DROP FUNCTION
Drop a stored function.

DROP FUNCTION UDF
Drop a user-defined function.

DROP INDEX
Drops an index from a table.

DROP LOGFILE GROUP
The DROP LOGFILE GROUP statement is not supported by MariaDB. It was origin...

DROP PACKAGE
Drops a stored package entirely.

DROP PACKAGE BODY
Drops a package body (i.e the implementation) previously created using the CREATE PACKAGE BODY.

DROP PROCEDURE
Drop stored procedure.

DROP ROLE
Drop a role.

DROP SEQUENCE
Deleting a SEQUENCE.

DROP SERVER
Dropping a server definition.

DROP TABLE
Removes definition and data from one or more tables.

DROP TABLESPACE
DROP TABLESPACE is not available in MariaDB.

DROP TRIGGER
Drops a trigger.

DROP USER
Remove one or more MariaDB accounts.

DROP VIEW
Removes one or more views.

There are 1 related questions.

1.1.1.3.3.1 DROP DATABASE

Syntax

```
DROP {DATABASE | SCHEMA} [IF EXISTS] db_name
```

Contents

Description

`DROP DATABASE` drops all tables in the database and deletes the database. Be very careful with this statement! To use DROP DATABASE, you need the DROP privilege on the database. `DROP SCHEMA` is a synonym for `DROP DATABASE`.

Important: When a database is dropped, user privileges on the database are not automatically dropped. See GRANT.

IF EXISTS

Use `IF EXISTS` to prevent an error from occurring for databases that do not exist. A `NOTE` is generated for each non-existent database when using `IF EXISTS`. See SHOW WARNINGS.

Atomic DDL

MariaDB starting with 10.6.1

MariaDB 10.6.1 supports Atomic DDL.

`DROP DATABASE` is implemented as

```
loop over all tables
  DROP TABLE table
```

Each individual DROP TABLE is atomic while `DROP DATABASE` as a whole is crash-safe.

Examples

```
DROP DATABASE bufg;
Query OK, 0 rows affected (0.39 sec)

DROP DATABASE bufg;
ERROR 1008 (HY000): Can't drop database 'bufg'; database doesn't exist

 \W
Show warnings enabled.

DROP DATABASE IF EXISTS bufg;
Query OK, 0 rows affected, 1 warning (0.00 sec)
Note (Code 1008): Can't drop database 'bufg'; database doesn't exist
```

1.1.1.3.3.2 DROP EVENT

Syntax

```
DROP EVENT [IF EXISTS] event_name
```

Description

This statement drops the event named `event_name`. The event immediately ceases being active, and is deleted completely from the server.

If the event does not exist, the error `ERROR 1517 (HY000): Unknown event 'event_name'` results. You can override this and cause the statement to generate a `NOTE` for non-existent events instead by using `IF EXISTS`. See `SHOW WARNINGS`.

This statement requires the `EVENT` privilege. In MySQL 5.1.11 and earlier, an event could be dropped only by its definer, or by a user having the `SUPER` privilege.

Examples

```
DROP EVENT myevent3;
```

Using the IF EXISTS clause:

```
DROP EVENT IF EXISTS myevent3;
Query OK, 0 rows affected, 1 warning (0.01 sec)

SHOW WARNINGS;
+-------+------+-------------------------------+
| Level | Code | Message                       |
+-------+------+-------------------------------+
| Note  | 1305 | Event myevent3 does not exist |
+-------+------+-------------------------------+
```

1.1.1.3.3.3 DROP FUNCTION

Syntax

```
DROP FUNCTION [IF EXISTS] f_name
```

Contents

Description

The DROP FUNCTION statement is used to drop a stored function or a user-defined function (UDF). That is, the specified routine is removed from the server, along with all privileges specific to the function. You must have the `ALTER ROUTINE` privilege for the routine in order to drop it. If the automatic_sp_privileges server system variable is set, both the `ALTER ROUTINE` and `EXECUTE` privileges are granted automatically to the routine creator - see Stored Routine Privileges.

IF EXISTS

The `IF EXISTS` clause is a MySQL/MariaDB extension. It prevents an error from occurring if the function does not exist. A `NOTE` is produced that can be viewed with SHOW WARNINGS.

For dropping a user-defined functions (UDF), see DROP FUNCTION UDF.

Examples

```
DROP FUNCTION hello;
Query OK, 0 rows affected (0.042 sec)

DROP FUNCTION hello;
ERROR 1305 (42000): FUNCTION test.hello does not exist

DROP FUNCTION IF EXISTS hello;
Query OK, 0 rows affected, 1 warning (0.000 sec)

SHOW WARNINGS;
+-------+------+-----------------------------------+
| Level | Code | Message                           |
+-------+------+-----------------------------------+
| Note  | 1305 | FUNCTION test.hello does not exist |
+-------+------+-----------------------------------+
```

1.1.1.3.3.4 DROP FUNCTION UDF

Syntax

```
DROP FUNCTION [IF EXISTS] function_name
```

Contents

Description

This statement drops the user-defined function (UDF) named `function_name`.

To drop a function, you must have the `DELETE privilege` for the mysql database. This is because `DROP FUNCTION` removes the row from the mysql.func system table that records the function's name, type and shared library name.

For dropping a stored function, see DROP FUNCTION.

Upgrading a UDF

To upgrade the UDF's shared library, first run a DROP FUNCTION statement, then upgrade the shared library and finally run the CREATE FUNCTION statement. If you upgrade without following this process, you may crash the server.

Examples

```
DROP FUNCTION jsoncontains_path;
```

IF EXISTS:

```
DROP FUNCTION jsoncontains_path;
ERROR 1305 (42000): FUNCTION test.jsoncontains_path does not exist

DROP FUNCTION IF EXISTS jsoncontains_path;
Query OK, 0 rows affected, 1 warning (0.00 sec)

SHOW WARNINGS;
+-------+------+-------------------------------------------------+
| Level | Code | Message                                         |
+-------+------+-------------------------------------------------+
| Note  | 1305 | FUNCTION test.jsoncontains_path does not exist |
+-------+------+-------------------------------------------------+
```

1.1.1.3.3.5 DROP INDEX

Syntax

```
DROP INDEX [IF EXISTS] index_name ON tbl_name
    [WAIT n |NOWAIT]
```

Contents

Description

`DROP INDEX` drops the index named `index_name` from the table `tbl_name`. This statement is mapped to an `ALTER TABLE` statement to drop the index.

If another connection is using the table, a metadata lock is active, and this statement will wait until the lock is released. This is also true for non-transactional tables.

See ALTER TABLE.

Another shortcut, CREATE INDEX, allows the creation of an index.

To remove the primary key, `` `PRIMARY` `` must be specified as index_name. Note that the quotes are necessary, because `PRIMARY` is a keyword.

Privileges

Executing the `DROP INDEX` statement requires the `INDEX` privilege for the table or the database.

Online DDL

Online DDL is used by default with InnoDB, when the drop index operation supports it.

See InnoDB Online DDL Overview for more information on online DDL with InnoDB.

DROP INDEX IF EXISTS ...

If the `IF EXISTS` clause is used, then MariaDB will return a warning instead of an error if the index does not exist.

WAIT/NOWAIT

Sets the lock wait timeout. See WAIT and NOWAIT.

Progress Reporting

MariaDB provides progress reporting for `DROP INDEX` statement for clients that support the new progress reporting protocol. For example, if you were using the mariadb client, then the progress report might look like this::

1.1.1.3.3.6 DROP LOGFILE GROUP

The `DROP LOGFILE GROUP` statement is not supported by MariaDB. It was originally inherited from MySQL NDB Cluster. See MDEV-19295 for more information.

1.1.1.3.3.7 DROP PACKAGE

MariaDB starting with 10.3.5
Oracle-style packages were introduced in MariaDB 10.3.5.

Syntax

```
DROP PACKAGE [IF EXISTS]  [ db_name . ] package_name
```

Contents
1. Syntax
2. Description

Description

The `DROP PACKAGE` statement can be used when Oracle SQL_MODE is set.

The `DROP PACKAGE` statement drops a stored package entirely:

- Drops the package specification (earlier created using the CREATE PACKAGE statement).
- Drops the package implementation, if the implementation was already created using the CREATE PACKAGE BODY statement.

1.1.1.3.3.8 DROP PACKAGE BODY

MariaDB starting with 10.3.5
Oracle-style packages were introduced in MariaDB 10.3.5.

Syntax

```
DROP PACKAGE BODY [IF EXISTS]  [ db_name . ] package_name
```

Contents
1. Syntax
2. Description

Description

The `DROP PACKAGE BODY` statement can be used when Oracle SQL_MODE is set.

The `DROP PACKAGE BODY` statement drops the package body (i.e the implementation), previously created using the CREATE PACKAGE BODY statement.

Note, `DROP PACKAGE BODY` drops only the package implementation, but does not drop the package specification. Use DROP PACKAGE to drop the package entirely (i.e. both implementation and specification).

1.1.1.3.3.9 DROP PROCEDURE

Syntax

```
DROP PROCEDURE [IF EXISTS] sp_name
```

Contents
1. Syntax
2. Description
3. Examples

Description

This statement is used to drop a stored procedure. That is, the specified routine is removed from the server along with all privileges specific to the procedure. You must have the `ALTER ROUTINE` privilege for the routine. If the `automatic_sp_privileges` server system variable is set, that privilege and `EXECUTE` are granted automatically to the routine creator - see Stored Routine Privileges.

The `IF EXISTS` clause is a MySQL/MariaDB extension. It prevents an error from occurring if the procedure or function does not exist. A `NOTE` is produced that can be viewed with `SHOW WARNINGS`.

While this statement takes effect immediately, threads which are executing a procedure can continue execution.

Examples

```
DROP PROCEDURE simpleproc;
```

IF EXISTS:

```
DROP PROCEDURE simpleproc;
ERROR 1305 (42000): PROCEDURE test.simpleproc does not exist

DROP PROCEDURE IF EXISTS simpleproc;
Query OK, 0 rows affected, 1 warning (0.00 sec)

SHOW WARNINGS;
+-------+------+-----------------------------------------+
| Level | Code | Message                                 |
+-------+------+-----------------------------------------+
| Note  | 1305 | PROCEDURE test.simpleproc does not exist |
+-------+------+-----------------------------------------+
```

1.1.1.1.9 DROP ROLE

1.1.6.5 DROP SEQUENCE

1.1.1.3.3.12 DROP SERVER

Syntax

```
DROP SERVER [ IF EXISTS ] server_name
```

Contents

Description

Drops the server definition for the server named *server_name*. The corresponding row within the mysql.servers table will be deleted. This statement requires the SUPER privilege or, from MariaDB 10.5.2, the FEDERATED ADMIN privilege.

Dropping a server for a table does not affect any FederatedX, FEDERATED, Connect or Spider tables that used this connection information when they were created.

DROP SERVER is not written to the binary log, irrespective of the binary log format being used. From MariaDB 10.1.13, Galera replicates the CREATE SERVER, ALTER SERVER and DROP SERVER statements.

IF EXISTS

If the IF EXISTS clause is used, MariaDB will not return an error if the server does not exist. Unlike all other statements,

DROP SERVER IF EXISTS does not issue a note if the server does not exist. See MDEV-9400.

Examples

```
DROP SERVER s;
```

IF EXISTS:

```
DROP SERVER s;
ERROR 1477 (HY000): The foreign server name you are trying to reference
  does not exist. Data source error:  s

DROP SERVER IF EXISTS s;
Query OK, 0 rows affected (0.00 sec)
```

1.1.1.2.1.8 DROP TABLE

1.1.1.3.3.14 DROP TABLESPACE

The `DROP TABLESPACE` statement is not supported by MariaDB. It was originally inherited from MySQL NDB Cluster. In MySQL 5.7 and later, the statement is also supported for InnoDB. However, MariaDB has chosen not to include that specific feature. See MDEV-19294 for more information.

1.1.1.3.3.15 DROP TRIGGER

Syntax

```
DROP TRIGGER [IF EXISTS] [schema_name.]trigger_name
```

Contents

Description

This statement drops a trigger. The schema (database) name is optional. If the schema is omitted, the trigger is dropped from the default schema. Its use requires the `TRIGGER` privilege for the table associated with the trigger.

Use `IF EXISTS` to prevent an error from occurring for a trigger that does not exist. A `NOTE` is generated for a non-existent trigger when using `IF EXISTS`. See SHOW WARNINGS.

Note: Triggers for a table are also dropped if you drop the table.

Atomic DDL

MariaDB starting with 10.6.1

MariaDB 10.6.1 supports Atomic DDL and `DROP TRIGGER` is atomic.

Examples

```
DROP TRIGGER test.example_trigger;
```

Using the IF EXISTS clause:

```
DROP TRIGGER IF EXISTS test.example_trigger;
Query OK, 0 rows affected, 1 warning (0.01 sec)

SHOW WARNINGS;
+-------+------+-----------------------+
| Level | Code | Message               |
+-------+------+-----------------------+
| Note  | 1360 | Trigger does not exist |
+-------+------+-----------------------+
```

1.1.1.1.3 DROP USER

1.1.1.3.3.17 DROP VIEW

Syntax

```
DROP VIEW [IF EXISTS]
    view_name [, view_name] ...
    [RESTRICT | CASCADE]
```

Contents

Description

`DROP VIEW` removes one or more views. You must have the DROP privilege for each view. If any of the views named in the argument list do not exist, MariaDB returns an error indicating by name which non-existing views it was unable to drop, but it also drops all of the views in the list that do exist.

The `IF EXISTS` clause prevents an error from occurring for views that don't exist. When this clause is given, a `NOTE` is generated for each non-existent view. See SHOW WARNINGS.

`RESTRICT` and `CASCADE`, if given, are parsed and ignored.

It is possible to specify view names as `db_name.view_name`. This is useful to delete views from multiple databases with one statement. See Identifier Qualifiers for details.

The DROP privilege is required to use `DROP TABLE` on non-temporary tables. For temporary tables, no privilege is required, because such tables are only visible for the current session.

If a view references another view, it will be possible to drop the referenced view. However, the other view will reference a view which does not exist any more. Thus, querying it will produce an error similar to the following:

```
ERROR 1356 (HY000): View 'db_name.view_name' references invalid table(s) or
column(s) or function(s) or definer/invoker of view lack rights to use them
```

This problem is reported in the output of CHECK TABLE.

Note that it is not necessary to use `DROP VIEW` to replace an existing view, because `CREATE VIEW` has an `OR REPLACE` clause.

Atomic DDL

MariaDB starting with 10.6.1

MariaDB 10.6.1 supports Atomic DDL and `DROP VIEW` for a singular view is atomic. Dropping multiple views is crash-safe.

Examples

```
DROP VIEW v,v2;
```

Given views `v` and `v2`, but no view `v3`

```
DROP VIEW v,v2,v3;
ERROR 1051 (42S02): Unknown table 'v3'
```

```
DROP VIEW IF EXISTS v,v2,v3;
Query OK, 0 rows affected, 1 warning (0.01 sec)

SHOW WARNINGS;
+-------+------+-------------------------+
| Level | Code | Message                 |
+-------+------+-------------------------+
| Note  | 1051 | Unknown table 'test.v3' |
+-------+------+-------------------------+
```

1.1.1.3.4 Atomic DDL

From MariaDB 10.6.1, we have improved readability for DDL (Data Definition Language) operations to make most of them atomic, and the rest crash-safe, even if the server crashes in the middle of an operation.

The design of Atomic/Crash-safe DDL (MDEV-17567) allows it to work with all storage engines.

Definitions

- Atomic means that either the operation succeeds (and is logged to the binary log or is completely reversed.
- Crash-safe means that in case of a crash, after the server has restarted, all tables are consistent, there are no temporary files or tables on disk and the binary log matches the status of the server.
- DDL Data definition language.
- DML Data manipulation language.
- 'DDL recovery log' or 'DDL log' for short, is the new log file, `ddl_recovery.log` by default, that stores all DDL operations in progress. This is used to recover the state of the server in case of sudden crash.

Background

Before 10.6, in case of a crash, there was a small possibility that one of the following things could happen:

- There could be temporary tables starting with `#sql-alter` or `#sql-shadow` or temporary files ending with '' left.
- The table in the storage engine and the table's .frm file could be out of sync.
- During a multi-table rename, only some of the tables were renamed.

Which DDL Operations are Now Atomic

- CREATE TABLE, except when used with CREATE OR REPLACE, which is only crash safe.
- RENAME TABLE and RENAME TABLES.
- CREATE VIEW
- CREATE SEQUENCE
- CREATE TRIGGER
- DROP TRIGGER
- DROP TABLE and DROP VIEW. Dropping multiple tables is only crash safe.
- ALTER TABLE
- ALTER SEQUENCE is not listed above as it is internally implemented as a DML.

Which DDL Operations are Now Crash Safe

DROP TABLE of Multiple Tables.

DROP TABLE over multiple tables is treated as if every DROP is a separate, atomic operation. This means that after a crash, all fully, or partly, dropped tables will be dropped and logged to the binary log. The undropped tables will be left untouched.

CREATE OR REPLACE TABLE

CREATE OR REPLACE TABLE foo is implemented as:

```
DROP TABLE IF EXISTS foo;
CREATE TABLE foo ...
```

This means that if there is a crash during `CREATE TABLE` then the original table 'foo' will be dropped even if the new table was not created. If the table was not re-created, the binary log will contain the `DROP TABLE` .

DROP DATABASE

DROP DATABASE is implemented as:

```
loop over all tables
  DROP TABLE table
```

Each DROP TABLE is atomic, but in case of a crash, things will work the same way as DROP TABLE with multiple tables.

Atomic with Different Storage Engines

Atomic/Crash-safe DDL works with all storage engines that either have atomic DDLs internally or are able to re-execute `DROP` or `RENAME` in case of failure.

This should be true for most storage engines. The ones that still need some work are:

- The S3 storage engine.
- The partitioning engine. Partitioning should be atomic for most cases, but there are still some known issues that need to be tested and fixed.

The DDL Log Recovery File

The new startup option --log-ddl-recovery=path (`ddl_recovery.log` by default) can be used to specify the place for the DDL log file. This is mainly useful in the case when one has a filesystem on persistent memory, as there is a lot of sync on this file during DDL operations.

This file contains all DDL operations that are in progress.

At MariaDB server startup, the DDL log file is copied to a file with the same base name but with a `-backup.log` suffix. This is mainly done to be able to find out what went wrong if recovery fails.

If the server crashes during recovery (unlikely but possible), the recovery will continue where it was before. The recovery will retry each entry up to 3 times before giving up and proceeding with the next entry.

Conclusions

- We believe that a clean separation of layers leads to an easier-to-maintain solution. The Atomic DDL implementation in MariaDB 10.6 introduced minimal changes to the storage engine API, mainly for native ALTER TABLE.
- In our InnoDB implementation, no file format changes were needed on top of the RENAME undo log that was introduced in MariaDB 10.2.19 for a backup-safe TRUNCATE re-implementation. Correct use of sound design principles (write-ahead logging and transactions; also file creation now follows the ARIES protocol) is sufficient. We removed the hacks (at most one CREATE or DROP per transaction) and correctly implemented `rollback` and `purge` triggers for the InnoDB SYS_INDEXES table.
- Numerous DDL recovery bugs in InnoDB were found and fixed quickly thanks to https://rr-project.org. We are still working on one: data files must not be deleted before the DDL transaction is committed.

Thanks to Atomic/Crash-safe DDL, the MariaDB server is now much more stable and reliable in unstable environments. There is still ongoing work to fix the few remaining issues mentioned above to make all DDL operations Atomic. The target for these is MariaDB 10.7.

1.1.1.3.5 CONSTRAINT

MariaDB supports the implementation of constraints at the table-level using either CREATE TABLE or ALTER TABLE statements. A table constraint restricts the data you can add to the table. If you attempt to insert invalid data on a column, MariaDB throws an error.

Syntax

```
[CONSTRAINT [symbol]] constraint_expression

constraint_expression:
  | PRIMARY KEY [index_type] (index_col_name, ...) [index_option] ...
  | FOREIGN KEY [index_name] (index_col_name, ...)
       REFERENCES tbl_name (index_col_name, ...)
       [ON DELETE reference_option]
       [ON UPDATE reference_option]
  | UNIQUE [INDEX|KEY] [index_name]
       [index_type] (index_col_name, ...) [index_option] ...
  | CHECK (check_constraints)

index_type:
  USING {BTREE | HASH | RTREE}

index_col_name:
  col_name [(length)] [ASC | DESC]

index_option:
  | KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'
  | CLUSTERING={YES|NO}

reference_option:
  RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT
```

Contents

Description

Constraints provide restrictions on the data you can add to a table. This allows you to enforce data integrity from MariaDB, rather than through application logic. When a statement violates a constraint, MariaDB throws an error.

There are four types of table constraints:

Constraint	Description
`PRIMARY KEY`	Sets the column for referencing rows. Values must be unique and not null.
`FOREIGN KEY`	Sets the column to reference the primary key on another table.
`UNIQUE`	Requires values in column or columns only occur once in the table.
`CHECK`	Checks whether the data meets the given condition.

The Information Schema TABLE_CONSTRAINTS Table contains information about tables that have constraints.

FOREIGN KEY Constraints

InnoDB supports foreign key constraints. The syntax for a foreign key constraint definition in InnoDB looks like this:

```
[CONSTRAINT [symbol]] FOREIGN KEY
    [index_name] (index_col_name, ...)
    REFERENCES tbl_name (index_col_name,...)
    [ON DELETE reference_option]
    [ON UPDATE reference_option]

reference_option:
    RESTRICT | CASCADE | SET NULL | NO ACTION
```

The Information Schema REFERENTIAL_CONSTRAINTS table has more information about foreign keys.

CHECK Constraints

Constraints are enforced. Before MariaDB 10.2.1 constraint expressions were accepted in the syntax but ignored.

You can define constraints in 2 different ways:

- `CHECK(expression)` given as part of a column definition.
- `CONSTRAINT [constraint_name] CHECK (expression)`

Before a row is inserted or updated, all constraints are evaluated in the order they are defined. If any constraint expression returns false, then the row will not be inserted or updated. One can use most deterministic functions in a constraint, including UDFs.

```
CREATE TABLE t1 (a INT CHECK (a>2), b INT CHECK (b>2), CONSTRAINT a_greater CHECK (a>b));
```

If you use the second format and you don't give a name to the constraint, then the constraint will get an automatically generated name. This is done so that you can later delete the constraint with ALTER TABLE DROP constraint_name.

One can disable all constraint expression checks by setting the check_constraint_checks variable to `OFF`. This is useful for example when loading a table that violates some constraints that you want to later find and fix in SQL.

Replication

In row-based replication, only the master checks constraints, and failed statements will not be replicated. In statement-based replication, the slaves will also check constraints. Constraints should therefore be identical, as well as deterministic, in a replication environment.

Auto_increment

auto_increment columns are not permitted in check constraints. Before MariaDB 10.2.6, they were permitted, but would not work correctly. See MDEV-11117.

Examples

```
CREATE TABLE product (category INT NOT NULL, id INT NOT NULL,
                      price DECIMAL,
                      PRIMARY KEY(category, id)) ENGINE=INNODB;
CREATE TABLE customer (id INT NOT NULL,
                       PRIMARY KEY (id)) ENGINE=INNODB;
CREATE TABLE product_order (no INT NOT NULL AUTO_INCREMENT,
                            product_category INT NOT NULL,
                            product_id INT NOT NULL,
                            customer_id INT NOT NULL,
                            PRIMARY KEY(no),
                            INDEX (product_category, product_id),
                            FOREIGN KEY (product_category, product_id)
                              REFERENCES product(category, id)
                              ON UPDATE CASCADE ON DELETE RESTRICT,
                            INDEX (customer_id),
                            FOREIGN KEY (customer_id)
                              REFERENCES customer(id)) ENGINE=INNODB;
```

The following examples will work from MariaDB 10.2.1 onwards.

Numeric constraints and comparisons:

```
CREATE TABLE t1 (a INT CHECK (a>2), b INT CHECK (b>2), CONSTRAINT a_greater CHECK (a>b));

INSERT INTO t1(a) VALUES (1);
ERROR 4022 (23000): CONSTRAINT `a` failed for `test`.`t1`

INSERT INTO t1(a,b) VALUES (3,4);
ERROR 4022 (23000): CONSTRAINT `a_greater` failed for `test`.`t1`

INSERT INTO t1(a,b) VALUES (4,3);
Query OK, 1 row affected (0.04 sec)
```

Dropping a constraint:

```
ALTER TABLE t1 DROP CONSTRAINT a_greater;
```

Adding a constraint:

```
ALTER TABLE t1 ADD CONSTRAINT a_greater CHECK (a>b);
```

Date comparisons and character length:

```
CREATE TABLE t2 (name VARCHAR(30) CHECK (CHAR_LENGTH(name)>2), start_date DATE,
  end_date DATE CHECK (start_date IS NULL OR end_date IS NULL OR start_date<end_date));

INSERT INTO t2(name, start_date, end_date) VALUES('Ione', '2003-12-15', '2014-11-09');
Query OK, 1 row affected (0.04 sec)

INSERT INTO t2(name, start_date, end_date) VALUES('Io', '2003-12-15', '2014-11-09');
ERROR 4022 (23000): CONSTRAINT `name` failed for `test`.`t2`

INSERT INTO t2(name, start_date, end_date) VALUES('Ione', NULL, '2014-11-09');
Query OK, 1 row affected (0.04 sec)

INSERT INTO t2(name, start_date, end_date) VALUES('Ione', '2015-12-15', '2014-11-09');
ERROR 4022 (23000): CONSTRAINT `end_date` failed for `test`.`t2`
```

A misplaced parenthesis:

```
CREATE TABLE t3 (name VARCHAR(30) CHECK (CHAR_LENGTH(name>2)), start_date DATE,
  end_date DATE CHECK (start_date IS NULL OR end_date IS NULL OR start_date<end_date));
Query OK, 0 rows affected (0.32 sec)

INSERT INTO t3(name, start_date, end_date) VALUES('Io', '2003-12-15', '2014-11-09');
Query OK, 1 row affected, 1 warning (0.04 sec)

SHOW WARNINGS;
+---------+------+-----------------------------------------+
| Level   | Code | Message                                 |
+---------+------+-----------------------------------------+
| Warning | 1292 | Truncated incorrect DOUBLE value: 'Io' |
+---------+------+-----------------------------------------+
```

Compare the definition of table *t2* to table *t3*. `CHAR_LENGTH(name)>2` is very different to `CHAR_LENGTH(name>2)` as the latter mistakenly performs a numeric comparison on the *name* field, leading to unexpected results.

5.3.11 MERGE

1.1.1.2.1.12 RENAME TABLE

1.1.1.2.1.19 TRUNCATE TABLE

1.1.1.4 Data Manipulation

SQL commands for querying and manipulating data, such as SELECT, UPDATE, DELETE etc.

Selecting Data

Documentation on the SELECT statement and related clauses.

Inserting & Loading Data

Documentation on the INSERT statement and related clauses.

Changing & Deleting Data

Documentation on the UPDATE, REPLACE, and DELETE Statements.

There are 4 related questions.

1.1.1.4.1 Selecting Data

The `SELECT` statement is used for retrieving data from tables, for select specific data, often based on a criteria given in the `WHERE` clause.

SELECT

SQL statement used primarily for retrieving data from a MariaDB database.

Joins & Subqueries

Documentation on the JOIN, UNION, EXCEPT and INTERSECT clauses, and on subqueries.

LIMIT

Documentation of the LIMIT clause.

ORDER BY

Order the results returned from a resultset.

GROUP BY

Aggregate data in a SELECT statement with the GROUP BY clause.

Common Table Expressions

Common table expressions are temporary named result sets.

SELECT WITH ROLLUP

Adds extra rows to the resultset that represent super-aggregate summaries

SELECT INTO OUTFILE

Write the resultset to a formatted file

SELECT INTO DUMPFILE

Write a binary string into file

FOR UPDATE

Acquires a lock on the rows

LOCK IN SHARE MODE

Acquires a write lock.

Optimizer Hints

Optimizer hints There are some options available in SELECT to affect the ex...

PROCEDURE

The PROCEDURE Clause of the SELECT Statement.

HANDLER

Direct access to reading rows from the storage engine.

DUAL

Dummy table name

SELECT ... OFFSET ... FETCH

Allows one to specify an offset, a number of rows to be returned, and wheth...

There are 2 related questions.

1.1.1.4.1.1 SELECT

Syntax

```
SELECT
    [ALL | DISTINCT | DISTINCTROW]
    [HIGH_PRIORITY]
    [STRAIGHT_JOIN]
    [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
    [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr ...]
    [ FROM table_references
      [WHERE where_condition]
      [GROUP BY {col_name | expr | position} [ASC | DESC], ... [WITH ROLLUP]]
      [HAVING where_condition]
      [ORDER BY {col_name | expr | position} [ASC | DESC], ...]
      [LIMIT {[offset,] row_count | row_count OFFSET offset  [ROWS EXAMINED rows_limit] } |
        [OFFSET start { ROW | ROWS }]
        [FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } { ONLY | WITH TIES }] ]
      procedure|[PROCEDURE procedure_name(argument_list)]
      [INTO OUTFILE 'file_name' [CHARACTER SET charset_name] [export_options] |
        INTO DUMPFILE 'file_name' | INTO var_name [, var_name] ]
      [FOR UPDATE lock_option | LOCK IN SHARE MODE lock_option]

export_options:
    [{FIELDS | COLUMNS}
        [TERMINATED BY 'string']
        [[OPTIONALLY] ENCLOSED BY 'char']
        [ESCAPED BY 'char']
    ]
    [LINES
        [STARTING BY 'string']
        [TERMINATED BY 'string']
    ]

lock_option:
    [WAIT n | NOWAIT | SKIP LOCKED]
```

Contents

Description

`SELECT` is used to retrieve rows selected from one or more tables, and can include UNION statements and subqueries.

- Each *select_expr* expression indicates a column or data that you want to retrieve. You must have at least one select expression. See Select Expressions below.
- The `FROM` clause indicates the table or tables from which to retrieve rows. Use either a single table name or a `JOIN` expression. See JOIN for details. If no table is involved, FROM DUAL can be specified.
- Each table can also be specified as `db_name.tabl_name`. Each column can also be specified as `tbl_name.col_name` or even `db_name.tbl_name.col_name`. This allows one to write queries which involve multiple databases. See Identifier Qualifiers for syntax details.
- The `WHERE` clause, if given, indicates the condition or conditions that rows must satisfy to be selected.

`where_condition` is an expression that evaluates to true for each row to be selected. The statement selects all rows if there is no WHERE clause.
    - In the `WHERE` clause, you can use any of the functions and operators that MariaDB supports, except for aggregate (summary) functions. See Functions and Operators and Functions and Modifiers for use with GROUP BY (aggregate).
- Use the ORDER BY clause to order the results.
- Use the LIMIT clause allows you to restrict the results to only a certain number of rows, optionally with an offset.
- Use the GROUP BY and `HAVING` clauses to group rows together when they have columns or computed values in common.

SELECT can also be used to retrieve rows computed without reference to any table.

Select Expressions

A `SELECT` statement must contain one or more select expressions, separated by commas. Each select expression can be one of the following:

- The name of a column.
- Any expression using functions and operators.
- `*` to select all columns from all tables in the `FROM` clause.
- `tbl_name.*` to select all columns from just the table *tbl_name*.

When specifying a column, you can either use just the column name or qualify the column name with the name of the table using `tbl_name.col_name`. The qualified form is useful if you are joining multiple tables in the `FROM` clause. If you do not qualify the column names when selecting from multiple tables, MariaDB will try to find the column in each table. It is an error if that column name exists in multiple tables.

You can quote column names using backticks. If you are qualifying column names with table names, quote each part separately as `` `tbl_name`.`col_name` ``.

If you use any grouping functions in any of the select expressions, all rows in your results will be implicitly grouped, as if you had used `GROUP BY NULL`.

DISTINCT

A query may produce some identical rows. By default, all rows are retrieved, even when their values are the same. To explicitly specify that you want to retrieve identical rows, use the `ALL` option. If you want duplicates to be removed from the resultset, use the `DISTINCT` option. `DISTINCTROW` is a synonym for `DISTINCT`. See also COUNT DISTINCT and SELECT UNIQUE in Oracle mode.

INTO

The `INTO` clause is used to specify that the query results should be written to a file or variable.

- SELECT INTO OUTFILE - formatting and writing the result to an external file.
- SELECT INTO DUMPFILE - binary-safe writing of the unformatted results to an external file.
- SELECT INTO Variable - selecting and setting variables.

The reverse of `SELECT INTO OUTFILE` is LOAD DATA.

LIMIT

Restricts the number of returned rows. See LIMIT and LIMIT ROWS EXAMINED for details.

LOCK IN SHARE MODE/FOR UPDATE

See LOCK IN SHARE MODE and FOR UPDATE for details on the respective locking clauses.

OFFSET ... FETCH

MariaDB starting with 10.6

See SELECT ... OFFSET ... FETCH.

ORDER BY

Order a resultset. See ORDER BY for details.

PARTITION

Specifies to the optimizer which partitions are relevant for the query. Other partitions will not be read. See Partition Pruning and Selection for details.

PROCEDURE

Passes the whole result set to a C Procedure. See PROCEDURE and PROCEDURE ANALYSE (the only built-in procedure not requiring the server to be recompiled).

SKIP LOCKED

MariaDB starting with 10.6

The SKIP LOCKED clause was introduced in MariaDB 10.6.0.

This causes those rows that couldn't be locked (LOCK IN SHARE MODE or FOR UPDATE) to be excluded from the result set. An explicit `NOWAIT` is implied here. This is only implemented on InnoDB tables and ignored otherwise.

SQL_CALC_FOUND_ROWS

When `SQL_CALC_FOUND_ROWS` is used, then MariaDB will calculate how many rows would have been in the result, if there would be no LIMIT clause. The result can be found by calling the function FOUND_ROWS() in your next sql statement.

max_statement_time clause

By using max_statement_time in conjunction with SET STATEMENT, it is possible to limit the execution time of individual queries. For example:

```
SET STATEMENT max_statement_time=100 FOR
  SELECT field1 FROM table_name ORDER BY field1;
```

WAIT/NOWAIT

Set the lock wait timeout. See WAIT and NOWAIT.

Examples

```
SELECT f1,f2 FROM t1 WHERE (f3<=10) AND (f4='y');
```

See Getting Data from MariaDB (Beginner tutorial), or the various sub-articles, for more examples.

1.1.1.4.1.2 Joins & Subqueries

Documentation on the JOIN, UNION, EXCEPT and INTERSECT clauses, and on subqueries.

Joins
Querying from multiple tables.

Subqueries
Queries within queries.

UNION
Combine the results from multiple SELECT statements into a single result set.

EXCEPT
Subtraction of two result sets.

INTERSECT

Records that are present in both result sets will be included in the result of the operation.

Precedence Control in Table Operations

Controlling order of execution in SELECT, UNION, EXCEPT, and INTERSECT.

MINUS

Synonym for EXCEPT.

1.1.1.4.1.2.1 Joins

Articles about joins in MariaDB.

Joining Tables with JOIN Clauses

An introductory tutorial on using the JOIN clause.

More Advanced Joins

A more advanced tutorial on JOINs.

JOIN Syntax

Description MariaDB supports the following JOIN syntaxes for the table_refe...

Comma vs JOIN

A query to grab the list of phone numbers for clients who ordered in the la...

There are 1 related questions.

6.2.5 Joining Tables with JOIN Clauses

1.1.1.4.1.2.1.2 More Advanced Joins

Contents

This article is a follow up to the Introduction to JOINs page. If you're just getting started with JOINs, go through that page first and then come back here.

The Employee Database

Let us begin by using an example employee database of a fairly small family business, which does not anticipate expanding in the future.

First, we create the table that will hold all of the employees and their contact information:

```
CREATE TABLE `Employees` (
  `ID` TINYINT(3) UNSIGNED NOT NULL AUTO_INCREMENT,
  `First_Name` VARCHAR(25) NOT NULL,
  `Last_Name` VARCHAR(25) NOT NULL,
  `Position` VARCHAR(25) NOT NULL,
  `Home_Address` VARCHAR(50) NOT NULL,
  `Home_Phone` VARCHAR(12) NOT NULL,
  PRIMARY KEY (`ID`)
) ENGINE=MyISAM;
```

Next, we add a few employees to the table:

```
INSERT INTO `Employees` (`First_Name`, `Last_Name`, `Position`, `Home_Address`, `Home_Phone`)
  VALUES
  ('Mustapha', 'Mond', 'Chief Executive Officer', '692 Promiscuous Plaza', '326-555-3492'),
  ('Henry', 'Foster', 'Store Manager', '314 Savage Circle', '326-555-3847'),
  ('Bernard', 'Marx', 'Cashier', '1240 Ambient Avenue', '326-555-8456'),
  ('Lenina', 'Crowne', 'Cashier', '281 Bumblepuppy Boulevard', '328-555-2349'),
  ('Fanny', 'Crowne', 'Restocker', '1023 Bokanovsky Lane', '326-555-6329'),
  ('Helmholtz', 'Watson', 'Janitor', '944 Soma Court', '329-555-2478');
```

Now, we create a second table, containing the hours which each employee clocked in and out during the week:

```
CREATE TABLE `Hours` (
  `ID` TINYINT(3) UNSIGNED NOT NULL,
  `Clock_In` DATETIME NOT NULL,
  `Clock_Out` DATETIME NOT NULL
) ENGINE=MyISAM;
```

Finally, although it is a lot of information, we add a full week of hours for each of the employees into the second table that we created:

```
INSERT INTO `Hours`
  VALUES
  ('1', '2005-08-08 07:00:42', '2005-08-08 17:01:36'),
  ('1', '2005-08-09 07:01:34', '2005-08-09 17:10:11'),
  ('1', '2005-08-10 06:59:56', '2005-08-10 17:09:29'),
  ('1', '2005-08-11 07:00:17', '2005-08-11 17:00:47'),
  ('1', '2005-08-12 07:02:29', '2005-08-12 16:59:12'),
  ('2', '2005-08-08 07:00:25', '2005-08-08 17:03:13'),
  ('2', '2005-08-09 07:00:57', '2005-08-09 17:05:09'),
  ('2', '2005-08-10 06:58:43', '2005-08-10 16:58:24'),
  ('2', '2005-08-11 07:01:58', '2005-08-11 17:00:45'),
  ('2', '2005-08-12 07:02:12', '2005-08-12 16:58:57'),
  ('3', '2005-08-08 07:00:12', '2005-08-08 17:01:32'),
  ('3', '2005-08-09 07:01:10', '2005-08-09 17:00:26'),
  ('3', '2005-08-10 06:59:53', '2005-08-10 17:02:53'),
  ('3', '2005-08-11 07:01:15', '2005-08-11 17:04:23'),
  ('3', '2005-08-12 07:00:51', '2005-08-12 16:57:52'),
  ('4', '2005-08-08 06:54:37', '2005-08-08 17:01:23'),
  ('4', '2005-08-09 06:58:23', '2005-08-09 17:00:54'),
  ('4', '2005-08-10 06:59:14', '2005-08-10 17:00:12'),
  ('4', '2005-08-11 07:00:49', '2005-08-11 17:00:34'),
  ('4', '2005-08-12 07:01:09', '2005-08-12 16:58:29'),
  ('5', '2005-08-08 07:00:04', '2005-08-08 17:01:43'),
  ('5', '2005-08-09 07:02:12', '2005-08-09 17:02:13'),
  ('5', '2005-08-10 06:59:39', '2005-08-10 17:03:37'),
  ('5', '2005-08-11 07:01:26', '2005-08-11 17:00:03'),
  ('5', '2005-08-12 07:02:15', '2005-08-12 16:59:02'),
  ('6', '2005-08-08 07:00:12', '2005-08-08 17:01:02'),
  ('6', '2005-08-09 07:03:44', '2005-08-09 17:00:00'),
  ('6', '2005-08-10 06:54:19', '2005-08-10 17:03:31'),
  ('6', '2005-08-11 07:00:05', '2005-08-11 17:02:57'),
  ('6', '2005-08-12 07:02:07', '2005-08-12 16:58:23');
```

Working with the Employee Database

Now that we have a cleanly structured database to work with, let us begin this tutorial by stepping up one notch from the last tutorial and filtering our information a little.

Filtering by Name

Earlier in the week, an anonymous employee reported that Helmholtz came into work almost four minutes late; to verify this, we will begin our investigation by filtering out employees whose first names are "Helmholtz":

```
SELECT
  `Employees`.`First_Name`,
  `Employees`.`Last_Name`,
  `Hours`.`Clock_In`,
  `Hours`.`Clock_Out`
FROM `Employees`
INNER JOIN `Hours` ON `Employees`.`ID` = `Hours`.`ID`
WHERE `Employees`.`First_Name` = 'Helmholtz';
```

The result:

```
+------------+-----------+---------------------+---------------------+
| First_Name | Last_Name | Clock_In            | Clock_Out           |
+------------+-----------+---------------------+---------------------+
| Helmholtz  | Watson    | 2005-08-08 07:00:12 | 2005-08-08 17:01:02 |
| Helmholtz  | Watson    | 2005-08-09 07:03:44 | 2005-08-09 17:00:00 |
| Helmholtz  | Watson    | 2005-08-10 06:54:19 | 2005-08-10 17:03:31 |
| Helmholtz  | Watson    | 2005-08-11 07:00:05 | 2005-08-11 17:02:57 |
| Helmholtz  | Watson    | 2005-08-12 07:02:07 | 2005-08-12 16:58:23 |
+------------+-----------+---------------------+---------------------+
5 rows in set (0.00 sec)
```

This is obviously more information than we care to trudge through, considering we only care about when he arrived past 7:00:59 on any given day within this week; thus, we need to add a couple more conditions to our WHERE clause.

Filtering by Name, Date and Time

In the following example, we will filter out all of the times which Helmholtz clocked in that were before 7:01:00 and during the work week that lasted from the 8th to the 12th of August:

```
SELECT
  `Employees`.`First_Name`,
  `Employees`.`Last_Name`,
  `Hours`.`Clock_In`,
  `Hours`.`Clock_Out`
FROM `Employees`
INNER JOIN `Hours` ON `Employees`.`ID` = `Hours`.`ID`
WHERE `Employees`.`First_Name` = 'Helmholtz'
AND DATE_FORMAT(`Hours`.`Clock_In`, '%Y-%m-%d') >= '2005-08-08'
AND DATE_FORMAT(`Hours`.`Clock_In`, '%Y-%m-%d') <= '2005-08-12'
AND DATE_FORMAT(`Hours`.`Clock_In`, '%H:%i:%S') > '07:00:59';
```

The result:

```
+------------+-----------+---------------------+---------------------+
| First_Name | Last_Name | Clock_In            | Clock_Out           |
+------------+-----------+---------------------+---------------------+
| Helmholtz  | Watson    | 2005-08-09 07:03:44 | 2005-08-09 17:00:00 |
| Helmholtz  | Watson    | 2005-08-12 07:02:07 | 2005-08-12 16:58:23 |
+------------+-----------+---------------------+---------------------+
2 rows in set (0.00 sec)
```

We have now, by merely adding a few more conditions, eliminated all of the irrelevant information; Helmholtz was late to work on the 9th and the 12th of August.

Displaying Total Work Hours per Day

Suppose you would like toâ€”based on the information stored in both of our tables in the employee databaseâ€”develop a quick list of the total hours each employee has worked for each day recorded; a simple way to estimate the time each employee worked per day is exemplified below:

```
SELECT
  `Employees`.`ID`,
  `Employees`.`First_Name`,
  `Employees`.`Last_Name`,
  `Hours`.`Clock_In`,
  `Hours`.`Clock_Out`,
DATE_FORMAT(`Hours`.`Clock_Out`, '%T')-DATE_FORMAT(`Hours`.`Clock_In`, '%T') AS 'Total_Hours'
FROM `Employees` INNER JOIN `Hours` ON `Employees`.`ID` = `Hours`.`ID`;
```

The result (limited by 10):

```
+----+------------+-----------+---------------------+---------------------+-------------+
| ID | First_Name | Last_Name | Clock_In            | Clock_Out           | Total_Hours |
+----+------------+-----------+---------------------+---------------------+-------------+
|  1 | Mustapha   | Mond      | 2005-08-08 07:00:42 | 2005-08-08 17:01:36 |          10 |
|  1 | Mustapha   | Mond      | 2005-08-09 07:01:34 | 2005-08-09 17:10:11 |          10 |
|  1 | Mustapha   | Mond      | 2005-08-10 06:59:56 | 2005-08-10 17:09:29 |          11 |
|  1 | Mustapha   | Mond      | 2005-08-11 07:00:17 | 2005-08-11 17:00:47 |          10 |
|  1 | Mustapha   | Mond      | 2005-08-12 07:02:29 | 2005-08-12 16:59:12 |           9 |
|  2 | Henry      | Foster    | 2005-08-08 07:00:25 | 2005-08-08 17:03:13 |          10 |
|  2 | Henry      | Foster    | 2005-08-09 07:00:57 | 2005-08-09 17:05:09 |          10 |
|  2 | Henry      | Foster    | 2005-08-10 06:58:43 | 2005-08-10 16:58:24 |          10 |
|  2 | Henry      | Foster    | 2005-08-11 07:01:58 | 2005-08-11 17:00:45 |          10 |
|  2 | Henry      | Foster    | 2005-08-12 07:02:12 | 2005-08-12 16:58:57 |           9 |
+----+------------+-----------+---------------------+---------------------+-------------+
10 rows in set (0.00 sec)
```

1.1.1.4.1.2.1.3 JOIN Syntax

Description

MariaDB supports the following `JOIN` syntaxes for the `table_references` part of `SELECT` statements and multiple-table `DELETE` and `UPDATE` statements:

```
table_references:
    table_reference [, table_reference] ...

table_reference:
    table_factor
  | join_table

table_factor:
    tbl_name [PARTITION (partition_list)]
        [query_system_time_period_specification] [[AS] alias] [index_hint_list]
  | table_subquery [query_system_time_period_specification] [AS] alias
  | ( table_references )
  | { ON table_reference LEFT OUTER JOIN table_reference
        ON conditional_expr }

join_table:
    table_reference [INNER | CROSS] JOIN table_factor [join_condition]
  | table_reference STRAIGHT_JOIN table_factor
  | table_reference STRAIGHT_JOIN table_factor ON conditional_expr
  | table_reference {LEFT|RIGHT} [OUTER] JOIN table_reference join_condition
  | table_reference NATURAL [{LEFT|RIGHT} [OUTER]] JOIN table_factor

join_condition:
    ON conditional_expr
  | USING (column_list)

query_system_time_period_specification:
    FOR SYSTEM_TIME AS OF point_in_time
  | FOR SYSTEM_TIME BETWEEN point_in_time AND point_in_time
  | FOR SYSTEM_TIME FROM point_in_time TO point_in_time
  | FOR SYSTEM_TIME ALL

point_in_time:
    [TIMESTAMP] expression
  | TRANSACTION expression

index_hint_list:
    index_hint [, index_hint] ...

index_hint:
    USE {INDEX|KEY}
      [{FOR {JOIN|ORDER BY|GROUP BY}] ([index_list])
  | IGNORE {INDEX|KEY}
      [{FOR {JOIN|ORDER BY|GROUP BY}] (index_list)
  | FORCE {INDEX|KEY}
      [{FOR {JOIN|ORDER BY|GROUP BY}] (index_list)

index_list:
    index_name [, index_name] ...
```

A table reference is also known as a join expression.

Each table can also be specified as `db_name.tabl_name`. This allows to write queries which involve multiple databases. See Identifier Qualifiers for syntax details.

The syntax of `table_factor` is extended in comparison with the SQL Standard. The latter accepts only `table_reference`, not a list of them inside a pair of parentheses.

This is a conservative extension if we consider each comma in a list of table_reference items as equivalent to an inner join. For example:

```
SELECT * FROM t1 LEFT JOIN (t2, t3, t4)
                 ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)
```

is equivalent to:

```
SELECT * FROM t1 LEFT JOIN (t2 CROSS JOIN t3 CROSS JOIN t4)
                 ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)
```

In MariaDB, `CROSS JOIN` is a syntactic equivalent to `INNER JOIN` (they can replace each other). In standard SQL, they are not equivalent. `INNER JOIN` is used with an `ON` clause, `CROSS JOIN` is used otherwise.

In general, parentheses can be ignored in join expressions containing only inner join operations. MariaDB also supports nested joins (see http://dev.mysql.com/doc/refman/5.1/en/nested-join-optimization.html).

See System-versioned tables for more information about `FOR SYSTEM_TIME` syntax.

Index hints can be specified to affect how the MariaDB optimizer makes use of indexes. For more information, see How to force query plans.

Examples

```
SELECT left_tbl.*
  FROM left_tbl LEFT JOIN right_tbl ON left_tbl.id = right_tbl.id
  WHERE right_tbl.id IS NULL;
```

1.1.1.4.1.2.1.4 Comma vs JOIN

A query to grab the list of phone numbers for clients who ordered in the last two weeks might be written in a couple of ways. Here are two:

```
SELECT *
FROM
  clients,
  orders,
  phoneNumbers
WHERE
  clients.id = orders.clientId
  AND clients.id = phoneNumbers.clientId
  AND orderPlaced >= NOW() - INTERVAL 2 WEEK;
```

```
SELECT *
FROM
  clients
  INNER JOIN orders ON clients.id = orders.clientId
  INNER JOIN phoneNumbers ON clients.id = phoneNumbers.clientId
WHERE
  orderPlaced >= NOW() - INTERVAL 2 WEEK;
```

Does it make a difference? Not much as written. But you should use the second form. Why?

- **Readability.** Once the WHERE clause contains more than two conditions, it becomes tedious to pick out the difference between business logic (only dates in the last two weeks) and relational logic (which fields relate clients to orders). Using the JOIN syntax with an ON clause makes the WHERE list shorter, and makes it very easy to see how tables relate to each other.
- **Flexibility.** Let's say we need to see all clients even if they don't have a phone number in the system. With the second version, it's easy; just change `INNER JOIN phoneNumbers` to `LEFT JOIN phoneNumbers`. Try that with the first version, and MySQL version 5.0.12+ will issue a syntax error because of the change in precedence between the comma operator and the JOIN keyword. The solution is to rearrange the FROM clause or add parentheses to override the precedence, and that quickly becomes frustrating.
- **Portability.** The changes in 5.0.12 were made to align with SQL:2003. If your queries use standard syntax, you will have an easier time switching to a different database should the need ever arise.

1.1.1.4.1.2.2 Subqueries

A subquery is a query nested in another query.

Scalar Subqueries

Subquery returning a single value.

Row Subqueries

Subquery returning a row.

Subqueries and ALL

Return true if the comparison returns true for each row, or the subquery returns no rows.

Subqueries and ANY

Return true if the comparison returns true for at least one row returned by the subquery.

Subqueries and EXISTS

Returns true if the subquery returns any rows.

Subqueries in a FROM Clause

Subqueries are more commonly placed in a WHERE clause, but can also form part of the FROM clause.

Subquery Optimizations

Articles about subquery optimizations in MariaDB.

Subqueries and JOINs

Rewriting subqueries as JOINs, and using subqueries instead of JOINs.

Subquery Limitations

There are a number of limitations regarding subqueries.

There are 1 related questions.

1.1.1.4.1.2.2.1 Scalar Subqueries

A scalar subquery is a subquery that returns a single value. This is the simplest form of a subquery, and can be used in most places a literal or single column value is valid.

The data type, length and character set and collation are all taken from the result returned by the subquery. The result of a subquery can always be NULL, that is, no result returned. Even if the original value is defined as NOT NULL, this is disregarded.

A subquery cannot be used where only a literal is expected, for example LOAD DATA INFILE expects a literal string containing the file name, and LIMIT requires a literal integer.

Examples

```
CREATE TABLE sq1 (num TINYINT);

CREATE TABLE sq2 (num TINYINT);

INSERT INTO sq1 VALUES (1);

INSERT INTO sq2 VALUES (10* (SELECT num FROM sq1));

SELECT * FROM sq2;
+------+
| num  |
+------+
|   10 |
+------+
```

Inserting a second row means the subquery is no longer a scalar, and this particular query is not valid:

```
INSERT INTO sq1 VALUES (2);

INSERT INTO sq2 VALUES (10* (SELECT num FROM sq1));
ERROR 1242 (21000): Subquery returns more than 1 row
```

No rows in the subquery, so the scalar is NULL:

```
INSERT INTO sq2 VALUES (10* (SELECT num FROM sq3 WHERE num='3'));

SELECT * FROM sq2;
+------+
| num  |
+------+
|   10 |
| NULL |
+------+
```

A more traditional scalar subquery, as part of a WHERE clause:

```
SELECT * FROM sq1 WHERE num = (SELECT MAX(num)/10 FROM sq2);
+------+
| num  |
+------+
|    1 |
+------+
```

1.1.1.4.1.2.2.2 Row Subqueries

A row subquery is a subquery returning a single row, as opposed to a scalar subquery, which returns a single column from a row, or a literal.

Examples

```
CREATE TABLE staff (name VARCHAR(10), age TINYINT);

CREATE TABLE customer (name VARCHAR(10), age TINYINT);

INSERT INTO staff VALUES ('Bilhah',37), ('Valerius',61), ('Maia',25);

INSERT INTO customer VALUES ('Thanasis',48), ('Valerius',61), ('Brion',51);

SELECT * FROM staff WHERE (name,age) = (SELECT name,age FROM customer WHERE name='Valerius');
+----------+------+
| name     | age  |
+----------+------+
| Valerius |   61 |
+----------+------+
```

Finding all rows in one table also in another:

```
SELECT name,age FROM staff WHERE (name,age) IN (SELECT name,age FROM customer);
+----------+------+
| name     | age  |
+----------+------+
| Valerius |   61 |
+----------+------+
```

1.1.1.4.1.2.2.3 Subqueries and ALL

Contents

Subqueries using the ALL keyword will return `true` if the comparison returns `true` for each row returned by the subquery, or the subquery returns no rows.

Syntax

```
scalar_expression comparison_operator ALL <Table subquery>
```

- `scalar_expression` may be any expression that evaluates to a single value
- `comparison_operator` may be any one of: `=`, `>`, `<`, `>=`, `<=`, `<>` or `!=`

`ALL` returns:

- `NULL` if the comparison operator returns `NULL` for at least one row returned by the Table subquery or scalar_expression returns `NULL`.
- `FALSE` if the comparison operator returns `FALSE` for at least one row returned by the Table subquery.
- `TRUE` if the comparison operator returns `TRUE` for all rows returned by the Table subquery, or if Table subquery returns no rows.

`NOT IN` is an alias for `<> ALL`.

Examples

```
CREATE TABLE sq1 (num TINYINT);

CREATE TABLE sq2 (num2 TINYINT);

INSERT INTO sq1 VALUES(100);

INSERT INTO sq2 VALUES(40),(50),(60);

SELECT * FROM sq1 WHERE num > ALL (SELECT * FROM sq2);
+------+
| num  |
+------+
|  100 |
+------+
```

Since `100` > all of `40`, `50` and `60`, the evaluation is true and the row is returned

Adding a second row to sq1, where the evaluation for that record is false:

```
INSERT INTO sq1 VALUES(30);

SELECT * FROM sq1 WHERE num > ALL (SELECT * FROM sq2);
+------+
| num  |
+------+
|  100 |
+------+
```

Adding a new row to sq2, causing all evaluations to be false:

```
INSERT INTO sq2 VALUES(120);

SELECT * FROM sq1 WHERE num > ALL (SELECT * FROM sq2);
Empty set (0.00 sec)
```

When the subquery returns no results, the evaluation is still true:

```
SELECT * FROM sq1 WHERE num > ALL (SELECT * FROM sq2 WHERE num2 > 300);
+------+
| num  |
+------+
|  100 |
|   30 |
+------+
```

Evaluating against a NULL will cause the result to be unknown, or not true, and therefore return no rows:

```
INSERT INTO sq2 VALUES (NULL);

SELECT * FROM sq1 WHERE num > ALL (SELECT * FROM sq2);
```

1.1.1.4.1.2.2.4 Subqueries and ANY

Contents

Subqueries using the ANY keyword will return `true` if the comparison returns `true` for at least one row returned by the subquery.

Syntax

The required syntax for an `ANY` or `SOME` quantified comparison is:

```
scalar_expression comparison_operator ANY <Table subquery>
```

Or:

```
scalar_expression comparison_operator SOME <Table subquery>
```

- `scalar_expression` may be any expression that evaluates to a single value.
- `comparison_operator` may be any one of `=`, `>`, `<`, `>=`, `<=`, `<>` or `!=`.

`ANY` returns:

- `TRUE` if the comparison operator returns `TRUE` for at least one row returned by the Table subquery.
- `FALSE` if the comparison operator returns `FALSE` for all rows returned by the Table subquery, or Table subquery has zero rows.
- `NULL` if the comparison operator returns `NULL` for at least one row returned by the Table subquery and doesn't returns `TRUE` for any of them, or if scalar_expression returns `NULL`.

`SOME` is a synmonym for `ANY`, and `IN` is a synonym for `= ANY`

Examples

```
CREATE TABLE sq1 (num TINYINT);

CREATE TABLE sq2 (num2 TINYINT);

INSERT INTO sq1 VALUES(100);

INSERT INTO sq2 VALUES(40),(50),(120);

SELECT * FROM sq1 WHERE num > ANY (SELECT * FROM sq2);
+------+
| num  |
+------+
|  100 |
+------+
```

`100` is greater than two of the three values, and so the expression evaluates as true.

SOME is a synonym for ANY:

```
SELECT * FROM sq1 WHERE num < SOME (SELECT * FROM sq2);
+------+
| num  |
+------+
|  100 |
+------+
```

`IN` is a synonym for `= ANY`, and here there are no matches, so no results are returned:

```
SELECT * FROM sq1 WHERE num IN (SELECT * FROM sq2);
Empty set (0.00 sec)
```

```
INSERT INTO sq2 VALUES(100);
Query OK, 1 row affected (0.05 sec)

SELECT * FROM sq1 WHERE num <> ANY (SELECT * FROM sq2);
+------+
| num  |
+------+
|  100 |
+------+
```

Reading this query, the results may be counter-intuitive. It may seem to read as "SELECT * FROM sq1 WHERE num does not match any results in sq2. Since it does match 100, it could seem that the results are incorrect. However, the query returns a result if the match does not match any *of* sq2. Since `100` already does not match `40`, the expression evaluates to true immediately, regardless of the 100's matching. It may be more easily readable to use SOME in a case such as this:

```
SELECT * FROM sq1 WHERE num <> SOME (SELECT * FROM sq2);
+------+
| num  |
+------+
|  100 |
+------+
```

1.1.1.4.1.2.2.5 Subqueries and EXISTS

Syntax

```
SELECT ... WHERE EXISTS <Table subquery>
```

Description

Subqueries using the `EXISTS` keyword will return `true` if the subquery returns any rows. Conversely, subqueries using `NOT EXISTS` will return `true` only if the subquery returns no rows from the table.

EXISTS subqueries ignore the columns specified by the SELECT of the subquery, since they're not relevant. For example,

```
SELECT col1 FROM t1 WHERE EXISTS (SELECT * FROM t2);
```

and

```
SELECT col1 FROM t1 WHERE EXISTS (SELECT col2 FROM t2);
```

produce identical results.

Examples

```
CREATE TABLE sq1 (num TINYINT);

CREATE TABLE sq2 (num2 TINYINT);

INSERT INTO sq1 VALUES(100);

INSERT INTO sq2 VALUES(40),(50),(60);

SELECT * FROM sq1 WHERE EXISTS (SELECT * FROM sq2 WHERE num2>50);
+------+
| num  |
+------+
|  100 |
+------+

SELECT * FROM sq1 WHERE NOT EXISTS (SELECT * FROM sq2 GROUP BY num2 HAVING MIN(num2)=40);
Empty set (0.00 sec)
```

1.1.1.4.1.2.2.6 Subqueries in a FROM Clause

Although subqueries are more commonly placed in a WHERE clause, they can also form part of the FROM clause. Such subqueries are commonly called derived tables.

If a subquery is used in this way, you must also use an AS clause to name the result of the subquery.

ORACLE mode

MariaDB starting with 10.6.0
From MariaDB 10.6.0, anonymous subqueries in a FROM clause (no AS clause) are permitted in ORACLE mode.

Examples

```
CREATE TABLE student (name CHAR(10), test CHAR(10), score TINYINT);

INSERT INTO student VALUES
  ('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),
  ('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),
  ('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),
  ('Tatiana', 'SQL', 87), ('Tatiana', 'Tuning', 83);
```

Assume that, given the data above, you want to return the average total for all students. In other words, the average of Chun's 148 (75+73), Esben's 74 (43+31), etc.

You cannot do the following:

```
SELECT AVG(SUM(score)) FROM student GROUP BY name;
ERROR 1111 (HY000): Invalid use of group function
```

A subquery in the FROM clause is however permitted:

```
SELECT AVG(sq_sum) FROM (SELECT SUM(score) AS sq_sum FROM student GROUP BY name) AS t;
+-------------+
| AVG(sq_sum) |
+-------------+
|    134.0000 |
+-------------+
```

From MariaDB 10.6 in ORACLE mode, the following is permitted:

```
SELECT * FROM (SELECT 1 FROM DUAL), (SELECT 2 FROM DUAL);
```

3.3.4.2 Subquery Optimizations

3.3.4.2.1 Subquery Optimizations Map

3.3.4.2.2 Semi-join Subquery Optimizations

3.3.4.2.3 Table Pullout Optimization

3.3.4.2.4 Non-semi-join Subquery Optimizations

3.3.4.2.5 Subquery Cache

3.3.4.2.6 Condition Pushdown Into IN subqueries

3.3.4.2.7 Conversion of Big IN Predicates Into Subqueries

3.3.4.2.8 EXISTS-to-IN Optimization

3.3.4.2.9 Optimizing GROUP BY and DISTINCE Clauses in Subqueries

1.1.1.4.1.2.2.8 Subqueries and JOINs

A subquery can quite often, but not in all cases, be rewritten as a JOIN.

Contents

Rewriting Subqueries as JOINS

A subquery using `IN` can be rewritten with the `DISTINCT` keyword, for example:

```
SELECT * FROM table1 WHERE col1 IN (SELECT col1 FROM table2);
```

can be rewritten as:

```
SELECT DISTINCT table1.* FROM table1, table2 WHERE table1.col1=table2.col1;
```

`NOT IN` or `NOT EXISTS` queries can also be rewritten. For example, these two queries returns the same result:

```
SELECT * FROM table1 WHERE col1 NOT IN (SELECT col1 FROM table2);
SELECT * FROM table1 WHERE NOT EXISTS (SELECT col1 FROM table2 WHERE table1.col1=table2.col1);
```

and both can be rewritten as:

```
SELECT table1.* FROM table1 LEFT JOIN table2 ON table1.id=table2.id WHERE table2.id IS NULL;
```

Subqueries that can be rewritten as a LEFT JOIN are sometimes more efficient.

Using Subqueries instead of JOINS

There are some scenarios, though, which call for subqueries rather than joins:

- When you want duplicates, but not false duplicates. Suppose `Table_1` has three rows — { `1` , `1` , `2` } — and `Table_2` has two rows — { `1` , `2` , `2` }. If you need to list the rows in `Table_1` which are also in `Table_2` , only this subquery-based `SELECT` statement will give the right answer (`1` , `1` , `2`):

```
SELECT Table_1.column_1
FROM   Table_1
WHERE  Table_1.column_1 IN
  (SELECT Table_2.column_1
    FROM   Table_2);
```

  This SQL statement won't work:

```
SELECT Table_1.column_1
FROM   Table_1,Table_2
WHERE  Table_1.column_1 = Table_2.column_1;
```

  because the result will be { `1` , `1` , `2` , `2` } — and the duplication of 2 is an error. This SQL statement won't work either:

```
SELECT DISTINCT Table_1.column_1
FROM   Table_1,Table_2
WHERE  Table_1.column_1 = Table_2.column_1;
```

  because the result will be { `1` , `2` } — and the removal of the duplicated 1 is an error too.

- When the outermost statement is not a query. The SQL statement:

```
UPDATE Table_1 SET column_1 = (SELECT column_1 FROM Table_2);
```

  can't be expressed using a join unless some rare SQL3 features are used.

- When the join is over an expression. The SQL statement:

```
SELECT * FROM Table_1
WHERE column_1 + 5 =
  (SELECT MAX(column_1) FROM Table_2);
```

  is hard to express with a join. In fact, the only way we can think of is this SQL statement:

```
SELECT Table_1.*
FROM   Table_1,
      (SELECT MAX(column_1) AS max_column_1 FROM Table_2) AS Table_2
WHERE  Table_1.column_1 + 5 = Table_2.max_column_1;
```

  which still involves a parenthesized query, so nothing is gained from the transformation.

- When you want to see the exception. For example, suppose the question is: what books are longer than Das Kapital? These two queries are effectively almost the same:

```
SELECT DISTINCT Bookcolumn_1.*
FROM   Books AS Bookcolumn_1 JOIN Books AS Bookcolumn_2 USING(page_count)
WHERE  title = 'Das Kapital';

SELECT DISTINCT Bookcolumn_1.*
FROM   Books AS Bookcolumn_1
WHERE  Bookcolumn_1.page_count >
  (SELECT DISTINCT page_count
  FROM   Books AS Bookcolumn_2
  WHERE  title = 'Das Kapital');
```

  The difference is between these two SQL statements is, if there are two editions of *Das Kapital* (with different page counts), then the self-join example will return the books which are longer than the shortest edition of *Das Kapital*. That might be the wrong answer, since the original question didn't ask for "... longer than `ANY` book named *Das Kapital*" (it seems to contain a false assumption that there's only one edition).

1.1.1.4.1.2.2.9 Subquery Limitations

Contents

There are a number of limitations regarding subqueries, which are discussed below.

The following tables and data will be used in the examples that follow:

```
CREATE TABLE staff(name VARCHAR(10),age TINYINT);

CREATE TABLE customer(name VARCHAR(10),age TINYINT);
```

```
INSERT INTO staff VALUES
('Bilhah',37), ('Valerius',61), ('Maia',25);

INSERT INTO customer VALUES
('Thanasis',48), ('Valerius',61), ('Brion',51);
```

ORDER BY and LIMIT

To use ORDER BY or limit LIMIT in subqueries both must be used.. For example:

```
SELECT * FROM staff WHERE name IN (SELECT name FROM customer ORDER BY name);
+----------+------+
| name     | age  |
+----------+------+
| Valerius |   61 |
+----------+------+
```

is valid, but

```
SELECT * FROM staff WHERE name IN (SELECT NAME FROM customer ORDER BY name LIMIT 1);
ERROR 1235 (42000): This version of MariaDB doesn't
  yet support 'LIMIT & IN/ALL/ANY/SOME subquery'
```

is not.

Modifying and Selecting from the Same Table

It's not possible to both modify and select from the same table in a subquery. For example:

```
DELETE FROM staff WHERE name = (SELECT name FROM staff WHERE age=61);
ERROR 1093 (HY000): Table 'staff' is specified twice, both
  as a target for 'DELETE' and as a separate source for data
```

Row Comparison Operations

There is only partial support for row comparison operations. The expression in

```
expr op {ALL|ANY|SOME} subquery,
```

must be scalar and the subquery can only return a single column.

However, because of the way `IN` is implemented (it is rewritten as a sequence of `=` comparisons and `AND`), the expression in

```
expression [NOT] IN subquery
```

is permitted to be an n-tuple and the subquery can return rows of n-tuples.

For example:

```
SELECT * FROM staff WHERE (name,age) NOT IN (
  SELECT name,age FROM customer WHERE age >=51]
);
+--------+------+
| name   | age  |
+--------+------+
| Bilhah |   37 |
| Maia   |   25 |
+--------+------+
```

is permitted, but

```
SELECT * FROM staff WHERE (name,age) = ALL (
  SELECT name,age FROM customer WHERE age >=51
);
ERROR 1241 (21000): Operand should contain 1 column(s)
```

is not.

Correlated Subqueries

Subqueries in the FROM clause cannot be correlated subqueries. They cannot be evaluated for each row of the outer query

since they are evaluated to produce a result set during when the query is executed.

Stored Functions

A subquery can refer to a stored function which modifies data. This is an extension to the SQL standard, but can result in indeterminate outcomes. For example, take:

```
SELECT ... WHERE x IN (SELECT f() ...);
```

where *f()* inserts rows. The function *f()* could be executed a different number of times depending on how the optimizer chooses to handle the query.

This sort of construct is therefore not safe to use in replication that is not row-based, as there could be different results on the master and the slave.

1.1.1.4.1.2.3 UNION

`UNION` is used to combine the results from multiple SELECT statements into a single result set.

Syntax

```
SELECT ...
UNION [ALL | DISTINCT] SELECT ...
[UNION [ALL | DISTINCT] SELECT ...]
[ORDER BY [column [, column ...]]]
[LIMIT {[offset,] row_count | row_count OFFSET offset}]
```

Contents

Description

`UNION` is used to combine the results from multiple SELECT statements into a single result set.

The column names from the first `SELECT` statement are used as the column names for the results returned. Selected columns listed in corresponding positions of each SELECT statement should have the same data type. (For example, the first column selected by the first statement should have the same type as the first column selected by the other statements.)

If they don't, the type and length of the columns in the result take into account the values returned by all of the SELECTs, so there is no need for explicit casting. Note that currently this is not the case for recursive CTEs - see MDEV-12325.

Table names can be specified as `db_name.tbl_name`. This permits writing `UNION`s which involve multiple databases. See Identifier Qualifiers for syntax details.

UNION queries cannot be used with aggregate functions.

`EXCEPT` and `UNION` have the same operation precedence and `INTERSECT` has a higher precedence, unless running in Oracle mode, in which case all three have the same precedence.

ALL/DISTINCT

The `ALL` keyword causes duplicate rows to be preserved. The `DISTINCT` keyword (the default if the keyword is omitted) causes duplicate rows to be removed by the results.

UNION ALL and UNION DISTINCT can both be present in a query. In this case, UNION DISTINCT will override any UNION ALLs to its left.

MariaDB starting with 10.1.1

Until MariaDB 10.1.1, all `UNION ALL` statements required the server to create a temporary table. Since MariaDB

10.1.1, the server can in most cases execute `UNION ALL` without creating a temporary table, improving performance (see MDEV-334).

ORDER BY and LIMIT

Individual SELECTs can contain their own ORDER BY and LIMIT clauses. In this case, the individual queries need to be wrapped between parentheses. However, this does not affect the order of the UNION, so they only are useful to limit the record read by one SELECT.

The UNION can have global ORDER BY and LIMIT clauses, which affect the whole resultset. If the columns retrieved by individual SELECT statements have an alias (AS), the ORDER BY must use that alias, not the real column names.

HIGH_PRIORITY

Specifying a query as HIGH_PRIORITY will not work inside a UNION. If applied to the first SELECT, it will be ignored. Applying to a later SELECT results in a syntax error:

```
ERROR 1234 (42000): Incorrect usage/placement of 'HIGH_PRIORITY'
```

SELECT ... INTO ...

Individual SELECTs cannot be written INTO DUMPFILE or INTO OUTFILE. If the last SELECT statement specifies INTO DUMPFILE or INTO OUTFILE, the entire result of the UNION will be written. Placing the clause after any other SELECT will result in a syntax error.

If the result is a single row, SELECT ... INTO @var_name can also be used.

MariaDB starting with 10.4.0

Parentheses

From MariaDB 10.4.0, parentheses can be used to specify precedence. Before this, a syntax error would be returned.

Examples

`UNION` between tables having different column names:

```
(SELECT e_name AS name, email FROM employees)
UNION
(SELECT c_name AS name, email FROM customers);
```

Specifying the `UNION` 's global order and limiting total rows:

```
(SELECT name, email FROM employees)
UNION
(SELECT name, email FROM customers)
ORDER BY name LIMIT 10;
```

Adding a constant row:

```
(SELECT 'John Doe' AS name, 'john.doe@example.net' AS email)
UNION
(SELECT name, email FROM customers);
```

Differing types:

```
SELECT CAST('x' AS CHAR(1)) UNION SELECT REPEAT('y',4);
+----------------------+
| CAST('x' AS CHAR(1)) |
+----------------------+
| x                    |
| yyyy                 |
+----------------------+
```

Returning the results in order of each individual SELECT by use of a sort column:

```
(SELECT 1 AS sort_column, e_name AS name, email FROM employees)
UNION
(SELECT 2, c_name AS name, email FROM customers) ORDER BY sort_column;
```

Difference between UNION, EXCEPT and INTERSECT. `INTERSECT ALL` and `EXCEPT ALL` are available from MariaDB 10.5.0.

```
CREATE TABLE seqs (i INT);
INSERT INTO seqs VALUES (1),(2),(2),(3),(3),(4),(5),(6);

SELECT i FROM seqs WHERE i <= 3 UNION SELECT i FROM seqs WHERE i>=3;
+------+
| i    |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|    5 |
|    6 |
+------+

SELECT i FROM seqs WHERE i <= 3 UNION ALL SELECT i FROM seqs WHERE i>=3;
+------+
| i    |
+------+
|    1 |
|    2 |
|    2 |
|    3 |
|    3 |
|    3 |
|    3 |
|    4 |
|    5 |
|    6 |
+------+

SELECT i FROM seqs WHERE i <= 3 EXCEPT SELECT i FROM seqs WHERE i>=3;
+------+
| i    |
+------+
|    1 |
|    2 |
+------+

SELECT i FROM seqs WHERE i <= 3 EXCEPT ALL SELECT i FROM seqs WHERE i>=3;
+------+
| i    |
+------+
|    1 |
|    2 |
|    2 |
+------+

SELECT i FROM seqs WHERE i <= 3 INTERSECT SELECT i FROM seqs WHERE i>=3;
+------+
| i    |
+------+
|    3 |
+------+

SELECT i FROM seqs WHERE i <= 3 INTERSECT ALL SELECT i FROM seqs WHERE i>=3;
+------+
| i    |
+------+
|    3 |
|    3 |
+------+
```

Parentheses for specifying precedence, from MariaDB 10.4.0

```
CREATE OR REPLACE TABLE t1 (a INT);
CREATE OR REPLACE TABLE t2 (b INT);
CREATE OR REPLACE TABLE t3 (c INT);

INSERT INTO t1 VALUES (1),(2),(3),(4);
INSERT INTO t2 VALUES (5),(6);
INSERT INTO t3 VALUES (1),(6);

((SELECT a FROM t1) UNION (SELECT b FROM t2)) INTERSECT (SELECT c FROM t3);
+------+
| a    |
+------+
|    1 |
|    6 |
+------+

(SELECT a FROM t1) UNION ((SELECT b FROM t2) INTERSECT (SELECT c FROM t3));
+------+
| a    |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|    6 |
+------+
```

1.1.1.4.1.2.4 EXCEPT

The result of `EXCEPT` is all records of the left `SELECT` result set except records which are in right `SELECT` result set, i.e. it is subtraction of two result sets. From MariaDB 10.6.1, `MINUS` is a synonym when SQL_MODE=ORACLE is set.

Syntax

```
SELECT ...
(INTERSECT [ALL | DISTINCT] | EXCEPT [ALL | DISTINCT] | UNION [ALL | DISTINCT])
  SELECT ...
[(INTERSECT [ALL | DISTINCT] | EXCEPT [ALL | DISTINCT] | UNION [ALL | DISTINCT])
  SELECT ...]
[ORDER BY [{col_name | expr | position} [ASC | DESC]
  [, {col_name | expr | position} [ASC | DESC] ...]]]
[LIMIT {[offset,] row_count | row_count OFFSET offset}
| OFFSET start { ROW | ROWS }
| FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } { ONLY | WITH TIES } ]
```

Contents

Please note:

- Brackets for explicit operation precedence are not supported; use a subquery in the `FROM` clause as a workaround).

Description

MariaDB has supported `EXCEPT` and INTERSECT in addition to UNION since MariaDB 10.3.

The queries before and after `EXCEPT` must be SELECT or VALUES statements.

All behavior for naming columns, `ORDER BY` and `LIMIT` is the same as for `UNION`. Note that the alternative SELECT ... OFFSET ... FETCH syntax is only supported. This allows us to use the `WITH TIES` clause.

`EXCEPT` implicitly supposes a `DISTINCT` operation.

The result of `EXCEPT` is all records of the left `SELECT` result except records which are in right `SELECT` result set, i.e. it is subtraction of two result sets.

`EXCEPT` and `UNION` have the same operation precedence and `INTERSECT` has a higher precedence, unless running in Oracle mode, in which case all three have the same precedence.

MariaDB starting with 10.4.0

Parentheses

From MariaDB 10.4.0, parentheses can be used to specify precedence. Before this, a syntax error would be returned.

MariaDB starting with 10.5.0

ALL/DISTINCT

`EXCEPT ALL` and `EXCEPT DISTINCT` were introduced in MariaDB 10.5.0. The `ALL` operator leaves duplicates intact, while the `DISTINCT` operator removes duplicates. `DISTINCT` is the default behavior if neither operator is supplied, and the only behavior prior to MariaDB 10.5.

Examples

Show customers which are not employees:

```
(SELECT e_name AS name, email FROM customers)
EXCEPT
(SELECT c_name AS name, email FROM employees);
```

Difference between UNION, EXCEPT and INTERSECT. `INTERSECT ALL` and `EXCEPT ALL` are available from MariaDB 10.5.0.

```
CREATE TABLE seqs (i INT);
INSERT INTO seqs VALUES (1),(2),(2),(3),(3),(4),(5),(6);

SELECT i FROM seqs WHERE i <= 3 UNION SELECT i FROM seqs WHERE i>=3;
+------+
| i    |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|    5 |
|    6 |
+------+

SELECT i FROM seqs WHERE i <= 3 UNION ALL SELECT i FROM seqs WHERE i>=3;
+------+
| i    |
+------+
|    1 |
|    2 |
|    2 |
|    3 |
|    3 |
|    3 |
|    3 |
|    4 |
|    5 |
|    6 |
+------+

SELECT i FROM seqs WHERE i <= 3 EXCEPT SELECT i FROM seqs WHERE i>=3;
+------+
| i    |
+------+
|    1 |
|    2 |
+------+

SELECT i FROM seqs WHERE i <= 3 EXCEPT ALL SELECT i FROM seqs WHERE i>=3;
+------+
| i    |
+------+
|    1 |
|    2 |
|    2 |
+------+

SELECT i FROM seqs WHERE i <= 3 INTERSECT SELECT i FROM seqs WHERE i>=3;
+------+
| i    |
+------+
|    3 |
+------+

SELECT i FROM seqs WHERE i <= 3 INTERSECT ALL SELECT i FROM seqs WHERE i>=3;
+------+
| i    |
+------+
|    3 |
|    3 |
+------+
```

Parentheses for specifying precedence, from MariaDB 10.4.0

```
CREATE OR REPLACE TABLE t1 (a INT);
CREATE OR REPLACE TABLE t2 (b INT);
CREATE OR REPLACE TABLE t3 (c INT);

INSERT INTO t1 VALUES (1),(2),(3),(4);
INSERT INTO t2 VALUES (5),(6);
INSERT INTO t3 VALUES (1),(6);

((SELECT a FROM t1) UNION (SELECT b FROM t2)) EXCEPT (SELECT c FROM t3);
+------+
| a    |
+------+
|    2 |
|    3 |
|    4 |
|    5 |
+------+

(SELECT a FROM t1) UNION ((SELECT b FROM t2) EXCEPT (SELECT c FROM t3));
+------+
| a    |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|    5 |
+------+
```

Here is an example that makes use of the SEQUENCE storage engine and the VALUES statement, to generate a numeric sequence and remove some arbitrary numbers from it:

```
(SELECT seq FROM seq_1_to_10) EXCEPT VALUES (2), (3), (4);
+-----+
| seq |
+-----+
|   1 |
|   5 |
|   6 |
|   7 |
|   8 |
|   9 |
|  10 |
+-----+
```

1.1.1.4.1.2.5 INTERSECT

MariaDB starting with 10.3.0

INTERSECT was introduced in MariaDB 10.3.0.

The result of an intersect is the intersection of right and left `SELECT` results, i.e. only records that are present in both result sets will be included in the result of the operation.

Syntax

```
SELECT ...
(INTERSECT [ALL | DISTINCT] | EXCEPT [ALL | DISTINCT] | UNION [ALL | DISTINCT]) SELECT ...
[(INTERSECT [ALL | DISTINCT] | EXCEPT [ALL | DISTINCT] | UNION [ALL | DISTINCT]) SELECT ...]
[ORDER BY [column [, column ...]]]
[LIMIT {[offset,] row_count | row_count OFFSET offset}]
```

Contents

Description

MariaDB has supported `INTERSECT` (as well as EXCEPT) in addition to UNION since MariaDB 10.3.

All behavior for naming columns, `ORDER BY` and `LIMIT` is the same as for UNION.

`INTERSECT` implicitly supposes a `DISTINCT` operation.

The result of an intersect is the intersection of right and left `SELECT` results, i.e. only records that are present in both result sets will be included in the result of the operation.

`INTERSECT` has higher precedence than `UNION` and `EXCEPT` (unless running running in Oracle mode, in which case all three have the same precedence). If possible it will be executed linearly but if not it will be translated to a subquery in the `FROM` clause:

```
(select a,b from t1)
union
(select c,d from t2)
intersect
(select e,f from t3)
union
(select 4,4);
```

will be translated to:

```
(select a,b from t1)
union
select c,d from
  ((select c,d from t2)
   intersect
   (select e,f from t3)) dummy_subselect
union
(select 4,4)
```

MariaDB starting with 10.4.0

Parentheses

From MariaDB 10.4.0, parentheses can be used to specify precedence. Before this, a syntax error would be returned.

MariaDB starting with 10.5.0

ALL/DISTINCT

`INTERSECT ALL` and `INTERSECT DISTINCT` were introduced in MariaDB 10.5.0. The `ALL` operator leaves duplicates intact, while the `DISTINCT` operator removes duplicates. `DISTINCT` is the default behavior if neither operator is supplied, and the only behavior prior to MariaDB 10.5.

Examples

Show customers which are employees:

```
(SELECT e_name AS name, email FROM employees)
INTERSECT
(SELECT c_name AS name, email FROM customers);
```

Difference between UNION, EXCEPT and INTERSECT. `INTERSECT ALL` and `EXCEPT ALL` are available from MariaDB

10.5.0.

```
CREATE TABLE seqs (i INT);
INSERT INTO seqs VALUES (1),(2),(2),(3),(3),(4),(5),(6);

SELECT i FROM seqs WHERE i <= 3 UNION SELECT i FROM seqs WHERE i>=3;
+------+
| i    |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|    5 |
|    6 |
+------+

SELECT i FROM seqs WHERE i <= 3 UNION ALL SELECT i FROM seqs WHERE i>=3;
+------+
| i    |
+------+
|    1 |
|    2 |
|    2 |
|    3 |
|    3 |
|    3 |
|    3 |
|    4 |
|    5 |
|    6 |
+------+

SELECT i FROM seqs WHERE i <= 3 EXCEPT SELECT i FROM seqs WHERE i>=3;
+------+
| i    |
+------+
|    1 |
|    2 |
+------+

SELECT i FROM seqs WHERE i <= 3 EXCEPT ALL SELECT i FROM seqs WHERE i>=3;
+------+
| i    |
+------+
|    1 |
|    2 |
|    2 |
+------+

SELECT i FROM seqs WHERE i <= 3 INTERSECT SELECT i FROM seqs WHERE i>=3;
+------+
| i    |
+------+
|    3 |
+------+

SELECT i FROM seqs WHERE i <= 3 INTERSECT ALL SELECT i FROM seqs WHERE i>=3;
+------+
| i    |
+------+
|    3 |
|    3 |
+------+
```

Parentheses for specifying precedence, from MariaDB 10.4.0

```
CREATE OR REPLACE TABLE t1 (a INT);
CREATE OR REPLACE TABLE t2 (b INT);
CREATE OR REPLACE TABLE t3 (c INT);

INSERT INTO t1 VALUES (1),(2),(3),(4);
INSERT INTO t2 VALUES (5),(6);
INSERT INTO t3 VALUES (1),(6);

((SELECT a FROM t1) UNION (SELECT b FROM t2)) INTERSECT (SELECT c FROM t3);
+------+
| a    |
+------+
|    1 |
|    6 |
+------+

(SELECT a FROM t1) UNION ((SELECT b FROM t2) INTERSECT (SELECT c FROM t3));
+------+
| a    |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|    6 |
+------+
```

1.1.1.4.1.2.6 Precedence Control in Table Operations

MariaDB starting with 10.4.0

Beginning in MariaDB 10.4, you can control the ordering of execution on table operations using parentheses.

Syntax

```
(  expression )
[ORDER BY [column[, column...]]]
[LIMIT {[offset,] row_count | row_count OFFSET offset}]
```

Contents

Description

Using parentheses in your SQL allows you to control the order of execution for `SELECT` statements and Table Value Constructor, including `UNION`, `EXCEPT`, and `INTERSECT` operations. MariaDB executes the parenthetical expression before the rest of the statement. You can then use `ORDER BY` and `LIMIT` clauses the further organize the result-set.

Note: In practice, the Optimizer may rearrange the exact order in which MariaDB executes different parts of the statement. When it calculates the result-set, however, it returns values as though the parenthetical expression were executed first.

Example

```
CREATE TABLE test.t1 (num INT);

INSERT INTO test.t1 VALUES (1),(2),(3);

(SELECT * FROM test.t1
 UNION
 VALUES (10))
INTERSECT
VALUES (1),(3),(10),(11);
+------+
| num  |
+------+
|    1 |
|    3 |
|   10 |
+------+

((SELECT * FROM test.t1
  UNION
  VALUES (10))
 INTERSECT
 VALUES (1),(3),(10),(11))
ORDER BY 1 DESC;
+------+
| num  |
+------+
|   10 |
|    3 |
|    1 |
+------+
```

1.1.1.4.1.2.7 MINUS

MariaDB starting with 10.6.1

`MINUS` was introduced as a synonym for EXCEPT from MariaDB 10.6.1 when SQL_MODE=ORACLE is set.

```
CREATE TABLE seqs (i INT);
INSERT INTO seqs VALUES (1),(2),(2),(3),(3),(4),(5),(6);

SET SQL_MODE='ORACLE';

SELECT i FROM seqs WHERE i <= 3 MINUS SELECT i FROM seqs WHERE i>=3;
+------+
| i    |
+------+
|    1 |
|    2 |
+------+
```

1.1.1.4.1.3 LIMIT

Contents

Description

Use the `LIMIT` clause to restrict the number of returned rows. When you use a single integer *n* with `LIMIT`, the first *n* rows will be returned. Use the ORDER BY clause to control which rows come first. You can also select a number of rows after an offset using either of the following:

```
LIMIT offset, row_count
LIMIT row_count OFFSET offset
```

When you provide an offset *m* with a limit *n*, the first *m* rows will be ignored, and the following *n* rows will be returned.

Executing an UPDATE with the `LIMIT` clause is not safe for replication. `LIMIT 0` is an exception to this rule (see MDEV-6170).

There is a LIMIT ROWS EXAMINED optimization which provides the means to terminate the execution of SELECT statements which examine too many rows, and thus use too many resources. See LIMIT ROWS EXAMINED.

Multi-Table Updates

Until MariaDB 10.3.1, it was not possible to use `LIMIT` (or ORDER BY) in a multi-table UPDATE statement. This restriction was lifted in MariaDB 10.3.2.

GROUP_CONCAT

Starting from MariaDB 10.3.3, it is possible to use `LIMIT` with GROUP_CONCAT().

Examples

```
CREATE TABLE members (name VARCHAR(20));
INSERT INTO members VALUES('Jagdish'),('Kenny'),('Rokurou'),('Immaculada');

SELECT * FROM members;
+------------+
| name       |
+------------+
| Jagdish    |
| Kenny      |
| Rokurou    |
| Immaculada |
+------------+
```

Select the first two names (no ordering specified):

```
SELECT * FROM members LIMIT 2;
+---------+
| name    |
+---------+
| Jagdish |
| Kenny   |
+---------+
```

All the names in alphabetical order:

```
SELECT * FROM members ORDER BY name;
+------------+
| name       |
+------------+
| Immaculada |
| Jagdish    |
| Kenny      |
| Rokurou    |
+------------+
```

The first two names, ordered alphabetically:

```
SELECT * FROM members ORDER BY name LIMIT 2;
+------------+
| name       |
+------------+
| Immaculada |
| Jagdish    |
+------------+
```

The third name, ordered alphabetically (the first name would be offset zero, so the third is offset two):

```
SELECT * FROM members ORDER BY name LIMIT 2,1;
+-------+
| name  |
+-------+
| Kenny |
+-------+
```

From MariaDB 10.3.2, `LIMIT` can be used in a multi-table update:

```
CREATE TABLE warehouse (product_id INT, qty INT);
INSERT INTO warehouse VALUES (1,100),(2,100),(3,100),(4,100);

CREATE TABLE store (product_id INT, qty INT);
INSERT INTO store VALUES (1,5),(2,5),(3,5),(4,5);

UPDATE warehouse,store SET warehouse.qty = warehouse.qty-2, store.qty = store.qty+2
  WHERE (warehouse.product_id = store.product_id AND store.product_id  >= 1)
    ORDER BY store.product_id DESC LIMIT 2;

SELECT * FROM warehouse;
+------------+------+
| product_id | qty  |
+------------+------+
|          1 |  100 |
|          2 |  100 |
|          3 |   98 |
|          4 |   98 |
+------------+------+

SELECT * FROM store;
+------------+------+
| product_id | qty  |
+------------+------+
|          1 |    5 |
|          2 |    5 |
|          3 |    7 |
|          4 |    7 |
+------------+------+
```

From MariaDB 10.3.3, `LIMIT` can be used with GROUP_CONCAT, so, for example, given the following table:

```
CREATE TABLE d (dd DATE, cc INT);

INSERT INTO d VALUES ('2017-01-01',1);
INSERT INTO d VALUES ('2017-01-02',2);
INSERT INTO d VALUES ('2017-01-04',3);
```

the following query:

```
SELECT SUBSTRING_INDEX(GROUP_CONCAT(CONCAT_WS(":",dd,cc) ORDER BY cc DESC),",",1) FROM d;
+-----------------------------------------------------------------------------+
| SUBSTRING_INDEX(GROUP_CONCAT(CONCAT_WS(":",dd,cc) ORDER BY cc DESC),",",1) |
+-----------------------------------------------------------------------------+
| 2017-01-04:3                                                                |
+-----------------------------------------------------------------------------+
```

can be more simply rewritten as:

```
SELECT GROUP_CONCAT(CONCAT_WS(":",dd,cc) ORDER BY cc DESC LIMIT 1) FROM d;
+-----------------------------------------------------------+
| GROUP_CONCAT(CONCAT_WS(":",dd,cc) ORDER BY cc DESC LIMIT 1) |
+-----------------------------------------------------------+
| 2017-01-04:3                                              |
+-----------------------------------------------------------+
```

1.1.1.4.1.4 ORDER BY

Contents

Description

Use the `ORDER BY` clause to order a resultset, such as that are returned from a SELECT statement. You can specify just a column or use any expression with functions. If you are using the `GROUP BY` clause, you can use grouping functions in `ORDER BY`. Ordering is done after grouping.

You can use multiple ordering expressions, separated by commas. Rows will be sorted by the first expression, then by the second expression if they have the same value for the first, and so on.

You can use the keywords `ASC` and `DESC` after each ordering expression to force that ordering to be ascending or descending, respectively. Ordering is ascending by default.

You can also use a single integer as the ordering expression. If you use an integer *n*, the results will be ordered by the *n*th column in the select expression.

When string values are compared, they are compared as if by the STRCMP function. `STRCMP` ignores trailing whitespace and may normalize characters and ignore case, depending on the collation in use.

Duplicated entries in the `ORDER BY` clause are removed.

`ORDER BY` can also be used to order the activities of a DELETE or UPDATE statement (usually with the LIMIT clause).

MariaDB starting with 10.3.2

Until MariaDB 10.3.1, it was not possible to use `ORDER BY` (or LIMIT) in a multi-table UPDATE statement. This restriction was lifted in MariaDB 10.3.2.

MariaDB starting with 10.5

From MariaDB 10.5, MariaDB allows packed sort keys and values of non-sorted fields in the sort buffer. This can make filesort temporary files much smaller when VARCHAR, CHAR or BLOBs are used, notably speeding up some ORDER BY sorts.

Examples

```
CREATE TABLE seq (i INT, x VARCHAR(1));
INSERT INTO seq VALUES (1,'a'), (2,'b'), (3,'b'), (4,'f'), (5,'e');

SELECT * FROM seq ORDER BY i;
+------+------+
| i    | x    |
+------+------+
|    1 | a    |
|    2 | b    |
|    3 | b    |
|    4 | f    |
|    5 | e    |
+------+------+

SELECT * FROM seq ORDER BY i DESC;
+------+------+
| i    | x    |
+------+------+
|    5 | e    |
|    4 | f    |
|    3 | b    |
|    2 | b    |
|    1 | a    |
+------+------+

SELECT * FROM seq ORDER BY x,i;
+------+------+
| i    | x    |
+------+------+
|    1 | a    |
|    2 | b    |
|    3 | b    |
|    5 | e    |
|    4 | f    |
+------+------+
```

ORDER BY in an UPDATE statement, in conjunction with LIMIT:

```
UPDATE seq SET x='z' WHERE x='b' ORDER BY i DESC LIMIT 1;

SELECT * FROM seq;
+------+------+
| i    | x    |
+------+------+
|    1 | a    |
|    2 | b    |
|    3 | z    |
|    4 | f    |
|    5 | e    |
+------+------+
```

From MariaDB 10.3.2, `ORDER BY` can be used in a multi-table update:

```
CREATE TABLE warehouse (product_id INT, qty INT);
INSERT INTO warehouse VALUES (1,100),(2,100),(3,100),(4,100);

CREATE TABLE store (product_id INT, qty INT);
INSERT INTO store VALUES (1,5),(2,5),(3,5),(4,5);

UPDATE warehouse,store SET warehouse.qty = warehouse.qty-2, store.qty = store.qty+2
  WHERE (warehouse.product_id = store.product_id AND store.product_id  >= 1)
    ORDER BY store.product_id DESC LIMIT 2;

SELECT * FROM warehouse;
+------------+------+
| product_id | qty  |
+------------+------+
|          1 |  100 |
|          2 |  100 |
|          3 |   98 |
|          4 |   98 |
+------------+------+

SELECT * FROM store;
+------------+------+
| product_id | qty  |
+------------+------+
|          1 |    5 |
|          2 |    5 |
|          3 |    7 |
|          4 |    7 |
+------------+------+
```

1.1.1.4.1.5 GROUP BY

Contents

Use the `GROUP BY` clause in a SELECT statement to group rows together that have the same value in one or more column, or the same computed value using expressions with any functions and operators except grouping functions. When you use a `GROUP BY` clause, you will get a single result row for each group of rows that have the same value for the expression given in `GROUP BY`.

When grouping rows, grouping values are compared as if by the `=` operator. For string values, the `=` operator ignores trailing whitespace and may normalize characters and ignore case, depending on the collation in use.

You can use any of the grouping functions in your select expression. Their values will be calculated based on all the rows that have been grouped together for each result row. If you select a non-grouped column or a value computed from a non-grouped column, it is undefined which row the returned value is taken from. This is not permitted if the `ONLY_FULL_GROUP_BY` SQL_MODE is used.

You can use multiple expressions in the `GROUP BY` clause, separated by commas. Rows are grouped together if they match on each of the expressions.

You can also use a single integer as the grouping expression. If you use an integer *n*, the results will be grouped by the *n*th column in the select expression.

The `WHERE` clause is applied before the `GROUP BY` clause. It filters non-aggregated rows before the rows are grouped together. To filter grouped rows based on aggregate values, use the `HAVING` clause. The `HAVING` clause takes any expression and evaluates it as a boolean, just like the `WHERE` clause. You can use grouping functions in the `HAVING` clause. As with the select expression, if you reference non-grouped columns in the `HAVING` clause, the behavior is undefined.

By default, if a `GROUP BY` clause is present, the rows in the output will be sorted by the expressions used in the `GROUP BY`. You can also specify `ASC` or `DESC` (ascending, descending) after those expressions, like in ORDER BY. The default is `ASC`.

If you want the rows to be sorted by another field, you can add an explicit ORDER BY. If you don't want the result to be ordered, you can add ORDER BY NULL.

WITH ROLLUP

The `WITH ROLLUP` modifer adds extra rows to the resultset that represent super-aggregate summaries. For a full description with examples, see SELECT WITH ROLLUP.

GROUP BY Examples

Consider the following table that records how many times each user has played and won a game:

```
CREATE TABLE plays (name VARCHAR(16), plays INT, wins INT);
INSERT INTO plays VALUES
  ("John", 20, 5),
  ("Robert", 22, 8),
  ("Wanda", 32, 8),
  ("Susan", 17, 3);
```

Get a list of win counts along with a count:

```
SELECT wins, COUNT(*) FROM plays GROUP BY wins;
+------+----------+
| wins | COUNT(*) |
+------+----------+
|    3 |        1 |
|    5 |        1 |
|    8 |        2 |
+------+----------+
3 rows in set (0.00 sec)
```

The `GROUP BY` expression can be a computed value, and can refer back to an identifer specified with `AS`. Get a list of win averages along with a count:

```
SELECT (wins / plays) AS winavg, COUNT(*) FROM plays GROUP BY winavg;
+--------+----------+
| winavg | COUNT(*) |
+--------+----------+
| 0.1765 |        1 |
| 0.2500 |        2 |
| 0.3636 |        1 |
+--------+----------+
3 rows in set (0.00 sec)
```

You can use any grouping function in the select expression. For each win average as above, get a list of the average play count taken to get that average:

```
SELECT (wins / plays) AS winavg, AVG(plays) FROM plays
  GROUP BY winavg;
+--------+------------+
| winavg | AVG(plays) |
+--------+------------+
| 0.1765 |    17.0000 |
| 0.2500 |    26.0000 |
| 0.3636 |    22.0000 |
+--------+------------+
3 rows in set (0.00 sec)
```

You can filter on aggregate information using the `HAVING` clause. The `HAVING` clause is applied after `GROUP BY` and allows you to filter on aggregate data that is not available to the `WHERE` clause. Restrict the above example to results that involve an average number of plays over 20:

```
SELECT (wins / plays) AS winavg, AVG(plays) FROM plays
  GROUP BY winavg HAVING AVG(plays) > 20;
+--------+------------+
| winavg | AVG(plays) |
+--------+------------+
| 0.2500 |    26.0000 |
| 0.3636 |    22.0000 |
+--------+------------+
2 rows in set (0.00 sec)
```

1.1.1.4.1.6 Common Table Expressions

WITH

Allows reference to subqueries as temporary tables within queries.

Non-Recursive Common Table Expressions Overview

Common Table Expressions (CTEs) are essentially Temporary Named Result Sets.

Recursive Common Table Expressions Overview

A recursive CTE will repeatedly execute subsets of the data until obtaining the complete results.

There are 1 related questions.

1.1.1.4.1.6.1 WITH

MariaDB starting with 10.2.1

Common Table Expressions were introduced in MariaDB 10.2.1.

Syntax

```
WITH [RECURSIVE] table_reference [(columns_list)] AS  (
  SELECT ...
)
[CYCLE cycle_column_list RESTRICT]
SELECT ...
```

Contents

Description

The `WITH` keyword signifies a Common Table Expression (CTE). It allows you to refer to a subquery expression many times in a query, as if having a temporary table that only exists for the duration of a query.

There are two kinds of CTEs:

- Non-Recursive
- Recursive (signified by the `RECURSIVE` keyword, supported since MariaDB 10.2.2)

You can use `table_reference` as any normal table in the external `SELECT` part. You can also use `WITH` in subqueries, as well as with EXPLAIN and SELECT.

Poorly-formed recursive CTEs can in theory cause infinite loops. The max_recursive_iterations system variable limits the number of recursions.

CYCLE ... RESTRICT

MariaDB starting with 10.5.2

The CYCLE clause enables CTE cycle detection, avoiding excessive or infinite loops, MariaDB supports a relaxed, non-standard grammar.

The SQL Standard permits a CYCLE clause, as follows:

```
WITH RECURSIVE ... (
   ...
)
CYCLE <cycle column list>
SET <cycle mark column> TO <cycle mark value> DEFAULT <non-cycle mark value>
USING <path column>
```

where all clauses are mandatory.

MariaDB does not support this, but from 10.5.2 permits a non-standard relaxed grammar, as follows:

```
WITH RECURSIVE ... (
  ...
)
CYCLE <cycle column list> RESTRICT
```

With the use of `CYCLE ... RESTRICT` it makes no difference whether the CTE uses `UNION ALL` or `UNION DISTINCT` anymore. `UNION ALL` means "all rows, but without cycles", which is exactly what the `CYCLE` clause enables. And `UNION DISTINCT` means all rows should be different, which, again, is what will happen — as uniqueness is enforced over a subset of columns, complete rows will automatically all be different.

Examples

Below is an example with the `WITH` at the top level:

```
WITH t AS (SELECT a FROM t1 WHERE b >= 'c')
  SELECT * FROM t2, t WHERE t2.c = t.a;
```

The example below uses `WITH` in a subquery:

```
SELECT t1.a, t1.b FROM t1, t2
  WHERE t1.a > t2.c
     AND t2.c IN(WITH t AS (SELECT * FROM t1 WHERE t1.a < 5)
                 SELECT t2.c FROM t2, t WHERE t2.c = t.a);
```

Below is an example of a Recursive CTE:

```
WITH RECURSIVE ancestors AS
 ( SELECT * FROM folks
   WHERE name="Alex"
   UNION
   SELECT f.*
   FROM folks AS f, ancestors AS a
   WHERE f.id = a.father OR f.id = a.mother )
SELECT * FROM ancestors;
```

Take the following structure, and data,

```
CREATE TABLE t1 (from_ int, to_ int);
INSERT INTO t1 VALUES (1,2), (1,100), (2,3), (3,4), (4,1);
SELECT * FROM t1;
+-------+------+
| from_ | to_  |
+-------+------+
|     1 |    2 |
|     1 |  100 |
|     2 |    3 |
|     3 |    4 |
|     4 |    1 |
+-------+------+
```

Given the above, the following query would theoretically result in an infinite loop due to the last record in t1 (note that max_recursive_iterations is set to 10 for the purposes of this example, to avoid the excessive number of cycles):

```
SET max_recursive_iterations=10;

WITH RECURSIVE cte (depth, from_, to_) AS (
  SELECT 0,1,1 UNION DISTINCT SELECT depth+1, t1.from_, t1.to_
    FROM t1, cte  WHERE t1.from_ = cte.to_
)
SELECT * FROM cte;
+-------+-------+------+
| depth | from_ | to_  |
+-------+-------+------+
|     0 |     1 |    1 |
|     1 |     1 |    2 |
|     1 |     1 |  100 |
|     2 |     2 |    3 |
|     3 |     3 |    4 |
|     4 |     4 |    1 |
|     5 |     1 |    2 |
|     5 |     1 |  100 |
|     6 |     2 |    3 |
|     7 |     3 |    4 |
|     8 |     4 |    1 |
|     9 |     1 |    2 |
|     9 |     1 |  100 |
|    10 |     2 |    3 |
+-------+-------+------+
```

However, the CYCLE ... RESTRICT clause (from MariaDB 10.5.2) can overcome this:

```
WITH RECURSIVE cte (depth, from_, to_) AS (
  SELECT 0,1,1 UNION SELECT depth+1, t1.from_, t1.to_
    FROM t1, cte WHERE t1.from_ = cte.to_
)
CYCLE from_, to_ RESTRICT
SELECT * FROM cte;
+-------+-------+------+
| depth | from_ | to_  |
+-------+-------+------+
|     0 |     1 |    1 |
|     1 |     1 |    2 |
|     1 |     1 |  100 |
|     2 |     2 |    3 |
|     3 |     3 |    4 |
|     4 |     4 |    1 |
+-------+-------+------+
```

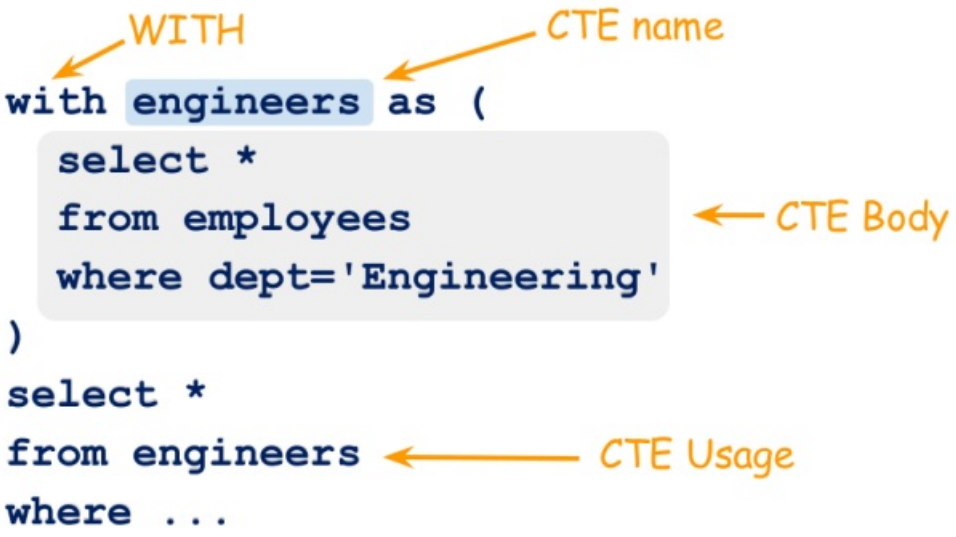
1.1.1.4.1.6.2 Non-Recursive Common Table Expressions Overview

Contents

Common Table Expressions (CTEs) are a standard SQL feature, and are essentially temporary named result sets. There are two kinds of CTEs: Non-Recursive, which this article covers; and Recursive.

Non-Recursive CTEs

The WITH keyword signifies a CTE. It is given a name, followed by a body (the main query) as follows:



CTEs are similar to derived tables. For example

```
WITH engineers AS
   ( SELECT * FROM employees
     WHERE dept = 'Engineering' )

SELECT * FROM engineers
WHERE ...
```

```
SELECT * FROM
   ( SELECT * FROM employees
     WHERE dept = 'Engineering' ) AS engineers
WHERE
...
```

A non-recursive CTE is basically a query-local VIEW. There are several advantages and caveats to them. The syntax is more readable than nested `FROM (SELECT ...)` . A CTE can refer to another and it can be referenced from multiple places.

A CTE referencing Another CTE

Using this format makes for a more readable SQL than a nested `FROM(SELECT ...)` clause. Below is an example of this:

```
WITH engineers AS (
SELECT * FROM employees
WHERE dept IN('Development','Support') ),
eu_engineers AS ( SELECT * FROM engineers WHERE country IN('NL',...) )
SELECT
...
FROM eu_engineers;
```

Multiple Uses of a CTE

This can be an 'anti-self join', for example:

```
WITH engineers AS (
SELECT * FROM employees
WHERE dept IN('Development','Support') )

SELECT * FROM engineers E1
WHERE NOT EXISTS
   (SELECT 1 FROM engineers E2
    WHERE E2.country=E1.country
    AND E2.name <> E1.name );
```

Or, for year-over-year comparisons, for example:

```
WITH sales_product_year AS (
SELECT product, YEAR(ship_date) AS year,
SUM(price) AS total_amt
FROM item_sales
GROUP BY product, year )

SELECT *
FROM sales_product_year CUR,
sales_product_year PREV,
WHERE CUR.product=PREV.product
AND  CUR.year=PREV.year + 1
AND CUR.total_amt > PREV.total_amt
```

Another use is to compare individuals against their group. Below is an example of how this might be executed:

```
WITH sales_product_year AS (
SELECT product,
YEAR(ship_date) AS year,
SUM(price) AS total_amt
FROM item_sales
GROUP BY product, year
)

SELECT *
FROM sales_product_year S1
WHERE
total_amt >
    (SELECT 0.1 * SUM(total_amt)
     FROM sales_product_year S2
     WHERE S2.year = S1.year)
```

1.1.1.4.1.6.3 Recursive Common Table Expressions Overview

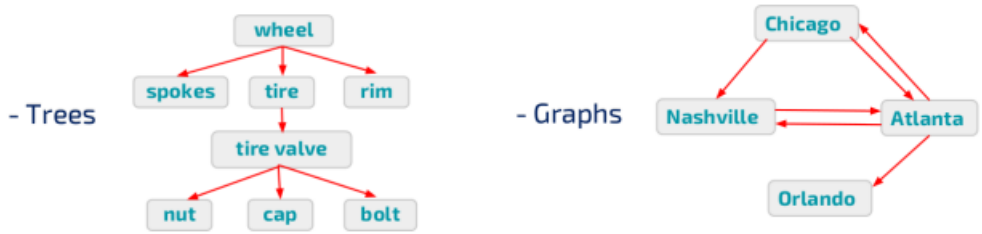
Contents

Common Table Expressions (CTEs) are a standard SQL feature, and are essentially temporary named result sets. CTEs first appeared in the SQL standard in 1999, and the first implementations began appearing in 2007.

There are two kinds of CTEs:

- Non-recursive
- Recursive, which this article covers.

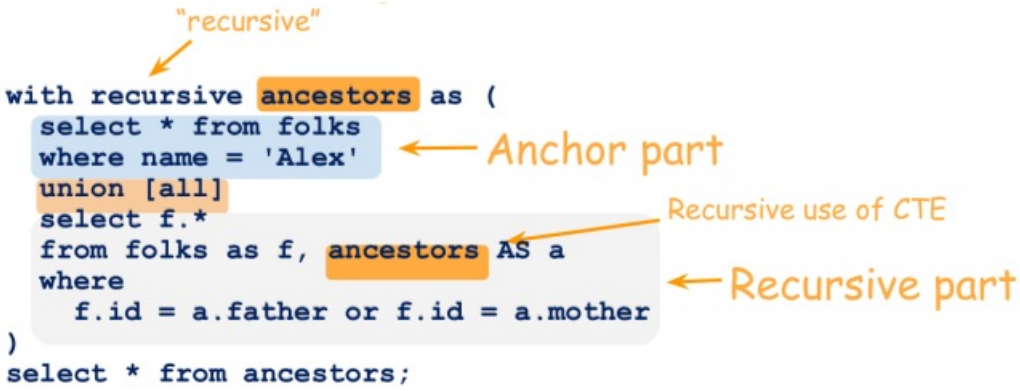
SQL is generally poor at recursive structures.

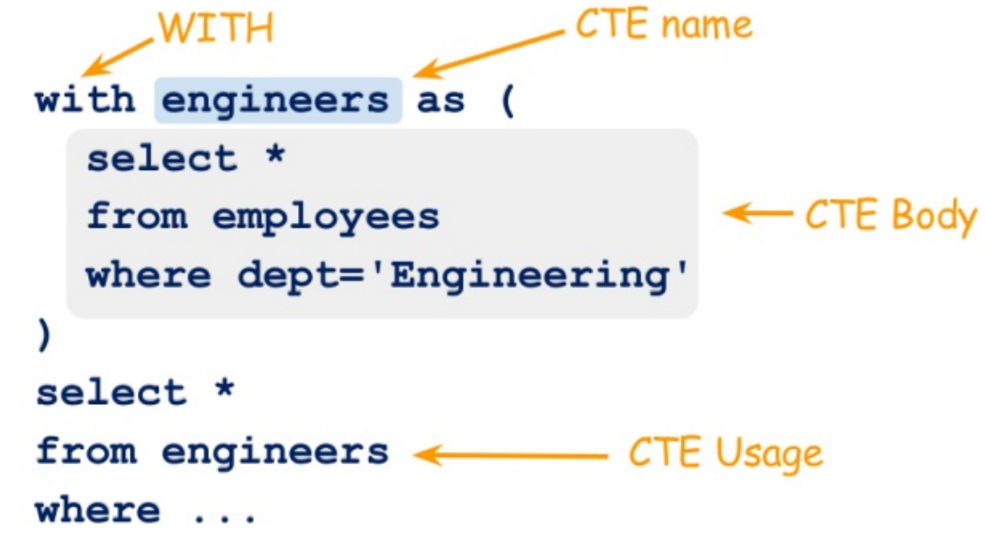


CTEs permit a query to reference itself. A recursive CTE will repeatedly execute subsets of the data until it obtains the complete result set. This makes it particularly useful for handing hierarchical or tree-structured data. max_recursive_iterations avoids infinite loops.

Syntax example

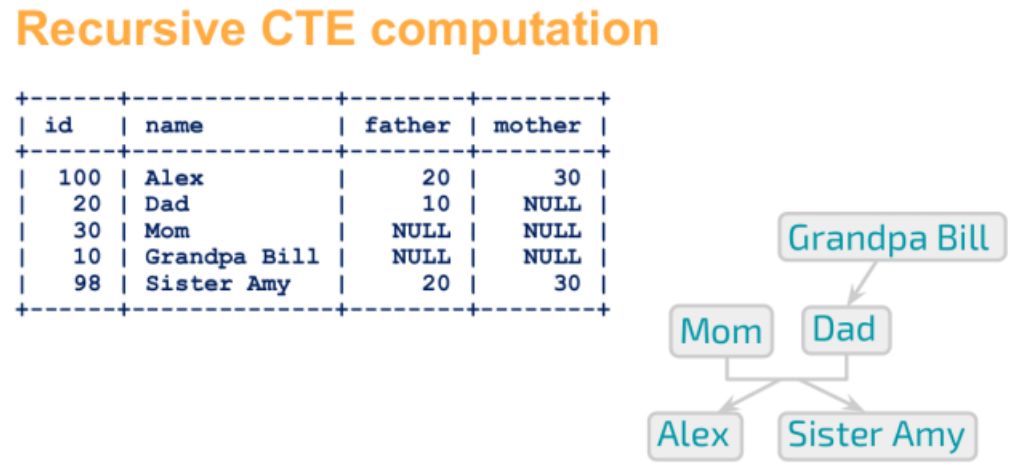
WITH RECURSIVE signifies a recursive CTE. It is given a name, followed by a body (the main query) as follows:





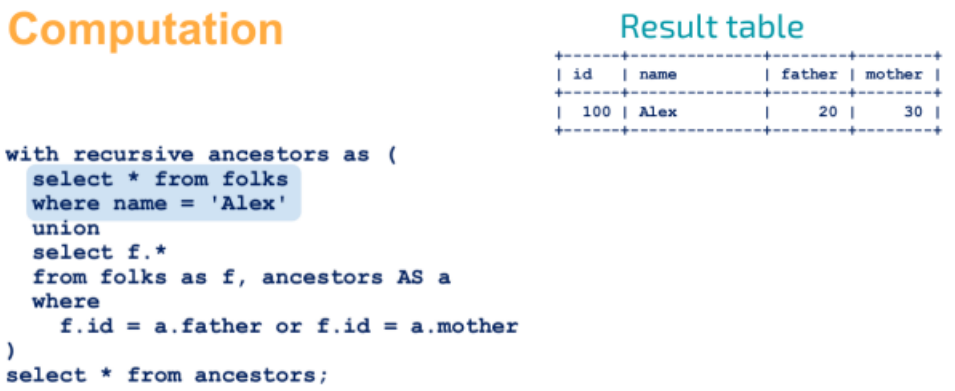
Computation

Given the following structure:



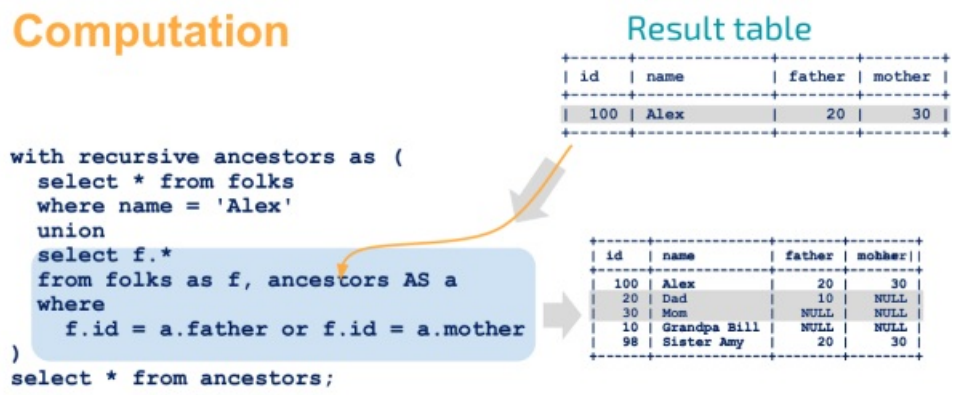
id	name	father	mother
100	Alex	20	30
20	Dad	10	NULL
30	Mom	NULL	NULL
10	Grandpa Bill	NULL	NULL
98	Sister Amy	20	30

First execute the anchor part of the query:

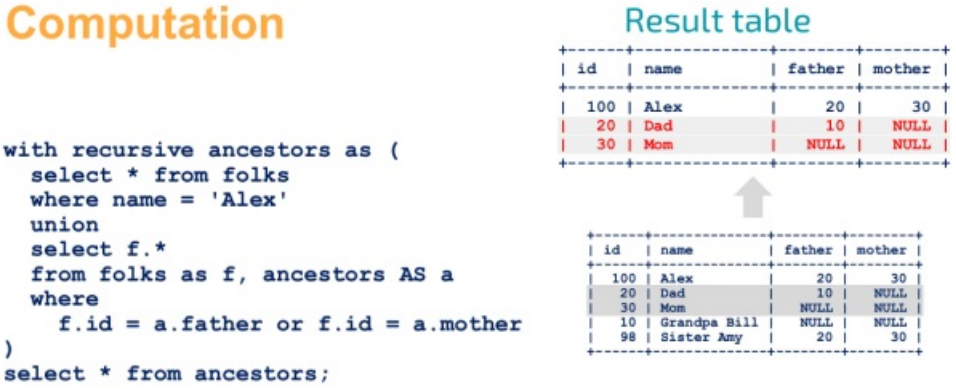


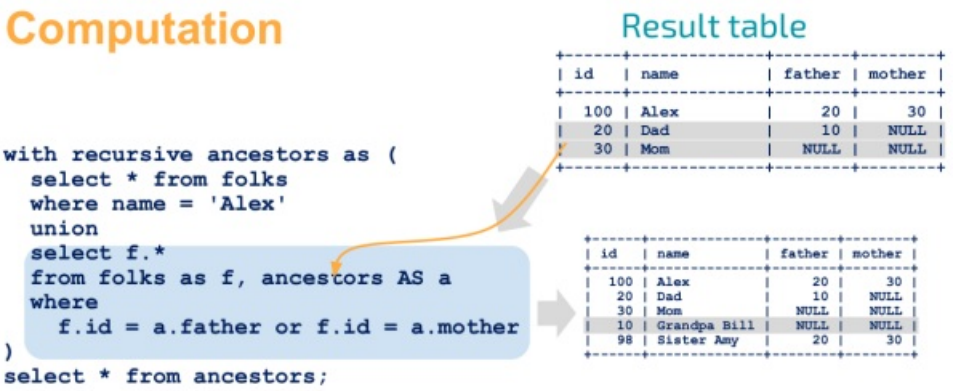
Step #1:
execution of the anchor part

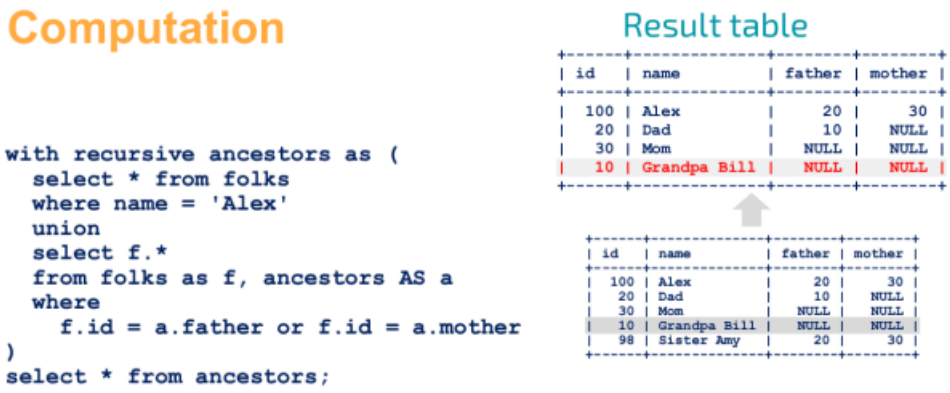
Next, execute the recursive part of the query:

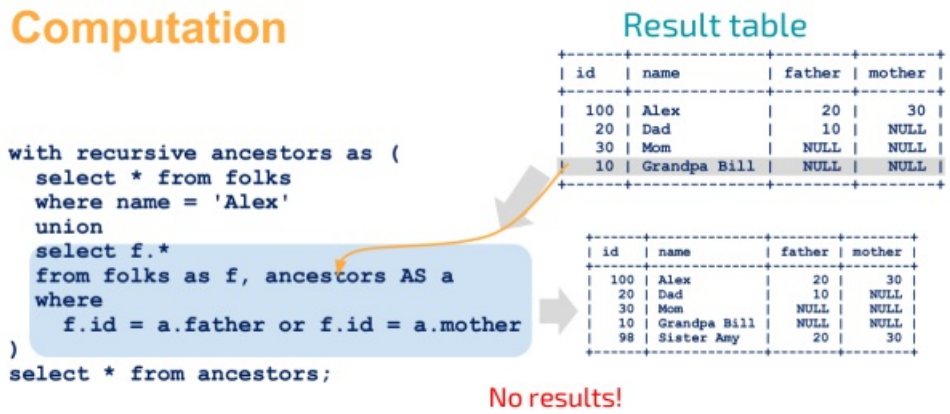


Step #2:
execution of the recursive part









Summary so far

```
with recursive R as (
  select anchor_data
  union [all]
  select recursive_part
  from R, ...
)
select ...
```

1. Compute anchor_data
2. Compute recursive_part to get the new data
3. if (new data is non-empty) goto 2;

CAST to avoid truncating data

As currently implemented by MariaDB and by the SQL Standard, data may be truncated if not correctly cast. It is necessary to CAST the column to the correct width if the CTE's recursive part produces wider values for a column than the CTE's nonrecursive part. Some other DBMS give an error in this situation, and MariaDB's behavior may change in future - see MDEV-12325. See the examples below.

Examples

Transitive closure - determining bus destinations

Sample data:

```
CREATE TABLE bus_routes (origin varchar(50), dst varchar(50));
INSERT INTO bus_routes VALUES
  ('New York', 'Boston'),
  ('Boston', 'New York'),
  ('New York', 'Washington'),
  ('Washington', 'Boston'),
  ('Washington', 'Raleigh');
```

Now, we want to return the bus destinations with New York as the origin:

```
WITH RECURSIVE bus_dst as (
    SELECT origin as dst FROM bus_routes WHERE origin='New York'
  UNION
    SELECT bus_routes.dst FROM bus_routes JOIN bus_dst ON bus_dst.dst= bus_routes.origin
)
SELECT * FROM bus_dst;
+------------+
| dst        |
+------------+
| New York   |
| Boston     |
| Washington |
| Raleigh    |
+------------+
```

The above example is computed as follows:

First, the anchor data is calculated:

- Starting from New York
- Boston and Washington are added

Next, the recursive part:

- Starting from Boston and then Washington
- Raleigh is added
- UNION excludes nodes that are already present.

Computing paths - determining bus routes

This time, we are trying to get bus routes such as “New York -> Washington -> Raleigh”.

Using the same sample data as the previous example:

```
WITH RECURSIVE paths (cur_path, cur_dest) AS (
    SELECT origin, origin FROM bus_routes WHERE origin='New York'
  UNION
    SELECT CONCAT(paths.cur_path, ',', bus_routes.dst), bus_routes.dst
     FROM paths
     JOIN bus_routes
       ON paths.cur_dest = bus_routes.origin AND
         NOT FIND_IN_SET(bus_routes.dst, paths.cur_path)
)
SELECT * FROM paths;
+---------------------------+------------+
| cur_path                  | cur_dest   |
+---------------------------+------------+
| New York                  | New York   |
| New York,Boston           | Boston     |
| New York,Washington       | Washington |
| New York,Washington,Boston | Boston     |
| New York,Washington,Raleigh | Raleigh    |
+---------------------------+------------+
```

CAST to avoid data truncation

In the following example, data is truncated because the results are not specifically cast to a wide enough type:

```
WITH RECURSIVE tbl AS (
  SELECT NULL AS col
  UNION
  SELECT "THIS NEVER SHOWS UP" AS col FROM tbl
)
SELECT col FROM tbl
+------+
| col  |
+------+
| NULL |
|      |
+------+
```

Explicitly use CAST to overcome this:

```
WITH RECURSIVE tbl AS (
  SELECT CAST(NULL AS CHAR(50)) AS col
  UNION SELECT "THIS NEVER SHOWS UP" AS col FROM tbl
)
SELECT * FROM tbl;
+---------------------+
| col                 |
+---------------------+
| NULL                |
| THIS NEVER SHOWS UP |
+---------------------+
```

1.1.1.4.1.7 SELECT WITH ROLLUP

Contents

Syntax

See SELECT for the full syntax.

Description

The `WITH ROLLUP` modifier adds extra rows to the resultset that represent super-aggregate summaries. The super-aggregated column is represented by a `NULL` value. Multiple aggregates over different columns will be added if there are multiple `GROUP BY` columns.

The LIMIT clause can be used at the same time, and is applied after the `WITH ROLLUP` rows have been added.

`WITH ROLLUP` cannot be used with ORDER BY. Some sorting is still possible by using `ASC` or `DESC` clauses with the `GROUP BY` column, although the super-aggregate rows will always be added last.

Examples

These examples use the following sample table

```
CREATE TABLE booksales (
  country VARCHAR(35), genre ENUM('fiction','non-fiction'), year YEAR, sales INT);

INSERT INTO booksales VALUES
  ('Senegal','fiction',2014,12234), ('Senegal','fiction',2015,15647),
  ('Senegal','non-fiction',2014,64980), ('Senegal','non-fiction',2015,78901),
  ('Paraguay','fiction',2014,87970), ('Paraguay','fiction',2015,76940),
  ('Paraguay','non-fiction',2014,8760), ('Paraguay','non-fiction',2015,9030);
```

The addition of the `WITH ROLLUP` modifier in this example adds an extra row that aggregates both years:

```
SELECT year, SUM(sales) FROM booksales GROUP BY year;
+------+------------+
| year | SUM(sales) |
+------+------------+
| 2014 |     173944 |
| 2015 |     180518 |
+------+------------+
2 rows in set (0.08 sec)

SELECT year, SUM(sales) FROM booksales GROUP BY year WITH ROLLUP;
+------+------------+
| year | SUM(sales) |
+------+------------+
| 2014 |     173944 |
| 2015 |     180518 |
| NULL |     354462 |
+------+------------+
```

In the following example, each time the genre, the year or the country change, another super-aggregate row is added:

```
SELECT country, year, genre, SUM(sales)
  FROM booksales GROUP BY country, year, genre;
+----------+------+-------------+------------+
| country  | year | genre       | SUM(sales) |
+----------+------+-------------+------------+
| Paraguay | 2014 | fiction     |      87970 |
| Paraguay | 2014 | non-fiction |       8760 |
| Paraguay | 2015 | fiction     |      76940 |
| Paraguay | 2015 | non-fiction |       9030 |
| Senegal  | 2014 | fiction     |      12234 |
| Senegal  | 2014 | non-fiction |      64980 |
| Senegal  | 2015 | fiction     |      15647 |
| Senegal  | 2015 | non-fiction |      78901 |
+----------+------+-------------+------------+

SELECT country, year, genre, SUM(sales)
  FROM booksales GROUP BY country, year, genre WITH ROLLUP;
+----------+------+-------------+------------+
| country  | year | genre       | SUM(sales) |
+----------+------+-------------+------------+
| Paraguay | 2014 | fiction     |      87970 |
| Paraguay | 2014 | non-fiction |       8760 |
| Paraguay | 2014 | NULL        |      96730 |
| Paraguay | 2015 | fiction     |      76940 |
| Paraguay | 2015 | non-fiction |       9030 |
| Paraguay | 2015 | NULL        |      85970 |
| Paraguay | NULL | NULL        |     182700 |
| Senegal  | 2014 | fiction     |      12234 |
| Senegal  | 2014 | non-fiction |      64980 |
| Senegal  | 2014 | NULL        |      77214 |
| Senegal  | 2015 | fiction     |      15647 |
| Senegal  | 2015 | non-fiction |      78901 |
| Senegal  | 2015 | NULL        |      94548 |
| Senegal  | NULL | NULL        |     171762 |
| NULL     | NULL | NULL        |     354462 |
+----------+------+-------------+------------+
```

The LIMIT clause, applied after WITH ROLLUP:

```
SELECT country, year, genre, SUM(sales)
  FROM booksales GROUP BY country, year, genre WITH ROLLUP LIMIT 4;
+----------+------+-------------+------------+
| country  | year | genre       | SUM(sales) |
+----------+------+-------------+------------+
| Paraguay | 2014 | fiction     |      87970 |
| Paraguay | 2014 | non-fiction |       8760 |
| Paraguay | 2014 | NULL        |      96730 |
| Paraguay | 2015 | fiction     |      76940 |
+----------+------+-------------+------------+
```

Sorting by year descending:

```
SELECT country, year, genre, SUM(sales)
  FROM booksales GROUP BY country, year DESC, genre WITH ROLLUP;
+----------+------+-------------+------------+
| country  | year | genre       | SUM(sales) |
+----------+------+-------------+------------+
| Paraguay | 2015 | fiction     |      76940 |
| Paraguay | 2015 | non-fiction |       9030 |
| Paraguay | 2015 | NULL        |      85970 |
| Paraguay | 2014 | fiction     |      87970 |
| Paraguay | 2014 | non-fiction |       8760 |
| Paraguay | 2014 | NULL        |      96730 |
| Paraguay | NULL | NULL        |     182700 |
| Senegal  | 2015 | fiction     |      15647 |
| Senegal  | 2015 | non-fiction |      78901 |
| Senegal  | 2015 | NULL        |      94548 |
| Senegal  | 2014 | fiction     |      12234 |
| Senegal  | 2014 | non-fiction |      64980 |
| Senegal  | 2014 | NULL        |      77214 |
| Senegal  | NULL | NULL        |     171762 |
| NULL     | NULL | NULL        |     354462 |
+----------+------+-------------+------------+
```

1.1.1.4.1.8 SELECT INTO OUTFILE

Syntax

```
SELECT ... INTO OUTFILE 'file_name'
        [CHARACTER SET charset_name]
        [export_options]

export_options:
    [{FIELDS | COLUMNS}
        [TERMINATED BY 'string']
        [[OPTIONALLY] ENCLOSED BY 'char']
        [ESCAPED BY 'char']
    ]
    [LINES
        [STARTING BY 'string']
        [TERMINATED BY 'string']
    ]
```

Contents

Description

`SELECT INTO OUTFILE` writes the resulting rows to a file, and allows the use of column and row terminators to specify a particular output format. The default is to terminate fields with tabs (`\t`) and lines with newlines (`\n`).

The file must not exist. It cannot be overwritten. A user needs the FILE privilege to run this statement. Also, MariaDB needs permission to write files in the specified location. If the secure_file_priv system variable is set to a non-empty directory name, the file can only be written to that directory.

The `LOAD DATA INFILE` statement complements `SELECT INTO OUTFILE` .

Character-sets

The `CHARACTER SET` clause specifies the character set in which the results are to be written. Without the clause, no conversion takes place (the binary character set). In this case, if there are multiple character sets, the output will contain these too, and may not easily be able to be reloaded.

In cases where you have two servers using different character-sets, using `SELECT INTO OUTFILE` to transfer data from one to the other can have unexpected results. To ensure that MariaDB correctly interprets the escape sequences, use the

`CHARACTER SET` clause on both the `SELECT INTO OUTFILE` statement and the subsequent `LOAD DATA INFILE` statement.

Example

The following example produces a file in the CSV format:

```
SELECT customer_id, firstname, surname from customer
  INTO OUTFILE '/exportdata/customers.txt'
  FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
  LINES TERMINATED BY '\n';
```

The following ANSI syntax is also supported for simple `SELECT` without `UNION`

```
SELECT customer_id, firstname, surname INTO OUTFILE '/exportdata/customers.txt'
  FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
  LINES TERMINATED BY '\n'
  FROM customers;
```

If you want to use the ANSI syntax with `UNION` or similar construct you have to use the syntax:

```
SELECT  * INTO OUTFILE "/tmp/skr3" FROM (SELECT * FROM t1 UNION SELECT * FROM t1);
```

1.1.1.4.1.9 SELECT INTO DUMPFILE

Contents

Syntax

```
SELECT ... INTO DUMPFILE 'file_path'
```

Description

`SELECT ... INTO DUMPFILE` is a SELECT clause which writes the resultset into a single unformatted row, without any separators, in a file. The results will not be returned to the client.

file_path can be an absolute path, or a relative path starting from the data directory. It can only be specified as a string literal, not as a variable. However, the statement can be dynamically composed and executed as a prepared statement to work around this limitation.

This statement is binary-safe and so is particularly useful for writing BLOB values to file. It can be used, for example, to copy an image or an audio document from the database to a file. SELECT ... INTO FILE can be used to save a text file.

The file must not exist. It cannot be overwritten. A user needs the FILE privilege to run this statement. Also, MariaDB needs permission to write files in the specified location. If the secure_file_priv system variable is set to a non-empty directory name, the file can only be written to that directory.

Since MariaDB 5.1, the character_set_filesystem system variable has controlled interpretation of file names that are given as literal strings.

Example

```
SELECT _utf8'Hello world!' INTO DUMPFILE '/tmp/world';

SELECT LOAD_FILE('/tmp/world') AS world;
+--------------+
| world        |
+--------------+
| Hello world! |
+--------------+
```

1.1.1.4.1.10 FOR UPDATE

InnoDB supports row-level locking. Selected rows can be locked using LOCK IN SHARE MODE or FOR UPDATE. In both cases, a lock is acquired on the rows read by the query, and it will be released when the current transaction is committed.

The `FOR UPDATE` clause of SELECT applies only when autocommit is set to 0 or the `SELECT` is enclosed in a transaction. A lock is acquired on the rows, and other transactions are prevented from writing the rows, acquire locks, and from reading them (unless their isolation level is `READ UNCOMMITTED`).

If `autocommit` is set to 1, the LOCK IN SHARE MODE and `FOR UPDATE` clauses have no effect in InnoDB. For non-transactional storage engines like MyISAM and ARIA, a table level lock will be taken even if autocommit is set to 1.

If the isolation level is set to SERIALIZABLE, all plain `SELECT` statements are converted to `SELECT ... LOCK IN SHARE MODE`.

Example

```
SELECT * FROM trans WHERE period=2001 FOR UPDATE;
```

1.1.1.4.1.11 LOCK IN SHARE MODE

InnoDB supports row-level locking. Selected rows can be locked using `LOCK IN SHARE MODE` or FOR UPDATE. In both cases, a lock is acquired on the rows read by the query, and it will be released when the current transaction is committed.

When `LOCK IN SHARE MODE` is specified in a SELECT statement, MariaDB will wait until all transactions that have modified the rows are committed. Then, a write lock is acquired. All transactions can read the rows, but if they want to modify them, they have to wait until your transaction is committed.

If autocommit is set to 1 (the default), the LOCK IN SHARE MODE and FOR UPDATE clauses have no effect in InnoDB. For non-transactional storage engines like MyISAM and ARIA, a table level lock will be taken even if autocommit is set to 1.

1.1.1.4.1.12 Optimizer Hints

Optimizer hints

There are some options available in SELECT to affect the execution plan. These are known as optimizer hints.

HIGH PRIORITY

`HIGH_PRIORITY` gives the statement a higher priority. If the table is locked, high priority `SELECT`s will be executed as soon as the lock is released, even if other statements are queued. `HIGH_PRIORITY` applies only if the storage engine only supports table-level locking (`MyISAM`, `MEMORY`, `MERGE`). See HIGH_PRIORITY and LOW_PRIORITY clauses for details.

SQL_CACHE / SQL_NO_CACHE

If the query_cache_type system variable is set to 2 or `DEMAND`, and the current statement is cacheable, `SQL_CACHE` causes the query to be cached and `SQL_NO_CACHE` causes the query not to be cached. For `UNION`s, `SQL_CACHE` or `SQL_NO_CACHE` should be specified for the first query. See also The Query Cache for more detail and a list of the types of statements that aren't cacheable.

SQL_BUFFER_RESULT

`SQL_BUFFER_RESULT` forces the optimizer to use a temporary table to process the result. This is useful to free locks as soon as possible.

SQL_SMALL_RESULT / SQL_BIG_RESULT

`SQL_SMALL_RESULT` and `SQL_BIG_RESULT` tell the optimizer whether the result is very big or not. Usually, `GROUP BY` and `DISTINCT` operations are performed using a temporary table. Only if the result is very big, using a temporary table is not convenient. The optimizer automatically knows if the result is too big, but you can force the optimizer to use a temporary table with `SQL_SMALL_RESULT`, or avoid the temporary table using `SQL_BIG_RESULT`.

STRAIGHT_JOIN

`STRAIGHT_JOIN` applies to the JOIN queries, and tells the optimizer that the tables must be read in the order they appear in the `SELECT`. For `const` and `system` table this options is sometimes ignored.

SQL_CALC_FOUND_ROWS

`SQL_CALC_FOUND_ROWS` is only applied when using the `LIMIT` clause. If this option is used, MariaDB will count how many rows would match the query, without the `LIMIT` clause. That number can be retrieved in the next query, using FOUND_ROWS().

USE/FORCE/IGNORE INDEX

`USE INDEX`, `FORCE INDEX` and `IGNORE INDEX` constrain the query planning to a specific index.

For further information about some of these options, see How to force query plans.

1.1.1.4.1.13 PROCEDURE

The `PROCEDURE` clause of SELECT passes the whole result set to a Procedure which will process it. These Procedures are not Stored Procedures, and can only be written in the C language, so it is necessary to recompile the server.

Currently, the only available procedure is ANALYSE, which examines the resultset and suggests the optimal datatypes for each column. It is defined in the `sql/sql_analyse.cc` file, and can be used as an example to create more Procedures.

This clause cannot be used in a view's definition.

1.1.4.2 HANDLER

1.1.1.4.1.15 DUAL

Description

You are allowed to specify `DUAL` as a dummy table name in situations where no tables are referenced, such as the following SELECT statement:

```
SELECT 1 + 1 FROM DUAL;
+-------+
| 1 + 1 |
+-------+
|     2 |
+-------+
```

`DUAL` is purely for the convenience of people who require that all `SELECT` statements should have `FROM` and possibly other clauses. MariaDB ignores the clauses. MariaDB does not require `FROM DUAL` if no tables are referenced.

FROM DUAL could be used when you only SELECT computed values, but require a WHERE clause, perhaps to test that a script correctly handles empty resultsets:

```
SELECT 1 FROM DUAL WHERE FALSE;
Empty set (0.00 sec)
```

1.1.1.4.1.16 SELECT ... OFFSET ... FETCH

MariaDB starting with 10.6.0

`SELECT ... OFFSET ... FETCH` was introduced in MariaDB 10.6.

Syntax

```
OFFSET start { ROW | ROWS }
FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } { ONLY | WITH TIES }
```

Description

The `OFFSET` clause allows one to return only those elements of a resultset that come after a specified offset. The `FETCH` clause specifies the number of rows to return, while `ONLY` or `WITH TIES` specifies whether or not to also return any further results that tie for last place according to the ordered resultset.

Either the singular `ROW` or the plural `ROWS` can be used after the `OFFSET` and `FETCH` clauses; the choice has no impact on the results.

In the case of `WITH TIES`, an ORDER BY clause is required, otherwise an ERROR will be returned.

```
SELECT i FROM t1 FETCH FIRST 2 ROWS WITH TIES;
ERROR 4180 (HY000): FETCH ... WITH TIES requires ORDER BY clause to be present
```

Examples

Given a table with 6 rows:

```
CREATE OR REPLACE TABLE t1 (i INT);
INSERT INTO t1 VALUES (1),(2),(3),(4), (4), (5);
SELECT i FROM t1 ORDER BY i ASC;
+------+
| i    |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|    4 |
|    5 |
+------+
```

`OFFSET 2` allows one to skip the first two results.

```
SELECT i FROM t1 ORDER BY i ASC OFFSET 2 ROWS;
+------+
| i    |
+------+
|    3 |
|    4 |
|    4 |
|    5 |
+------+
```

`FETCH FIRST 3 ROWS ONLY` limits the results to three rows only

```
SELECT i FROM t1 ORDER BY i ASC OFFSET 1 ROWS FETCH FIRST 3 ROWS ONLY;
+------+
| i    |
+------+
|    2 |
|    3 |
|    4 |
+------+
```

The same outcome can also be achieved with the LIMIT clause:

```
SELECT i FROM t1 ORDER BY i ASC LIMIT 3 OFFSET 1;
+------+
| i    |
+------+
|    2 |
|    3 |
|    4 |
+------+
```

`WITH TIES` ensures the tied result `4` is also returned.

```
SELECT i FROM t1 ORDER BY i ASC OFFSET 1 ROWS FETCH FIRST 3 ROWS WITH TIES;
+------+
| i    |
+------+
|    2 |
|    3 |
|    4 |
|    4 |
+------+
```

1.1.1.4.2 Inserting & Loading Data

The `INSERT` statement is the primary SQL statement for adding data into a table in MariaDB.

INSERT

Insert rows into a table.

INSERT DELAYED

Queue row to be inserted when thread is free.

INSERT SELECT

Insert the rows returned by a SELECT into a table

LOAD Data into Tables or Index

Loading data quickly into MariaDB

Concurrent Inserts

Under some circumstances, MyISAM allows INSERTs and SELECTs to be executed concurrently.

HIGH_PRIORITY and LOW_PRIORITY

Modifying statement priority in storage engines supporting table-level locks.

IGNORE

Suppress errors while trying to violate a UNIQUE constraint.

INSERT - Default & Duplicate Values

Default and duplicate values when inserting.

INSERT IGNORE

Convert errors to warnings, permitting inserts of additional rows to continue.

INSERT ON DUPLICATE KEY UPDATE

INSERT if no duplicate key is found, otherwise UPDATE.

INSERT...RETURNING

Returns a resultset of the inserted rows.

There are 3 related questions.

1.1.1.4.2.1 INSERT

Syntax

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
 [INTO] tbl_name [PARTITION (partition_list)] [(col,...)]
 {VALUES | VALUE} ({expr | DEFAULT},...),(...),...
 [ ON DUPLICATE KEY UPDATE
   col=expr
     [, col=expr] ... ] [RETURNING select_expr
      [, select_expr ...]]
```

Or:

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name [PARTITION (partition_list)]
    SET col={expr | DEFAULT}, ...
    [ ON DUPLICATE KEY UPDATE
      col=expr
        [, col=expr] ... ] [RETURNING select_expr
      [, select_expr ...]]
```

Or:

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name [PARTITION (partition_list)] [(col,...)]
    SELECT ...
    [ ON DUPLICATE KEY UPDATE
      col=expr
        [, col=expr] ... ] [RETURNING select_expr
      [, select_expr ...]]
```

Contents

The `INSERT` statement is used to insert new rows into an existing table. The `INSERT ... VALUES` and `INSERT ... SET` forms of the statement insert rows based on explicitly specified values. The `INSERT ... SELECT` form inserts rows selected from another table or tables. `INSERT ... SELECT` is discussed further in the `INSERT ... SELECT` article.

The table name can be specified in the form `db_name . tbl_name` or, if a default database is selected, in the form `tbl_name` (see Identifier Qualifiers). This allows to use INSERT ... SELECT to copy rows between different databases.

The PARTITION clause can be used in both the INSERT and the SELECT part. See Partition Pruning and Selection for details.

MariaDB starting with 10.5

The RETURNING clause was introduced in MariaDB 10.5.

The columns list is optional. It specifies which values are explicitly inserted, and in which order. If this clause is not specified, all values must be explicitly specified, in the same order they are listed in the table definition.

The list of value follow the `VALUES` or `VALUE` keyword (which are interchangeable, regardless how much values you want to insert), and is wrapped by parenthesis. The values must be listed in the same order as the columns list. It is possible to specify more than one list to insert more than one rows with a single statement. If many rows are inserted, this is a speed optimization.

For one-row statements, the `SET` clause may be more simple, because you don't need to remember the columns order. All values are specified in the form `col = expr .`

Values can also be specified in the form of a SQL expression or subquery. However, the subquery cannot access the same table that is named in the `INTO` clause.

If you use the `LOW_PRIORITY` keyword, execution of the `INSERT` is delayed until no other clients are reading from the table. If you use the `HIGH_PRIORITY` keyword, the statement has the same priority as `SELECT` s. This affects only storage engines that use only table-level locking (MyISAM, MEMORY, MERGE). However, if one of these keywords is specified, concurrent inserts cannot be used. See HIGH_PRIORITY and LOW_PRIORITY clauses for details.

INSERT DELAYED

For more details on the `DELAYED` option, see INSERT DELAYED.

HIGH PRIORITY and LOW PRIORITY

See HIGH_PRIORITY and LOW_PRIORITY.

Defaults and Duplicate Values

See INSERT - Default & Duplicate Values for details..

INSERT IGNORE

See INSERT IGNORE.

INSERT ON DUPLICATE KEY UPDATE

See INSERT ON DUPLICATE KEY UPDATE.

Examples

Specifying the column names:

```
INSERT INTO person (first_name, last_name) VALUES ('John', 'Doe');
```

Inserting more than 1 row at a time:

```
INSERT INTO tbl_name VALUES (1, "row 1"), (2, "row 2");
```

Using the `SET` clause:

```
INSERT INTO person SET first_name = 'John', last_name = 'Doe';
```

SELECTing from another table:

```
INSERT INTO contractor SELECT * FROM person WHERE status = 'c';
```

See INSERT ON DUPLICATE KEY UPDATE and INSERT IGNORE for further examples.

INSERT ... RETURNING

`INSERT ... RETURNING` returns a resultset of the inserted rows.

This returns the listed columns for all the rows that are inserted, or alternatively, the specified SELECT expression. Any SQL expressions which can be calculated can be used in the select expression for the RETURNING clause, including virtual columns and aliases, expressions which use various operators such as bitwise, logical and arithmetic operators, string functions, date-time functions, numeric functions, control flow functions, secondary functions and stored functions. Along with this, statements which have subqueries and prepared statements can also be used.

Examples

Simple INSERT statement

```
INSERT INTO t2 VALUES (1,'Dog'),(2,'Lion'),(3,'Tiger'),(4,'Leopard')
RETURNING id2,id2+id2,id2&id2,id2||id2;
+-----+---------+---------+----------+
| id2 | id2+id2 | id2&id2 | id2||id2 |
+-----+---------+---------+----------+
|   1 |       2 |       1 |        1 |
|   2 |       4 |       2 |        1 |
|   3 |       6 |       3 |        1 |
|   4 |       8 |       4 |        1 |
+-----+---------+---------+----------+
```

Using stored functions in RETURNING

```
DELIMITER |
CREATE FUNCTION f(arg INT) RETURNS INT
    BEGIN
       RETURN (SELECT arg+arg);
    END|

DELIMITER ;

PREPARE stmt FROM "INSERT INTO t1 SET id1=1, animal1='Bear' RETURNING f(id1), UPPER(animal1)";

EXECUTE stmt;
+---------+----------------+
| f(id1)  | UPPER(animal1) |
+---------+----------------+
|       2 | BEAR           |
+---------+----------------+
```

Subqueries in the RETURNING clause that return more than one row or column cannot be used.

Aggregate functions cannot be used in the RETURNING clause. Since aggregate functions work on a set of values, and if the purpose is to get the row count, ROW_COUNT() with SELECT can be used or it can be used in INSERT...SELECT...RETURNING if the table in the RETURNING clause is not the same as the INSERT table.

1.1.1.4.2.2 INSERT DELAYED

Syntax

```
INSERT DELAYED ...
```

Contents

Description

The `DELAYED` option for the `INSERT` statement is a MariaDB/MySQL extension to standard SQL that is very useful if you have clients that cannot or need not wait for the `INSERT` to complete. This is a common situation when you use MariaDB for logging and you also periodically run `SELECT` and `UPDATE` statements that take a long time to complete.

When a client uses `INSERT DELAYED`, it gets an okay from the server at once, and the row is queued to be inserted when the table is not in use by any other thread.

Another major benefit of using `INSERT DELAYED` is that inserts from many clients are bundled together and written in one block. This is much faster than performing many separate inserts.

Note that `INSERT DELAYED` is slower than a normal `INSERT` if the table is not otherwise in use. There is also the additional overhead for the server to handle a separate thread for each table for which there are delayed rows. This means that you should use `INSERT DELAYED` only when you are really sure that you need it.

The queued rows are held only in memory until they are inserted into the table. This means that if you terminate mysqld forcibly (for example, with kill -9) or if mysqld dies unexpectedly, any queued rows that have not been written to disk are lost.

The number of concurrent `INSERT DELAYED` threads is limited by the `max_delayed_threads` server system variables. If

it is set to 0, `INSERT DELAYED` is disabled. The session value can be equal to the global value, or 0 to disable this statement for the current session. If this limit has been reached, the `DELAYED` clause will be silently ignore for subsequent statements (no error will be produced).

Limitations

There are some limitations on the use of `DELAYED`:

- `INSERT DELAYED` works only with MyISAM, MEMORY, ARCHIVE, and BLACKHOLE tables. If you execute INSERT DELAYED with another storage engine, you will get an error like this: `ERROR 1616 (HY000): DELAYED option not supported for table 'tab_name'`
- For MyISAM tables, if there are no free blocks in the middle of the data file, concurrent SELECT and INSERT statements are supported. Under these circumstances, you very seldom need to use `INSERT DELAYED` with MyISAM.
- `INSERT DELAYED` should be used only for `INSERT` statements that specify value lists. The server ignores `DELAYED` for `INSERT ... SELECT` or `INSERT ... ON DUPLICATE KEY UPDATE` statements.
- Because the `INSERT DELAYED` statement returns immediately, before the rows are inserted, you cannot use `LAST_INSERT_ID()` to get the `AUTO_INCREMENT` value that the statement might generate.
- `DELAYED` rows are not visible to `SELECT` statements until they actually have been inserted.
- After `INSERT DELAYED`, ROW_COUNT() returns the number of the rows you tried to insert, not the number of the successful writes.
- `DELAYED` is ignored on slave replication servers, so that `INSERT DELAYED` is treated as a normal `INSERT` on slaves. This is because `DELAYED` could cause the slave to have different data than the master. `INSERT DELAYED` statements are not safe for replication.
- Pending `INSERT DELAYED` statements are lost if a table is write locked and ALTER TABLE is used to modify the table structure.
- `INSERT DELAYED` is not supported for views. If you try, you will get an error like this: `ERROR 1347 (HY000): 'view_name' is not BASE TABLE`
- `INSERT DELAYED` is not supported for partitioned tables.
- `INSERT DELAYED` is not supported within stored programs.
- `INSERT DELAYED` does not work with triggers.
- `INSERT DELAYED` does not work if there is a check constraint in place.
- `INSERT DELAYED` does not work if skip-new mode is active.

1.1.1.4.2.3 INSERT SELECT

Syntax

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name [(col_name,...)]
    SELECT ...
    [ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

Contents

Description

With `INSERT ... SELECT`, you can quickly insert many rows into a table from one or more other tables. For example:

```
INSERT INTO tbl_temp2 (fld_id)
  SELECT tbl_temp1.fld_order_id
  FROM tbl_temp1 WHERE tbl_temp1.fld_order_id > 100;
```

`tbl_name` can also be specified in the form `db_name.tbl_name` (see Identifier Qualifiers). This allows to copy rows between different databases.

If the new table has a primary key or UNIQUE indexes, you can use IGNORE to handle duplicate key errors during the query. The newer values will not be inserted if an identical value already exists.

`REPLACE` can be used instead of `INSERT` to prevent duplicates on `UNIQUE` indexes by deleting old values. In that case, `ON DUPLICATE KEY UPDATE` cannot be used.

`INSERT ... SELECT` works for tables which already exist. To create a table for a given resultset, you can use CREATE TABLE ... SELECT.

1.1.1.4.2.4 LOAD Data into Tables or Index

Loading data quickly into MariaDB

LOAD DATA INFILE

Read rows from a text file into a table.

LOAD INDEX

Loads one or more indexes from one or more MyISAM/Aria tables into a key buffer.

LOAD XML

Load XML data into a table.

LOAD_FILE

Returns file contents as a string.

There are 4 related questions.

1.1.1.4.2.4.1 LOAD DATA INFILE

Syntax

```
LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'file_name'
    [REPLACE | IGNORE]
    INTO TABLE tbl_name
    [CHARACTER SET charset_name]
    [{FIELDS | COLUMNS}
        [TERMINATED BY 'string']
        [[OPTIONALLY] ENCLOSED BY 'char']
        [ESCAPED BY 'char']
    ]
    [LINES
        [STARTING BY 'string']
        [TERMINATED BY 'string']
    ]
    [IGNORE number {LINES|ROWS}]
    [(col_name_or_user_var,...)]
    [SET col_name = expr,...]
```

Contents

Description

`LOAD DATA INFILE` is unsafe for statement-based replication.

Reads rows from a text file into the designated table on the database at a very high speed. The file name must be given as a

literal string.

Files are written to disk using the SELECT INTO OUTFILE statement. You can then read the files back into a table using the `LOAD DATA INFILE` statement. The `FIELDS` and `LINES` clauses are the same in both statements. These clauses are optional, but if both are specified then the `FIELDS` clause must precede `LINES`.

Executing this statement activates `INSERT` triggers.

One must have the FILE privilege to be able to execute LOAD DATA INFILE. This is to ensure normal users cannot read system files. LOAD DATA LOCAL INFILE does not have this requirement.

If the secure_file_priv system variable is set (by default it is not), the loaded file must be present in the specified directory.

Note that MariaDB's systemd unit file restricts access to `/home`, `/root`, and `/run/user` by default. See Configuring access to home directories.

`LOAD DATA LOCAL INFILE`

When you execute the `LOAD DATA INFILE` statement, MariaDB Server attempts to read the input file from its own file system. By contrast, when you execute the `LOAD DATA LOCAL INFILE` statement, the client attempts to read the input file from its file system, and it sends the contents of the input file to the MariaDB Server. This allows you to load files from the client's local file system into the database.

If you don't want to permit this operation (perhaps for security reasons), you can disable the `LOAD DATA LOCAL INFILE` statement on either the server or the client.

- The `LOAD DATA LOCAL INFILE` statement can be disabled on the server by setting the local_infile system variable to `0`.
- The `LOAD DATA LOCAL INFILE` statement can be disabled on the client. If you are using MariaDB Connector/C, this can be done by unsetting the `CLIENT_LOCAL_FILES` capability flag with the mysql_real_connect function or by unsetting the `MYSQL_OPT_LOCAL_INFILE` option with mysql_optionsv function. If you are using a different client or client library, then see the documentation for your specific client or client library to determine how it handles the `LOAD DATA LOCAL INFILE` statement.

If the `LOAD DATA LOCAL INFILE` statement is disabled by either the server or the client and if the user attempts to execute it, then the server will cause the statement to fail with the following error message:

```
The used command is not allowed with this MariaDB version
```

Note that it is not entirely accurate to say that the MariaDB version does not support the command. It would be more accurate to say that the MariaDB configuration does not support the command. See MDEV-20500 for more information.

From MariaDB 10.5.2, the error message is more accurate:

```
The used command is not allowed because the MariaDB server or client
  has disabled the local infile capability
```

`REPLACE` and `IGNORE`

If you load data from a file into a table that already contains data and has a primary key, you may encounter issues where the statement attempts to insert a row with a primary key that already exists. When this happens, the statement fails with Error 1064, protecting the data already on the table. If you want MariaDB to overwrite duplicates, use the `REPLACE` keyword.

The `REPLACE` keyword works like the REPLACE statement. Here, the statement attempts to load the data from the file. If the row does not exist, it adds it to the table. If the row contains an existing primary key, it replaces the table data. That is, in the event of a conflict, it assumes the file contains the desired row.

This operation can cause a degradation in load speed by a factor of 20 or more if the part that has already been loaded is larger than the capacity of the InnoDB Buffer Pool. This happens because it causes a lot of turnaround in the buffer pool.

Use the `IGNORE` keyword when you want to skip any rows that contain a conflicting primary key. Here, the statement attempts to load the data from the file. If the row does not exist, it adds it to the table. If the row contains an existing primary key, it ignores the addition request and moves on to the next. That is, in the event of a conflict, it assumes the table contains the desired row.

Character-sets

When the statement opens the file, it attempts to read the contents using the default character-set, as defined by the character_set_database system variable.

In the cases where the file was written using a character-set other than the default, you can specify the character-set to use with the `CHARACTER SET` clause in the statement. It ignores character-sets specified by the SET NAMES statement and by the character_set_client system variable. Setting the `CHARACTER SET` clause to a value of `binary` indicates "no conversion."

The statement interprets all fields in the file as having the same character-set, regardless of the column data type. To properly interpret file contents, you must ensure that it was written with the correct character-set. If you write a data file with mariadb-dump -T or with the SELECT INTO OUTFILE statement with the mariadb client, be sure to use the `--default-character-set` option, so that the output is written with the desired character-set.

When using mixed character sets, use the `CHARACTER SET` clause in both SELECT INTO OUTFILE and `LOAD DATA INFILE` to ensure that MariaDB correctly interprets the escape sequences.

The character_set_filesystem system variable controls the interpretation of the filename.

It is currently not possible to load data files that use the `ucs2` character set.

Preprocessing Inputs

col_name_or_user_var can be a column name, or a user variable. In the case of a variable, the SET statement can be used to preprocess the value before loading into the table.

Priority and Concurrency

In storage engines that perform table-level locking (MyISAM, MEMORY and MERGE), using the LOW_PRIORITY keyword, MariaDB delays insertions until no other clients are reading from the table. Alternatively, when using the MyISAM storage engine, you can use the CONCURRENT keyword to perform concurrent insertion.

The `LOW_PRIORITY` and `CONCURRENT` keywords are mutually exclusive. They cannot be used in the same statement.

Progress Reporting

The `LOAD DATA INFILE` statement supports progress reporting. You may find this useful when dealing with long-running operations. Using another client you can issue a SHOW PROCESSLIST query to check the progress of the data load.

Using mariadb-import

MariaDB ships with a separate utility for loading data from files: mariadb-import (or `mysqlimport` before MariaDB 10.5). It operates by sending `LOAD DATA INFILE` statements to the server.

Using mariadb-import you can compress the file using the `--compress` option, to get better performance over slow networks, providing both the client and server support the compressed protocol. Use the `--local` option to load from the local file system.

Indexing

In cases where the storage engine supports ALTER TABLE... DISABLE KEYS statements (MyISAM and Aria), the `LOAD DATA INFILE` statement automatically disables indexes during the execution.

Examples

You have a file with this content (note the the separator is ',', not tab, which is the default):

```
2,2
3,3
4,4
5,5
6,8
```

```
CREATE TABLE t1 (a int, b int, c int, d int, PRIMARY KEY (a));
LOAD DATA LOCAL INFILE
 '/tmp/loaddata7.dat' INTO TABLE t1 FIELDS TERMINATED BY ',' (a,b) SET c=a+b;
SELECT * FROM t1;
+------+------+------+
| a    | b    | c    |
+------+------+------+
|    2 |    2 |    4 |
|    3 |    3 |    6 |
|    4 |    4 |    8 |
|    5 |    5 |   10 |
|    6 |    8 |   14 |
+------+------+------+
```

Another example, given the following data (the separator is a tab):

```
1       a
2       b
```

The value of the first column is doubled before loading:

```
LOAD DATA INFILE 'ld.txt' INTO TABLE ld (@i,v) SET i=@i*2;

SELECT * FROM ld;
+------+------+
| i    | v    |
+------+------+
|    2 | a    |
|    4 | b    |
+------+------+
```

1.1.1.2.17 LOAD INDEX

1.1.1.4.2.4.3 LOAD XML

Syntax

```
LOAD XML [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'file_name'
    [REPLACE | IGNORE]
    INTO TABLE [db_name.]tbl_name
    [CHARACTER SET charset_name]
    [ROWS IDENTIFIED BY '<tagname>']
    [IGNORE number {LINES | ROWS}]
    [(column_or_user_var,...)]
    [SET col_name = expr,...]
```

Description

The LOAD XML statement reads data from an XML file into a table. The `file_name` must be given as a literal string. The `tagname` in the optional ROWS IDENTIFIED BY clause must also be given as a literal string, and must be surrounded by angle brackets (< and >).

LOAD XML acts as the complement of running the mariadb client in XML output mode (that is, starting the client with the --xml option). To write data from a table to an XML file, use a command such as the following one from the system shell:

```
shell> mariadb --xml -e 'SELECT * FROM mytable' > file.xml
```

To read the file back into a table, use LOAD XML INFILE. By default, the <row> element is considered to be the equivalent of a database table row; this can be changed using the ROWS IDENTIFIED BY clause.

This statement supports three different XML formats:

- Column names as attributes and column values as attribute values:

```
<row column1="value1" column2="value2" .../>
```

- Column names as tags and column values as the content of these tags:

```
<row>
  <column1>value1</column1>
  <column2>value2</column2>
</row>
```

- Column names are the name attributes of <field> tags, and values are the contents of these tags:

```
<row>
  <field name='column1'>value1</field>
  <field name='column2'>value2</field>
</row>
```

This is the format used by other tools, such as mariadb-dump.

All 3 formats can be used in the same XML file; the import routine automatically detects the format for each row and interprets it correctly. Tags are matched based on the tag or attribute name and the column name.

The following clauses work essentially the same way for LOAD XML as they do for LOAD DATA:

- LOW_PRIORITY or CONCURRENT
- LOCAL
- REPLACE or IGNORE
- CHARACTER SET
- (column_or_user_var,...)
- SET

See LOAD DATA for more information about these clauses.

The IGNORE number LINES or IGNORE number ROWS clause causes the first number rows in the XML file to be skipped. It is analogous to the LOAD DATA statement's IGNORE ... LINES clause.

If the `LOW_PRIORITY` keyword is used, insertions are delayed until no other clients are reading from the table. The `CONCURRENT` keyword allows the use of concurrent inserts. These clauses cannot be specified together.

This statement activates INSERT triggers.

1.1.1.4.2.4.4 LOAD_FILE

Syntax

```
LOAD_FILE(file_name)
```

Contents

Description

Reads the file and returns the file contents as a string. To use this function, the file must be located on the server host, you must specify the full path name to the file, and you must have the FILE privilege. The file must be readable by all and it must be less than the size, in bytes, of the max_allowed_packet system variable. If the secure_file_priv system variable is set to a non-empty directory name, the file to be loaded must be located in that directory.

If the file does not exist or cannot be read because one of the preceding conditions is not satisfied, the function returns NULL.

Since MariaDB 5.1, the character_set_filesystem system variable has controlled interpretation of file names that are given as literal strings.

Statements using the LOAD_FILE() function are not safe for statement based replication. This is because the slave will execute the LOAD_FILE() command itself. If the file doesn't exist on the slave, the function will return NULL.

Examples

```
UPDATE t SET blob_col=LOAD_FILE('/tmp/picture') WHERE id=1;
```

1.1.1.4.2.5 Concurrent Inserts

Contents

The MyISAM storage engine supports concurrent inserts. This feature allows SELECT statements to be executed during INSERT operations, reducing contention.

Whether concurrent inserts can be used or not depends on the value of the concurrent_insert server system variable:

- `NEVER` (0) disables concurrent inserts.
- `AUTO` (1) allows concurrent inserts only when the target table has no free blocks (no data in the middle of the table has been deleted after the last OPTIMIZE TABLE). This is the default.
- `ALWAYS` (2) always enables concurrent inserts, in which case new rows are added at the end of a table if the table is being used by another thread.

If the binary log is used, CREATE TABLE ... SELECT and INSERT ... SELECT statements cannot use concurrent inserts. These statements acquire a read lock on the table, so concurrent inserts will need to wait. This way the log can be safely used to restore data.

Concurrent inserts are not used by replicas with the row based replication (see binary log formats).

If an INSERT statement contain the HIGH_PRIORITY clause, concurrent inserts cannot be used. INSERT ... DELAYED is usually unneeded if concurrent inserts are enabled.

LOAD DATA INFILE uses concurrent inserts if the `CONCURRENT` keyword is specified and concurrent_insert is not `NEVER`. This makes the statement slower (even if no other sessions access the table) but reduces contention.

LOCK TABLES allows non-conflicting concurrent inserts if a `READ LOCAL` lock is used. Concurrent inserts are not allowed if the `LOCAL` keyword is omitted.

Notes

The decision to enable concurrent insert for a table is done when the table is opened. If you change the value of concurrent_insert it will only affect new opened tables. If you want it to work for also for tables in use or cached, you should do FLUSH TABLES after setting the variable.

1.1.1.4.2.6 HIGH_PRIORITY and LOW_PRIORITY

Contents

The InnoDB storage engine uses row-level locking to ensure data integrity. However some storage engines (such as MEMORY, MyISAM, Aria and MERGE) lock the whole table to prevent conflicts. These storage engines use two separate queues to remember pending statements; one is for SELECTs and the other one is for write statements (INSERT, DELETE, UPDATE). By default, the latter has a higher priority.

To give write operations a lower priority, the low_priority_updates server system variable can be set to `ON`. The option is available on both the global and session levels, and it can be set at startup or via the SET statement.

When too many table locks have been set by write statements, some pending SELECTs are executed. The maximum number of write locks that can be acquired before this happens is determined by the max_write_lock_count server system variable, which is dynamic.

If write statements have a higher priority (default), the priority of individual write statements (INSERT, REPLACE, UPDATE, DELETE) can be changed via the `LOW_PRIORITY` attribute, and the priority of a `SELECT` statement can be raised via the `HIGH_PRIORITY` attribute. Also, LOCK TABLES supports a `LOW_PRIORITY` attribute for `WRITE` locks.

If read statements have a higher priority, the priority of an `INSERT` can be changed via the `HIGH_PRIORITY` attribute. However, the priority of other write statements cannot be raised individually.

The use of `LOW_PRIORITY` or `HIGH_PRIORITY` for an `INSERT` prevents Concurrent Inserts from being used.

1.1.1.2.1.21 IGNORE

1.1.1.4.2.8 INSERT - Default & Duplicate Values

Contents

Default Values

If the `SQL_MODE` contains `STRICT_TRANS_TABLES` and you are inserting into a transactional table (like InnoDB), or if the SQL_MODE contains `STRICT_ALL_TABLES`, all `NOT NULL` columns which do not have a `DEFAULT` value (and are not AUTO_INCREMENT) must be explicitly referenced in `INSERT` statements. If not, an error like this is produced:

```
ERROR 1364 (HY000): Field 'col' doesn't have a default value
```

In all other cases, if a `NOT NULL` column without a `DEFAULT` value is not referenced, an empty value will be inserted (for example, 0 for `INTEGER` columns and '' for `CHAR` columns). See NULL Values in MariaDB:Inserting for examples.

If a `NOT NULL` column having a `DEFAULT` value is not referenced, `NULL` will be inserted.

If a `NULL` column having a `DEFAULT` value is not referenced, its default value will be inserted. It is also possible to explicitly assign the default value using the `DEFAULT` keyword or the `DEFAULT()` function.

If the `DEFAULT` keyword is used but the column does not have a `DEFAULT` value, an error like this is produced:

```
ERROR 1364 (HY000): Field 'col' doesn't have a default value
```

Duplicate Values

By default, if you try to insert a duplicate row and there is a `UNIQUE` index, `INSERT` stops and an error like this is produced:

```
ERROR 1062 (23000): Duplicate entry 'dup_value' for key 'col'
```

To handle duplicates you can use the IGNORE clause, INSERT ON DUPLICATE KEY UPDATE or the REPLACE statement. Note that the IGNORE and DELAYED options are ignored when you use ON DUPLICATE KEY UPDATE.

1.1.1.4.2.9 INSERT IGNORE

Contents

Ignoring Errors

Normally INSERT stops and rolls back when it encounters an error.

By using the IGNORE keyword all errors are converted to warnings, which will not stop inserts of additional rows.

Invalid values are changed to the closest valid value and inserted, with a warning produced.

The IGNORE and DELAYED options are ignored when you use ON DUPLICATE KEY UPDATE.

Prior to MySQL and MariaDB 5.5.28, no warnings were issued for duplicate key errors when using `IGNORE`. You can get the old behavior if you set OLD_MODE to `NO_DUP_KEY_WARNINGS_WITH_IGNORE`.

See IGNORE for a full description of effects.

Examples

```
CREATE TABLE t1 (x INT UNIQUE);

INSERT INTO t1 VALUES(1),(2);

INSERT INTO t1 VALUES(2),(3);
ERROR 1062 (23000): Duplicate entry '2' for key 'x'
SELECT * FROM t1;
+------+
| x    |
+------+
|    1 |
|    2 |
+------+

INSERT IGNORE INTO t1 VALUES(2),(3);
Query OK, 1 row affected, 1 warning (0.04 sec)

SHOW WARNINGS;
+---------+------+----------------------------------+
| Level   | Code | Message                          |
+---------+------+----------------------------------+
| Warning | 1062 | Duplicate entry '2' for key 'x' |
+---------+------+----------------------------------+

SELECT * FROM t1;
+------+
| x    |
+------+
|    1 |
|    2 |
|    3 |
+------+
```

Converting values:

```
CREATE OR REPLACE TABLE t2(id INT, t VARCHAR(2) NOT NULL, n INT NOT NULL);

INSERT INTO t2(id) VALUES (1),(2);
ERROR 1364 (HY000): Field 't' doesn't have a default value

INSERT IGNORE INTO t2(id) VALUES (1),(2);
Query OK, 2 rows affected, 2 warnings (0.026 sec)
Records: 2  Duplicates: 0  Warnings: 2

SHOW WARNINGS;
+---------+------+---------------------------------------+
| Level   | Code | Message                               |
+---------+------+---------------------------------------+
| Warning | 1364 | Field 't' doesn't have a default value |
| Warning | 1364 | Field 'n' doesn't have a default value |
+---------+------+---------------------------------------+

SELECT * FROM t2;
+------+---+---+
| id   | t | n |
+------+---+---+
|    1 |   | 0 |
|    2 |   | 0 |
+------+---+---+
```

See INSERT ON DUPLICATE KEY UPDATE for further examples using that syntax.

1.1.1.4.2.10 INSERT ON DUPLICATE KEY UPDATE

Syntax

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
  [INTO] tbl_name [PARTITION (partition_list)] [(col,...)]
  {VALUES | VALUE} ({expr | DEFAULT},...),(...),...
  [ ON DUPLICATE KEY UPDATE
    col=expr
      [, col=expr] ... ]
```

Or:

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name [PARTITION (partition_list)]
    SET col={expr | DEFAULT}, ...
    [ ON DUPLICATE KEY UPDATE
      col=expr
        [, col=expr] ... ]
```

Or:

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name [PARTITION (partition_list)] [(col,...)]
    SELECT ...
    [ ON DUPLICATE KEY UPDATE
      col=expr
        [, col=expr] ... ]
```

Contents

Description

INSERT ... ON DUPLICATE KEY UPDATE is a MariaDB/MySQL extension to the INSERT statement that, if it finds a duplicate unique or primary key, will instead perform an UPDATE.

The row/s affected value is reported as 1 if a row is inserted, and 2 if a row is updated, unless the API's `CLIENT_FOUND_ROWS` flag is set.

If more than one unique index is matched, only the first is updated. It is not recommended to use this statement on tables with more than one unique index.

If the table has an AUTO_INCREMENT primary key and the statement inserts or updates a row, the LAST_INSERT_ID() function returns its AUTO_INCREMENT value.

The VALUES() function can only be used in a `ON DUPLICATE KEY UPDATE` clause and has no meaning in any other context. It returns the column values from the `INSERT` portion of the statement. This function is particularly useful for multi-rows inserts.

The IGNORE and DELAYED options are ignored when you use `ON DUPLICATE KEY UPDATE`.

See Partition Pruning and Selection for details on the PARTITION clause.

This statement activates INSERT and UPDATE triggers. See Trigger Overview for details.

See also a similar statement, REPLACE.

Examples

```
CREATE TABLE ins_duplicate (id INT PRIMARY KEY, animal VARCHAR(30));
INSERT INTO ins_duplicate VALUES (1,'Aardvark'), (2,'Cheetah'), (3,'Zebra');
```

If there is no existing key, the statement runs as a regular INSERT:

```
INSERT INTO ins_duplicate VALUES (4,'Gorilla')
  ON DUPLICATE KEY UPDATE animal='Gorilla';
Query OK, 1 row affected (0.07 sec)
```

```
SELECT * FROM ins_duplicate;
+----+----------+
| id | animal   |
+----+----------+
|  1 | Aardvark |
|  2 | Cheetah  |
|  3 | Zebra    |
|  4 | Gorilla  |
+----+----------+
```

A regular INSERT with a primary key value of 1 will fail, due to the existing key:

```
INSERT INTO ins_duplicate VALUES (1,'Antelope');
ERROR 1062 (23000): Duplicate entry '1' for key 'PRIMARY'
```

However, we can use an INSERT ON DUPLICATE KEY UPDATE instead:

```
INSERT INTO ins_duplicate VALUES (1,'Antelope')
  ON DUPLICATE KEY UPDATE animal='Antelope';
Query OK, 2 rows affected (0.09 sec)
```

Note that there are two rows reported as affected, but this refers only to the UPDATE.

```
SELECT * FROM ins_duplicate;
+----+----------+
| id | animal   |
+----+----------+
|  1 | Antelope |
|  2 | Cheetah  |
|  3 | Zebra    |
|  4 | Gorilla  |
+----+----------+
```

Adding a second unique column:

```
ALTER TABLE ins_duplicate ADD id2 INT;
UPDATE ins_duplicate SET id2=id+10;
ALTER TABLE ins_duplicate ADD UNIQUE KEY(id2);
```

Where two rows match the unique keys match, only the first is updated. This can be unsafe and is not recommended unless you are certain what you are doing.

```
INSERT INTO ins_duplicate VALUES (2,'Lion',13)
  ON DUPLICATE KEY UPDATE animal='Lion';
Query OK, 2 rows affected (0.004 sec)

SELECT * FROM ins_duplicate;
+----+----------+------+
| id | animal   | id2  |
+----+----------+------+
|  1 | Antelope |   11 |
|  2 | Lion     |   12 |
|  3 | Zebra    |   13 |
|  4 | Gorilla  |   14 |
+----+----------+------+
```

Although the third row with an id of 3 has an id2 of 13, which also matched, it was not updated.

Changing id to an auto_increment field. If a new row is added, the auto_increment is moved forward. If the row is updated, it remains the same.

```
ALTER TABLE `ins_duplicate` CHANGE `id` `id` INT( 11 ) NOT NULL AUTO_INCREMENT;
ALTER TABLE ins_duplicate DROP id2;
SELECT Auto_increment FROM INFORMATION_SCHEMA.TABLES
  WHERE TABLE_NAME='ins_duplicate';
+----------------+
| Auto_increment |
+----------------+
|              5 |
+----------------+

INSERT INTO ins_duplicate VALUES (2,'Leopard')
  ON DUPLICATE KEY UPDATE animal='Leopard';
Query OK, 2 rows affected (0.00 sec)

SELECT Auto_increment FROM INFORMATION_SCHEMA.TABLES
  WHERE TABLE_NAME='ins_duplicate';
+----------------+
| Auto_increment |
+----------------+
|              5 |
+----------------+

INSERT INTO ins_duplicate VALUES (5,'Wild Dog')
  ON DUPLICATE KEY UPDATE animal='Wild Dog';
Query OK, 1 row affected (0.09 sec)

SELECT * FROM ins_duplicate;
+----+----------+
| id | animal   |
+----+----------+
|  1 | Antelope |
|  2 | Leopard  |
|  3 | Zebra    |
|  4 | Gorilla  |
|  5 | Wild Dog |
+----+----------+

SELECT Auto_increment FROM INFORMATION_SCHEMA.TABLES
  WHERE TABLE_NAME='ins_duplicate';
+----------------+
| Auto_increment |
+----------------+
|              6 |
+----------------+
```

Refering to column values from the INSERT portion of the statement:

```
INSERT INTO table (a,b,c) VALUES (1,2,3),(4,5,6)
    ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);
```

See the VALUES() function for more.

1.1.1.4.2.11 INSERT...RETURNING

MariaDB starting with 10.5.0

INSERT ... RETURNING was added in MariaDB 10.5.0, and returns a resultset of the inserted rows.

Syntax

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
 [INTO] tbl_name [PARTITION (partition_list)] [(col,...)]
 {VALUES | VALUE} ({expr | DEFAULT},...),(...),...
 [ ON DUPLICATE KEY UPDATE
   col=expr
     [, col=expr] ... ] [RETURNING select_expr
      [, select_expr ...]]
```

Or:

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name [PARTITION (partition_list)]
    SET col={expr | DEFAULT}, ...
    [ ON DUPLICATE KEY UPDATE
      col=expr
        [, col=expr] ... ] [RETURNING select_expr
      [, select_expr ...]]
```

Or:

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name [PARTITION (partition_list)] [(col,...)]
    SELECT ...
    [ ON DUPLICATE KEY UPDATE
      col=expr
        [, col=expr] ... ] [RETURNING select_expr
      [, select_expr ...]]
```

Contents

Description

`INSERT ... RETURNING` returns a resultset of the inserted rows.

This returns the listed columns for all the rows that are inserted, or alternatively, the specified SELECT expression. Any SQL expressions which can be calculated can be used in the select expression for the RETURNING clause, including virtual columns and aliases, expressions which use various operators such as bitwise, logical and arithmetic operators, string functions, date-time functions, numeric functions, control flow functions, secondary functions and stored functions. Along with this, statements which have subqueries and prepared statements can also be used.

Examples

Simple INSERT statements:

```
CREATE OR REPLACE TABLE t2 (id INT, animal VARCHAR(20), t TIMESTAMP);

INSERT INTO t2 (id) VALUES (2),(3) RETURNING id,t;
+------+---------------------+
| id   | t                   |
+------+---------------------+
|    2 | 2021-04-28 00:59:32 |
|    3 | 2021-04-28 00:59:32 |
+------+---------------------+
```

```
INSERT INTO t2(id,animal) VALUES (1,'Dog'),(2,'Lion'),(3,'Tiger'),(4,'Leopard')
  RETURNING id,id+id,id&id,id||id;
+------+-------+-------+--------+
| id   | id+id | id&id | id||id |
+------+-------+-------+--------+
|    1 |     2 |     1 |      1 |
|    2 |     4 |     2 |      1 |
|    3 |     6 |     3 |      1 |
|    4 |     8 |     4 |      1 |
+------+-------+-------+--------+
```

Using stored functions in RETURNING

```
DELIMITER |
CREATE FUNCTION f(arg INT) RETURNS INT
    BEGIN
       RETURN (SELECT arg+arg);
    END|

DELIMITER ;

PREPARE stmt FROM "INSERT INTO t1 SET id1=1, animal1='Bear' RETURNING f(id1), UPPER(animal1)";

EXECUTE stmt;
+---------+----------------+
| f(id1)  | UPPER(animal1) |
+---------+----------------+
|       2 | BEAR           |
+---------+----------------+
```

Subqueries in the RETURNING clause that return more than one row or column cannot be used.

Aggregate functions cannot be used in the RETURNING clause. Since aggregate functions work on a set of values, and if the purpose is to get the row count, ROW_COUNT() with SELECT can be used or it can be used in INSERT...SELECT...RETURNING if the table in the RETURNING clause is not the same as the INSERT table.

1.1.1.4.3 Changing & Deleting Data

DELETE
Delete rows from one or more tables.

HIGH_PRIORITY and LOW_PRIORITY
Modifying statement priority in storage engines supporting table-level locks.

IGNORE
Suppress errors while trying to violate a UNIQUE constraint.

REPLACE
Equivalent to DELETE + INSERT, or just an INSERT if no rows are returned.

REPLACE...RETURNING
Returns a resultset of the replaced rows.

TRUNCATE TABLE
DROP and re-CREATE a table.

UPDATE
Modify rows in one or more tables.

1.1.1.2.1.7 DELETE

1.1.1.4.2.6 HIGH_PRIORITY and LOW_PRIORITY

1.1.1.2.1.21 IGNORE

1.1.1.2.1.15 REPLACE

1.1.1.4.3.5 REPLACE...RETURNING

MariaDB starting with 10.5.0
REPLACE ... RETURNING was added in MariaDB 10.5.0, and returns a resultset of the replaced rows.

Syntax

```
REPLACE [LOW_PRIORITY | DELAYED]
 [INTO] tbl_name [PARTITION (partition_list)] [(col,...)]
 {VALUES | VALUE} ({expr | DEFAULT},...),(...),...
[RETURNING select_expr
      [, select_expr ...]]
```

Or:

```
REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name [PARTITION (partition_list)]
    SET col={expr | DEFAULT}, ...
[RETURNING select_expr
      [, select_expr ...]]
```

Or:

```
REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name [PARTITION (partition_list)] [(col,...)]
    SELECT ...
[RETURNING select_expr
      [, select_expr ...]]
```

Contents

Description

`REPLACE ... RETURNING` returns a resultset of the replaced rows.

This returns the listed columns for all the rows that are replaced, or alternatively, the specified SELECT expression. Any SQL expressions which can be calculated can be used in the select expression for the RETURNING clause, including virtual columns and aliases, expressions which use various operators such as bitwise, logical and arithmetic operators, string functions, date-time functions, numeric functions, control flow functions, secondary functions and stored functions. Along with this, statements which have subqueries and prepared statements can also be used.

Examples

Simple REPLACE statement

```
REPLACE INTO t2 VALUES (1,'Leopard'),(2,'Dog') RETURNING id2, id2+id2
as Total ,id2|id2, id2&&id2;
+-----+-------+---------+----------+
| id2 | Total | id2|id2 | id2&&id2 |
+-----+-------+---------+----------+
|   1 |     2 |       1 |        1 |
|   2 |     4 |       2 |        1 |
+-----+-------+---------+----------+
```

Using stored functions in RETURNING

```
DELIMITER |
CREATE FUNCTION f(arg INT) RETURNS INT
    BEGIN
      RETURN (SELECT arg+arg);
    END|

DELIMITER ;
PREPARE stmt FROM "REPLACE INTO t2 SET id2=3, animal2='Fox' RETURNING f2(id2),
UPPER(animal2)";

EXECUTE stmt;
+---------+----------------+
| f2(id2) | UPPER(animal2) |
+---------+----------------+
|       6 | FOX            |
+---------+----------------+
```

Subqueries in the statement

```
REPLACE INTO t1 SELECT * FROM t2 RETURNING (SELECT id2 FROM t2 WHERE
id2 IN (SELECT id2 FROM t2 WHERE id2=1)) AS new_id;
+--------+
| new_id |
+--------+
|      1 |
|      1 |
|      1 |
|      1 |
+--------+
```

Subqueries in the RETURNING clause that return more than one row or column cannot be used..

Aggregate functions cannot be used in the RETURNING clause. Since aggregate functions work on a set of values and if the purpose is to get the row count, ROW_COUNT() with SELECT can be used, or it can be used in REPLACE...SELECT...RETURNING if the table in the RETURNING clause is not the same as the REPLACE table.

1.1.1.2.1.19 TRUNCATE TABLE

1.1.1.2.1.20 UPDATE

1.1.1.5 Prepared Statements

In addition to using prepared statements from the libmysqld, you can also do prepared statements from any client by using the text based prepared statement interface.

You first prepare the statement with PREPARE, execute with EXECUTE, and release it with DEALLOCATE.

PREPARE Statement

Define a prepare statement.

EXECUTE Statement

Executes a previously PREPAREd statement

DEALLOCATE / DROP PREPARE

Deallocates a prepared statement.

EXECUTE IMMEDIATE

Immediately execute a dynamic SQL statement

Out Parameters in PREPARE

Using question mark placeholders for out-parameters in the PREPARE statement

There are 1 related questions.

1.1.1.5.1 PREPARE Statement

Syntax

```
PREPARE stmt_name FROM preparable_stmt
```

Contents

Description

The `PREPARE` statement prepares a statement and assigns it a name, `stmt_name`, by which to refer to the statement later. Statement names are not case sensitive. `preparable_stmt` is either a string literal or a user variable (not a local variable, an SQL expression or a subquery) that contains the text of the statement. The text must represent a single SQL statement, not multiple statements. Within the statement, "?" characters can be used as parameter markers to indicate where data values are to be bound to the query later when you execute it. The "?" characters should not be enclosed within quotes, even if you intend to bind them to string values. Parameter markers can be used only where expressions should appear, not for SQL keywords, identifiers, and so forth.

The scope of a prepared statement is the session within which it is created. Other sessions cannot see it.

If a prepared statement with the given name already exists, it is deallocated implicitly before the new statement is prepared. This means that if the new statement contains an error and cannot be prepared, an error is returned and no statement with the given name exists.

Prepared statements can be PREPAREd and EXECUTEd in a stored procedure, but not in a stored function or trigger. Also, even if the statement is PREPAREd in a procedure, it will not be deallocated when the procedure execution ends.

A prepared statement can access user-defined variables, but not local variables or procedure's parameters.

If the prepared statement contains a syntax error, PREPARE will fail. As a side effect, stored procedures can use it to check if a statement is valid. For example:

```
CREATE PROCEDURE `test_stmt`(IN sql_text TEXT)
BEGIN
        DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
                SELECT CONCAT(sql_text, ' is not valid');
        END;
        SET @SQL := sql_text;
        PREPARE stmt FROM @SQL;
        DEALLOCATE PREPARE stmt;
END;
```

The FOUND_ROWS() and ROW_COUNT() functions, if called immediatly after EXECUTE, return the number of rows read or affected by the prepared statements; however, if they are called after DEALLOCATE PREPARE, they provide information about this statement. If the prepared statement produces errors or warnings, GET DIAGNOSTICS return information about them. DEALLOCATE PREPARE shouldn't clear the diagnostics area, unless it produces an error.

A prepared statement is executed with `EXECUTE` and released with `DEALLOCATE PREPARE`.

The max_prepared_stmt_count server system variable determines the number of allowed prepared statements that can be prepared on the server. If it is set to 0, prepared statements are not allowed. If the limit is reached, an error similar to the following will be produced:

```
ERROR 1461 (42000): Can't create more than max_prepared_stmt_count statements
  (current value: 0)
```

Oracle Mode

MariaDB starting with 10.3

In Oracle mode from MariaDB 10.3, `PREPARE stmt FROM 'SELECT :1, :2'` is used, instead of `?`.

Permitted Statements

MariaDB starting with 10.6.2
All statements can be prepared, except PREPARE, EXECUTE, and DEALLOCATE / DROP PREPARE.

Prior to this, not all statements can be prepared. Only the following SQL commands are permitted:

- ALTER TABLE
- ANALYZE TABLE
- BINLOG
- CACHE INDEX
- CALL
- CHANGE MASTER
- CHECKSUM {TABLE | TABLES}
- COMMIT
- {CREATE | DROP} DATABASE
- {CREATE | DROP} INDEX
- {CREATE | RENAME | DROP} TABLE
- {CREATE | RENAME | DROP} USER
- {CREATE | DROP} VIEW
- DELETE
- DESCRIBE
- DO
- EXPLAIN
- FLUSH {TABLE | TABLES | TABLES WITH READ LOCK | HOSTS | PRIVILEGES | LOGS | STATUS | MASTER | SLAVE | DES_KEY_FILE | USER_RESOURCES | QUERY CACHE | TABLE_STATISTICS | INDEX_STATISTICS | USER_STATISTICS | CLIENT_STATISTICS}
- GRANT
- INSERT
- INSTALL {PLUGIN | SONAME}
- HANDLER READ
- KILL
- LOAD INDEX INTO CACHE
- OPTIMIZE TABLE
- REPAIR TABLE
- REPLACE
- RESET {MASTER | SLAVE | QUERY CACHE}
- REVOKE
- ROLLBACK
- SELECT
- SET
- SET GLOBAL SQL_SLAVE_SKIP_COUNTER
- SET ROLE
- SET SQL_LOG_BIN
- SET TRANSACTION ISOLATION LEVEL
- SHOW EXPLAIN
- SHOW {DATABASES | TABLES | OPEN TABLES | TABLE STATUS | COLUMNS | INDEX | TRIGGERS | EVENTS | GRANTS | CHARACTER SET | COLLATION | ENGINES | PLUGINS [SONAME] | PRIVILEGES | PROCESSLIST | PROFILE | PROFILES | VARIABLES | STATUS | WARNINGS | ERRORS | TABLE_STATISTICS | INDEX_STATISTICS | USER_STATISTICS | CLIENT_STATISTICS | AUTHORS | CONTRIBUTORS}
- SHOW CREATE {DATABASE | TABLE | VIEW | PROCEDURE | FUNCTION | TRIGGER | EVENT}
- SHOW {FUNCTION | PROCEDURE} CODE
- SHOW BINLOG EVENTS
- SHOW SLAVE HOSTS
- SHOW {MASTER | BINARY} LOGS
- SHOW {MASTER | SLAVE | TABLES | INNODB | FUNCTION | PROCEDURE} STATUS
- SLAVE {START | STOP}
- TRUNCATE TABLE
- SHUTDOWN
- UNINSTALL {PLUGIN | SONAME}
- UPDATE

Synonyms are not listed here, but can be used. For example, DESC can be used instead of DESCRIBE.

Compound statements can be prepared too.

Note that if a statement can be run in a stored routine, it will work even if it is called by a prepared statement. For example, SIGNAL can't be directly prepared. However, it is allowed in stored routines. If the x() procedure contains SIGNAL, you can

still prepare and execute the 'CALL x();' prepared statement.

PREPARE supports most kinds of expressions as well, for example:

```
PREPARE stmt FROM CONCAT('SELECT * FROM ', table_name);
```

When PREPARE is used with a statement which is not supported, the following error is produced:

```
ERROR 1295 (HY000): This command is not supported in the prepared statement protocol yet
```

Example

```
create table t1 (a int,b char(10));
insert into t1 values (1,"one"),(2, "two"),(3,"three");
prepare test from "select * from t1 where a=?";
set @param=2;
execute test using @param;
+------+------+
| a    | b    |
+------+------+
|    2 | two  |
+------+------+
set @param=3;
execute test using @param;
+------+-------+
| a    | b     |
+------+-------+
|    3 | three |
+------+-------+
deallocate prepare test;
```

Since identifiers are not permitted as prepared statements parameters, sometimes it is necessary to dynamically compose an SQL statement. This technique is called *dynamic SQL*). The following example shows how to use dynamic SQL:

```
CREATE PROCEDURE test.stmt_test(IN tab_name VARCHAR(64))
BEGIN
 SET @sql = CONCAT('SELECT COUNT(*) FROM ', tab_name);
 PREPARE stmt FROM @sql;
 EXECUTE stmt;
 DEALLOCATE PREPARE stmt;
END;

CALL test.stmt_test('mysql.user');
+----------+
| COUNT(*) |
+----------+
|        4 |
+----------+
```

Use of variables in prepared statements:

```
PREPARE stmt FROM 'SELECT @x;';

SET @x = 1;

EXECUTE stmt;
+------+
| @x   |
+------+
|    1 |
+------+

SET @x = 0;

EXECUTE stmt;
+------+
| @x   |
+------+
|    0 |
+------+

DEALLOCATE PREPARE stmt;
```

1.1.1.5.2 Out Parameters in PREPARE

MariaDB 10.1.1

Out parameters in PREPARE were only available in MariaDB 10.1.1

One can use question mark placeholders for out-parameters in the PREPARE statement. Only SELECT ... INTO can be used this way:

```
prepare test from "select id into ? from t1 where val=?";
execute test using @out, @in;
```

This is particularly convenient when used with compound statements:

```
PREPARE stmt FROM "BEGIN NOT ATOMIC
  DECLARE v_res INT;
  SELECT COUNT(*) INTO v_res FROM t1;
  SELECT 'Hello World', v_res INTO ?,?;
END"|
```

1.1.1.2.14 EXECUTE STATEMENT

1.1.1.5.4 DEALLOCATE / DROP PREPARE

Syntax

```
{DEALLOCATE | DROP} PREPARE stmt_name
```

Description

To deallocate a prepared statement produced with `PREPARE`, use a `DEALLOCATE PREPARE` statement that refers to the prepared statement name.

A prepared statement is implicitly deallocated when a new `PREPARE` command is issued. In that case, there is no need to use `DEALLOCATE`.

Attempting to execute a prepared statement after deallocating it results in an error, as if it was not prepared at all:

```
ERROR 1243 (HY000): Unknown prepared statement handler (stmt_name) given to EXECUTE
```

If the specified statement has not been PREPAREd, an error similar to the following will be produced:

```
ERROR 1243 (HY000): Unknown prepared statement handler (stmt_name) given to DEALLOCATE PREPARE
```

Example

See example in PREPARE.

1.1.1.5.5 EXECUTE IMMEDIATE

MariaDB starting with 10.2.3
EXECUTE IMMEDIATE was introduced in MariaDB 10.2.3.

Syntax

```
EXECUTE IMMEDIATE statement
    [USING param[, param] ...]

param:
    expression | IGNORE | DEFAULT
```

Description

`EXECUTE IMMEDIATE` executes a dynamic SQL statement created on the fly, which can reduce performance overhead.

For example:

```
EXECUTE IMMEDIATE 'SELECT 1'
```

which is shorthand for:

```
prepare stmt from "select 1";
execute stmt;
deallocate prepare stmt;
```

EXECUTE IMMEDIATE supports complex expressions as prepare source and parameters:

```
EXECUTE IMMEDIATE CONCAT('SELECT COUNT(*) FROM ', 't1', ' WHERE a=?') USING 5+5;
```

Limitations: subselects and stored function calls are not supported as a prepare source.

The following examples return an error:

```
CREATE OR REPLACE FUNCTION f1() RETURNS VARCHAR(64) RETURN 'SELECT * FROM t1';
EXECUTE IMMEDIATE f1();
ERROR 1970 (42000): EXECUTE IMMEDIATE does not support subqueries or stored functions

EXECUTE IMMEDIATE (SELECT 'SELECT * FROM t1');
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that
  corresponds to your MariaDB server version for the right syntax to use near
  'SELECT 'SELECT * FROM t1')' at line 1

CREATE OR REPLACE FUNCTION f1() RETURNS INT RETURN 10;
EXECUTE IMMEDIATE 'SELECT * FROM t1 WHERE a=?' USING f1();
ERROR 1970 (42000): EXECUTE..USING does not support subqueries or stored functions

EXECUTE IMMEDIATE 'SELECT * FROM t1 WHERE a=?' USING (SELECT 10);
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that
  corresponds to your MariaDB server version for the right syntax to use near
  'SELECT 10)' at line 1
```

One can use a user or an SP variable as a workaround:

```
CREATE OR REPLACE FUNCTION f1() RETURNS VARCHAR(64) RETURN 'SELECT * FROM t1';
SET @stmt=f1();
EXECUTE IMMEDIATE @stmt;

SET @stmt=(SELECT 'SELECT 1');
EXECUTE IMMEDIATE @stmt;

CREATE OR REPLACE FUNCTION f1() RETURNS INT RETURN 10;
SET @param=f1();
EXECUTE IMMEDIATE 'SELECT * FROM t1 WHERE a=?' USING @param;

SET @param=(SELECT 10);
EXECUTE IMMEDIATE 'SELECT * FROM t1 WHERE a=?' USING @param;
```

EXECUTE IMMEDIATE supports user variables and SP variables as OUT parameters

```
DELIMITER $$
CREATE OR REPLACE PROCEDURE p1(OUT a INT)
BEGIN
  SET a:= 10;
END;
$$
DELIMITER ;
SET @a=2;
EXECUTE IMMEDIATE 'CALL p1(?)' USING @a;
SELECT @a;
+------+
| @a   |
+------+
|   10 |
+------+
```

Similar to PREPARE, EXECUTE IMMEDIATE is allowed in stored procedures but is not allowed in stored functions.

This example uses EXECUTE IMMEDIATE inside a stored procedure:

```
DELIMITER $$
CREATE OR REPLACE PROCEDURE p1()
BEGIN
  EXECUTE IMMEDIATE 'SELECT 1';
END;
$$
DELIMITER ;
CALL p1;
+---+
| 1 |
+---+
| 1 |
+---+
```

This script returns an error:

```
DELIMITER $$
CREATE FUNCTION f1() RETURNS INT
BEGIN
  EXECUTE IMMEDIATE 'DO 1';
  RETURN 1;
END;
$$
ERROR 1336 (0A000): Dynamic SQL is not allowed in stored function or trigger
```

EXECUTE IMMEDIATE can use DEFAULT and IGNORE indicators as bind parameters:

```
CREATE OR REPLACE TABLE t1 (a INT DEFAULT 10);
EXECUTE IMMEDIATE 'INSERT INTO t1 VALUES (?)' USING DEFAULT;
SELECT * FROM t1;
+------+
| a    |
+------+
|   10 |
+------+
```

EXECUTE IMMEDIATE increments the Com_execute_immediate status variable, as well as the Com_stmt_prepare, Com_stmt_execute and Com_stmt_close status variables.

Note, EXECUTE IMMEDIATE does not increment the Com_execute_sql status variable. *Com_execute_sql* is used only for PREPARE..EXECUTE.

This session screenshot demonstrates how EXECUTE IMMEDIATE affects status variables:

```
SELECT * FROM INFORMATION_SCHEMA.SESSION_STATUS WHERE VARIABLE_NAME RLIKE
  ('COM_(EXECUTE|STMT_PREPARE|STMT_EXECUTE|STMT_CLOSE)');

+-----------------------+----------------+
| VARIABLE_NAME         | VARIABLE_VALUE |
+-----------------------+----------------+
| COM_EXECUTE_IMMEDIATE | 0              |
| COM_EXECUTE_SQL       | 0              |
| COM_STMT_CLOSE        | 0              |
| COM_STMT_EXECUTE      | 0              |
| COM_STMT_PREPARE      | 0              |
+-----------------------+----------------+

EXECUTE IMMEDIATE 'SELECT 1';
+---+
| 1 |
+---+
| 1 |
+---+

SELECT * FROM INFORMATION_SCHEMA.SESSION_STATUS WHERE VARIABLE_NAME RLIKE
  ('COM_(EXECUTE|STMT_PREPARE|STMT_EXECUTE|STMT_CLOSE)');
+-----------------------+----------------+
| VARIABLE_NAME         | VARIABLE_VALUE |
+-----------------------+----------------+
| COM_EXECUTE_IMMEDIATE | 1              |
| COM_EXECUTE_SQL       | 0              |
| COM_STMT_CLOSE        | 1              |
| COM_STMT_EXECUTE      | 1              |
| COM_STMT_PREPARE      | 1              |
+-----------------------+----------------+
```

1.1.1.6 Programmatic & Compound Statements

Compound statements in MariaDB can be used both inside and outside of stored programs.

Using Compound Statements Outside of Stored Programs

Compound statements are not just for stored programs.

BEGIN END

How to write compound statements.

CASE Statement

Conditional construct with multiple choices.

DECLARE CONDITION

For declaring a named error condition (SQLSTATE or error code).

DECLARE HANDLER

Construct to declare how errors are handled.

DECLARE Variable

Declare local variables within stored programs.

FOR

FOR loops allow code to be executed a fixed number of times.

GOTO

Jump to the given label.

IF

A basic conditional construct statement.

ITERATE

Used to repeat the execution of the current loop.

Labels

Identifiers used to identify a BEGIN ... END construct.

LEAVE

Used to exit a code block.

LOOP

Used to loop within a code block without a condition.

REPEAT LOOP

Used to repeat statements until a search condition is true.

RESIGNAL

Used to send a SIGNAL again for the previous error.

RETURN

Statement to terminate execution of a stored function and return a value.

SELECT INTO

SQL statement for inserting values into variables.

SET Variable

Used to insert a value into a variable with a code block.

SIGNAL

May be used to produce a custom error message.

WHILE

Used to repeat a block of SQL statements while a search condition is true.

Cursors

Structure for traversing and processing results, sequentially.

Diagnostics

Error conditions and statement information.

1.1.1.6.1 Using Compound Statements Outside of Stored Programs

Compound statements can also be used outside of stored programs.

```
delimiter |
IF @have_innodb THEN
  CREATE TABLE IF NOT EXISTS innodb_index_stats (
    database_name    VARCHAR(64) NOT NULL,
    table_name       VARCHAR(64) NOT NULL,
    index_name       VARCHAR(64) NOT NULL,
    last_update      TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    stat_name        VARCHAR(64) NOT NULL,
    stat_value       BIGINT UNSIGNED NOT NULL,
    sample_size      BIGINT UNSIGNED,
    stat_description VARCHAR(1024) NOT NULL,
    PRIMARY KEY (database_name, table_name, index_name, stat_name)
  ) ENGINE=INNODB DEFAULT CHARSET=utf8 COLLATE=utf8_bin STATS_PERSISTENT=0;
END IF|
Query OK, 0 rows affected, 2 warnings (0.00 sec)
```

Note, that using compound statements this way is subject to following limitations:

- Only BEGIN, IF, CASE, LOOP, WHILE, REPEAT statements may start a compound statement outside of stored programs.
- BEGIN must use the `BEGIN NOT ATOMIC` syntax (otherwise it'll be confused with BEGIN that starts a transaction).
- A compound statement might not start with a label.
- A compound statement is parsed completely—note "2 warnings" in the above example, even if the condition was false (InnoDB was, indeed, disabled), and the CREATE TABLE statement was not executed, it was still parsed and the parser produced "Unknown storage engine" warning.

Inside a compound block first three limitations do not apply, one can use anything that can be used inside a stored program — including labels, condition handlers, variables, and so on:

```
BEGIN NOT ATOMIC
    DECLARE foo CONDITION FOR 1146;
    DECLARE x INT DEFAULT 0;
    DECLARE CONTINUE HANDLER FOR SET x=1;
    INSERT INTO test.t1 VALUES ("hndlr1", val, 2);
    END|
```

Example how to use `IF` :

```
IF (1>0) THEN BEGIN NOT ATOMIC SELECT 1; END ; END IF;;
```

Example of how to use `WHILE` loop:

```
DELIMITER |
BEGIN NOT ATOMIC
    DECLARE x INT DEFAULT 0;
    WHILE x <= 10 DO
        SET x = x + 1;
        SELECT x;
    END WHILE;
END|
DELIMITER ;
```

1.1.1.6.2 BEGIN END

Syntax

```
[begin_label:] BEGIN [NOT ATOMIC]
    [statement_list]
END [end_label]
```

Contents

`NOT ATOMIC` is required when used outside of a stored procedure. Inside stored procedures or within an anonymous block, `BEGIN` alone starts a new anonymous block.

Description

`BEGIN ... END` syntax is used for writing compound statements. A compound statement can contain multiple statements, enclosed by the `BEGIN` and `END` keywords. statement_list represents a list of one or more statements, each terminated by a semicolon (i.e., `;`) statement delimiter. statement_list is optional, which means that the empty compound statement (`BEGIN END`) is legal.

Note that `END` will perform a commit. If you are running in autocommit mode, every statement will be committed separately. If you are not running in `autocommit` mode, you must execute a COMMIT or ROLLBACK after `END` to get the database up to date.

Use of multiple statements requires that a client is able to send statement strings containing the ; statement delimiter. This is handled in the mysql command-line client with the DELIMITER command. Changing the `;` end-of-statement delimiter (for example, to `//`) allows `;` to be used in a program body.

A compound statement within a stored program can be labeled. `end_label` cannot be given unless `begin_label` also is present. If both are present, they must be the same.

`BEGIN ... END` constructs can be nested. Each block can define its own variables, a `CONDITION` , a `HANDLER` and a CURSOR, which don't exist in the outer blocks. The most local declarations override the outer objects which use the same name (see example below).

The declarations order is the following:

- `DECLARE` local variables;
- `DECLARE CONDITION` s;
- `DECLARE CURSOR` s;
- `DECLARE HANDLER` s;

Note that `DECLARE HANDLER` contains another `BEGIN ... END` construct.

Here is an example of a very simple, anonymous block:

```
BEGIN NOT ATOMIC
SET @a=1;
CREATE TABLE test.t1(a INT);
END|
```

Below is an example of nested blocks in a stored procedure:

```
CREATE PROCEDURE t( )
BEGIN
   DECLARE x TINYINT UNSIGNED DEFAULT 1;
   BEGIN
      DECLARE x CHAR(2) DEFAULT '02';
       DECLARE y TINYINT UNSIGNED DEFAULT 10;
       SELECT x, y;
   END;
   SELECT x;
END;
```

In this example, a TINYINT variable, `x` is declared in the outter block. But in the inner block `x` is re-declared as a CHAR and an `y` variable is declared. The inner SELECT shows the "new" value of `x` , and the value of `y` . But when x is selected in the outer block, the "old" value is returned. The final SELECT doesn't try to read `y` , because it doesn't exist in that context.

1.1.1.6.3 CASE Statement

Syntax

```
CASE case_value
    WHEN when_value THEN statement_list
    [WHEN when_value THEN statement_list] ...
    [ELSE statement_list]
END CASE
```

Or:

```
CASE
    WHEN search_condition THEN statement_list
    [WHEN search_condition THEN statement_list] ...
    [ELSE statement_list]
END CASE
```

Description

The text on this page describes the `CASE` statement for stored programs. See the CASE OPERATOR for details on the CASE operator outside of stored programs.

The `CASE` statement for stored programs implements a complex conditional construct. If a `search_condition` evaluates to true, the corresponding SQL statement list is executed. If no search condition matches, the statement list in the `ELSE` clause is executed. Each `statement_list` consists of one or more statements.

The `CASE` statement cannot have an `ELSE NULL` clause, and it is terminated with `END CASE` instead of `END`. implements a complex conditional construct. If a `search_condition` evaluates to true, the corresponding SQL statement list is executed. If no search condition matches, the statement list in the `ELSE` clause is executed. Each `statement_list` consists of one or more statements.

If no when_value or search_condition matches the value tested and the `CASE` statement contains no `ELSE` clause, a Case not found for `CASE` statement error results.

Each statement_list consists of one or more statements; an empty `statement_list` is not allowed. To handle situations where no value is matched by any `WHEN` clause, use an `ELSE` containing an empty BEGIN ... END block, as shown in this example:

```
DELIMITER |
CREATE PROCEDURE p()
BEGIN
  DECLARE v INT DEFAULT 1;
  CASE v
    WHEN 2 THEN SELECT v;
    WHEN 3 THEN SELECT 0;
    ELSE BEGIN END;
  END CASE;
END;
|
```

The indentation used here in the `ELSE` clause is for purposes of clarity only, and is not otherwise significant. See Delimiters in the mariadb client for more on the use of the delimiter command.

Note: The syntax of the `CASE` statement used inside stored programs differs slightly from that of the SQL CASE expression described in CASE OPERATOR. The `CASE` statement cannot have an `ELSE NULL` clause, and it is terminated with `END CASE` instead of `END`.

1.1.1.6.4 DECLARE CONDITION

Syntax

```
DECLARE condition_name CONDITION FOR condition_value

condition_value:
    SQLSTATE [VALUE] sqlstate_value
  | mysql_error_code
```

Description

The `DECLARE ... CONDITION` statement defines a named error condition. It specifies a condition that needs specific handling and associates a name with that condition. Later, the name can be used in a DECLARE ... HANDLER, SIGNAL or RESIGNAL statement (as long as the statement is located in the same BEGIN ... END block).

Conditions must be declared after local variables, but before CURSORs and HANDLERs.

A condition_value for `DECLARE ... CONDITION` can be an SQLSTATE value (a 5-character string literal) or a MySQL error code (a number). You should not use SQLSTATE value '00000' or MySQL error code 0, because those indicate sucess rather than an error condition. If you try, or if you specify an invalid SQLSTATE value, an error like this is produced:

```
ERROR 1407 (42000): Bad SQLSTATE: '00000'
```

For a list of SQLSTATE values and MariaDB error codes, see MariaDB Error Codes.

1.1.1.6.5 DECLARE HANDLER

Syntax

```
DECLARE handler_type HANDLER
    FOR condition_value [, condition_value] ...
    statement

handler_type:
    CONTINUE
  | EXIT
  | UNDO

condition_value:
    SQLSTATE [VALUE] sqlstate_value
  | condition_name
  | SQLWARNING
  | NOT FOUND
  | SQLEXCEPTION
  | mariadb_error_code
```

Description

The `DECLARE ... HANDLER` statement specifies handlers that each may deal with one or more conditions. If one of these conditions occurs, the specified statement is executed. statement can be a simple statement (for example, `SET var_name = value`), or it can be a compound statement written using BEGIN and END.

Handlers must be declared after local variables, a `CONDITION` and a CURSOR.

For a `CONTINUE` handler, execution of the current program continues after execution of the handler statement. For an EXIT handler, execution terminates for the BEGIN ... END compound statement in which the handler is declared. (This is true even if the condition occurs in an inner block.) The `UNDO` handler type statement is not supported.

If a condition occurs for which no handler has been declared, the default action is `EXIT` .

A condition_value for `DECLARE ... HANDLER` can be any of the following values:

- An SQLSTATE value (a 5-character string literal) or a MariaDB error code (a number). You should not use `SQLSTATE` value '00000' or MariaDB error code 0, because those indicate sucess rather than an error condition. For a list of `SQLSTATE` values and MariaDB error codes, see MariaDB Error Codes.
- A condition name previously specified with `DECLARE ... CONDITION` . It must be in the same stored program. See DECLARE CONDITION.
- `SQLWARNING` is shorthand for the class of SQLSTATE values that begin with '01'.
- `NOT FOUND` is shorthand for the class of `SQLSTATE` values that begin with '02'. This is relevant only the context of cursors and is used to control what happens when a cursor reaches the end of a data set. If no more rows are available, a No Data condition occurs with `SQLSTATE` value 02000. To detect this condition, you can set up a handler for it (or for a `NOT FOUND` condition). An example is shown in Cursor Overview. This condition also occurs for SELECT ... INTO var_list statements that retrieve no rows.
- SQLEXCEPTION is shorthand for the class of SQLSTATE values that do not begin with '00', '01', or '02'.

When an error raises, in some cases it could be handled by multiple `HANDLER`s. For example, there may be an handler for 1050 error, a separate handler for the 42S01 SQLSTATE, and another separate handler for the `SQLEXCEPTION` class: in theory all occurrences of `HANDLER` may catch the 1050 error, but MariaDB chooses the `HANDLER` with the highest precedence. Here are the precedence rules:

- Handlers which refer to an error code have the highest precedence.
- Handlers which refer to a SQLSTATE come next.
- Handlers which refer to an error class have the lowest precedence.

In some cases, a statement could produce multiple errors. If this happens, in some cases multiple handlers could have the highest precedence. In such cases, the choice of the handler is indeterminate.

Note that if an error occurs within a `CONTINUE HANDLER` block, it can be handled by another `HANDLER`. However, a `HANDLER` which is already in the stack (that is, it has been called to handle an error and its execution didn't finish yet) cannot handle new errors—this prevents endless loops. For example, suppose that a stored procedure contains a `CONTINUE HANDLER` for `SQLWARNING` and another `CONTINUE HANDLER` for `NOT FOUND`. At some point, a NOT FOUND error occurs, and the execution enters the `NOT FOUND HANDLER`. But within that handler, a warning occurs, and the execution enters the `SQLWARNING HANDLER`. If another `NOT FOUND` error occurs, it cannot be handled again by the `NOT FOUND HANDLER`, because its execution is not finished.

When a `DECLARE HANDLER` block can handle more than one error condition, it may be useful to know which errors occurred. To do so, you can use the GET DIAGNOSTICS statement.

An error that is handled by a `DECLARE HANDLER` construct can be issued again using the RESIGNAL statement.

Below is an example using `DECLARE HANDLER`:

```
CREATE TABLE test.t (s1 INT, PRIMARY KEY (s1));

DELIMITER //

CREATE PROCEDURE handlerdemo ( )
     BEGIN
       DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x2 = 1;
       SET @x = 1;
       INSERT INTO test.t VALUES (1);
       SET @x = 2;
       INSERT INTO test.t VALUES (1);
       SET @x = 3;
     END;
     //

DELIMITER ;

CALL handlerdemo( );

SELECT @x;
+------+
| @x   |
+------+
|    3 |
+------+
```

1.1.1.6.6 DECLARE Variable

Syntax

```
DECLARE var_name [, var_name] ... [[ROW] TYPE OF]] type [DEFAULT value]
```

Contents

Description

This statement is used to declare local variables within stored programs. To provide a default value for the variable, include a `DEFAULT` clause. The value can be specified as an expression (even subqueries are permitted); it need not be a constant. If the `DEFAULT` clause is missing, the initial value is `NULL`.

Local variables are treated like stored routine parameters with respect to data type and overflow checking. See CREATE PROCEDURE.

Local variables must be declared before `CONDITION`s, CURSORs and `HANDLER`s.

Local variable names are not case sensitive.

The scope of a local variable is within the `BEGIN ... END` block where it is declared. The variable can be referred to in blocks nested within the declaring block, except those blocks that declare a variable with the same name.

TYPE OF / ROW TYPE OF

MariaDB starting with 10.3

`TYPE OF` and `ROW TYPE OF` anchored data types for stored routines were introduced in MariaDB 10.3.

Anchored data types allow a data type to be defined based on another object, such as a table row, rather than specifically set in the declaration. If the anchor object changes, so will the anchored data type. This can lead to routines being easier to maintain, so that if the data type in the table is changed, it will automatically be changed in the routine as well.

Variables declared with `ROW TYPE OF` will have the same features as implicit ROW variables. It is not possible to use `ROW TYPE OF` variables in a LIMIT clause.

The real data type of `TYPE OF` and `ROW TYPE OF table_name` will become known at the very beginning of the stored routine call. ALTER TABLE or DROP TABLE statements performed inside the current routine on the tables that appear in anchors won't affect the data type of the anchored variables, even if the variable is declared after an ALTER TABLE or DROP TABLE statement.

The real data type of a `ROW TYPE OF cursor_name` variable will become known when execution enters into the block where the variable is declared. Data type instantiation will happen only once. In a cursor `ROW TYPE OF` variable that is declared inside a loop, its data type will become known on the very first iteration and won't change on further loop iterations.

The tables referenced in `TYPE OF` and `ROW TYPE OF` declarations will be checked for existence at the beginning of the stored routine call. CREATE PROCEDURE or CREATE FUNCTION will not check the referenced tables for existence.

Examples

`TYPE OF` and `ROW TYPE OF` from MariaDB 10.3:

```
DECLARE tmp TYPE OF t1.a; -- Get the data type from the column {{a}} in the table {{t1}}

DECLARE rec1 ROW TYPE OF t1; -- Get the row data type from the table {{t1}}

DECLARE rec2 ROW TYPE OF cur1; -- Get the row data type from the cursor {{cur1}}
```

1.1.1.6.7 FOR

MariaDB starting with 10.3

FOR loops were introduced in MariaDB 10.3.

Syntax

Integer range FOR loop:

```
[begin_label:]
FOR var_name IN [ REVERSE ] lower_bound .. upper_bound
DO statement_list
END FOR [ end_label ]
```

Explicit cursor FOR loop

```
[begin_label:]
FOR record_name IN cursor_name [ ( cursor_actual_parameter_list)]
DO statement_list
END FOR [ end_label ]
```

Explicit cursor FOR loop (Oracle mode)

```
[begin_label:]
FOR record_name IN cursor_name [ ( cursor_actual_parameter_list)]
LOOP
  statement_list
END LOOP [ end_label ]
```

Implicit cursor FOR loop

```
[begin_label:]
FOR record_name IN ( select_statement )
DO statement_list
END FOR [ end_label ]
```

Contents

Description

FOR loops allow code to be executed a fixed number of times.

In an integer range FOR loop, MariaDB will compare the lower bound and upper bound values, and assign the lower bound value to a counter. If REVERSE is not specified, and the upper bound value is greater than or equal to the counter, the counter will be incremented and the statement will continue, after which the loop is entered again. If the upper bound value is greater than the counter, the loop will be exited.

If REVERSE is specified, the counter is decremented, and the upper bound value needs to be less than or equal for the loop to continue.

Examples

Intger range FOR loop:

```
CREATE TABLE t1 (a INT);

DELIMITER //

FOR i IN 1..3
DO
  INSERT INTO t1 VALUES (i);
END FOR;
//

DELIMITER ;

SELECT * FROM t1;
+------+
| a    |
+------+
|    1 |
|    2 |
|    3 |
+------+
```

REVERSE integer range FOR loop:

```
CREATE OR REPLACE TABLE t1 (a INT);

DELIMITER //
FOR i IN REVERSE 4..12
    DO
    INSERT INTO t1 VALUES (i);
END FOR;
//
Query OK, 9 rows affected (0.422 sec)


DELIMITER ;

SELECT * FROM t1;
+------+
| a    |
+------+
|   12 |
|   11 |
|   10 |
|    9 |
|    8 |
|    7 |
|    6 |
|    5 |
|    4 |
+------+
```

Explicit cursor in Oracle mode:

```
SET sql_mode=ORACLE;

CREATE OR REPLACE TABLE t1 (a INT, b VARCHAR(32));

INSERT INTO t1 VALUES (10,'b0');
INSERT INTO t1 VALUES (11,'b1');
INSERT INTO t1 VALUES (12,'b2');

DELIMITER //

CREATE OR REPLACE PROCEDURE p1(pa INT) AS
  CURSOR cur(va INT) IS
    SELECT a, b FROM t1 WHERE a=va;
BEGIN
  FOR rec IN cur(pa)
  LOOP
    SELECT rec.a, rec.b;
  END LOOP;
END;
//

DELIMITER ;

CALL p1(10);
+-------+-------+
| rec.a | rec.b |
+-------+-------+
|    10 | b0    |
+-------+-------+

CALL p1(11);
+-------+-------+
| rec.a | rec.b |
+-------+-------+
|    11 | b1    |
+-------+-------+

CALL p1(12);
+-------+-------+
| rec.a | rec.b |
+-------+-------+
|    12 | b2    |
+-------+-------+

CALL p1(13);
Query OK, 0 rows affected (0.000 sec)
```

1.1.1.6.8 GOTO

MariaDB starting with 10.3

The GOTO statement was introduced in MariaDB 10.3 for Oracle compatibility.

Syntax

```
GOTO label
```

Contents

Description

The `GOTO` statement causes the code to jump to the specified label, and continue operating from there. It is only accepted when in Oracle mode.

Example

```
SET sql_mode=ORACLE;

DELIMITER //

CREATE OR REPLACE PROCEDURE p1 AS

BEGIN

  SELECT 1;
  GOTO label;
  SELECT 2;
  <<label>>
  SELECT 3;

END;

//

DELIMITER

call p1();
+---+
| 1 |
+---+
| 1 |
+---+
1 row in set (0.000 sec)

+---+
| 3 |
+---+
| 3 |
+---+
1 row in set (0.000 sec)
```

1.1.1.6.9 IF

Syntax

```
IF search_condition THEN statement_list
    [ELSEIF search_condition THEN statement_list] ...
    [ELSE statement_list]
END IF;
```

Description

`IF` implements a basic conditional construct. If the `search_condition` evaluates to true, the corresponding SQL statement list is executed. If no `search_condition` matches, the statement list in the `ELSE` clause is executed. Each statement_list consists of one or more statements.

1.1.1.6.10 ITERATE

Syntax

```
ITERATE label
```

`ITERATE` can appear only within LOOP, REPEAT, and WHILE statements. `ITERATE` means "do the loop again", and uses the statement's label to determine which statements to repeat. The label must be in the same stored program, not in a caller procedure.

If you try to use `ITERATE` with a non-existing label, or if the label is associated to a construct which is not a loop, the following error will be produced:

```
ERROR 1308 (42000): ITERATE with no matching label: <label_name>
```

Below is an example of how `ITERATE` might be used:

```
CREATE PROCEDURE doiterate(p1 INT)
BEGIN
  label1: LOOP
    SET p1 = p1 + 1;
    IF p1 < 10 THEN ITERATE label1; END IF;
    LEAVE label1;
  END LOOP label1;
  SET @x = p1;
END
```

1.1.1.6.11 Labels

Syntax

```
label: <construct>
[label]
```

Labels are MariaDB identifiers which can be used to identify a `BEGIN ... END` construct or a loop. They have a maximum length of 16 characters and can be quoted with backticks (i.e.., `` ` ``).

Labels have a start part and an end part. The start part must precede the portion of code it refers to, must be followed by a colon (`:`) and can be on the same or different line. The end part is optional and adds nothing, but can make the code more readable. If used, the end part must precede the construct's delimiter (`;`). Constructs identified by a label can be nested. Each construct can be identified by only one label.

Labels need not be unique in the stored program they belong to. However, a label for an inner loop cannot be identical to a label for an outer loop. In this case, the following error would be produced:

```
ERROR 1309 (42000): Redefining label <label_name>
```

LEAVE and ITERATE statements can be used to exit or repeat a portion of code identified by a label. They must be in the same Stored Routine, Trigger or Event which contains the target label.

Below is an example using a simple label that is used to exit a LOOP:

```
CREATE PROCEDURE `test_sp`()
BEGIN
   `my_label`:
   LOOP
      SELECT 'looping';
      LEAVE `my_label`;
   END LOOP;
   SELECT 'out of loop';
END;
```

The following label is used to exit a procedure, and has an end part:

```
CREATE PROCEDURE `test_sp`()
`my_label`:
BEGIN
   IF @var = 1 THEN
      LEAVE `my_label`;
   END IF;
   DO something();
END `my_label`;
```

1.1.1.6.12 LEAVE

Syntax

```
LEAVE label
```

This statement is used to exit the flow control construct that has the given label. The label must be in the same stored program, not in a caller procedure. `LEAVE` can be used within BEGIN ... END or loop constructs (LOOP, REPEAT, WHILE). In Stored Procedures, Triggers and Events, LEAVE can refer to the outmost BEGIN ... END construct; in that case, the program exits the procedure. In Stored Functions, RETURN can be used instead.

Note that LEAVE cannot be used to exit a DECLARE HANDLER block.

If you try to LEAVE a non-existing label, or if you try to LEAVE a HANDLER block, the following error will be produced:

```
ERROR 1308 (42000): LEAVE with no matching label: <label_name>
```

The following example uses `LEAVE` to exit the procedure if a condition is true:

```
CREATE PROCEDURE proc(IN p TINYINT)
CONTAINS SQL
`whole_proc`:
BEGIN
   SELECT 1;
   IF p < 1 THEN
      LEAVE `whole_proc`;
   END IF;
   SELECT 2;
END;

CALL proc(0);
+---+
| 1 |
+---+
| 1 |
+---+
```

1.1.1.6.13 LOOP

Syntax

```
[begin_label:] LOOP
    statement_list
END LOOP [end_label]
```

Contents

Description

`LOOP` implements a simple loop construct, enabling repeated execution of the statement list, which consists of one or more statements, each terminated by a semicolon (i.e., `;`) statement delimiter. The statements within the loop are repeated until the loop is exited; usually this is accomplished with a LEAVE statement.

A `LOOP` statement can be labeled. `end_label` cannot be given unless `begin_label` also is present. If both are present, they must be the same.

See Delimiters in the mariadb client for more on delimiter usage in the client.

1.1.1.6.14 REPEAT LOOP

Syntax

```
[begin_label:] REPEAT
    statement_list
UNTIL search_condition
END REPEAT [end_label]
```

The statement list within a `REPEAT` statement is repeated until the search_condition is true. Thus, a `REPEAT` always enters the loop at least once. statement_list consists of one or more statements, each terminated by a semicolon (i.e., `;`) statement delimiter.

A `REPEAT` statement can be labeled. end_label cannot be given unless begin_label also is present. If both are present, they must be the same.

See Delimiters in the mariadb client for more on client delimiter usage.

```
DELIMITER //

CREATE PROCEDURE dorepeat(p1 INT)
  BEGIN
    SET @x = 0;
    REPEAT SET @x = @x + 1; UNTIL @x > p1 END REPEAT;
  END
//

CALL dorepeat(1000)//

SELECT @x//
+------+
| @x   |
+------+
| 1001 |
+------+
```

1.1.1.6.15 RESIGNAL

Syntax

```
RESIGNAL [error_condition]
    [SET error_property
    [, error_property] ...]

error_condition:
    SQLSTATE [VALUE] 'sqlstate_value'
  | condition_name

error_property:
    error_property_name = <error_property_value>

error_property_name:
    CLASS_ORIGIN
  | SUBCLASS_ORIGIN
  | MESSAGE_TEXT
  | MYSQL_ERRNO
  | CONSTRAINT_CATALOG
  | CONSTRAINT_SCHEMA
  | CONSTRAINT_NAME
  | CATALOG_NAME
  | SCHEMA_NAME
  | TABLE_NAME
  | COLUMN_NAME
  | CURSOR_NAME
```

Contents

Description

The syntax of `RESIGNAL` and its semantics are very similar to SIGNAL. This statement can only be used within an error HANDLER. It produces an error, like SIGNAL. `RESIGNAL` clauses are the same as SIGNAL, except that they all are optional, even SQLSTATE. All the properties which are not specified in `RESIGNAL`, will be identical to the properties of the error that was received by the error HANDLER. For a description of the clauses, see diagnostics area.

Note that `RESIGNAL` does not empty the diagnostics area: it just appends another error condition.

`RESIGNAL`, without any clauses, produces an error which is identical to the error that was received by HANDLER.

If used out of a HANDLER construct, RESIGNAL produces the following error:

```
ERROR 1645 (0K000): RESIGNAL when handler not active
```

In MariaDB 5.5, if a HANDLER contained a CALL to another procedure, that procedure could use `RESIGNAL`. Since MariaDB 10.0, trying to do this raises the above error.

For a list of `SQLSTATE` values and MariaDB error codes, see MariaDB Error Codes.

The following procedure tries to query two tables which don't exist, producing a 1146 error in both cases. Those errors will trigger the HANDLER. The first time the error will be ignored and the client will not receive it, but the second time, the error is re-signaled, so the client will receive it.

```
CREATE PROCEDURE test_error( )
BEGIN
   DECLARE CONTINUE HANDLER
      FOR 1146
   BEGIN
   IF @hide_errors IS FALSE THEN
      RESIGNAL;
   END IF;
   END;
   SET @hide_errors = TRUE;
   SELECT 'Next error will be ignored' AS msg;
   SELECT `c` FROM `temptab_one`;
   SELECT 'Next error won''t be ignored' AS msg;
   SET @hide_errors = FALSE;
   SELECT `c` FROM `temptab_two`;
END;

CALL test_error( );

+----------------------------+
| msg                        |
+----------------------------+
| Next error will be ignored |
+----------------------------+

+-----------------------------+
| msg                         |
+-----------------------------+
| Next error won't be ignored |
+-----------------------------+

ERROR 1146 (42S02): Table 'test.temptab_two' doesn't exist
```

The following procedure re-signals an error, modifying only the error message to clarify the cause of the problem.

```
CREATE PROCEDURE test_error()
BEGIN
   DECLARE CONTINUE HANDLER
   FOR 1146
   BEGIN
      RESIGNAL SET
      MESSAGE_TEXT = '`temptab` does not exist';
   END;
   SELECT `c` FROM `temptab`;
END;

CALL test_error( );
ERROR 1146 (42S02): `temptab` does not exist
```

As explained above, this works on MariaDB 5.5, but produces a 1645 error since 10.0.

```
CREATE PROCEDURE handle_error()
BEGIN
  RESIGNAL;
END;
CREATE PROCEDURE p()
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION CALL p();
  SIGNAL SQLSTATE '45000';
END;
```

1.1.1.6.16 RETURN

Syntax

```
RETURN expr
```

The `RETURN` statement terminates execution of a stored function and returns the value *`expr`* to the function caller. There must be at least one `RETURN` statement in a stored function. If the function has multiple exit points, all exit points must have a `RETURN`.

This statement is not used in stored procedures, triggers, or events. LEAVE can be used instead.

The following example shows that `RETURN` can return the result of a scalar subquery:

```
CREATE FUNCTION users_count() RETURNS BOOL
   READS SQL DATA
BEGIN
   RETURN (SELECT COUNT(DISTINCT User) FROM mysql.user);
END;
```

1.1.1.6.17 SELECT INTO

Syntax

```
SELECT col_name [, col_name] ...
    INTO var_name [, var_name] ...
    table_expr
```

Contents

Description

SELECT ... INTO enables selected columns to be stored directly into variables. No resultset is produced. The query should return a single row. If the query returns no rows, a warning with error code 1329 occurs (No data), and the variable values remain unchanged. If the query returns multiple rows, error 1172 occurs (Result consisted of more than one row). If it is possible that the statement may retrieve multiple rows, you can use `LIMIT 1` to limit the result set to a single row.

The INTO clause can also be specified at the end of the statement.

In the context of such statements that occur as part of events executed by the Event Scheduler, diagnostics messages (not only errors, but also warnings) are written to the error log, and, on Windows, to the application event log.

This statement can be used with both local variables and user-defined variables.

For the complete syntax, see SELECT.

Another way to set a variable's value is the SET statement.

`SELECT ... INTO` results are not stored in the query cache even if `SQL_CACHE` is specified.

Examples

```
SELECT id, data INTO @x,@y
FROM test.t1 LIMIT 1;
SELECT * from t1 where t1.a=@x and t1.b=@y
```

If you want to use this construct with `UNION` you have to use the syntax:

```
SELECT  * INTO @x FROM (SELECT t1.a FROM t1 UNION SELECT t2.a FROM t2) dt;
```

1.1.1.2.7.10 SET Variable

1.1.1.6.19 SIGNAL

Syntax

```
SIGNAL error_condition
    [SET error_property
    [, error_property] ...]

error_condition:
    SQLSTATE [VALUE] 'sqlstate_value'
  | condition_name

error_property:
    error_property_name = <error_property_value>

error_property_name:
    CLASS_ORIGIN
  | SUBCLASS_ORIGIN
  | MESSAGE_TEXT
  | MYSQL_ERRNO
  | CONSTRAINT_CATALOG
  | CONSTRAINT_SCHEMA
  | CONSTRAINT_NAME
  | CATALOG_NAME
  | SCHEMA_NAME
  | TABLE_NAME
  | COLUMN_NAME
  | CURSOR_NAME
```

Contents

`SIGNAL` empties the diagnostics area and produces a custom error. This statement can be used anywhere, but is

generally useful when used inside a stored program. When the error is produced, it can be caught by a HANDLER. If not, the current stored program, or the current statement, will terminate with the specified error.

Sometimes an error HANDLER just needs to SIGNAL the same error it received, optionally with some changes. Usually the RESIGNAL statement is the most convenient way to do this.

`error_condition` can be an SQLSTATE value or a named error condition defined via DECLARE CONDITION. SQLSTATE must be a constant string consisting of five characters. These codes are standard to ODBC and ANSI SQL. For customized errors, the recommended SQLSTATE is '45000'. For a list of SQLSTATE values used by MariaDB, see the MariaDB Error Codes page. The SQLSTATE can be read via the API method `mysql_sqlstate( )`.

To specify error properties user-defined variables and local variables can be used, as well as character set conversions (but you can't set a collation).

The error properties, their type and their default values are explained in the diagnostics area page.

Errors

If the SQLSTATE is not valid, the following error like this will be produced:

```
ERROR 1407 (42000): Bad SQLSTATE: '123456'
```

If a property is specified more than once, an error like this will be produced:

```
ERROR 1641 (42000): Duplicate condition information item 'MESSAGE_TEXT'
```

If you specify a condition name which is not declared, an error like this will be produced:

```
ERROR 1319 (42000): Undefined CONDITION: cond_name
```

If MYSQL_ERRNO is out of range, you will get an error like this:

```
ERROR 1231 (42000): Variable 'MYSQL_ERRNO' can't be set to the value of '0'
```

Examples

Here's what happens if SIGNAL is used in the client to generate errors:

```
SIGNAL SQLSTATE '01000';
Query OK, 0 rows affected, 1 warning (0.00 sec)

SHOW WARNINGS;

+---------+------+----------------------------------------+
| Level   | Code | Message                                |
+---------+------+----------------------------------------+
| Warning | 1642 | Unhandled user-defined warning condition |
+---------+------+----------------------------------------+
1 row in set (0.06 sec)

SIGNAL SQLSTATE '02000';
ERROR 1643 (02000): Unhandled user-defined not found condition
```

How to specify MYSQL_ERRNO and MESSAGE_TEXT properties:

```
SIGNAL SQLSTATE '45000' SET MYSQL_ERRNO=30001, MESSAGE_TEXT='H
ello, world!';

ERROR 30001 (45000): Hello, world!
```

The following code shows how to use user variables, local variables and character set conversion with SIGNAL:

```
CREATE PROCEDURE test_error(x INT)
BEGIN
   DECLARE errno SMALLINT UNSIGNED DEFAULT 31001;
   SET @errmsg = 'Hello, world!';
   IF x = 1 THEN
      SIGNAL SQLSTATE '45000' SET
      MYSQL_ERRNO = errno,
      MESSAGE_TEXT = @errmsg;
   ELSE
      SIGNAL SQLSTATE '45000' SET
      MYSQL_ERRNO = errno,
      MESSAGE_TEXT = _utf8'Hello, world!';
   END IF;
END;
```

How to use named error conditions:

```
CREATE PROCEDURE test_error(n INT)
BEGIN
   DECLARE `too_big` CONDITION FOR SQLSTATE '45000';
   IF n > 10 THEN
      SIGNAL `too_big`;
   END IF;
END;
```

In this example, we'll define a HANDLER for an error code. When the error occurs, we SIGNAL a more informative error which makes sense for our procedure:

```
CREATE PROCEDURE test_error()
BEGIN
   DECLARE EXIT HANDLER
   FOR 1146
   BEGIN
      SIGNAL SQLSTATE '45000' SET
      MESSAGE_TEXT = 'Temporary tables not found; did you call init() procedure?';
   END;
   -- this will produce a 1146 error
   SELECT `c` FROM `temptab`;
END;
```

1.1.1.6.20 WHILE

Syntax

```
[begin_label:] WHILE search_condition DO
    statement_list
END WHILE [end_label]
```

Description

The statement list within a `WHILE` statement is repeated as long as the `search_condition` is true. statement_list consists of one or more statements. If the loop must be executed at least once, `REPEAT ... LOOP` can be used instead.

A `WHILE` statement can be labeled. end_label cannot be given unless begin_label also is present. If both are present, they must be the same.

Examples

```
CREATE PROCEDURE dowhile()
BEGIN
  DECLARE v1 INT DEFAULT 5;

  WHILE v1 > 0 DO
    ...
    SET v1 = v1 - 1;
  END WHILE;
END
```

1.1.1.6.21 Cursors

A cursor is a structure that allows you to go over records sequentially, and perform processing based on the result.

Cursor Overview

Structure for traversing and processing results sequentially.

DECLARE CURSOR

Declares a cursor which can be used inside stored programs.

OPEN

Open a previously declared cursor.

FETCH

Fetch a row from a cursor.

CLOSE

Close a previously opened cursor.

There are 1 related questions.

1.1.1.6.21.1 Cursor Overview

Contents

Description

A cursor is a structure that allows you to go over records sequentially, and perform processing based on the result.

MariaDB permits cursors inside stored programs, and MariaDB cursors are non-scrollable, read-only and asensitive.

- Non-scrollable means that the rows can only be fetched in the order specified by the SELECT statement. Rows cannot be skipped, you cannot jump to a specific row, and you cannot fetch rows in reverse order.
- Read-only means that data cannot be updated through the cursor.
- Asensitive means that the cursor points to the actual underlying data. This kind of cursor is quicker than the alternative, an insensitive cursor, as no data is copied to a temporary table. However, changes to the data being used by the cursor will affect the cursor data.

Cursors are created with a DECLARE CURSOR statement and opened with an OPEN statement. Rows are read with a FETCH statement before the cursor is finally closed with a CLOSE statement.

When FETCH is issued and there are no more rows to extract, the following error is produced:

```
ERROR 1329 (02000): No data - zero rows fetched, selected, or processed
```

To avoid problems, a DECLARE HANDLER statement is generally used. The HANDLER should handler the 1329 error, or the '02000' SQLSTATE, or the NOT FOUND error class.

Only SELECT statements are allowed for cursors, and they cannot be contained in a variable - so, they cannot be composed dynamically. However, it is possible to SELECT from a view. Since the CREATE VIEW statement can be executed as a prepared statement, it is possible to dynamically create the view that is queried by the cursor.

From MariaDB 10.3.0, cursors can have parameters. Cursor parameters can appear in any part of the DECLARE CURSOR select_statement where a stored procedure variable is allowed (select list, WHERE, HAVING, LIMIT etc). See DECLARE CURSOR and OPEN for syntax, and below for an example:

Examples

```
CREATE TABLE c1(i INT);

CREATE TABLE c2(i INT);

CREATE TABLE c3(i INT);

DELIMITER //

CREATE PROCEDURE p1()
BEGIN
  DECLARE done INT DEFAULT FALSE;
  DECLARE x, y INT;
  DECLARE cur1 CURSOR FOR SELECT i FROM test.c1;
  DECLARE cur2 CURSOR FOR SELECT i FROM test.c2;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

  OPEN cur1;
  OPEN cur2;

  read_loop: LOOP
    FETCH cur1 INTO x;
    FETCH cur2 INTO y;
    IF done THEN
      LEAVE read_loop;
    END IF;
    IF x < y THEN
      INSERT INTO test.c3 VALUES (x);
    ELSE
      INSERT INTO test.c3 VALUES (y);
    END IF;
  END LOOP;

  CLOSE cur1;
  CLOSE cur2;
END; //

DELIMITER ;

INSERT INTO c1 VALUES(5),(50),(500);

INSERT INTO c2 VALUES(10),(20),(30);

CALL p1;

SELECT * FROM c3;
+------+
| i    |
+------+
|    5 |
|   20 |
|   30 |
+------+
```

From MariaDB 10.3.0

```
DROP PROCEDURE IF EXISTS p1;
DROP TABLE IF EXISTS t1;
CREATE TABLE t1 (a INT, b VARCHAR(10));

INSERT INTO t1 VALUES (1,'old'),(2,'old'),(3,'old'),(4,'old'),(5,'old');

DELIMITER //

CREATE PROCEDURE p1(min INT,max INT)
BEGIN
  DECLARE done INT DEFAULT FALSE;
  DECLARE va INT;
  DECLARE cur CURSOR(pmin INT, pmax INT) FOR SELECT a FROM t1 WHERE a BETWEEN pmin AND pmax;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done=TRUE;
  OPEN cur(min,max);
  read_loop: LOOP
    FETCH cur INTO va;
    IF done THEN
      LEAVE read_loop;
    END IF;
    INSERT INTO t1 VALUES (va,'new');
  END LOOP;
  CLOSE cur;
END;
//

DELIMITER ;

CALL p1(2,4);

SELECT * FROM t1;
+------+------+
| a    | b    |
+------+------+
|    1 | old  |
|    2 | old  |
|    3 | old  |
|    4 | old  |
|    5 | old  |
|    2 | new  |
|    3 | new  |
|    4 | new  |
+------+------+
```

1.1.1.6.21.2 DECLARE CURSOR

Syntax

<= MariaDB 10.2

```
DECLARE cursor_name CURSOR FOR select_statement
```

From MariaDB 10.3

```
DECLARE cursor_name CURSOR [(cursor_formal_parameter[,...])] FOR select_statement

cursor_formal_parameter:
    name type [collate clause]
```

From MariaDB 10.8

```
DECLARE cursor_name CURSOR [(cursor_formal_parameter[,...])] FOR select_statement

cursor_formal_parameter:
    [IN] name type [collate clause]
```

Contents
1. Syntax
2. Description
   1. Parameters
   2. IN

Description

This statement declares a cursor. Multiple cursors may be declared in a stored program, but each cursor in a given block must have a unique name.

`select_statement` is not executed until the OPEN statement is executed. It is important to remember this if the query produces an error, or calls functions which have side effects.

A `SELECT` associated to a cursor can use variables, but the query itself cannot be a variable, and cannot be dynamically composed. The `SELECT` statement cannot have an `INTO` clause.

Cursors must be declared before HANDLERs, but after local variables and CONDITIONs.

Parameters

MariaDB starting with 10.3.0

From MariaDB 10.3.0, cursors can have parameters. This is a non-standard SQL extension. Cursor parameters can appear in any part of the DECLARE CURSOR select_statement where a stored procedure variable is allowed (select list, WHERE, HAVING, LIMIT etc).

IN

MariaDB starting with 10.8.0

From MariaDB 10.8.0 preview release, the `IN` qualifier is supported in the `cursor_format_parameter` part of the syntax.

See Cursor Overview for an example.

1.1.1.6.21.3 OPEN

Syntax

<= MariaDB 10.2

```
OPEN cursor_name
```

From MariaDB 10.3

```
OPEN cursor_name [expression[,...]];
```

Contents
1. Syntax
2. Description

Description

This statement opens a cursor which was previously declared with DECLARE CURSOR.

The query associated to the DECLARE CURSOR is executed when OPEN is executed. It is important to remember this if the query produces an error, or calls functions which have side effects.

This is necessary in order to FETCH rows from a cursor.

See Cursor Overview for an example.

1.1.1.6.21.4 FETCH

Syntax

```
FETCH cursor_name INTO var_name [, var_name] ...
```

Contents

Description

This statement fetches the next row (if a row exists) using the specified open cursor, and advances the cursor pointer.

`var_name` can be a local variable, but *not* a user-defined variable.

If no more rows are available, a No Data condition occurs with `SQLSTATE` value `02000`. To detect this condition, you can set up a handler for it (or for a `NOT FOUND` condition).

See Cursor Overview for an example.

1.1.1.6.21.5 CLOSE

Syntax

```
CLOSE cursor_name
```

Contents

Description

This statement closes a previously opened cursor. The cursor must have been previously opened or else an error occurs.

If not closed explicitly, a cursor is closed at the end of the compound statement in which it was declared.

See Cursor Overview for an example.

1.1.1.7 Stored Routine Statements

CALL

Invokes a stored procedure.

DO

Executes expressions without returning results.

There are 1 related questions.

1.1.1.7.1 CALL

Syntax

```
CALL sp_name([parameter[,...]])
CALL sp_name[()]
```

Description

The `CALL` statement invokes a stored procedure that was defined previously with CREATE PROCEDURE.

Stored procedure names can be specified as `database_name.procedure_name`. Procedure names and database names can be quoted with backticks (). This is necessary if they are reserved words, or contain special characters. See identifier qualifiers for details.

`CALL p()` and `CALL p` are equivalent.

If parentheses are used, any number of spaces, tab characters and newline characters are allowed between the procedure's name and the open parenthesis.

`CALL` can pass back values to its caller using parameters that are declared as `OUT` or `INOUT` parameters. If no value is assigned to an `OUT` parameter, `NULL` is assigned (and its former value is lost). To pass such values from another stored program you can use user-defined variables, local variables or routine's parameters; in other contexts, you can only use user-defined variables.

`CALL` can also be executed as a prepared statement. Placeholders can be used for `IN` parameters in all versions of MariaDB; for `OUT` and `INOUT` parameters, placeholders can be used since MariaDB 5.5.

When the procedure returns, a client program can also obtain the number of rows affected for the final statement executed within the routine: At the SQL level, call the `ROW_COUNT()` function; from the C API, call the `mysql_affected_rows()` function.

If the `CLIENT_MULTI_RESULTS` API flag is set, `CALL` can return any number of resultsets and the called stored procedure can execute prepared statements. If it is not set, at most one resultset can be returned and prepared statements cannot be used within procedures.

1.1.1.7.2 DO

Syntax

```
DO expr [, expr] ...
```

Description

`DO` executes the expressions but does not return any results. In most respects, `DO` is shorthand for `SELECT expr, ...,` but has the advantage that it is slightly faster when you do not care about the result.

`DO` is useful primarily with functions that have side effects, such as `RELEASE_LOCK()`.

1.1.1.2.1 Table Statements

1.1.1.9 Transactions

"An SQL-transaction (transaction) is a sequence of executions of SQL-statements that is atomic with respect to recovery. That is to say: either the execution result is completely successful, or it has no effect on any SQL-schemas or SQL-data."

— The SQL Standard

The InnoDB storage engine supports ACID-compliant transactions.

Transaction Articles

START TRANSACTION

Basic transaction control statements.

COMMIT

Ends a transaction, making changes visible to subsequent transactions

ROLLBACK

Cancel current transaction and the changes to data

SET TRANSACTION
Sets the transaction isolation level.

LOCK TABLES
Explicitly lock tables.

SAVEPOINT
SAVEPOINT for a ROLLBACK.

Metadata Locking
A lock which protects each transaction from external DDL statements.

SQL statements That Cause an Implicit Commit
List of statements which implicitly commit the current transaction

Transaction Timeouts
Timing out idle transactions

UNLOCK TABLES
Explicitly releases any table locks held by the current session.

WAIT and NOWAIT
Extended syntax so that it is possible to set lock wait timeout for certain statements.

XA Transactions
Transactions designed to allow distributed transactions.

READ COMMITTED
Each consistent read, even within the same transaction, sets and reads its own fresh snapshot.

READ UNCOMMITTED
SELECT statements are performed in a non-locking fashion, but a possible ea...

REPEATABLE READ
All consistent reads within the same transaction read the snapshot established by the first read.

SERIALIZABLE
Similar to REPEATABLE READ, with SELECT ... LOCK IN SHARE MODE if autocommit is disabled.

1.1.1.9.1 START TRANSACTION

Syntax

```
START TRANSACTION [transaction_property [, transaction_property] ...] | BEGIN [WORK]
COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
SET autocommit = {0 | 1}

transaction_property:
    WITH CONSISTENT SNAPSHOT
  | READ WRITE
  | READ ONLY
```

Contents

Description

The `START TRANSACTION` or `BEGIN` statement begins a new transaction. COMMIT commits the current transaction, making its changes permanent. ROLLBACK rolls back the current transaction, canceling its changes. The SET autocommit statement disables or enables the default autocommit mode for the current session.

START TRANSACTION and SET autocommit = 1 implicitly commit the current transaction, if any.

The optional `WORK` keyword is supported for `COMMIT` and `ROLLBACK`, as are the `CHAIN` and `RELEASE` clauses. `CHAIN` and `RELEASE` can be used for additional control over transaction completion. The value of the completion_type system variable determines the default completion behavior.

The `AND CHAIN` clause causes a new transaction to begin as soon as the current one ends, and the new transaction has the same isolation level as the just-terminated transaction. The `RELEASE` clause causes the server to disconnect the current client session after terminating the current transaction. Including the `NO` keyword suppresses `CHAIN` or `RELEASE` completion, which can be useful if the completion_type system variable is set to cause chaining or release completion by default.

Access Mode

The access mode specifies whether the transaction is allowed to write data or not. By default, transactions are in `READ WRITE` mode (see the tx_read_only system variable). `READ ONLY` mode allows the storage engine to apply optimizations that cannot be used for transactions which write data. Note that unlike the global `read_only` mode, `READ_ONLY ADMIN` (and `SUPER` before MariaDB 10.11.0) privilege doesn't allow writes and DDL statements on temporary tables are not allowed either.

It is not permitted to specify both `READ WRITE` and `READ ONLY` in the same statement.

`READ WRITE` and `READ ONLY` can also be specified in the `SET TRANSACTION` statement, in which case the specified mode is valid for all sessions, or for all subsequent transaction used by the current session.

autocommit

By default, MariaDB runs with autocommit mode enabled. This means that as soon as you execute a statement that updates (modifies) a table, MariaDB stores the update on disk to make it permanent. To disable autocommit mode, use the following statement:

```
SET autocommit=0;
```

After disabling autocommit mode by setting the autocommit variable to zero, changes to transaction-safe tables (such as those for InnoDB or `NDBCLUSTER`) are not made permanent immediately. You must use `COMMIT` to store your changes to disk or ROLLBACK to ignore the changes.

To disable autocommit mode for a single series of statements, use the `START TRANSACTION` statement.

DDL Statements

DDL statements (`CREATE`, `ALTER`, `DROP`) and administrative statements (`FLUSH`, `RESET`, `OPTIMIZE`, `ANALYZE`, `CHECK`, `REPAIR`, `CACHE INDEX`), transaction management statements (`BEGIN`, `START TRANSACTION`) and `LOAD DATA INFILE`, cause an implicit `COMMIT` and start a new transaction. An exception to this rule are the DDL that operate on temporary tables: you can `CREATE`, `ALTER` and `DROP` them without causing any `COMMIT`, but those actions cannot be rolled back. This means that if you call `ROLLBACK`, the temporary tables you created in the transaction will remain, while the rest of the transaction will be rolled back.

Transactions cannot be used in Stored Functions or Triggers. In Stored Procedures and Events BEGIN is not allowed, so you should use START TRANSACTION instead.

A transaction acquires a metadata lock on every table it accesses to prevent other connections from altering their structure. The lock is released at the end of the transaction. This happens even with non-transactional storage engines (like MEMORY or CONNECT), so it makes sense to use transactions with non-transactional tables.

in_transaction

The in_transaction system variable is a session-only, read-only variable that returns `1` inside a transaction, and `0` if not in a transaction.

WITH CONSISTENT SNAPSHOT

The `WITH CONSISTENT SNAPSHOT` option starts a consistent read for storage engines such as InnoDB that can do so, the same as if a START TRANSACTION followed by a SELECT from any InnoDB table was issued.

See Enhancements for START TRANSACTION WITH CONSISTENT SNAPSHOT.

Examples

```
START TRANSACTION;
SELECT @A:=SUM(salary) FROM table1 WHERE type=1;
UPDATE table2 SET summary=@A WHERE type=1;
COMMIT;
```

1.1.1.9.2 COMMIT

The `COMMIT` statement ends a transaction, saving any changes to the data so that they become visible to subsequent transactions. Also, unlocks metadata changed by current transaction. If autocommit is set to 1, an implicit commit is performed after each statement. Otherwise, all transactions which don't end with an explicit `COMMIT` are implicitly rollbacked and the changes are lost. The `ROLLBACK` statement can be used to do this explicitly.

The required syntax for the `COMMIT` statement is as follows:

```
COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
```

`COMMIT` is the more important transaction terminator, as well as the more interesting one. The basic form of the `COMMIT` statement is simply the keyword `COMMIT` (the keyword `WORK` is simply noise and can be omitted without changing the effect).

The optional `AND CHAIN` clause is a convenience for initiating a new transaction as soon as the old transaction terminates. If `AND CHAIN` is specified, then there is effectively nothing between the old and new transactions, although they remain separate. The characteristics of the new transaction will be the same as the characteristics of the old one — that is, the new transaction will have the same access mode, isolation level and diagnostics area size (we'll discuss all of these shortly) as the transaction just terminated.

`RELEASE` tells the server to disconnect the client immediately after the current transaction.

There are `NO RELEASE` and `AND NO CHAIN` options. By default, commits do not `RELEASE` or `CHAIN`, but it's possible to change this default behavior with the completion_type server system variable. In this case, the `AND NO CHAIN` and `NO RELEASE` options override the server default.

1.1.1.9.3 ROLLBACK

The `ROLLBACK` statement rolls back (ends) a transaction, destroying any changes to SQL-data so that they never become visible to subsequent transactions. The required syntax for the `ROLLBACK` statement is as follows.

```
ROLLBACK [ WORK ] [ AND [ NO ] CHAIN ]
[ TO [ SAVEPOINT ] {<savepoint name> | <simple target specification>} ]
```

The `ROLLBACK` statement will either end a transaction, destroying all data changes that happened during any of the transaction, or it will just destroy any data changes that happened since you established a savepoint. The basic form of the `ROLLBACK` statement is just the keyword `ROLLBACK` (the keyword `WORK` is simply noise and can be omitted without changing the effect).

The optional `AND CHAIN` clause is a convenience for initiating a new transaction as soon as the old transaction terminates. If `AND CHAIN` is specified, then there is effectively nothing between the old and new transactions, although they remain separate. The characteristics of the new transaction will be the same as the characteristics of the old one — that is, the new transaction will have the same access mode, isolation level and diagnostics area size (we'll discuss all of these shortly) as the transaction just terminated. The `AND NO CHAIN` option just tells your DBMS to end the transaction — that is, these four SQL statements are equivalent:

```
ROLLBACK;
ROLLBACK WORK;
ROLLBACK AND NO CHAIN;
ROLLBACK WORK AND NO CHAIN;
```

All of them end a transaction without saving any transaction characteristics. The only other options, the equivalent statements:

```
ROLLBACK AND CHAIN;
ROLLBACK WORK AND CHAIN;
```

both tell your DBMS to end a transaction, but to save that transaction's characteristics for the next transaction.

`ROLLBACK` is much simpler than `COMMIT` : it may involve no more than a few deletions (of Cursors, locks, prepared SQL statements and log-file entries). It's usually assumed that `ROLLBACK` can't fail, although such a thing is conceivable (for example, an encompassing transaction might reject an attempt to `ROLLBACK` because it's lining up for a `COMMIT`).

`ROLLBACK` cancels all effects of a transaction. It does not cancel effects on objects outside the DBMS's control (for example the values in host program variables or the settings made by some SQL/CLI function calls). But in general, it is a convenient statement for those situations when you say "oops, this isn't working" or when you simply don't care whether your temporary work becomes permanent or not.

Here is a moot question. If all you've been doing is `SELECT` s, so that there have been no data changes, should you end the transaction with `ROLLBACK` or `COMMIT` ? It shouldn't really matter because both `ROLLBACK` and `COMMIT` do the same transaction-terminating job. However, the popular conception is that `ROLLBACK` implies failure, so after a successful series of `SELECT` statements the convention is to end the transaction with `COMMIT` rather than `ROLLBACK` .

MariaDB (and most other DBMSs) supports rollback of SQL-data change statements, but not of SQL-Schema statements. This means that if you use any of `CREATE` , `ALTER` , `DROP` , `GRANT` , `REVOKE` , you are implicitly committing at execution time.

```
INSERT INTO Table_2 VALUES(5);
DROP TABLE Table_3 CASCADE;
ROLLBACK;
```

The result will be that both the `INSERT` and the `DROP` will go through as separate transactions so the `ROLLBACK` will have no effect.

1.1.1.9.4 SET TRANSACTION

Syntax

```
SET [GLOBAL | SESSION] TRANSACTION
    transaction_property [, transaction_property] ...

transaction_property:
    ISOLATION LEVEL level
  | READ WRITE
  | READ ONLY

level:
     REPEATABLE READ
   | READ COMMITTED
   | READ UNCOMMITTED
   | SERIALIZABLE
```

Contents

Description

This statement sets the transaction isolation level or the transaction access mode globally, for the current session, or for the next transaction:

- With the `GLOBAL` keyword, the statement sets the default transaction level globally for all subsequent sessions. Existing sessions are unaffected.
- With the `SESSION` keyword, the statement sets the default transaction level for all subsequent transactions performed within the current session.
- Without any `SESSION` or `GLOBAL` keyword, the statement sets the isolation level for only the next (not started) transaction performed within the current session. After that it reverts to using the session value.

A change to the global default isolation level requires the `SUPER` privilege. Any session is free to change its session isolation level (even in the middle of a transaction), or the isolation level for its next transaction.

Isolation Level

To set the global default isolation level at server startup, use the --transaction-isolation=level option on the command line or in an option file. Values of level for this option use dashes rather than spaces, so the allowable values are READ_UNCOMMITTED, READ-COMMITTED, REPEATABLE-READ, or SERIALIZABLE. For example, to set the default isolation level to `REPEATABLE READ`, use these lines in the [mariadb] section of an option file:

```
[mariadb]
transaction-isolation = REPEATABLE-READ
```

To determine the global and session transaction isolation levels at runtime, check the value of the tx_isolation system variable (note that the variable has been renamed transaction_isolation from MariaDB 11.1.1, to match the option, and the old name deprecated).

```
SELECT @@GLOBAL.tx_isolation, @@tx_isolation;
```

From MariaDB 11.1.1:

```
SELECT @@GLOBAL.transaction_isolation, @@transaction_isolation;
```

InnoDB supports each of the translation isolation levels described here using different locking strategies. The default level is `REPEATABLE READ`. For additional information about InnoDB record-level locks and how it uses them to execute various types of statements, see InnoDB Lock Modes, and http://dev.mysql.com/doc/refman/en/innodb-locks-set.html.

Isolation Levels

The following sections describe how MariaDB supports the different transaction levels.

READ UNCOMMITTED

`SELECT` statements are performed in a non-locking fashion, but a possible earlier version of a row might be used. Thus, using this isolation level, such reads are not consistent. This is also called a "dirty read." Otherwise, this isolation level works like `READ COMMITTED`.

READ COMMITTED

A somewhat Oracle-like isolation level with respect to consistent (non-locking) reads: Each consistent read, even within the same transaction, sets and reads its own fresh snapshot. See http://dev.mysql.com/doc/refman/en/innodb-consistent-

read.html.

For locking reads (`SELECT` with `FOR UPDATE` or `LOCK IN SHARE MODE`), InnoDB locks only index records, not the gaps before them, and thus allows the free insertion of new records next to locked records. For `UPDATE` and `DELETE` statements, locking depends on whether the statement uses a unique index with a unique search condition (such as `WHERE id = 100`), or a range-type search condition (such as `WHERE id > 100`). For a unique index with a unique search condition, InnoDB locks only the index record found, not the gap before it. For range-type searches, InnoDB locks the index range scanned, using gap locks or next-key (gap plus index-record) locks to block insertions by other sessions into the gaps covered by the range. This is necessary because "phantom rows" must be blocked for MariaDB replication and recovery to work.

Note: If the `READ COMMITTED` isolation level is used or the innodb_locks_unsafe_for_binlog system variable is enabled, there is no InnoDB gap locking except for foreign-key constraint checking and duplicate-key checking. Also, record locks for non-matching rows are released after MariaDB has evaluated the `WHERE` condition.If you use `READ COMMITTED` or enable innodb_locks_unsafe_for_binlog, you must use row-based binary logging.

REPEATABLE READ

This is the default isolation level for InnoDB. For consistent reads, there is an important difference from the `READ COMMITTED` isolation level: All consistent reads within the same transaction read the snapshot established by the first read. This convention means that if you issue several plain (non-locking) `SELECT` statements within the same transaction, these `SELECT` statements are consistent also with respect to each other. See http://dev.mysql.com/doc/refman/en/innodb-consistent-read.html.

For locking reads (SELECT with FOR UPDATE or LOCK IN SHARE MODE), UPDATE, and DELETE statements, locking depends on whether the statement uses a unique index with a unique search condition, or a range-type search condition. For a unique index with a unique search condition, InnoDB locks only the index record found, not the gap before it. For other search conditions, InnoDB locks the index range scanned, using gap locks or next-key (gap plus index-record) locks to block insertions by other sessions into the gaps covered by the range.

This is the minimum isolation level for non-distributed XA transactions.

SERIALIZABLE

This level is like REPEATABLE READ, but InnoDB implicitly converts all plain SELECT statements to SELECT ... LOCK IN SHARE MODE if autocommit is disabled. If autocommit is enabled, the SELECT is its own transaction. It therefore is known to be read only and can be serialized if performed as a consistent (non-locking) read and need not block for other transactions. (This means that to force a plain SELECT to block if other transactions have modified the selected rows, you should disable autocommit.)

Distributed XA transactions should always use this isolation level.

Access Mode

The access mode specifies whether the transaction is allowed to write data or not. By default, transactions are in `READ WRITE` mode (see the tx_read_only system variable). `READ ONLY` mode allows the storage engine to apply optimizations that cannot be used for transactions which write data. Note that unlike the global read_only mode, READ_ONLY ADMIN (and SUPER before MariaDB 10.11.0) privilege doesn't allow writes and DDL statements on temporary tables are not allowed either.

It is not permitted to specify both `READ WRITE` and `READ ONLY` in the same statement.

`READ WRITE` and `READ ONLY` can also be specified in the START TRANSACTION statement, in which case the specified mode is only valid for one transaction.

Examples

```
SET GLOBAL TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

Attempting to set the isolation level within an existing transaction without specifying `GLOBAL` or `SESSION` .

```
START TRANSACTION;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
ERROR 1568 (25001): Transaction characteristics can't be changed while a transaction is in progre
```

1.1.1.9.5 LOCK TABLES

Syntax

```
LOCK TABLE[S]
    tbl_name [[AS] alias] lock_type
    [, tbl_name [[AS] alias] lock_type] ...
    [WAIT n|NOWAIT]

lock_type:
    READ [LOCAL]
  | [LOW_PRIORITY] WRITE
  | WRITE CONCURRENT

UNLOCK TABLES
```

Contents

Description

The *lock_type* can be one of:

Option	Description
READ	Read lock, no writes allowed
READ LOCAL	Read lock, but allow concurrent inserts
WRITE	Exclusive write lock. No other connections can read or write to this table
LOW_PRIORITY WRITE	Exclusive write lock, but allow new read locks on the table until we get the write lock.
WRITE CONCURRENT	Exclusive write lock, but allow READ LOCAL locks to the table.

MariaDB enables client sessions to acquire table locks explicitly for the purpose of cooperating with other sessions for access to tables, or to prevent other sessions from modifying tables during periods when a session requires exclusive access to them. A session can acquire or release locks only for itself. One session cannot acquire locks for another session or release locks held by another session.

Locks may be used to emulate transactions or to get more speed when updating tables.

`LOCK TABLES` explicitly acquires table locks for the current client session. Table locks can be acquired for base tables or views. To use `LOCK TABLES`, you must have the `LOCK TABLES` privilege, and the `SELECT` privilege for each object to be locked. See `GRANT`

For view locking, `LOCK TABLES` adds all base tables used in the view to the set of tables to be locked and locks them automatically. If you lock a table explicitly with `LOCK TABLES`, any tables used in triggers are also locked implicitly, as described in Triggers and Implicit Locks.

UNLOCK TABLES explicitly releases any table locks held by the current session.

MariaDB starting with 10.3.0

WAIT/NOWAIT

Set the lock wait timeout. See WAIT and NOWAIT.

Limitations

- `LOCK TABLES` doesn't work when using Galera cluster. You may experience crashes or locks when used with Galera.
- `LOCK TABLES` works on XtraDB/InnoDB tables only if the innodb_table_locks system variable is set to 1 (the default) and autocommit is set to 0 (1 is default). Please note that no error message will be returned on LOCK TABLES with

innodb_table_locks = 0.

- `LOCK TABLES` implicitly commits the active transaction, if any. Also, starting a transaction always releases all table locks acquired with LOCK TABLES. This means that there is no way to have table locks and an active transaction at the same time. The only exceptions are the transactions in autocommit mode. To preserve the data integrity between transactional and non-transactional tables, the GET_LOCK() function can be used.
- When using `LOCK TABLES` on a `TEMPORARY` table, it will always be locked with a `WRITE` lock.
- While a connection holds an explicit read lock on a table, it cannot modify it. If you try, the following error will be produced:

```
ERROR 1099 (HY000): Table 'tab_name' was locked with a READ lock and can't be updated
```

- While a connection holds an explicit lock on a table, it cannot access a non-locked table. If you try, the following error will be produced:

```
ERROR 1100 (HY000): Table 'tab_name' was not locked with LOCK TABLES
```

- While a connection holds an explicit lock on a table, it cannot issue the following: INSERT DELAYED, CREATE TABLE, CREATE TABLE ... LIKE, and DDL statements involving stored programs and views (except for triggers). If you try, the following error will be produced:

```
ERROR 1192 (HY000): Can't execute the given command because you have active locked tables or an
active transaction
```

- `LOCK TABLES` can not be used in stored routines - if you try, the following error will be produced on creation:

```
ERROR 1314 (0A000): LOCK is not allowed in stored procedures
```

1.1.1.9.6 SAVEPOINT

Syntax

```
SAVEPOINT identifier
ROLLBACK [WORK] TO [SAVEPOINT] identifier
RELEASE SAVEPOINT identifier
```

Contents

Description

InnoDB supports the SQL statements `SAVEPOINT`, `ROLLBACK TO SAVEPOINT`, `RELEASE SAVEPOINT` and the optional `WORK` keyword for `ROLLBACK`.

Each savepoint must have a legal MariaDB identifier. A savepoint is a named sub-transaction.

Normally ROLLBACK undoes the changes performed by the whole transaction. When used with the TO clause, it undoes the changes performed after the specified savepoint, and erases all subsequent savepoints. However, all locks that have been acquired after the save point will survive. RELEASE SAVEPOINT does not rollback or commit any changes, but removes the specified savepoint.

When the execution of a trigger or a stored function begins, it is not possible to use statements which reference a savepoint which was defined from out of that stored program.

When a COMMIT (including implicit commits) or a ROLLBACK statement (with no TO clause) is performed, they act on the whole transaction, and all savepoints are removed.

Errors

If COMMIT or ROLLBACK is issued and no transaction was started, no error is reported.

If SAVEPOINT is issued and no transaction was started, no error is reported but no savepoint is created. When ROLLBACK

TO SAVEPOINT or RELEASE SAVEPOINT is called for a savepoint that does not exist, an error like this is issued:

```
ERROR 1305 (42000): SAVEPOINT svp_name does not exist
```

1.1.1.9.7 Metadata Locking

MariaDB supports metadata locking. This means that when a transaction (including XA transactions) uses a table, it locks its metadata until the end of transaction. Non-transactional tables are also locked, as well as views and objects which are related to locked tables/views (stored functions, triggers, etc). When a connection tries to use a DDL statement (like an ALTER TABLE) which modifies a table that is locked, that connection is queued, and has to wait until it's unlocked. Using savepoints and performing a partial rollback does not release metadata locks.

LOCK TABLES ... WRITE are also queued. Some wrong statements which produce an error may not need to wait for the lock to be freed.

The metadata lock's timeout is determined by the value of the lock_wait_timeout server system variable (in seconds). However, note that its default value is 31536000 (1 year, MariaDB <= 10.2.3), or 86400 (1 day, MariaDB >= 10.2.4). If this timeout is exceeded, the following error is returned:

```
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
```

If the metadata_lock_info plugin is installed, the Information Schema metadata_lock_info table stores information about existing metadata locks.

MariaDB starting with 10.5.2

From MariaDB 10.5, the Performance Schema metadata_locks table contains metadata lock information.

Example

Let's use the following MEMORY (non-transactional) table:

```
CREATE TABLE t (a INT) ENGINE = MEMORY;
```

Connection 1 starts a transaction, and INSERTs a row into t:

```
START TRANSACTION;

INSERT INTO t SET a=1;
```

`t` 's metadata is now locked by connection 1. Connection 2 tries to alter `t` , but has to wait:

```
ALTER TABLE t ADD COLUMN b INT;
```

Connection 2's prompt is blocked now.

Now connection 1 ends the transaction:

```
COMMIT;
```

...and connection 2 finally gets the output of its command:

```
Query OK, 1 row affected (35.23 sec)
Records: 1  Duplicates: 0  Warnings: 0
```

1.1.1.9.8 SQL statements That Cause an Implicit Commit

Some SQL statements cause an implicit commit. As a rule of thumb, such statements are DDL statements. The same statements (except for SHUTDOWN) produce a 1400 error (SQLSTATE 'XAE09') if a XA transaction is in effect.

Here is the list:

```
ALTER DATABASE ... UPGRADE DATA DIRECTORY NAME
ALTER EVENT
ALTER FUNCTION
ALTER PROCEDURE
ALTER SEQUENCE
ALTER SERVER
ALTER TABLE
ALTER VIEW
ANALYZE TABLE
BEGIN
CACHE INDEX
CHANGE MASTER TO
CHECK TABLE
CREATE DATABASE
CREATE EVENT
CREATE FUNCTION
CREATE INDEX
CREATE PROCEDURE
CREATE ROLE
CREATE SEQUENCE
CREATE SERVER
CREATE TABLE
CREATE TRIGGER
CREATE USER
CREATE VIEW
DROP DATABASE
DROP EVENT
DROP FUNCTION
DROP INDEX
DROP PROCEDURE
DROP ROLE
DROP SEQUENCE
DROP SERVER
DROP TABLE
DROP TRIGGER
DROP USER
DROP VIEW
FLUSH
GRANT
LOAD INDEX INTO CACHE
LOCK TABLES
OPTIMIZE TABLE
RENAME TABLE
RENAME USER
REPAIR TABLE
RESET
REVOKE
SET PASSWORD
SHUTDOWN
START SLAVE
START TRANSACTION
STOP SLAVE
TRUNCATE TABLE
```

`SET autocommit = 1` causes an implicit commit if the value was 0.

All these statements cause an implicit commit before execution. This means that, even if the statement fails with an error, the transaction is committed. Some of them, like `CREATE TABLE ... SELECT`, also cause a commit immediatly after execution. Such statements couldn't be rollbacked in any case.

If you are not sure whether a statement has implicitly committed the current transaction, you can query the in_transaction server system variable.

Note that when a transaction starts (not in autocommit mode), all locks acquired with LOCK TABLES are released. And acquiring such locks always commits the current transaction. To preserve the data integrity between transactional and non-transactional tables, the GET_LOCK() function can be used.

Exceptions

These statements do not cause an implicit commit in the following cases:

- CREATE TABLE and DROP TABLE, when the `TEMPORARY` keyword is used.

    - However, TRUNCATE TABLE causes an implicit commit even when used on a temporary table.
- CREATE FUNCTION and DROP FUNCTION, when used to create a UDF (instead of a stored function). However, CREATE INDEX and DROP INDEX cause commits even when used with temporary tables.
- UNLOCK TABLES causes a commit only if a LOCK TABLES was used on non-transactional tables.
- START SLAVE, STOP SLAVE, RESET SLAVE and CHANGE MASTER TO did not cause implicit commits prior to MariaDB 10.0.

1.1.1.9.9 Transaction Timeouts

MariaDB has always had the wait_timeout and interactive_timeout settings, which close connections after a certain period of inactivity.

However, these are by default set to a long wait period. In situations where transactions may be started, but not committed or rolled back, more granular control and a shorter timeout may be desirable so as to avoid locks being held for too long.

MariaDB 10.3 introduced three new variables to handle this situation.

- idle_transaction_timeout (all transactions)
- idle_write_transaction_timeout (write transactions - called `idle_readwrite_transaction_timeout` until MariaDB 10.3.2)
- idle_readonly_transaction_timeout (read transactions)

These accept a time in seconds to time out, by closing the connection, transactions that are idle for longer than this period. By default all are set to zero, or no timeout.

idle_transaction_timeout affects all transactions, idle_write_transaction_timeout affects write transactions only and idle_readonly_transaction_timeout affects read transactions only. The latter two variables work independently. However, if either is set along with idle_transaction_timeout, the settings for idle_write_transaction_timeout or idle_readonly_transaction_timeout will take precedence.

Examples

```
SET SESSION idle_transaction_timeout=2;
BEGIN;
SELECT * FROM t;
Empty set (0.000 sec)
## wait 3 seconds
SELECT * FROM t;
ERROR 2006 (HY000): MySQL server has gone away
```

```
SET SESSION idle_write_transaction_timeout=2;
BEGIN;
SELECT * FROM t;
Empty set (0.000 sec)
## wait 3 seconds
SELECT * FROM t;
Empty set (0.000 sec)
INSERT INTO t VALUES(1);
## wait 3 seconds
SELECT * FROM t;
ERROR 2006 (HY000): MySQL server has gone away
```

```
SET SESSION idle_transaction_timeout=2, SESSION idle_readonly_transaction_timeout=10;
BEGIN;
SELECT * FROM t;
Empty set (0.000 sec)
 ## wait 3 seconds
SELECT * FROM t;
Empty set (0.000 sec)
## wait 11 seconds
SELECT * FROM t;
ERROR 2006 (HY000): MySQL server has gone away
```

1.1.1.9.10 UNLOCK TABLES

Syntax

```
UNLOCK TABLES
```

Contents

Description

`UNLOCK TABLES` explicitly releases any table locks held by the current session. See `LOCK TABLES` for more information.

In addition to releasing table locks acquired by the `LOCK TABLES` statement, the `UNLOCK TABLES` statement also releases the global read lock acquired by the `FLUSH TABLES WITH READ LOCK` statement. The `FLUSH TABLES WITH READ LOCK` statement is very useful for performing backups. See `FLUSH` for more information about `FLUSH TABLES WITH READ LOCK`.

1.1.1.9.11 WAIT and NOWAIT

Extended syntax so that it is possible to set innodb_lock_wait_timeout and lock_wait_timeout for the following statements:

Syntax

```
ALTER TABLE tbl_name [WAIT n|NOWAIT] ...
CREATE ... INDEX ON tbl_name (index_col_name, ...) [WAIT n|NOWAIT] ...
DROP INDEX ... [WAIT n|NOWAIT]
DROP TABLE tbl_name [WAIT n|NOWAIT] ...
LOCK TABLE ... [WAIT n|NOWAIT]
OPTIMIZE TABLE tbl_name [WAIT n|NOWAIT]
RENAME TABLE tbl_name [WAIT n|NOWAIT] ...
SELECT ... FOR UPDATE [WAIT n|NOWAIT]
SELECT ... LOCK IN SHARE MODE [WAIT n|NOWAIT]
TRUNCATE TABLE tbl_name [WAIT n|NOWAIT]
```

Description

The lock wait timeout can be explicitly set in the statement by using either `WAIT n` (to set the wait in seconds) or `NOWAIT`, in which case the statement will immediately fail if the lock cannot be obtained. `WAIT 0` is equivalent to `NOWAIT`.

1.1.1.9.12 XA Transactions

Contents

Overview

The MariaDB XA implementation is based on the X/Open CAE document Distributed Transaction Processing: The XA Specification. This document is published by The Open Group and available at http://www.opengroup.org/public/pubs/catalog/c193.htm.

XA transactions are designed to allow distributed transactions, where a transaction manager (the application) controls a transaction which involves multiple resources. Such resources are usually DBMSs, but could be resources of any type. The whole set of required transactional operations is called a global transaction. Each subset of operations which involve a

single resource is called a local transaction. XA used a 2-phases commit (2PC). With the first commit, the transaction manager tells each resource to prepare an effective commit, and waits for a confirm message. The changes are not still made effective at this point. If any of the resources encountered an error, the transaction manager will rollback the global transaction. If all resources communicate that the first commit is successful, the transaction manager can require a second commit, which makes the changes effective.

In MariaDB, XA transactions can only be used with storage engines that support them. At least InnoDB, TokuDB, SPIDER and MyRocks support them. For InnoDB, until MariaDB 10.2, XA transactions can be disabled by setting the innodb_support_xa server system variable to 0. From MariaDB 10.3, XA transactions are always supported.

Like regular transactions, XA transactions create metadata locks on accessed tables.

XA transactions require REPEATABLE READ as a minimum isolation level. However, distributed transactions should always use SERIALIZABLE.

Trying to start more than one XA transaction at the same time produces a 1400 error (SQLSTATE 'XAE09'). The same error is produced when attempting to start an XA transaction while a regular transaction is in effect. Trying to start a regular transaction while an XA transaction is in effect produces a 1399 error (SQLSTATE 'XAE07').

The statements that cause an implicit COMMIT for regular transactions produce a 1400 error (SQLSTATE 'XAE09') if a XA transaction is in effect.

Internal XA vs External XA

XA transactions are an overloaded term in MariaDB. If a storage engine is XA-capable, it can mean one or both of these:

- It supports MariaDB's internal two-phase commit API. This is transparent to the user. Sometimes this is called "internal XA", since MariaDB's internal transaction coordinator log can handle coordinating these transactions.
- It supports XA transactions, with the `XA START`, `XA PREPARE`, `XA COMMIT`, etc. statements. Sometimes this is called "external XA", since it requires the use of an external transaction coordinator to use this feature properly.

Transaction Coordinator Log

If you have two or more XA-capable storage engines enabled, then a transaction coordinator log must be available.

There are currently two implementations of the transaction coordinator log:

- Binary log-based transaction coordinator log
- Memory-mapped file-based transaction coordinator log

If the binary log is enabled on a server, then the server will use the binary log-based transaction coordinator log. Otherwise, it will use the memory-mapped file-based transaction coordinator log.

See Transaction Coordinator Log for more information.

Syntax

```
XA {START|BEGIN} xid [JOIN|RESUME]

XA END xid [SUSPEND [FOR MIGRATE]]

XA PREPARE xid

XA COMMIT xid [ONE PHASE]

XA ROLLBACK xid

XA RECOVER [FORMAT=['RAW'|'SQL']]

xid: gtrid [, bqual [, formatID ]]
```

The interface to XA transactions is a set of SQL statements starting with `XA`. Each statement changes a transaction's state, determining which actions it can perform. A transaction which does not exist is in the `NON-EXISTING` state.

`XA START` (or `BEGIN`) starts a transaction and defines its `xid` (a transaction identifier). The `JOIN` or `RESUME` keywords have no effect. The new transaction will be in `ACTIVE` state.

The `xid` can have 3 components, though only the first one is mandatory. `gtrid` is a quoted string representing a global transaction identifier. `bqual` is a quoted string representing a local transaction identifier. `formatID` is an unsigned integer indicating the format used for the first two components; if not specified, defaults to 1. MariaDB does not interpret in any way

these components, and only uses them to identify a transaction. `xid`s of transactions in effect must be unique.

`XA END` declares that the specified `ACTIVE` transaction is finished and it changes its state to `IDLE`. `SUSPEND [FOR MIGRATE]` has no effect.

`XA PREPARE` prepares an `IDLE` transaction for commit, changing its state to `PREPARED`. This is the first commit.

`XA COMMIT` definitely commits and terminates a transaction which has already been `PREPARED`. If the `ONE PHASE` clause is specified, this statements performs a 1-phase commit on an `IDLE` transaction.

`XA ROLLBACK` rolls back and terminates an `IDLE` or `PREPARED` transaction.

`XA RECOVER` shows information about all `PREPARED` transactions.

When trying to execute an operation which is not allowed for the transaction's current state, an error is produced:

```
XA COMMIT 'test' ONE PHASE;
ERROR 1399 (XAE07): XAER_RMFAIL: The command cannot be executed when global transaction is in
the  ACTIVE state

XA COMMIT 'test2';
ERROR 1399 (XAE07): XAER_RMFAIL: The command cannot be executed when global transaction is in
the  NON-EXISTING state
```

XA RECOVER

The `XA RECOVER` statement shows information about all transactions which are in the `PREPARED` state. It does not matter which connection created the transaction: if it has been `PREPARED`, it appears. But this does not mean that a connection can commit or rollback a transaction which was started by another connection. Note that transactions using a 1-phase commit are never in the `PREPARED` state, so they cannot be shown by `XA RECOVER`.

`XA RECOVER` produces four columns:

```
XA RECOVER;
+----------+--------------+--------------+------+
| formatID | gtrid_length | bqual_length | data |
+----------+--------------+--------------+------+
|        1 |            4 |            0 | test |
+----------+--------------+--------------+------+
```

MariaDB starting with 10.3.3

You can use `XA RECOVER FORMAT='SQL'` to get the data in a human readable form that can be directly copy-pasted into `XA COMMIT` or `XA ROLLBACK`. This is particularly useful for binary `xid` generated by some transaction coordinators.

`formatID` is the `formatID` part of `xid`.

`data` are the `gtrid` and `bqual` parts of `xid`, concatenated.

`gtrid_length` and `bqual_length` are the lengths of `gtrid` and `bqual`, respectevely.

Examples

2-phases commit:

```
XA START 'test';

INSERT INTO t VALUES (1,2);

XA END 'test';

XA PREPARE 'test';

XA COMMIT 'test';
```

1-phase commit:

```
XA START 'test';

INSERT INTO t VALUES (1,2);

XA END 'test';

XA COMMIT 'test' ONE PHASE;
```

Human-readable:

```
xa start '12\r34\t67\v78', 'abc\ndef', 3;

insert t1 values (40);

xa end '12\r34\t67\v78', 'abc\ndef', 3;

xa prepare '12\r34\t67\v78', 'abc\ndef', 3;

xa recover format='RAW';
+----------+--------------+--------------+--------------------+
| formatID | gtrid_length | bqual_length | data               |
+----------+--------------+--------------+--------------------+
34      67v78abc       11 |            7 | 12
def |
+----------+--------------+--------------+--------------------+

xa recover format='SQL';
+----------+--------------+--------------+-----------------------------------------------------+
| formatID | gtrid_length | bqual_length | data                                                |
+----------+--------------+--------------+-----------------------------------------------------+
|        3 |           11 |            7 | X'31320d3334093637763738',X'6162630a646566',3 |
+----------+--------------+--------------+-----------------------------------------------------+

xa rollback X'31320d3334093637763738',X'6162630a646566',3;
```

Known Issues

MariaDB Galera Cluster

MariaDB Galera Cluster does not support XA transactions.

However, MariaDB Galera Cluster builds include a built-in plugin called `wsrep`. Prior to MariaDB 10.4.3, this plugin was internally considered an XA-capable storage engine. Consequently, these MariaDB Galera Cluster builds have multiple XA-capable storage engines by default, even if the only "real" storage engine that supports external XA transactions enabled on these builds by default is InnoDB. Therefore, when using one these builds MariaDB would be forced to use a transaction coordinator log by default, which could have performance implications.

See Transaction Coordinator Log Overview: MariaDB Galera Cluster for more information.

1.1.1.9.13 READ COMMITTED

`READ COMMITTED` is one of the transaction isolation levels. Each consistent read, even within the same transaction, sets and reads its own fresh snapshot.

See SET TRANSACTION#Isolation Levels for details.

1.1.1.9.14 READ UNCOMMITTED

`READ UNCOMMITTED` is one of the transaction isolation levels. SELECT statements are performed in a non-locking fashion, but a possible earlier version of a row might be used

See SET TRANSACTION#Isolation Levels for details.

1.1.1.9.15 REPEATABLE READ

`REPEATABLE READ` is one of the transaction isolation levels. All consistent reads within the same transaction read the

snapshot established by the first read.

See SET TRANSACTION#Isolation Levels for details.

1.1.1.9.16 SERIALIZABLE

`SERIALIZABLE` is one of the transaction isolation levels. Similar to REPEATABLE READ, but InnoDB implicitly converts all plain SELECT statements to SELECT ... LOCK IN SHARE MODE if autocommit is disabled.

See SET TRANSACTION#Isolation Levels for details.

1.1.1.2.15 HELP Command

1.1.1.11 Comment Syntax

There are three supported comment styles in MariaDB:

1. From a ' `#` ' to the end of a line:
   ```
   SELECT * FROM users; # This is a comment
   ```
2. From a ' `--` ' to the end of a line. The space after the two dashes is required (as in MySQL).
   ```
   SELECT * FROM users; -- This is a comment
   ```
3. C style comments from an opening ' `/*` ' to a closing ' `*/` '. Comments of this form can span multiple lines:
   ```
   SELECT * FROM users; /* This is a
   multi-line
   comment */
   ```

Nested comments are possible in some situations, but they are not supported or recommended.

Executable Comments

As an aid to portability between different databases, MariaDB supports executable comments. These special comments allow you to embed SQL code which will not execute when run on other databases, but will execute when run on MariaDB.

MariaDB supports both MySQL's executable comment format, and a slightly modified version specific to MariaDB. This way, if you have SQL code that works on MySQL and MariaDB, but not other databases, you can wrap it in a MySQL executable comment, and if you have code that specifically takes advantage of features only available in MariaDB you can use the MariaDB specific format to hide the code from MySQL.

Executable Comment Syntax

MySQL and MariaDB executable comment syntax:

```
/*! MySQL or MariaDB-specific code */
```

Code that should be executed only starting from a specific MySQL or MariaDB version:

```
/*!##### MySQL or MariaDB-specific code */
```

The numbers, represented by ' `######` ' in the syntax examples above specify the specific the minimum versions of MySQL and MariaDB that should execute the comment. The first number is the major version, the second 2 numbers are the minor version and the last 2 is the patch level.

For example, if you want to embed some code that should only execute on MySQL or MariaDB starting from 5.1.0, you would do the following:

```
/*!50100 MySQL and MariaDB 5.1.0 (and above) code goes here. */
```

MariaDB-only executable comment syntax (starting from MariaDB 5.3.1):

```
/*M! MariaDB-specific code */
/*M!###### MariaDB-specific code */
```

MariaDB ignores MySQL-style executable comments that have a version number in the range `50700..99999`. This is needed to skip features introduced in MySQL-5.7 that are not ported to MariaDB 10.x yet.

```
/*!50701 MariaDB-10.x ignores MySQL-5.7 specific code */
```

Note: comments which have a version number in the range `50700..99999` that use MariaDB-style executable comment syntax are still executed.

```
/*M!50701 MariaDB-10.x does not ignore this */
```

Statement delimiters cannot be used within executable comments.

Examples

In MySQL all the following will return 2: In MariaDB, the last 2 queries would return 3.

```
SELECT 2 /* +1 */;
SELECT 1 /*! +1 */;
SELECT 1 /*!50101 +1 */;
SELECT 2 /*M! +1 */;
SELECT 2 /*M!50301 +1 */;
```

The following executable statement will not work due to the delimiter inside the executable portion:

```
/*M!100100 select 1 ; */
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to
your MariaDB server version for the right syntax to use near '' at line 1
```

Instead, the delimiter should be placed outside the executable portion:

```
/*M!100100 select 1 */;
+---+
| 1 |
+---+
| 1 |
+---+
```

1.2 Built-in Functions

1.1.2 SQL Language Structure

SQL language structure rules.

Identifier Names

Naming rules for identifiers.

Identifier Case-sensitivity

Whether objects are case-sensitive or not is partly determined by the under...

Binary Literals

Binary literals can be written in one of the following formats.

Boolean Literals

True and false.

Date and Time Literals

Literals regarding date and time.

Hexadecimal Literals
Hexadecimal literals can be written using any of the following syntaxes

Identifier Qualifiers
How to reference an object and its context in an SQL statement.

Identifier to File Name Mapping
Some identifiers map to a file name on the filesystem. Databases each have ...

MariaDB Error Codes
MariaDB error codes reference list.

Numeric Literals
Numeric literals are written as a sequence of digits from 0 to 9

Reserved Words
List of reserved words in MariaDB.

SQLSTATE
A string which identifies a condition's class and subclass

String Literals
Strings are sequences of characters and are enclosed with quotes.

Table Value Constructors
Documents adding arbitrary values to the result-set.

User-Defined Variables
Variables which exist within a session.

There are 5 related questions.

1.1.2.1 Identifier Names

Contents

Databases, tables, indexes, columns, aliases, views, stored routines, triggers, events, variables, partitions, tablespaces, savepoints, labels, users, roles, are collectively known as identifiers, and have certain rules for naming.

Identifiers may be quoted using the backtick character - `` ` ``. Quoting is optional for identifiers that don't contain special characters, or for identifiers that are not reserved words. If the `ANSI_QUOTES` SQL_MODE flag is set, double quotes (`"`) can also be used to quote identifiers. If the `MSSQL` flag is set, square brackets (`[` and `]`) can be used for quoting.

Even when using reserved words as names, fully qualified names do not need to be quoted. For example, `test.select` has only one possible meaning, so it is correctly parsed even without quotes.

Unquoted

The following characters are valid, and allow identifiers to be unquoted:

- ASCII: [0-9,a-z,A-Z$_] (numerals 0-9, basic Latin letters, both lowercase and uppercase, dollar sign, underscore)
- Extended: U+0080 .. U+FFFF

Quoted

The following characters are valid, but identifiers using them must be quoted:

- ASCII: U+0001 .. U+007F (full Unicode Basic Multilingual Plane (BMP) except for U+0000)
- Extended: U+0080 .. U+FFFF
- Identifier quotes can themselves be used as part of an identifier, as long as they are quoted.

Further Rules

There are a number of other rules for identifiers:

- Identifiers are stored as Unicode (UTF-8)
- Identifiers may or may not be case-sensitive. See Indentifier Case-sensitivity.
- Database, table and column names can't end with space characters
- Identifier names may begin with a numeral, but can't only contain numerals unless quoted.
- An identifier starting with a numeral, followed by an 'e', may be parsed as a floating point number, and needs to be quoted.
- Identifiers are not permitted to contain the ASCII NUL character (U+0000) and supplementary characters (U+10000 and higher).
- Names such as 5e6, 9e are not prohibited, but it's strongly recommended not to use them, as they could lead to ambiguity in certain contexts, being treated as a number or expression.
- User variables cannot be used as part of an identifier, or as an identifier in an SQL statement.

Quote Character

The regular quote character is the backtick character - `` ` ``, but if the `ANSI_QUOTES` SQL_MODE option is specified, a regular double quote - `"` may be used as well.

The backtick character can be used as part of an identifier. In that case the identifier needs to be quoted. The quote character can be the backtick, but in that case, the backtick in the name must be escaped with another backtick.

Maximum Length

- Databases, tables, columns, indexes, constraints, stored routines, triggers, events, views, tablespaces, servers and log file groups have a maximum length of 64 characters.
- Compound statement labels have a maximum length of 16 characters
- Aliases have a maximum length of 256 characters, except for column aliases in CREATE VIEW statements, which are checked against the maximum column length of 64 characters (not the maximum alias length of 256 characters).
- Users have a maximum length of 80 characters.
- Roles have a maximum length of 128 characters.
- Multi-byte characters do not count extra towards towards the character limit.

Multiple Identifiers

MariaDB allows the column name to be used on its own if the reference will be unambiguous, or the table name to be used with the column name, or all three of the database, table and column names. A period is used to separate the identifiers, and the period can be surrounded by spaces.

Examples

Using the period to separate identifiers:

```
CREATE TABLE t1 (i int);

INSERT INTO t1(i) VALUES (10);

SELECT i FROM t1;
+------+
| i    |
+------+
|   10 |
+------+

SELECT t1.i FROM t1;
+------+
| i    |
+------+
|   10 |
+------+

SELECT test.t1.i FROM t1;
+------+
| i    |
+------+
|   10 |
+------+
```

The period can be separated by spaces:

```
SELECT test . t1 . i FROM t1;
+------+
| i    |
+------+
|   10 |
+------+
```

Resolving ambiguity:

```
CREATE TABLE t2 (i int);

SELECT i FROM t1 LEFT JOIN t2 ON t1.i=t2.i;
ERROR 1052 (23000): Column 'i' in field list is ambiguous

SELECT t1.i FROM t1 LEFT JOIN t2 ON t1.i=t2.i;
+------+
| i    |
+------+
|   10 |
+------+
```

Creating a table with characters that require quoting:

```
CREATE TABLE 123% (i int);
ERROR 1064 (42000): You have an error in your SQL syntax;
  check the manual that corresponds to your MariaDB server version for the right syntax
  to use near '123% (i int)' at line 1

CREATE TABLE `123%` (i int);
Query OK, 0 rows affected (0.85 sec)

CREATE TABLE `TABLE` (i int);
Query OK, 0 rows affected (0.36 sec)
```

Using double quotes as a quoting character:

```
CREATE TABLE "SELECT" (i int);
ERROR 1064 (42000): You have an error in your SQL syntax;
  check the manual that corresponds to your MariaDB server version for the right syntax
  to use near '"SELECT" (i int)' at line 1

SET sql_mode='ANSI_QUOTES';
Query OK, 0 rows affected (0.03 sec)

CREATE TABLE "SELECT" (i int);
Query OK, 0 rows affected (0.46 sec)
```

Using an identifier quote as part of an identifier name:

```
SHOW VARIABLES LIKE 'sql_mode';
+---------------+-------------+
| Variable_name | Value       |
+---------------+-------------+
| sql_mode      | ANSI_QUOTES |
+---------------+-------------+

CREATE TABLE "fg`d" (i int);
Query OK, 0 rows affected (0.34 sec)
```

Creating the table named `*` (Unicode number: U+002A) requires quoting.

```
CREATE TABLE `*` (a INT);
```

Floating point ambiguity:

```
CREATE TABLE 8984444cce5d (x INT);
Query OK, 0 rows affected (0.38 sec)

CREATE TABLE 8981e56cce5d (x INT);
ERROR 1064 (42000): You have an error in your SQL syntax;
  check the manual that corresponds to your MariaDB server version for the right syntax
  to use near '8981e56cce5d (x INT)' at line 1

CREATE TABLE `8981e56cce5d` (x INT);
Query OK, 0 rows affected (0.39 sec)
```

1.1.2.2 Identifier Case-sensitivity

Whether objects are case-sensitive or not is partly determined by the underlying operating system. Unix-based systems are case-sensitive, Windows is not, while Mac OS X is usually case-insensitive by default, but devices can be configured as case-sensitive using Disk Utility.

Database, table, table aliases and trigger names are affected by the systems case-sensitivity, while index, column, column aliases, stored routine and event names are never case sensitive.

Log file group name are case sensitive.

The lower_case_table_names server system variable plays a key role. It determines whether table names, aliases and database names are compared in a case-sensitive manner. If set to 0 (the default on Unix-based systems), table names and aliases and database names are compared in a case-sensitive manner. If set to 1 (the default on Windows), names are stored in lowercase and not compared in a case-sensitive manner. If set to 2 (the default on Mac OS X), names are stored as declared, but compared in lowercase.

It is thus possible to make Unix-based systems behave like Windows and ignore case-sensitivity, but the reverse is not true, as the underlying Windows filesystem can not support this.

Even on case-insensitive systems, you are required to use the same case consistently within the same statement. The following statement fails, as it refers to the table name in a different case.

```
SELECT * FROM a_table WHERE A_table.id>10;
```

For a full list of identifier naming rules, see Identifier Names.

Please note that lower_case_table_names is a database initialization parameter. This means that, along with innodb_page_size, this variable must be set before running mariadb-install-db, and will not change the behavior of servers

unless applied before the creation of core system databases.

1.1.2.3 Binary Literals

Binary literals can be written in one of the following formats: `b'value'`, `B'value'` or `0bvalue`, where `value` is a string composed by `0` and `1` digits.

Binary literals are interpreted as binary strings, and are convenient to represent VARBINARY, BINARY or BIT values.

To convert a binary literal into an integer, just add 0.

Examples

Printing the value as a binary string:

```
SELECT 0b1000001;
+-----------+
| 0b1000001 |
+-----------+
| A         |
+-----------+
```

Converting the same value into a number:

```
SELECT 0b1000001+0;
+-------------+
| 0b1000001+0 |
+-------------+
|          65 |
+-------------+
```

1.1.2.4 Boolean Literals

In MariaDB, `FALSE` is a synonym of 0 and `TRUE` is a synonym of 1. These constants are case insensitive, so `TRUE`, `True`, and `true` are equivalent.

These terms are not synonyms of 0 and 1 when used with the `IS` operator. So, for example, `10 IS TRUE` returns 1, while `10 = TRUE` returns 0 (because 1 != 10).

The `IS` operator accepts a third constant exists: `UNKNOWN`. It is always a synonym of NULL.

`TRUE` and `FALSE` are reserved words, while `UNKNOWN` is not.

1.1.2.5 Date and Time Literals

Contents

Standard syntaxes

MariaDB supports the SQL standard and ODBC syntaxes for DATE, TIME and TIMESTAMP literals.

SQL standard syntax:

- DATE 'string'
- TIME 'string'
- TIMESTAMP 'string'

ODBC syntax:

- {d 'string'}
- {t 'string'}
- {ts 'string'}

The timestamp literals are treated as DATETIME literals, because in MariaDB the range of `DATETIME` is closer to the `TIMESTAMP` range in the SQL standard.

`string` is a string in a proper format, as explained below.

`DATE` literals

A `DATE` string is a string in one of the following formats: `'YYYY-MM-DD'` or `'YY-MM-DD'`. Note that any punctuation character can be used as delimiter. All delimiters must consist of 1 character. Different delimiters can be used in the same string. Delimiters are optional (but if one delimiter is used, all delimiters must be used).

A `DATE` literal can also be an integer, in one of the following formats: `YYYYMMDD` or `YYMMDD`.

All the following `DATE` literals are valid, and they all represent the same value:

```
'19940101'
'940101'
'1994-01-01'
'94/01/01'
'1994-01/01'
'94:01!01'
19940101
940101
```

`DATETIME` literals

A `DATETIME` string is a string in one of the following formats: `'YYYY-MM-DD HH:MM:SS'` or `'YY-MM-DD HH:MM:SS'`. Note that any punctuation character can be used as delimiter for the date part and for the time part. All delimiters must consist of 1 character. Different delimiters can be used in the same string. The hours, minutes and seconds parts can consist of one character. For this reason, delimiters are mandatory for `DATETIME` literals.

The delimiter between the date part and the time part can be a `T` or any sequence of space characters (including tabs, new lines and carriage returns).

A `DATETIME` literal can also be a number, in one of the following formats: `YYYYMMDDHHMMSS`, `YYMMDDHHMMSS`, `YYYYMMDD` or `YYMMDD`. In this case, all the time subparts must consist of 2 digits.

All the following `DATE` literals are valid, and they all represent the same value:

```
'1994-01-01T12:30:03'
'1994/01/01\n\t 12+30+03'
'1994/01\\01\n\t 12+30-03'
'1994-01-01 12:30:3'
```

`TIME` literals

A `TIME` string is a string in one of the following formats: `'D HH:MM:SS'`, `'HH:MM:SS`, `'D HH:MM'`, `'HH:MM'`, `'D HH'`, or `'SS'`. `D` is a value from 0 to 34 which represents days. `:` is the only allowed delimiter for `TIME` literals. Delimiters are mandatory, with an exception: the `'HHMMSS'` format is allowed. When delimiters are used, each part of the literal can consist of one character.

A `TIME` literal can also be a number in one of the following formats: `HHMMSS`, `MMSS`, or `SS`.

The following literals are equivalent:

```
'09:05:00'
'9:05:0'
'9:5:0'
'090500'
```

- 2-digit years

The year part in `DATE` and `DATETIME` literals is determined as follows:

- `70 - 99 = 1970 - 1999`
- `00 - 69 = 2000 - 2069`

Microseconds

`DATETIME` and `TIME` literals can have an optional microseconds part. For both string and numeric forms, it is expressed as a decimal part. Up to 6 decimal digits are allowed. Examples:

```
'12:30:00.123456'
123000.123456
```

See Microseconds in MariaDB for details.

Date and time literals and the `SQL_MODE`

Unless the SQL_MODE `NO_ZERO_DATE` flag is set, some special values are allowed: the `'0000-00-00'` `DATE`, the `'00:00:00'` `TIME`, and the `0000-00-00 00:00:00` `DATETIME`.

If the `ALLOW_INVALID_DATES` flag is set, the invalid dates (for example, 30th February) are allowed. If not, if the `NO_ZERO_DATE` is set, an error is produced; otherwise, a zero-date is returned.

Unless the `NO_ZERO_IN_DATE` flag is set, each subpart of a date or time value (years, hours...) can be set to 0.

1.1.2.6 Hexadecimal Literals

Contents

Hexadecimal literals can be written using any of the following syntaxes:

- x'`value`'
- X'`value`' (SQL standard)
- 0x`value` (ODBC)

`value` is a sequence of hexadecimal digits (from `0` to `9` and from `A` to `F`). The case of the digits does not matter. With the first two syntaxes, `value` must consist of an even number of digits. With the last syntax, digits can be even, and they are treated as if they had an extra 0 at the beginning.

Normally, hexadecimal literals are interpreted as binary string, where each pair of digits represents a character. When used in a numeric context, they are interpreted as integers. (See the example below). In no case can a hexadecimal literal be a decimal number.

The first two syntaxes; `X'value'` and x'`value`', follow the SQL standard, and behave as a string in all contexts in MariaDB since MariaDB 10.0.3 and MariaDB 5.5.31 (fixing MDEV-4489). The latter syntax, 0x`value`, is a MySQL/MariaDB extension for hex hybrids and behaves as a string or as a number depending on context. MySQL treats all syntaxes the same, so there may be different results in MariaDB and MySQL (see below).

Examples

Representing the `a` character with the three syntaxes explained above:

```
SELECT x'61', X'61', 0x61;
+-------+-------+------+
| x'61' | X'61' | 0x61 |
+-------+-------+------+
| a     | a     | a    |
+-------+-------+------+
```

Hexadecimal literals in a numeric context:

```
SELECT 0 + 0xF, -0xF;
+---------+------+
| 0 + 0xF | -0xF |
+---------+------+
|      15 |  -15 |
+---------+------+
```

Fun with Types

```
CREATE TABLE t1 (a INT, b VARCHAR(10));
INSERT INTO t1 VALUES (0x31, 0x61),(COALESCE(0x31), COALESCE(0x61));

SELECT * FROM t1;
+------+------+
| a    | b    |
+------+------+
|   49 | a    |
|    1 | a    |
+------+------+
```

The reason for the differing results above is that when 0x31 is inserted directly to the column, it's treated as a number, while when 0x31 is passed to COALESCE(), it's treated as a string, because:

- HEX values have a string data type by default.
- COALESCE() has the same data type as the argument.

Differences Between MariaDB and MySQL

```
SELECT x'0a'+0;
+---------+
| x'0a'+0 |
+---------+
|       0 |
+---------+
1 row in set, 1 warning (0.00 sec)

Warning (Code 1292): Truncated incorrect DOUBLE value: '\x0A'

SELECT X'0a'+0;
+---------+
| X'0a'+0 |
+---------+
|       0 |
+---------+
1 row in set, 1 warning (0.00 sec)

Warning (Code 1292): Truncated incorrect DOUBLE value: '\x0A'

SELECT 0x0a+0;
+--------+
| 0x0a+0 |
+--------+
|     10 |
+--------+
```

In MySQL (up until at least MySQL 8.0.26):

```
SELECT x'0a'+0;
+---------+
| x'0a'+0 |
+---------+
|      10 |
+---------+

SELECT X'0a'+0;
+---------+
| X'0a'+0 |
+---------+
|      10 |
+---------+

SELECT 0x0a+0;
+--------+
| 0x0a+0 |
+--------+
|     10 |
+--------+
```

1.1.2.7 Identifier Qualifiers

Contents

Qualifiers are used within SQL statements to reference data structures, such as databases, tables, or columns. For example, typically a SELECT query contains references to some columns and at least one table.

Qualifiers can be composed by one or more identifiers, where the initial parts affect the context within which the final identifier is interpreted:

- For a database, only the database identifier needs to be specified.
- For objects which are contained in a database (like tables, views, functions, etc) the database identifier can be specified. If no database is specified, the current database is assumed (see USE and DATABASE() for more details). If there is no default database and no database is specified, an error is issued.
- For column names, the table and the database are generally obvious from the context of the statement. It is however possible to specify the table identifier, or the database identifier plus the table identifier.
- An identifier is fully-qualified if it contains all possible qualifiers, for example, the following column is fully qualified: `db_name.tbl_name.col_name`.

If a qualifier is composed by more than one identifier, a dot (.) must be used as a separator. All identifiers can be quoted individually. Extra spacing (including new lines and tabs) is allowed.

All the following examples are valid:

- db_name.tbl_name.col_name
- tbl_name
- `db_name`.`tbl_name`.`col_name`
- `db_name` . `tbl_name`
- db_name. tbl_name

If a table identifier is prefixed with a dot (.), the default database is assumed. This syntax is supported for ODBC compliance, but has no practical effect on MariaDB. These qualifiers are equivalent:

- tbl_name
- . tbl_name
- .`tbl_name`
- . `tbl_name`

For DML statements, it is possible to specify a list of the partitions using the PARTITION clause. See Partition Pruning and Selection for details.

1.1.2.8 Identifier to File Name Mapping

Some identifiers map to a file name on the filesystem. Databases each have their own directory, while, depending on the storage engine, table names and index names may map to a file name.

Not all characters that are allowed in table names can be used in file names. Every filesystem has its own rules of what characters can be used in file names. To let the user create tables using all characters allowed in the SQL Standard and to not depend on whatever particular filesystem a particular database resides, MariaDB encodes "potentially unsafe"

characters in the table name to derive the corresponding file name.

This is implemented using a special character set. MariaDB converts a table name to the "filename" character set to get the file name for this table. And it converts the file name from the "filename" character set to, for example, utf8 to get the table name for this file name.

The conversion rules are as follows: if the identifier is made up only of basic Latin numbers, letters and/or the underscore character, the encoding matches the name (see however Identifier Case Sensitivity). Otherwise they are encoded according to the following table:

Code Range	Pattern	Number	Used	Unused	Blocks
00C0..017F	[@][0..4][g..z]	5*20= 100	97	3	Latin-1 Supplement + Latin Extended-A
0370..03FF	[@][5..9][g..z]	5*20= 100	88	12	Greek and Coptic
0400..052F	[@][g..z][0..6]	20*7= 140	137	3	Cyrillic + Cyrillic Supplement
0530..058F	[@][g..z][7..8]	20*2= 40	38	2	Armenian
2160..217F	[@][g..z][9]	20*1= 20	16	4	Number Forms
0180..02AF	[@][g..z][a..k]	20*11=220	203	17	Latin Extended-B + IPA Extensions
1E00..1EFF	[@][g..z][l..r]	20*7= 140	136	4	Latin Extended Additional
1F00..1FFF	[@][g..z][s..z]	20*8= 160	144	16	Greek Extended
....	[@][a..f][g..z]	6*20= 120	0	120	RESERVED
24B6..24E9	[@][@][a..z]	26	26	0	Enclosed Alphanumerics
FF21..FF5A	[@][a..z][@]	26	26	0	Halfwidth and Fullwidth forms

Code Range values are UCS-2.

All of this encoding happens transparently at the filesystem level with one exception. Until MySQL 5.1.6, an old encoding was used. Identifiers created in a version before MySQL 5.1.6, and which haven't been updated to the new encoding, the server prefixes `mysql50` to their name.

Examples

Find the file name for a table with a non-Latin1 name:

```
select cast(convert("this_is_таблица" USING filename) as binary);
+-----------------------------------------------------------------+
| cast(convert("this_is_таблица" USING filename) as binary)       |
+-----------------------------------------------------------------+
| this_is_@y0@g0@h0@r0@o0@i1@g0                                   |
+-----------------------------------------------------------------+
```

Find the table name for a file name:

```
select convert(_filename "this_is_@y0@g0@h0@r0@o0@i1@g0" USING utf8);
+-------------------------------------------------------------------+
| convert(_filename "this_is_@y0@g0@h0@r0@o0@i1@g0" USING utf8) |
+-------------------------------------------------------------------+
| this_is_таблица                                                   |
+-------------------------------------------------------------------+
```

An old table created before MySQL 5.1.6, with the old encoding:

```
SHOW TABLES;
+--------------------+
| Tables_in_test     |
+--------------------+
| #mysql50#table@1   |
+--------------------+
```

The prefix needs to be supplied to reference this table:

```
SHOW COLUMNS FROM `table@1`;
ERROR 1146 (42S02): Table 'test.table@1' doesn't exist

SHOW COLUMNS FROM `#mysql50#table@1`;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| i     | int(11) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
```

1.1.2.9 MariaDB Error Codes

MariaDB shares error codes with MySQL, as well as adding a number of new error codes specific to MariaDB.

An example of an error code is as follows:

```
SELECT * FROM x;
ERROR 1046 (3D000): No database selected
```

Contents

There are three pieces of information returned in an error:

- A numeric error code, in this case `1046`. Error codes from 1900 and up are specific to MariaDB, while error codes from 1000 to 1800 are shared by MySQL and MariaDB.
- An SQLSTATE value, consisting of five characters, in this case `3D000`. These codes are standard to ODBC and ANSI SQL. When MariaDB cannot allocate a standard SQLSTATE code, a generic `HY000`, or general error, is used.
- A string describing the error, in this case `No database selected`.

New error codes are being continually being added as new features are added. For a definitive list, see the file `sql/share/errmsg-utf8.txt`, as well as `include/mysqld_error.h` in the build directory, generated by the `comp_err` tool. Also, the perror tool can be used to get the error message which is associated with a given error code.

Shared MariaDB/MySQL error codes

Error Code	SQLSTATE	Error	Description
1000	HY000	ER_HASHCHK	hashchk
1001	HY000	ER_NISAMCHK	isamchk
1002	HY000	ER_NO	NO
1003	HY000	ER_YES	YES
1004	HY000	ER_CANT_CREATE_FILE	Can't create file '%s' (errno: %d)
1005	HY000	ER_CANT_CREATE_TABLE	Can't create table '%s' (errno: %d)
1006	HY000	ER_CANT_CREATE_DB	Can't create database '%s' (errno: %d
1007	HY000	ER_DB_CREATE_EXISTS	Can't create database '%s'; database exists
1008	HY000	ER_DB_DROP_EXISTS	Can't drop database '%s'; database doesn't exist
1009	HY000	ER_DB_DROP_DELETE	Error dropping database (can't delete '%s', errno: %d)
1010	HY000	ER_DB_DROP_RMDIR	Error dropping database (can't rmdir '%s', errno: %d)
1011	HY000	ER_CANT_DELETE_FILE	Error on delete of '%s' (errno: %d)
1012	HY000	ER_CANT_FIND_SYSTEM_REC	Can't read record in system table
1013	HY000	ER_CANT_GET_STAT	Can't get status of '%s' (errno: %d)
1014	HY000	ER_CANT_GET_WD	Can't get working directory (errno: %d)
1015	HY000	ER_CANT_LOCK	Can't lock file (errno: %d)

1016	HY000	ER_CANT_OPEN_FILE	Can't open file: '%s' (errno: %d)
1017	HY000	ER_FILE_NOT_FOUND	Can't find file: '%s' (errno: %d)
1018	HY000	ER_CANT_READ_DIR	Can't read dir of '%s' (errno: %d)
1019	HY000	ER_CANT_SET_WD	Can't change dir to '%s' (errno: %d)
1020	HY000	ER_CHECKREAD	Record has changed since last read in table '%s'
1021	HY000	ER_DISK_FULL	Disk full (%s); waiting for someone to free some space...
1022	23000	ER_DUP_KEY	Can't write; duplicate key in table '%s'
1023	HY000	ER_ERROR_ON_CLOSE	Error on close of '%s' (errno: %d)
1024	HY000	ER_ERROR_ON_READ	Error reading file '%s' (errno: %d)
1025	HY000	ER_ERROR_ON_RENAME	Error on rename of '%s' to '%s' (errno: %d)
1026	HY000	ER_ERROR_ON_WRITE	Error writing file '%s' (errno: %d)
1027	HY000	ER_FILE_USED	'%s' is locked against change
1028	HY000	ER_FILSORT_ABORT	Sort aborted
1029	HY000	ER_FORM_NOT_FOUND	View '%s' doesn't exist for '%s'
1030	HY000	ER_GET_ERRN	Got error %d from storage engine
1031	HY000	ER_ILLEGAL_HA	Table storage engine for '%s' doesn't have this option
1032	HY000	ER_KEY_NOT_FOUND	Can't find record in '%s'
1033	HY000	ER_NOT_FORM_FILE	Incorrect information in file: '%s'
1034	HY000	ER_NOT_KEYFILE	Incorrect key file for table '%s'; try to repair it
1035	HY000	ER_OLD_KEYFILE	Old key file for table '%s'; repair it!
1036	HY000	ER_OPEN_AS_READONLY	Table '%s' is read only
1037	HY001	ER_OUTOFMEMORY	Out of memory; restart server and try again (needed %d bytes)
1038	HY001	ER_OUT_OF_SORTMEMORY	Out of sort memory, consider increasing server sort buffer size
1039	HY000	ER_UNEXPECTED_EOF	Unexpected EOF found when reading file '%s' (Errno: %d)
1040	08004	ER_CON_COUNT_ERROR	Too many connections
1041	HY000	ER_OUT_OF_RESOURCES	Out of memory; check if mysqld or some other process uses all available memory; if not, you may have to use 'ulimit' to allow mysqld to use more memory or you can add more swap space
1042	08S01	ER_BAD_HOST_ERROR	Can't get hostname for your address
1043	08S01	ER_HANDSHAKE_ERROR	Bad handshake
1044	42000	ER_DBACCESS_DENIED_ERROR	Access denied for user '%s'@'%s' to database '%s'
1045	28000	ER_ACCESS_DENIED_ERROR	Access denied for user '%s'@'%s' (using password: %s)
1046	3D000	ER_NO_DB_ERROR	No database selected
1047	08S01	ER_UNKNOWN_COM_ERROR	Unknown command
1048	23000	ER_BAD_NULL_ERROR	Column '%s' cannot be null
1049	42000	ER_BAD_DB_ERROR	Unknown database '%s'
1050	42S01	ER_TABLE_EXISTS_ERROR	Table '%s' already exists
1051	42S02	ER_BAD_TABLE_ERROR	Unknown table '%s'
1052	23000	ER_NON_UNIQ_ERROR	Column '%s' in %s is ambiguous
1053	08S01	ER_SERVER_SHUTDOWN	Server shutdown in progress
1054	42S22	ER_BAD_FIELD_ERROR	Unknown column '%s' in '%s'

1055	42000	ER_WRONG_FIELD_WITH_GROUP	'%s' isn't in GROUP BY
1056	42000	ER_WRONG_GROUP_FIELD	Can't group on '%s'
1057	42000	ER_WRONG_SUM_SELECT	Statement has sum functions and columns in same statement
1058	21S01	ER_WRONG_VALUE_COUNT	Column count doesn't match value count
1059	42000	ER_TOO_LONG_IDENT	Identifier name '%s' is too long
1060	42S21	ER_DUP_FIELDNAME	Duplicate column name '%s'
1061	42000	ER_DUP_KEYNAME	Duplicate key name '%s'
1062	23000	ER_DUP_ENTRY	Duplicate entry '%s' for key %d
1063	42000	ER_WRONG_FIELD_SPEC	Incorrect column specifier for column '%s'
1064	42000	ER_PARSE_ERROR	%s near '%s' at line %d
1065	42000	ER_EMPTY_QUERY	Query was empty
1066	42000	ER_NONUNIQ_TABLE	Not unique table/alias: '%s'
1067	42000	ER_INVALID_DEFAULT	Invalid default value for '%s'
1068	42000	ER_MULTIPLE_PRI_KEY	Multiple primary key defined
1069	42000	ER_TOO_MANY_KEYS	Too many keys specified; max %d keys allowed
1070	42000	ER_TOO_MANY_KEY_PARTS	Too many key parts specified; max %d parts allowed
1071	42000	ER_TOO_LONG_KEY	Specified key was too long; max key length is %d bytes
1072	42000	ER_KEY_COLUMN_DOES_NOT_EXITS	Key column '%s' doesn't exist in table
1073	42000	ER_BLOB_USED_AS_KEY	BLOB column '%s' can't be used in key specification with the used table type
1074	42000	ER_TOO_BIG_FIELDLENGTH	Column length too big for column '%s' (max = %lu); use BLOB or TEXT instead
1075	42000	ER_WRONG_AUTO_KEY	Incorrect table definition; there can be only one auto column and it must be defined as a key
1076	HY000	ER_READY	%s: ready for connections. Version: '%s' socket: '%s' port: %d
1077	HY000	ER_NORMAL_SHUTDOWN	%s: Normal shutdown
1078	HY000	ER_GOT_SIGNAL	%s: Got signal %d. Aborting!
1079	HY000	ER_SHUTDOWN_COMPLETE	%s: Shutdown complete
1080	08S01	ER_FORCING_CLOSE	%s: Forcing close of thread %ld user: '%s'
1081	08S01	ER_IPSOCK_ERROR	Can't create IP socket
1082	42S12	ER_NO_SUCH_INDEX	Table '%s' has no index like the one used in CREATE INDEX; recreate the table
1083	42000	ER_WRONG_FIELD_TERMINATORS	Field separator argument is not what is expected; check the manual
1084	42000	ER_BLOBS_AND_NO_TERMINATED	You can't use fixed rowlength with BLOBs; please use 'fields terminated by'
1085	HY000	ER_TEXTFILE_NOT_READABLE	The file '%s' must be in the database directory or be readable by all
1086	HY000	ER_FILE_EXISTS_ERROR	File '%s' already exists
1087	HY000	ER_LOAD_INF	Records: %ld Deleted: %ld Skipped: %ld Warnings: %ld
1088	HY000	ER_ALTER_INF	Records: %ld Duplicates: %ld
1089	HY000	ER_WRONG_SUB_KEY	Incorrect prefix key; the used key part isn't a string, the used length is longer than the key part, or the storage engine doesn't support unique prefix keys

1090	42000	ER_CANT_REMOVE_ALL_FIELDS	You can't delete all columns with ALTER TABLE; use DROP TABLE instead
1091	42000	ER_CANT_DROP_FIELD_OR_KEY	Can't DROP '%s'; check that column/key exists
1092	HY000	ER_INSERT_INF	Records: %ld Duplicates: %ld Warnings: %ld
1093	HY000	ER_UPDATE_TABLE_USED	You can't specify target table '%s' for update in FROM clause
1094	HY000	ER_NO_SUCH_THREAD	Unknown thread id: %lu
1095	HY000	ER_KILL_DENIED_ERROR	You are not owner of thread %lu
1096	HY000	ER_NO_TABLES_USED	No tables used
1097	HY000	ER_TOO_BIG_SET	Too many strings for column %s and SET
1098	HY000	ER_NO_UNIQUE_LOGFILE	Can't generate a unique log-filename %s.(1-999)
1099	HY000	ER_TABLE_NOT_LOCKED_FOR_WRITE	Table '%s' was locked with a READ lock and can't be updated

Error Code	SQLSTATE	Error	Description
1100	HY000	ER_TABLE_NOT_LOCKED	Table '%s' was not locked with LOCK TABLES
1101		ER_UNUSED_17	You should never see it
1102	42000	ER_WRONG_DB_NAME	Incorrect database name '%s'
1103	42000	ER_WRONG_TABLE_NAME	Incorrect table name '%s'
1104	42000	ER_TOO_BIG_SELECT	The SELECT would examine more than MAX_JOIN_SIZE rows; check your WHERE and use SET SQL_BIG_SELECTS=1 or SET MAX_JOIN_SIZE=# if the SELECT is okay
1105	HY000	ER_UNKNOWN_ERROR	Unknown error
1106	42000	ER_UNKNOWN_PROCEDURE	Unknown procedure '%s'
1107	42000	ER_WRONG_PARAMCOUNT_TO_PROCEDURE	Incorrect parameter count to procedure '%s'
1108	HY000	ER_WRONG_PARAMETERS_TO_PROCEDURE	Incorrect parameters to procedure '%s'
1109	42S02	ER_UNKNOWN_TABLE	Unknown table '%s' in %s
1110	42000	ER_FIELD_SPECIFIED_TWICE	Column '%s' specified twice
1111	HY000	ER_INVALID_GROUP_FUNC_USE	Invalid use of group function
1112	42000	ER_UNSUPPORTED_EXTENSION	Table '%s' uses an extension that doesn't exist in this MariaDB version
1113	42000	ER_TABLE_MUST_HAVE_COLUMNS	A table must have at least 1 column
1114	HY000	ER_RECORD_FILE_FULL	The table '%s' is full
1115	42000	ER_UNKNOWN_CHARACTER_SET	Unknown character set: '%s'
1116	HY000	ER_TOO_MANY_TABLES	Too many tables; MariaDB can only use %d tables in a join
1117	HY000	ER_TOO_MANY_FIELDS	Too many columns
1118	42000	ER_TOO_BIG_ROWSIZE	Row size too large. The maximum row size for the used table type, not counting BLOBs, is %ld. You have to change some columns to TEXT or BLOBs
1119	HY000	ER_STACK_OVERRUN	Thread stack overrun: Used: %ld of a %ld stack. Use 'mysqld --thread_stack=#' to specify a bigger stack if needed
1120	42000	ER_WRONG_OUTER_JOIN	Cross dependency found in OUTER JOIN; examine your ON conditions

1121	42000	ER_NULL_COLUMN_IN_INDEX	Table handler doesn't support NULL in given index. Please change column '%s' to be NOT NULL or use another handler
1122	HY000	ER_CANT_FIND_UDF	Can't load function '%s'
1123	HY000	ER_CANT_INITIALIZE_UDF	Can't initialize function '%s'; %s
1124	HY000	ER_UDF_NO_PATHS	No paths allowed for shared library
1125	HY000	ER_UDF_EXISTS	Function '%s' already exists
1126	HY000	ER_CANT_OPEN_LIBRARY	Can't open shared library '%s' (Errno: %d %s)
1127	HY000	ER_CANT_FIND_DL_ENTRY	Can't find symbol '%s' in library
1128	HY000	ER_FUNCTION_NOT_DEFINED	Function '%s' is not defined
1129	HY000	ER_HOST_IS_BLOCKED	Host '%s' is blocked because of many connection errors; unblock with 'mysqladmin flush-hosts'
1130	HY000	ER_HOST_NOT_PRIVILEGED	Host '%s' is not allowed to connect to this MariaDB server
1131	42000	ER_PASSWORD_ANONYMOUS_USER	You are using MariaDB as an anonymous user and anonymous users are not allowed to change passwords
1132	42000	ER_PASSWORD_NOT_ALLOWED	You must have privileges to update tables in the mysql database to be able to change passwords for others
1133	42000	ER_PASSWORD_NO_MATCH	Can't find any matching row in the user table
1134	HY000	ER_UPDATE_INF	Rows matched: %ld Changed: %ld Warnings: %ld
1135	HY000	ER_CANT_CREATE_THREAD	Can't create a new thread (Errno %d); if you are not out of available memory, you can consult the manual for a possible OS-dependent bug
1136	21S01	ER_WRONG_VALUE_COUNT_ON_ROW	Column count doesn't match value count at row %ld
1137	HY000	ER_CANT_REOPEN_TABLE	Can't reopen table: '%s'
1138	22004	ER_INVALID_USE_OF_NULL	Invalid use of NULL value
1139	42000	ER_REGEXP_ERROR	Got error '%s' from regexp
1140	42000	ER_MIX_OF_GROUP_FUNC_AND_FIELDS	Mixing of GROUP columns (MIN(),MAX(),COUNT(),...) with no GROUP columns is illegal if there is no GROUP BY clause
1141	42000	ER_NONEXISTING_GRANT	There is no such grant defined for user '%s' on host '%s'
1142	42000	ER_TABLEACCESS_DENIED_ERROR	%s command denied to user '%s'@'%s' for table '%s'
1143	42000	ER_COLUMNACCESS_DENIED_ERROR	%s command denied to user '%s'@'%s' for column '%s' in table '%s'
1144	42000	ER_ILLEGAL_GRANT_FOR_TABLE	Illegal GRANT/REVOKE command; please consult the manual to see which privileges can be used
1145	42000	ER_GRANT_WRONG_HOST_OR_USER	The host or user argument to GRANT is too long
1146	42S02	ER_NO_SUCH_TABLE	Table '%s.%s' doesn't exist
1147	42000	ER_NONEXISTING_TABLE_GRANT	There is no such grant defined for user '%s' on host '%s' on table '%s'

1148	42000	ER_NOT_ALLOWED_COMMAND	The used command is not allowed with this MariaDB version
1149	42000	ER_SYNTAX_ERROR	You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use
1150	HY000	ER_DELAYED_CANT_CHANGE_LOCK	Delayed insert thread couldn't get requested lock for table %s
1151	HY000	ER_TOO_MANY_DELAYED_THREADS	Too many delayed threads in use
1152	08S01	ER_ABORTING_CONNECTION	Aborted connection %ld to db: '%s' user: '%s' (%s)
1153	08S01	ER_NET_PACKET_TOO_LARGE	Got a packet bigger than 'max_allowed_packet' bytes
1154	08S01	ER_NET_READ_ERROR_FROM_PIPE	Got a read error from the connection pipe
1155	08S01	ER_NET_FCNTL_ERROR	Got an error from fcntl()
1156	08S01	ER_NET_PACKETS_OUT_OF_ORDER	Got packets out of order
1157	08S01	ER_NET_UNCOMPRESS_ERROR	Couldn't uncompress communication packet
1158	08S01	ER_NET_READ_ERROR	Got an error reading communication packets
1159	08S01	ER_NET_READ_INTERRUPTED	Got timeout reading communication packets
1160	08S01	ER_NET_ERROR_ON_WRITE	Got an error writing communication packets
1161	08S01	ER_NET_WRITE_INTERRUPTED	Got timeout writing communication packets
1162	42000	ER_TOO_LONG_STRING	Result string is longer than 'max_allowed_packet' bytes
1163	42000	ER_TABLE_CANT_HANDLE_BLOB	The used table type doesn't support BLOB/TEXT columns
1164	42000	ER_TABLE_CANT_HANDLE_AUTO_INCREMENT	The used table type doesn't support AUTO_INCREMENT columns
1165	HY000	ER_DELAYED_INSERT_TABLE_LOCKED	INSERT DELAYED can't be used with table '%s' because it is locked with LOCK TABLES
1166	42000	ER_WRONG_COLUMN_NAME	Incorrect column name '%s'
1167	42000	ER_WRONG_KEY_COLUMN	The used storage engine can't index column '%s'
1168	HY000	ER_WRONG_MRG_TABLE	Unable to open underlying table which is differently defined or of non-MyISAM type or doesn't exist
1169	23000	ER_DUP_UNIQUE	Can't write, because of unique constraint, to table '%s'
1170	42000	ER_BLOB_KEY_WITHOUT_LENGTH	BLOB/TEXT column '%s' used in key specification without a key length
1171	42000	ER_PRIMARY_CANT_HAVE_NULL	All parts of a PRIMARY KEY must be NOT NULL; if you need NULL in a key, use UNIQUE instead
1172	42000	ER_TOO_MANY_ROWS	Result consisted of more than one row
1173	42000	ER_REQUIRES_PRIMARY_KEY	This table type requires a primary key
1174	HY000	ER_NO_RAID_COMPILED	This version of MariaDB is not compiled with RAID support
1175	HY000	ER_UPDATE_WITHOUT_KEY_IN_SAFE_MODE	You are using safe update mode and you tried to update a table without a WHERE that uses a KEY column
1176	42000	ER_KEY_DOES_NOT_EXITS	Key '%s' doesn't exist in table '%s'
1177	42000	ER_CHECK_NO_SUCH_TABLE	Can't open table

1178	42000	ER_CHECK_NOT_IMPLEMENTED	The storage engine for the table doesn't support %s
1179	25000	ER_CANT_DO_THIS_DURING_AN_TRANSACTION	You are not allowed to execute this command in a transaction
1180	HY000	ER_ERROR_DURING_COMMIT	Got error %d during COMMIT
1181	HY000	ER_ERROR_DURING_ROLLBACK	Got error %d during ROLLBACK
1182	HY000	ER_ERROR_DURING_FLUSH_LOGS	Got error %d during FLUSH_LOGS
1183	HY000	ER_ERROR_DURING_CHECKPOINT	Got error %d during CHECKPOINT
1184	08S01	ER_NEW_ABORTING_CONNECTION	Aborted connection %ld to db: '%s' user: '%s' host: '%s' (%s)
1185		ER_UNUSED_10	You should never see it
1186	HY000	ER_FLUSH_MASTER_BINLOG_CLOSED	Binlog closed, cannot RESET MASTER
1187	HY000	ER_INDEX_REBUILD	Failed rebuilding the index of dumped table '%s'
1188	HY000	ER_MASTER	Error from master: '%s'
1189	08S01	ER_MASTER_NET_READ	Net error reading from master
1190	08S01	ER_MASTER_NET_WRITE	Net error writing to master
1191	HY000	ER_FT_MATCHING_KEY_NOT_FOUND	Can't find FULLTEXT index matching the column list
1192	HY000	ER_LOCK_OR_ACTIVE_TRANSACTION	Can't execute the given command because you have active locked tables or an active transaction
1193	HY000	ER_UNKNOWN_SYSTEM_VARIABLE	Unknown system variable '%s'
1194	HY000	ER_CRASHED_ON_USAGE	Table '%s' is marked as crashed and should be repaired
1195	HY000	ER_CRASHED_ON_REPAIR	Table '%s' is marked as crashed and last (automatic?) repair failed
1196	HY000	ER_WARNING_NOT_COMPLETE_ROLLBACK	Some non-transactional changed tables couldn't be rolled back
1197	HY000	ER_TRANS_CACHE_FULL	Multi-statement transaction required more than 'max_binlog_cache_size' bytes of storage; increase this mysqld variable and try again
1198	HY000	ER_SLAVE_MUST_STOP	This operation cannot be performed with a running slave; run STOP SLAVE first
1199	HY000	ER_SLAVE_NOT_RUNNING	This operation requires a running slave; configure slave and do START SLAVE

Error Code	SQLSTATE	Error	Description
1200	HY000	ER_BAD_SLAVE	The server is not configured as slave; fix in config file or with CHANGE MASTER TO
1201	HY000	ER_MASTER_INFO	Could not initialize master info structure; more error messages can be found in the MariaDB error log
1202	HY000	ER_SLAVE_THREAD	Could not create slave thread; check system resources
1203	42000	ER_TOO_MANY_USER_CONNECTIONS	User %s already has more than 'max_user_connections' active connections
1204	HY000	ER_SET_CONSTANTS_ONLY	You may only use constant expressions with SET

1205	HY000	ER_LOCK_WAIT_TIMEOUT	Lock wait timeout exceeded; try restarting transaction
1206	HY000	ER_LOCK_TABLE_FULL	The total number of locks exceeds the lock table size
1207	25000	ER_READ_ONLY_TRANSACTION	Update locks cannot be acquired during a READ UNCOMMITTED transaction
1208	HY000	ER_DROP_DB_WITH_READ_LOCK	DROP DATABASE not allowed while thread is holding global read lock
1209	HY000	ER_CREATE_DB_WITH_READ_LOCK	CREATE DATABASE not allowed while thread is holding global read lock
1210	HY000	ER_WRONG_ARGUMENTS	Incorrect arguments to %s
1211	42000	ER_NO_PERMISSION_TO_CREATE_USER	'%s'@'%s' is not allowed to create new users
1212	HY000	ER_UNION_TABLES_IN_DIFFERENT_DIR	Incorrect table definition; all MERGE tables must be in the same database
1213	40001	ER_LOCK_DEADLOCK	Deadlock found when trying to get lock; try restarting transaction
1214	HY000	ER_TABLE_CANT_HANDLE_FT	The used table type doesn't support FULLTEXT indexes
1215	HY000	ER_CANNOT_ADD_FOREIGN	Cannot add foreign key constraint
1216	23000	ER_NO_REFERENCED_ROW	Cannot add or update a child row: a foreign key constraint fails
1217	23000	ER_ROW_IS_REFERENCED	Cannot delete or update a parent row: a foreign key constraint fails
1218	08S01	ER_CONNECT_TO_MASTER	Error connecting to master: %s
1219	HY000	ER_QUERY_ON_MASTER	Error running query on master: %s
1220	HY000	ER_ERROR_WHEN_EXECUTING_COMMAND	Error when executing command %s: %s
1221	HY000	ER_WRONG_USAGE	Incorrect usage of %s and %s
1222	21000	ER_WRONG_NUMBER_OF_COLUMNS_IN_SELECT	The used SELECT statements have a different number of columns
1223	HY000	ER_CANT_UPDATE_WITH_READLOCK	Can't execute the query because you have a conflicting read lock
1224	HY000	ER_MIXING_NOT_ALLOWED	Mixing of transactional and non-transactional tables is disabled
1225	HY000	ER_DUP_ARGUMENT	Option '%s' used twice in statement
1226	42000	ER_USER_LIMIT_REACHED	User '%s' has exceeded the '%s' resource (current value: %ld)
1227	42000	ER_SPECIFIC_ACCESS_DENIED_ERROR	Access denied; you need (at least one of) the %s privilege(s) for this operation
1228	HY000	ER_LOCAL_VARIABLE	Variable '%s' is a SESSION variable and can't be used with SET GLOBAL
1229	HY000	ER_GLOBAL_VARIABLE	Variable '%s' is a GLOBAL variable and should be set with SET GLOBAL
1230	42000	ER_NO_DEFAULT	Variable '%s' doesn't have a default value
1231	42000	ER_WRONG_VALUE_FOR_VAR	Variable '%s' can't be set to the value of '%s'
1232	42000	ER_WRONG_TYPE_FOR_VAR	Incorrect argument type to variable '%s'
1233	HY000	ER_VAR_CANT_BE_READ	Variable '%s' can only be set, not read
1234	42000	ER_CANT_USE_OPTION_HERE	Incorrect usage/placement of '%s'

1235	42000	ER_NOT_SUPPORTED_YET	This version of MariaDB doesn't yet support '%s'
1236	HY000	ER_MASTER_FATAL_ERROR_READING_BINLOG	Got fatal error %d from master when reading data from binary log: '%s'
1237	HY000	ER_SLAVE_IGNORED_TABLE	Slave SQL thread ignored the query because of replicate-*-table rules
1238	HY000	ER_INCORRECT_GLOBAL_LOCAL_VAR	Variable '%s' is a %s variable
1239	42000	ER_WRONG_FK_DEF	Incorrect foreign key definition for '%s': %s
1240	HY000	ER_KEY_REF_DO_NOT_MATCH_TABLE_REF	Key reference and table reference don't match
1241	21000	ER_OPERAND_COLUMNS	Operand should contain %d column(s)
1242	21000	ER_SUBQUERY_NO_1_ROW	Subquery returns more than 1 row
1243	HY000	ER_UNKNOWN_STMT_HANDLER	Unknown prepared statement handler (%.*s) given to %s
1244	HY000	ER_CORRUPT_HELP_DB	Help database is corrupt or does not exist
1245	HY000	ER_CYCLIC_REFERENCE	Cyclic reference on subqueries
1246	HY000	ER_AUTO_CONVERT	Converting column '%s' from %s to %s
1247	42S22	ER_ILLEGAL_REFERENCE	Reference '%s' not supported (%s)
1248	42000	ER_DERIVED_MUST_HAVE_ALIAS	Every derived table must have its own alias
1249	01000	ER_SELECT_REDUCED	Select %u was reduced during optimization
1250	42000	ER_TABLENAME_NOT_ALLOWED_HERE	Table '%s' from one of the SELECTs cannot be used in %s
1251	08004	ER_NOT_SUPPORTED_AUTH_MODE	Client does not support authentication protocol requested by server; consider upgrading MariaDB client
1252	42000	ER_SPATIAL_CANT_HAVE_NULL	All parts of a SPATIAL index must be NOT NULL
1253	42000	ER_COLLATION_CHARSET_MISMATCH	COLLATION '%s' is not valid for CHARACTER SET '%s'
1254	HY000	ER_SLAVE_WAS_RUNNING	Slave is already running
1255	HY000	ER_SLAVE_WAS_NOT_RUNNING	Slave already has been stopped
1256	HY000	ER_TOO_BIG_FOR_UNCOMPRESS	Uncompressed data size too large; the maximum size is %d (probably, length of uncompressed data was corrupted)
1257	HY000	ER_ZLIB_Z_MEM_ERROR	ZLIB: Not enough memory
1258	HY000	ER_ZLIB_Z_BUF_ERROR	ZLIB: Not enough room in the output buffer (probably, length of uncompressed data was corrupted)
1259	HY000	ER_ZLIB_Z_DATA_ERROR	ZLIB: Input data corrupted
1260	HY000	ER_CUT_VALUE_GROUP_CONCAT	Row %u was cut by GROUP_CONCAT()
1261	01000	ER_WARN_TOO_FEW_RECORDS	Row %ld doesn't contain data for all columns
1262	01000	ER_WARN_TOO_MANY_RECORDS	Row %ld was truncated; it contained more data than there were input columns
1263	22004	ER_WARN_NULL_TO_NOTNULL	Column set to default value; NULL supplied to NOT NULL column '%s' at row %ld
1264	22003	ER_WARN_DATA_OUT_OF_RANGE	Out of range value for column '%s' at row %ld
1265	01000	WARN_DATA_TRUNCATED	Data truncated for column '%s' at row %ld

1266	HY000	ER_WARN_USING_OTHER_HANDLER	Using storage engine %s for table '%s'
1267	HY000	ER_CANT_AGGREGATE_2COLLATIONS	Illegal mix of collations (%s,%s) and (%s,%s) for operation '%s'
1268	HY000	ER_DROP_USER	Cannot drop one or more of the requested users
1269	HY000	ER_REVOKE_GRANTS	Can't revoke all privileges for one or more of the requested users
1270	HY000	ER_CANT_AGGREGATE_3COLLATIONS	Illegal mix of collations (%s,%s), (%s,%s), (%s,%s) for operation '%s'
1271	HY000	ER_CANT_AGGREGATE_NCOLLATIONS	Illegal mix of collations for operation '%s'
1272	HY000	ER_VARIABLE_IS_NOT_STRUCT	Variable '%s' is not a variable component (can't be used as XXXX.variable_name)
1273	HY000	ER_UNKNOWN_COLLATION	Unknown collation: '%s'
1274	HY000	ER_SLAVE_IGNORED_SSL_PARAMS	SSL parameters in CHANGE MASTER are ignored because this MariaDB slave was compiled without SSL support; they can be used later if MariaDB slave with SSL is started
1275	HY000	ER_SERVER_IS_IN_SECURE_AUTH_MODE	Server is running in --secure-auth mode, but '%s'@'%s' has a password in the old format; please change the password to the new format
1276	HY000	ER_WARN_FIELD_RESOLVED	Field or reference '%s%s%s%s%s' of SELECT #%d was resolved in SELECT #%d
1277	HY000	ER_BAD_SLAVE_UNTIL_COND	Incorrect parameter or combination of parameters for START SLAVE UNTIL
1278	HY000	ER_MISSING_SKIP_SLAVE	It is recommended to use --skip-slave-start when doing step-by-step replication with START SLAVE UNTIL; otherwise, you will get problems if you get an unexpected slave's mysqld restart
1279	HY000	ER_UNTIL_COND_IGNORED	SQL thread is not to be started so UNTIL options are ignored
1280	42000	ER_WRONG_NAME_FOR_INDEX	Incorrect index name '%s'
1281	42000	ER_WRONG_NAME_FOR_CATALOG	Incorrect catalog name '%s'
1282	HY000	ER_WARN_QC_RESIZE	Query cache failed to set size %lu; new query cache size is %lu
1283	HY000	ER_BAD_FT_COLUMN	Column '%s' cannot be part of FULLTEXT index
1284	HY000	ER_UNKNOWN_KEY_CACHE	Unknown key cache '%s'
1285	HY000	ER_WARN_HOSTNAME_WONT_WORK	MariaDB is started in --skip-name-resolve mode; you must restart it without this switch for this grant to work
1286	42000	ER_UNKNOWN_STORAGE_ENGINE	Unknown storage engine '%s'
1287	HY000	ER_WARN_DEPRECATED_SYNTAX	'%s' is deprecated and will be removed in a future release. Please use %s instead
1288	HY000	ER_NON_UPDATABLE_TABLE	The target table %s of the %s is not updatable
1289	HY000	ER_FEATURE_DISABLED	The '%s' feature is disabled; you need MariaDB built with '%s' to have it working

1290	HY000	ER_OPTION_PREVENTS_STATEMENT	The MariaDB server is running with the %s option so it cannot execute this statement
1291	HY000	ER_DUPLICATED_VALUE_IN_TYPE	Column '%s' has duplicated value '%s' in %s
1292	22007	ER_TRUNCATED_WRONG_VALUE	Truncated incorrect %s value: '%s'
1293	HY000	ER_TOO_MUCH_AUTO_TIMESTAMP_COLS	Incorrect table definition; there can be only one TIMESTAMP column with CURRENT_TIMESTAMP in DEFAULT or ON UPDATE clause
1294	HY000	ER_INVALID_ON_UPDATE	Invalid ON UPDATE clause for '%s' column
1295	HY000	ER_UNSUPPORTED_PS	This command is not supported in the prepared statement protocol yet
1296	HY000	ER_GET_ERRMSG	Got error %d '%s' from %s
1297	HY000	ER_GET_TEMPORARY_ERRMSG	Got temporary error %d '%s' from %s
1298	HY000	ER_UNKNOWN_TIME_ZONE	Unknown or incorrect time zone: '%s'
1299	HY000	ER_WARN_INVALID_TIMESTAMP	Invalid TIMESTAMP value in column '%s' at row %ld

Error Code	SQLSTATE	Error	Description
1300	HY000	ER_INVALID_CHARACTER_STRING	Invalid %s character string: '%s'
1301	HY000	ER_WARN_ALLOWED_PACKET_OVERFLOWED	Result of %s() was larger than max_allowed_packet (%ld) - truncated
1302	HY000	ER_CONFLICTING_DECLARATIONS	Conflicting declarations: '%s%s' and '%s%s'
1303	2F003	ER_SP_NO_RECURSIVE_CREATE	Can't create a %s from within another stored routine
1304	42000	ER_SP_ALREADY_EXISTS	%s %s already exists
1305	42000	ER_SP_DOES_NOT_EXIST	%s %s does not exist
1306	HY000	ER_SP_DROP_FAILED	Failed to DROP %s %s
1307	HY000	ER_SP_STORE_FAILED	Failed to CREATE %s %s
1308	42000	ER_SP_LILABEL_MISMATCH	%s with no matching label: %s
1309	42000	ER_SP_LABEL_REDEFINE	Redefining label %s
1310	42000	ER_SP_LABEL_MISMATCH	End-label %s without match
1311	01000	ER_SP_UNINIT_VAR	Referring to uninitialized variable %s
1312	0A000	ER_SP_BADSELECT	PROCEDURE %s can't return a result set in the given context
1313	42000	ER_SP_BADRETURN	RETURN is only allowed in a FUNCTION
1314	0A000	ER_SP_BADSTATEMENT	%s is not allowed in stored procedures
1315	42000	ER_UPDATE_LOG_DEPRECATED_IGNORED	The update log is deprecated and replaced by the binary log; SET SQL_LOG_UPDATE has been ignored. This option will be removed in MariaDB 5.6.
1316	42000	ER_UPDATE_LOG_DEPRECATED_TRANSLATED	The update log is deprecated and replaced by the binary log; SET SQL_LOG_UPDATE has been translated to SET SQL_LOG_BIN. This option will be removed in MariaDB 5.6.
1317	70100	ER_QUERY_INTERRUPTED	Query execution was interrupted
1318	42000	ER_SP_WRONG_NO_OF_ARGS	Incorrect number of arguments for %s %s; expected %u, got %u
1319	42000	ER_SP_COND_MISMATCH	Undefined CONDITION: %s

1320	42000	ER_SP_NORETURN	No RETURN found in FUNCTION %s
1321	2F005	ER_SP_NORETURNEND	FUNCTION %s ended without RETURN
1322	42000	ER_SP_BAD_CURSOR_QUERY	Cursor statement must be a SELECT
1323	42000	ER_SP_BAD_CURSOR_SELECT	Cursor SELECT must not have INTO
1324	42000	ER_SP_CURSOR_MISMATCH	Undefined CURSOR: %s
1325	24000	ER_SP_CURSOR_ALREADY_OPEN	Cursor is already open
1326	24000	ER_SP_CURSOR_NOT_OPEN	Cursor is not open
1327	42000	ER_SP_UNDECLARED_VAR	Undeclared variable: %s
1328	HY000	ER_SP_WRONG_NO_OF_FETCH_ARGS	Incorrect number of FETCH variables
1329	02000	ER_SP_FETCH_NO_DATA	No data - zero rows fetched, selected, or processed
1330	42000	ER_SP_DUP_PARAM	Duplicate parameter: %s
1331	42000	ER_SP_DUP_VAR	Duplicate variable: %s
1332	42000	ER_SP_DUP_COND	Duplicate condition: %s
1333	42000	ER_SP_DUP_CURS	Duplicate cursor: %s
1334	HY000	ER_SP_CANT_ALTER	Failed to ALTER %s %s
1335	0A000	ER_SP_SUBSELECT_NYI	Subquery value not supported
1336	0A000	ER_STMT_NOT_ALLOWED_IN_SF_OR_TRG	%s is not allowed in stored function or trigger
1337	42000	ER_SP_VARCOND_AFTER_CURSHNDLR	Variable or condition declaration after cursor or handler declaration
1338	42000	ER_SP_CURSOR_AFTER_HANDLER	Cursor declaration after handler declaration
1339	20000	ER_SP_CASE_NOT_FOUND	Case not found for CASE statement
1340	HY000	ER_FPARSER_TOO_BIG_FILE	Configuration file '%s' is too big
1341	HY000	ER_FPARSER_BAD_HEADER	Malformed file type header in file '%s'
1342	HY000	ER_FPARSER_EOF_IN_COMMENT	Unexpected end of file while parsing comment '%s'
1343	HY000	ER_FPARSER_ERROR_IN_PARAMETER	Error while parsing parameter '%s' (line: '%s')
1344	HY000	ER_FPARSER_EOF_IN_UNKNOWN_PARAMETER	Unexpected end of file while skipping unknown parameter '%s'
1345	HY000	ER_VIEW_NO_EXPLAIN	EXPLAIN/SHOW can not be issued; lacking privileges for underlying table
1346	HY000	ER_FRM_UNKNOWN_TYPE	File '%s' has unknown type '%s' in its header
1347	HY000	ER_WRONG_OBJECT	'%s.%s' is not %s
1348	HY000	ER_NONUPDATEABLE_COLUMN	Column '%s' is not updatable
1349	HY000	ER_VIEW_SELECT_DERIVED	View's SELECT contains a subquery in the FROM clause
1350	HY000	ER_VIEW_SELECT_CLAUSE	View's SELECT contains a '%s' clause
1351	HY000	ER_VIEW_SELECT_VARIABLE	View's SELECT contains a variable or parameter
1352	HY000	ER_VIEW_SELECT_TMPTABLE	View's SELECT refers to a temporary table '%s'
1353	HY000	ER_VIEW_WRONG_LIST	View's SELECT and view's field list have different column counts
1354	HY000	ER_WARN_VIEW_MERGE	View merge algorithm can't be used here for now (assumed undefined algorithm)

1355	HY000	ER_WARN_VIEW_WITHOUT_KEY	View being updated does not have complete key of underlying table in it
1356	HY000	ER_VIEW_INVALID	View '%s.%s' references invalid table(s) or column(s) or function(s) or definer/invoker of view lack rights to use them
1357	HY000	ER_SP_NO_DROP_SP	Can't drop or alter a %s from within another stored routine
1358	HY000	ER_SP_GOTO_IN_HNDLR	GOTO is not allowed in a stored procedure handler
1359	HY000	ER_TRG_ALREADY_EXISTS	Trigger already exists
1360	HY000	ER_TRG_DOES_NOT_EXIST	Trigger does not exist
1361	HY000	ER_TRG_ON_VIEW_OR_TEMP_TABLE	Trigger's '%s' is view or temporary table
1362	HY000	ER_TRG_CANT_CHANGE_ROW	Updating of %s row is not allowed in %strigger
1363	HY000	ER_TRG_NO_SUCH_ROW_IN_TRG	There is no %s row in %s trigger
1364	HY000	ER_NO_DEFAULT_FOR_FIELD	Field '%s' doesn't have a default value
1365	22012	ER_DIVISION_BY_ZER	Division by 0
1366	HY000	ER_TRUNCATED_WRONG_VALUE_FOR_FIELD	Incorrect %s value: '%s' for column '%s' at row %ld
1367	22007	ER_ILLEGAL_VALUE_FOR_TYPE	Illegal %s '%s' value found during parsing
1368	HY000	ER_VIEW_NONUPD_CHECK	CHECK OPTION on non-updatable view '%s.%s'
1369	HY000	ER_VIEW_CHECK_FAILED	CHECK OPTION failed '%s.%s'
1370	42000	ER_PROCACCESS_DENIED_ERROR	%s command denied to user '%s'@'%s' for routine '%s'
1371	HY000	ER_RELAY_LOG_FAIL	Failed purging old relay logs: %s
1372	HY000	ER_PASSWD_LENGTH	Password hash should be a %d-digit hexadecimal number
1373	HY000	ER_UNKNOWN_TARGET_BINLOG	Target log not found in binlog index
1374	HY000	ER_IO_ERR_LOG_INDEX_READ	I/O error reading log index file
1375	HY000	ER_BINLOG_PURGE_PROHIBITED	Server configuration does not permit binlog purge
1376	HY000	ER_FSEEK_FAIL	Failed on fseek()
1377	HY000	ER_BINLOG_PURGE_FATAL_ERR	Fatal error during log purge
1378	HY000	ER_LOG_IN_USE	A purgeable log is in use, will not purge
1379	HY000	ER_LOG_PURGE_UNKNOWN_ERR	Unknown error during log purge
1380	HY000	ER_RELAY_LOG_INIT	Failed initializing relay log position: %s
1381	HY000	ER_NO_BINARY_LOGGING	You are not using binary logging
1382	HY000	ER_RESERVED_SYNTAX	The '%s' syntax is reserved for purposes internal to the MariaDB server
1383	HY000	ER_WSAS_FAILED	WSAStartup Failed
1384	HY000	ER_DIFF_GROUPS_PROC	Can't handle procedures with different groups yet
1385	HY000	ER_NO_GROUP_FOR_PROC	Select must have a group with this procedure
1386	HY000	ER_ORDER_WITH_PROC	Can't use ORDER clause with this procedure
1387	HY000	ER_LOGGING_PROHIBIT_CHANGING_OF	Binary logging and replication forbid changing the global server %s
1388	HY000	ER_NO_FILE_MAPPING	Can't map file: %s, errno: %d

1389	HY000	ER_WRONG_MAGIC	Wrong magic in %s
1390	HY000	ER_PS_MANY_PARAM	Prepared statement contains too many placeholders
1391	HY000	ER_KEY_PART_0	Key part '%s' length cannot be 0
1392	HY000	ER_VIEW_CHECKSUM	View text checksum failed
1393	HY000	ER_VIEW_MULTIUPDATE	Can not modify more than one base table through a join view '%s.%s'
1394	HY000	ER_VIEW_NO_INSERT_FIELD_LIST	Can not insert into join view '%s.%s' without fields list
1395	HY000	ER_VIEW_DELETE_MERGE_VIEW	Can not delete from join view '%s.%s'
1396	HY000	ER_CANNOT_USER	Operation %s failed for %s
1397	XAE04	ER_XAER_NOTA	XAER_NOTA: Unknown XID
1398	XAE05	ER_XAER_INVAL	XAER_INVAL: Invalid arguments (or unsupported command)
1399	XAE07	ER_XAER_RMFAIL	XAER_RMFAIL: The command cannot be executed when global transaction is in the %s state

Error Code	SQLSTATE	Error	Description
1400	XAE09	ER_XAER_OUTSIDE	XAER_OUTSIDE: Some work is done outside global transaction
1401	XAE03	ER_XAER_RMERR	XAER_RMERR: Fatal error occurred in the transaction branch - check your data for consistency
1402	XA100	ER_XA_RBROLLBACK	XA_RBROLLBACK: Transaction branch was rolled back
1403	42000	ER_NONEXISTING_PROC_GRANT	There is no such grant defined for user '%s' on host '%s' on routine '%s'
1404	HY000	ER_PROC_AUTO_GRANT_FAIL	Failed to grant EXECUTE and ALTER ROUTINE privileges
1405	HY000	ER_PROC_AUTO_REVOKE_FAIL	Failed to revoke all privileges to dropped routine
1406	22001	ER_DATA_TOO_LONG	Data too long for column '%s' at row %ld
1407	42000	ER_SP_BAD_SQLSTATE	Bad SQLSTATE: '%s'
1408	HY000	ER_STARTUP	%s: ready for connections. Version: '%s' socket: '%s' port: %d %s
1409	HY000	ER_LOAD_FROM_FIXED_SIZE_ROWS_TO_VAR	Can't load value from file with fixed size rows to variable
1410	42000	ER_CANT_CREATE_USER_WITH_GRANT	You are not allowed to create a user with GRANT
1411	HY000	ER_WRONG_VALUE_FOR_TYPE	Incorrect %s value: '%s' for function %s
1412	HY000	ER_TABLE_DEF_CHANGED	Table definition has changed, please retry transaction
1413	42000	ER_SP_DUP_HANDLER	Duplicate handler declared in the same block
1414	42000	ER_SP_NOT_VAR_ARG	OUT or INOUT argument %d for routine %s is not a variable or NEW pseudo-variable in BEFORE trigger

1415	0A000	ER_SP_NO_RETSET	Not allowed to return a result set from a %s
1416	22003	ER_CANT_CREATE_GEOMETRY_OBJECT	Cannot get geometry object from data you send to the GEOMETRY field
1417	HY000	ER_FAILED_ROUTINE_BREAK_BINLOG	A routine failed and has neither NO SQL nor READS SQL DATA in its declaration and binary logging is enabled; if non-transactional tables were updated, the binary log will miss their changes
1418	HY000	ER_BINLOG_UNSAFE_ROUTINE	This function has none of DETERMINISTIC, NO SQL, or READS SQL DATA in its declaration and binary logging is enabled (you *might* want to use the less safe log_bin_trust_function_creators variable)
1419	HY000	ER_BINLOG_CREATE_ROUTINE_NEED_SUPER	You do not have the SUPER privilege and binary logging is enabled (you *might* want to use the less safe log_bin_trust_function_creators variable)
1420	HY000	ER_EXEC_STMT_WITH_OPEN_CURSOR	You can't execute a prepared statement which has an open cursor associated with it. Reset the statement to re-execute it.
1421	HY000	ER_STMT_HAS_NO_OPEN_CURSOR	The statement (%lu) has no open cursor.
1422	HY000	ER_COMMIT_NOT_ALLOWED_IN_SF_OR_TRG	Explicit or implicit commit is not allowed in stored function or trigger.
1423	HY000	ER_NO_DEFAULT_FOR_VIEW_FIELD	Field of view '%s.%s' underlying table doesn't have a default value
1424	HY000	ER_SP_NO_RECURSION	Recursive stored functions and triggers are not allowed.
1425	42000	ER_TOO_BIG_SCALE	Too big scale %d specified for column '%s'. Maximum is %lu.
1426	42000	ER_TOO_BIG_PRECISION	Too big precision %d specified for column '%s'. Maximum is %lu.
1427	42000	ER_M_BIGGER_THAN_D	For float(M,D, double(M,D or decimal(M,D, M must be >= D (column '%s').
1428	HY000	ER_WRONG_LOCK_OF_SYSTEM_TABLE	You can't combine write-locking of system tables with other tables or lock types
1429	HY000	ER_CONNECT_TO_FOREIGN_DATA_SOURCE	Unable to connect to foreign data source: %s
1430	HY000	ER_QUERY_ON_FOREIGN_DATA_SOURCE	There was a problem processing the query on the foreign data source. Data source error: %s
1431	HY000	ER_FOREIGN_DATA_SOURCE_DOESNT_EXIST	The foreign data source you are trying to reference does not exist. Data source error: %s
1432	HY000	ER_FOREIGN_DATA_STRING_INVALID_CANT_CREATE	Can't create federated table. The data source connection string '%s' is not in the correct format

1433	HY000	ER_FOREIGN_DATA_STRING_INVALID	The data source connection string '%s' is not in the correct format
1434	HY000	ER_CANT_CREATE_FEDERATED_TABLE	Can't create federated table. Foreign data src error: %s
1435	HY000	ER_TRG_IN_WRONG_SCHEMA	Trigger in wrong schema
1436	HY000	ER_STACK_OVERRUN_NEED_MORE	Thread stack overrun: %ld bytes used of a %ld byte stack, and %ld bytes needed. Use 'mysqld --thread_stack=#' to specify a bigger stack.
1437	42000	ER_TOO_LONG_BODY	Routine body for '%s' is too long
1438	HY000	ER_WARN_CANT_DROP_DEFAULT_KEYCACHE	Cannot drop default keycache
1439	42000	ER_TOO_BIG_DISPLAYWIDTH	Display width out of range for column '%s' (max = %lu)
1440	XAE08	ER_XAER_DUPID	XAER_DUPID: The XID already exists
1441	22008	ER_DATETIME_FUNCTION_OVERFLOW	Datetime function: %s field overflow
1442	HY000	ER_CANT_UPDATE_USED_TABLE_IN_SF_OR_TRG	Can't update table '%s' in stored function/trigger because it is already used by statement which invoked this stored function/trigger.
1443	HY000	ER_VIEW_PREVENT_UPDATE	The definition of table '%s' prevents operation %s on table '%s'.
1444	HY000	ER_PS_NO_RECURSION	The prepared statement contains a stored routine call that refers to that same statement. It's not allowed to execute a prepared statement in such a recursive manner
1445	HY000	ER_SP_CANT_SET_AUTOCOMMIT	Not allowed to set autocommit from a stored function or trigger
1446	HY000	ER_MALFORMED_DEFINER	Definer is not fully qualified
1447	HY000	ER_VIEW_FRM_NO_USER	View '%s'.'%s' has no definer information (old table format). Current user is used as definer. Please recreate the view!
1448	HY000	ER_VIEW_OTHER_USER	You need the SUPER privilege for creation view with '%s'@'%s' definer
1449	HY000	ER_NO_SUCH_USER	The user specified as a definer ('%s'@'%s') does not exist
1450	HY000	ER_FORBID_SCHEMA_CHANGE	Changing schema from '%s' to '%s' is not allowed.
1451	23000	ER_ROW_IS_REFERENCED_2	Cannot delete or update a parent row: a foreign key constraint fails (%s)
1452	23000	ER_NO_REFERENCED_ROW_2	Cannot add or update a child row: a foreign key constraint fails (%s)
1453	42000	ER_SP_BAD_VAR_SHADOW	Variable '%s' must be quoted with `...`, or renamed
1454	HY000	ER_TRG_NO_DEFINER	No definer attribute for trigger '%s'.'%s'. The trigger will be activated under the authorization of the invoker, which may have insufficient privileges. Please recreate the trigger.
1455	HY000	ER_OLD_FILE_FORMAT	'%s' has an old format, you should re-create the '%s' object(s)

1456	HY000	ER_SP_RECURSION_LIMIT	Recursive limit %d (as set by the max_sp_recursion_depth variable) was exceeded for routine %s
1457	HY000	ER_SP_PROC_TABLE_CORRUPT	Failed to load routine %s. The table mysql.proc is missing, corrupt, or contains bad data (internal code %d)
1458	42000	ER_SP_WRONG_NAME	Incorrect routine name '%s'
1459	HY000	ER_TABLE_NEEDS_UPGRADE	Table upgrade required. Please do "REPAIR TABLE `%s`" or dump/reload to fix it!
1460	42000	ER_SP_NO_AGGREGATE	AGGREGATE is not supported for stored functions
1461	42000	ER_MAX_PREPARED_STMT_COUNT_REACHED	Can't create more than max_prepared_stmt_count statements (current value: %lu)
1462	HY000	ER_VIEW_RECURSIVE	`%s`.`%s` contains view recursion
1463	42000	ER_NON_GROUPING_FIELD_USED	Non-grouping field '%s' is used in %s clause
1464	HY000	ER_TABLE_CANT_HANDLE_SPKEYS	The used table type doesn't support SPATIAL indexes
1465	HY000	ER_NO_TRIGGERS_ON_SYSTEM_SCHEMA	Triggers can not be created on system tables
1466	HY000	ER_REMOVED_SPACES	Leading spaces are removed from name '%s'
1467	HY000	ER_AUTOINC_READ_FAILED	Failed to read auto-increment value from storage engine
1468	HY000	ER_USERNAME	user name
1469	HY000	ER_HOSTNAME	host name
1470	HY000	ER_WRONG_STRING_LENGTH	String '%s' is too long for %s (should be no longer than %d)
1471	HY000	ER_NON_INSERTABLE_TABLE	The target table %s of the %s is not insertable-into
1472	HY000	ER_ADMIN_WRONG_MRG_TABLE	Table '%s' is differently defined or of non-MyISAM type or doesn't exist
1473	HY000	ER_TOO_HIGH_LEVEL_OF_NESTING_FOR_SELECT	Too high level of nesting for select
1474	HY000	ER_NAME_BECOMES_EMPTY	Name '%s' has become ''
1475	HY000	ER_AMBIGUOUS_FIELD_TERM	First character of the FIELDS TERMINATED string is ambiguous; please use non-optional and non-empty FIELDS ENCLOSED BY
1476	HY000	ER_FOREIGN_SERVER_EXISTS	The foreign server, %s, you are trying to create already exists.
1477	HY000	ER_FOREIGN_SERVER_DOESNT_EXIST	The foreign server name you are trying to reference does not exist. Data source error: %s
1478	HY000	ER_ILLEGAL_HA_CREATE_OPTION	Table storage engine '%s' does not support the create option '%s'
1479	HY000	ER_PARTITION_REQUIRES_VALUES_ERROR	Syntax error: %s PARTITIONING requires definition of VALUES %s for each partition
1480	HY000	ER_PARTITION_WRONG_VALUES_ERROR	Only %s PARTITIONING can use VALUES %s in partition definition

1481	HY000	ER_PARTITION_MAXVALUE_ERROR	MAXVALUE can only be used in last partition definition
1482	HY000	ER_PARTITION_SUBPARTITION_ERROR	Subpartitions can only be hash partitions and by key
1483	HY000	ER_PARTITION_SUBPART_MIX_ERROR	Must define subpartitions on all partitions if on one partition
1484	HY000	ER_PARTITION_WRONG_NO_PART_ERROR	Wrong number of partitions defined, mismatch with previous setting
1485	HY000	ER_PARTITION_WRONG_NO_SUBPART_ERROR	Wrong number of subpartitions defined, mismatch with previous setting
1486	HY000	ER_CONST_EXPR_IN_PARTITION_FUNC_ERROR	Constant/Random expression in (sub)partitioning function is not allowed
1486	HY000	ER_WRONG_EXPR_IN_PARTITION_FUNC_ERROR	Constant, random or timezone-dependent expressions in (sub)partitioning function are not allowed
1487	HY000	ER_NO_CONST_EXPR_IN_RANGE_OR_LIST_ERROR	Expression in RANGE/LIST VALUES must be constant
1488	HY000	ER_FIELD_NOT_FOUND_PART_ERROR	Field in list of fields for partition function not found in table
1489	HY000	ER_LIST_OF_FIELDS_ONLY_IN_HASH_ERROR	List of fields is only allowed in KEY partitions
1490	HY000	ER_INCONSISTENT_PARTITION_INFO_ERROR	The partition info in the frm file is not consistent with what can be written into the frm file
1491	HY000	ER_PARTITION_FUNC_NOT_ALLOWED_ERROR	The %s function returns the wrong type
1492	HY000	ER_PARTITIONS_MUST_BE_DEFINED_ERROR	For %s partitions each partition must be defined
1493	HY000	ER_RANGE_NOT_INCREASING_ERROR	VALUES LESS THAN value must be strictly increasing for each partition
1494	HY000	ER_INCONSISTENT_TYPE_OF_FUNCTIONS_ERROR	VALUES value must be of same type as partition function
1495	HY000	ER_MULTIPLE_DEF_CONST_IN_LIST_PART_ERROR	Multiple definition of same constant in list partitioning
1496	HY000	ER_PARTITION_ENTRY_ERROR	Partitioning can not be used stand-alone in query
1497	HY000	ER_MIX_HANDLER_ERROR	The mix of handlers in the partitions is not allowed in this version of MariaDB
1498	HY000	ER_PARTITION_NOT_DEFINED_ERROR	For the partitioned engine it is necessary to define all %s
1499	HY000	ER_TOO_MANY_PARTITIONS_ERROR	Too many partitions (including subpartitions) were defined

Error Code	SQLSTATE	Error	Description
1500	HY000	ER_SUBPARTITION_ERROR	It is only possible to mix RANGE/LIST partitioning with HASH/KEY partitioning for subpartitioning
1501	HY000	ER_CANT_CREATE_HANDLER_FILE	Failed to create specific handler file
1502	HY000	ER_BLOB_FIELD_IN_PART_FUNC_ERROR	A BLOB field is not allowed in partition function
1503	HY000	ER_UNIQUE_KEY_NEED_ALL_FIELDS_IN_PF	A %s must include all columns in the table's partitioning function

1504	HY000	ER_NO_PARTS_ERROR	Number of %s = 0 is not an allowed value
1505	HY000	ER_PARTITION_MGMT_ON_NONPARTITIONED	Partition management on a not partitioned table is not possible
1506	HY000	ER_FOREIGN_KEY_ON_PARTITIONED	Foreign key clause is not yet supported in conjunction with partitioning
1507	HY000	ER_DROP_PARTITION_NON_EXISTENT	Error in list of partitions to %s
1508	HY000	ER_DROP_LAST_PARTITION	Cannot remove all partitions, use DROP TABLE instead
1509	HY000	ER_COALESCE_ONLY_ON_HASH_PARTITION	COALESCE PARTITION can only be used on HASH/KEY partitions
1510	HY000	ER_REORG_HASH_ONLY_ON_SAME_N	REORGANIZE PARTITION can only be used to reorganize partitions not to change their numbers
1511	HY000	ER_REORG_NO_PARAM_ERROR	REORGANIZE PARTITION without parameters can only be used on auto-partitioned tables using HASH PARTITIONs
1512	HY000	ER_ONLY_ON_RANGE_LIST_PARTITION	%s PARTITION can only be used on RANGE/LIST partitions
1513	HY000	ER_ADD_PARTITION_SUBPART_ERROR	Trying to Add partition(s) with wrong number of subpartitions
1514	HY000	ER_ADD_PARTITION_NO_NEW_PARTITION	At least one partition must be added
1515	HY000	ER_COALESCE_PARTITION_NO_PARTITION	At least one partition must be coalesced
1516	HY000	ER_REORG_PARTITION_NOT_EXIST	More partitions to reorganize than there are partitions
1517	HY000	ER_SAME_NAME_PARTITION	Duplicate partition name %s
1518	HY000	ER_NO_BINLOG_ERROR	It is not allowed to shut off binlog on this command
1519	HY000	ER_CONSECUTIVE_REORG_PARTITIONS	When reorganizing a set of partitions they must be in consecutive order
1520	HY000	ER_REORG_OUTSIDE_RANGE	Reorganize of range partitions cannot change total ranges except for last partition where it can extend the range
1521	HY000	ER_PARTITION_FUNCTION_FAILURE	Partition function not supported in this version for this handler
1522	HY000	ER_PART_STATE_ERROR	Partition state cannot be defined from CREATE/ALTER TABLE
1523	HY000	ER_LIMITED_PART_RANGE	The %s handler only supports 32 bit integers in VALUES
1524	HY000	ER_PLUGIN_IS_NOT_LOADED	Plugin '%s' is not loaded
1525	HY000	ER_WRONG_VALUE	Incorrect %s value: '%s'
1526	HY000	ER_NO_PARTITION_FOR_GIVEN_VALUE	Table has no partition for value %s
1527	HY000	ER_FILEGROUP_OPTION_ONLY_ONCE	It is not allowed to specify %s more than once
1528	HY000	ER_CREATE_FILEGROUP_FAILED	Failed to create %s
1529	HY000	ER_DROP_FILEGROUP_FAILED	Failed to drop %s
1530	HY000	ER_TABLESPACE_AUTO_EXTEND_ERROR	The handler doesn't support autoextend of tablespaces

1531	HY000	ER_WRONG_SIZE_NUMBER	A size parameter was incorrectly specified, either number or on the form 10M
1532	HY000	ER_SIZE_OVERFLOW_ERROR	The size number was correct but we don't allow the digit part to be more than 2 billion
1533	HY000	ER_ALTER_FILEGROUP_FAILED	Failed to alter: %s
1534	HY000	ER_BINLOG_ROW_LOGGING_FAILED	Writing one row to the row-based binary log failed
1535	HY000	ER_BINLOG_ROW_WRONG_TABLE_DEF	Table definition on master and slave does not match: %s
1536	HY000	ER_BINLOG_ROW_RBR_TO_SBR	Slave running with --log-slave-updates must use row-based binary logging to be able to replicate row-based binary log events
1537	HY000	ER_EVENT_ALREADY_EXISTS	Event '%s' already exists
1538	HY000	ER_EVENT_STORE_FAILED	Failed to store event %s. Error code %d from storage engine.
1539	HY000	ER_EVENT_DOES_NOT_EXIST	Unknown event '%s'
1540	HY000	ER_EVENT_CANT_ALTER	Failed to alter event '%s'
1541	HY000	ER_EVENT_DROP_FAILED	Failed to drop %s
1542	HY000	ER_EVENT_INTERVAL_NOT_POSITIVE_OR_TOO_BIG	INTERVAL is either not positive or too big
1543	HY000	ER_EVENT_ENDS_BEFORE_STARTS	ENDS is either invalid or before STARTS
1544	HY000	ER_EVENT_EXEC_TIME_IN_THE_PAST	Event execution time is in the past. Event has been disabled
1545	HY000	ER_EVENT_OPEN_TABLE_FAILED	Failed to open mysql.event
1546	HY000	ER_EVENT_NEITHER_M_EXPR_NOR_M_AT	No datetime expression provided
1547	HY000	ER_COL_COUNT_DOESNT_MATCH_CORRUPTED	Column count of mysql.%s is wrong. Expected %d, found %d. The table is probably corrupted
1548	HY000	ER_CANNOT_LOAD_FROM_TABLE	Cannot load from mysql.%s. The table is probably corrupted
1549	HY000	ER_EVENT_CANNOT_DELETE	Failed to delete the event from mysql.event
1550	HY000	ER_EVENT_COMPILE_ERROR	Error during compilation of event's body
1551	HY000	ER_EVENT_SAME_NAME	Same old and new event name
1552	HY000	ER_EVENT_DATA_TOO_LONG	Data for column '%s' too long
1553	HY000	ER_DROP_INDEX_FK	Cannot drop index '%s': needed in a foreign key constraint
1554	HY000	ER_WARN_DEPRECATED_SYNTAX_WITH_VER	The syntax '%s' is deprecated and will be removed in MariaDB %s. Please use %s instead
1555	HY000	ER_CANT_WRITE_LOCK_LOG_TABLE	You can't write-lock a log table. Only read access is possible
1556	HY000	ER_CANT_LOCK_LOG_TABLE	You can't use locks with log tables.
1557	23000	ER_FOREIGN_DUPLICATE_KEY	Upholding foreign key constraints for table '%s', entry '%s', key %d would lead to a duplicate entry

1558	HY000	ER_COL_COUNT_DOESNT_MATCH_PLEASE_UPDATE	Column count of mysql.%s is wrong. Expected %d, found %d. Created with MariaDB %d, now running %d. Please use mysql_upgrade to fix this error.
1559	HY000	ER_TEMP_TABLE_PREVENTS_SWITCH_OUT_OF_RBR	Cannot switch out of the row-based binary log format when the session has open temporary tables
1560	HY000	ER_STORED_FUNCTION_ PREVENTS_SWITCH_BINLOG_FORMAT	Cannot change the binary logging format inside a stored function or trigger
1561		ER_UNUSED_13	You should never see it
1562	HY000	ER_PARTITION_NO_TEMPORARY	Cannot create temporary table with partitions
1563	HY000	ER_PARTITION_CONST_DOMAIN_ERROR	Partition constant is out of partition function domain
1564	HY000	ER_PARTITION_FUNCTION_IS_NOT_ALLOWED	This partition function is not allowed
1565	HY000	ER_DDL_LOG_ERROR	Error in DDL log
1566	HY000	ER_NULL_IN_VALUES_LESS_THAN	Not allowed to use NULL value in VALUES LESS THAN
1567	HY000	ER_WRONG_PARTITION_NAME	Incorrect partition name
1568	25001	ER_CANT_CHANGE_TX_ISOLATION	Transaction isolation level can't be changed while a transaction is in progress
1569	HY000	ER_DUP_ENTRY_AUTOINCREMENT_CASE	ALTER TABLE causes auto_increment resequencing, resulting in duplicate entry '%s' for key '%s'
1570	HY000	ER_EVENT_MODIFY_QUEUE_ERROR	Internal scheduler error %d
1571	HY000	ER_EVENT_SET_VAR_ERROR	Error during starting/stopping of the scheduler. Error code %u
1572	HY000	ER_PARTITION_MERGE_ERROR	Engine cannot be used in partitioned tables
1573	HY000	ER_CANT_ACTIVATE_LOG	Cannot activate '%s' log
1574	HY000	ER_RBR_NOT_AVAILABLE	The server was not built with row-based replication
1575	HY000	ER_BASE64_DECODE_ERROR	Decoding of base64 string failed
1576	HY000	ER_EVENT_RECURSION_FORBIDDEN	Recursion of EVENT DDL statements is forbidden when body is present
1577	HY000	ER_EVENTS_DB_ERROR	Cannot proceed because system tables used by Event Scheduler were found damaged at server start
1578	HY000	ER_ONLY_INTEGERS_ALLOWED	Only integers allowed as number here
1579	HY000	ER_UNSUPORTED_LOG_ENGINE	This storage engine cannot be used for log tables"
1580	HY000	ER_BAD_LOG_STATEMENT	You cannot '%s' a log table if logging is enabled
1581	HY000	ER_CANT_RENAME_LOG_TABLE	Cannot rename '%s'. When logging enabled, rename to/from log table must rename two tables: the log table to an archive table and another table back to '%s'
1582	42000	ER_WRONG_PARAMCOUNT_TO_NATIVE_FCT	Incorrect parameter count in the call to native function '%s'

1583	42000	ER_WRONG_PARAMETERS_TO_NATIVE_FCT	Incorrect parameters in the call to native function '%s'
1584	42000	ER_WRONG_PARAMETERS_TO_STORED_FCT	Incorrect parameters in the call to stored function '%s'
1585	HY000	ER_NATIVE_FCT_NAME_COLLISION	This function '%s' has the same name as a native function
1586	23000	ER_DUP_ENTRY_WITH_KEY_NAME	Duplicate entry '%s' for key '%s'
1587	HY000	ER_BINLOG_PURGE_EMFILE	Too many files opened, please execute the command again
1588	HY000	ER_EVENT_CANNOT_CREATE_IN_THE_PAST	Event execution time is in the past and ON COMPLETION NOT PRESERVE is set. The event was dropped immediately after creation.
1589	HY000	ER_EVENT_CANNOT_ALTER_IN_THE_PAST	Event execution time is in the past and ON COMPLETION NOT PRESERVE is set. The event was dropped immediately after creation.
1590	HY000	ER_SLAVE_INCIDENT	The incident %s occured on the master. Message: %s
1591	HY000	ER_NO_PARTITION_FOR_GIVEN_VALUE_SILENT	Table has no partition for some existing values
1592	HY000	ER_BINLOG_UNSAFE_STATEMENT	Unsafe statement written to the binary log using statement format since BINLOG_FORMAT = STATEMENT. %s
1593	HY000	ER_SLAVE_FATAL_ERROR	Fatal error: %s
1594	HY000	ER_SLAVE_RELAY_LOG_READ_FAILURE	Relay log read failure: %s
1595	HY000	ER_SLAVE_RELAY_LOG_WRITE_FAILURE	Relay log write failure: %s
1596	HY000	ER_SLAVE_CREATE_EVENT_FAILURE	Failed to create %s
1597	HY000	ER_SLAVE_MASTER_COM_FAILURE	Master command %s failed: %s
1598	HY000	ER_BINLOG_LOGGING_IMPOSSIBLE	Binary logging not possible. Message: %s
1599	HY000	ER_VIEW_NO_CREATION_CTX	View `%s`.`%s` has no creation context

Error Code	SQLSTATE	Error	Description
1600	HY000	ER_VIEW_INVALID_CREATION_CTX	Creation context of view `%s`.`%s' is invalid
1601	HY000	ER_SR_INVALID_CREATION_CTX	Creation context of stored routine `%s`.`%s` is inva
1602	HY000	ER_TRG_CORRUPTED_FILE	Corrupted TRG file for table `%s`.`%s`
1603	HY000	ER_TRG_NO_CREATION_CTX	Triggers for table `%s`.`%s` have no creation conte
1604	HY000	ER_TRG_INVALID_CREATION_CTX	Trigger creation context of table `%s`.`%s` is invalic
1605	HY000	ER_EVENT_INVALID_CREATION_CTX	Creation context of event `%s`.`%s` is invalid
1606	HY000	ER_TRG_CANT_OPEN_TABLE	Cannot open table for trigger `%s`.`%s`
1607	HY000	ER_CANT_CREATE_SROUTINE	Cannot create stored routine `%s`. Check warnings
1608		ER_UNUSED_11	You should never see it
1609	HY000	ER_NO_FORMAT_DESCRIPTION_EVENT _BEFORE_BINLOG_STATEMENT	The BINLOG statement of type `%s` was not preceded by a format description BINLOG statemer
1610	HY000	ER_SLAVE_CORRUPT_EVENT	Corrupted replication event was detected
1611	HY000	ER_LOAD_DATA_INVALID_COLUMN	Invalid column reference (%s) in LOAD DATA
1612	HY000	ER_LOG_PURGE_NO_FILE	Being purged log %s was not found
1613	XA106	ER_XA_RBTIMEOUT	XA_RBTIMEOUT: Transaction branch was rolled back: took too long

1614	XA102	ER_XA_RBDEADLOCK	XA_RBDEADLOCK: Transaction branch was rolled back: deadlock was detected
1615	HY000	ER_NEED_REPREPARE	Prepared statement needs to be re-prepared
1616	HY000	ER_DELAYED_NOT_SUPPORTED	DELAYED option not supported for table '%s'
1617	HY000	WARN_NO_MASTER_INF	The master info structure does not exist
1618	HY000	WARN_OPTION_IGNORED	<%s> option ignored
1619	HY000	WARN_PLUGIN_DELETE_BUILTIN	Built-in plugins cannot be deleted
1620	HY000	WARN_PLUGIN_BUSY	Plugin is busy and will be uninstalled on shutdown
1621	HY000	ER_VARIABLE_IS_READONLY	%s variable '%s' is read-only. Use SET %s to assig the value
1622	HY000	ER_WARN_ENGINE_TRANSACTION_ROLLBACK	Storage engine %s does not support rollback for thi statement. Transaction rolled back and must be restarted
1623	HY000	ER_SLAVE_HEARTBEAT_FAILURE	Unexpected master's heartbeat data: %s
1624	HY000	ER_SLAVE_HEARTBEAT_VALUE_OUT_OF_RANGE	The requested value for the heartbeat period is eith negative or exceeds the maximum allowed (%s seconds).
1625		ER_UNUSED_14	You should never see it
1626	HY000	ER_CONFLICT_FN_PARSE_ERROR	Error in parsing conflict function. Message: %s
1627	HY000	ER_EXCEPTIONS_WRITE_ERROR	Write to exceptions table failed. Message: %s"
1628	HY000	ER_TOO_LONG_TABLE_COMMENT	Comment for table '%s' is too long (max = %lu)
1629	HY000	ER_TOO_LONG_FIELD_COMMENT	Comment for field '%s' is too long (max = %lu)
1630	42000	ER_FUNC_INEXISTENT_NAME_COLLISION	FUNCTION %s does not exist. Check the 'Function Name Parsing and Resolution' section in the Reference Manual
1631	HY000	ER_DATABASE_NAME	Database
1632	HY000	ER_TABLE_NAME	Table
1633	HY000	ER_PARTITION_NAME	Partition
1634	HY000	ER_SUBPARTITION_NAME	Subpartition
1635	HY000	ER_TEMPORARY_NAME	Temporary
1636	HY000	ER_RENAMED_NAME	Renamed
1637	HY000	ER_TOO_MANY_CONCURRENT_TRXS	Too many active concurrent transactions
1638	HY000	WARN_NON_ASCII_SEPARATOR_NOT_IMPLEMENTED	Non-ASCII separator arguments are not fully supported
1639	HY000	ER_DEBUG_SYNC_TIMEOUT	debug sync point wait timed out
1640	HY000	ER_DEBUG_SYNC_HIT_LIMIT	debug sync point hit limit reached
1641	42000	ER_DUP_SIGNAL_SET	Duplicate condition information item '%s'
1642	01000	ER_SIGNAL_WARN	Unhandled user-defined warning condition
1643	02000	ER_SIGNAL_NOT_FOUND	Unhandled user-defined not found condition
1644	HY000	ER_SIGNAL_EXCEPTION	Unhandled user-defined exception condition
1645	0K000	ER_RESIGNAL_WITHOUT_ACTIVE_HANDLER	RESIGNAL when handler not active
1646	HY000	ER_SIGNAL_BAD_CONDITION_TYPE	SIGNAL/RESIGNAL can only use a CONDITION defined with SQLSTATE
1647	HY000	WARN_COND_ITEM_TRUNCATED	Data truncated for condition item '%s'
1648	HY000	ER_COND_ITEM_TOO_LONG	Data too long for condition item '%s'
1649	HY000	ER_UNKNOWN_LOCALE	Unknown locale: '%s'
1650	HY000	ER_SLAVE_IGNORE_SERVER_IDS	The requested server id %d clashes with the slave startup option --replicate-same-server-id
1651	HY000	ER_QUERY_CACHE_DISABLED	Query cache is disabled; restart the server with query_cache_type=1 to enable it

1652	HY000	ER_SAME_NAME_PARTITION_FIELD	Duplicate partition field name '%s'
1653	HY000	ER_PARTITION_COLUMN_LIST_ERROR	Inconsistency in usage of column lists for partitionir
1654	HY000	ER_WRONG_TYPE_COLUMN_VALUE_ERROR	Partition column values of incorrect type
1655	HY000	ER_TOO_MANY_PARTITION_FUNC_FIELDS_ERROR	Too many fields in '%s'
1656	HY000	ER_MAXVALUE_IN_VALUES_IN	Cannot use MAXVALUE as value in VALUES IN
1657	HY000	ER_TOO_MANY_VALUES_ERROR	Cannot have more than one value for this type of % partitioning
1658	HY000	ER_ROW_SINGLE_PARTITION_FIELD_ERROR	Row expressions in VALUES IN only allowed for multi-field column partitioning
1659	HY000	ER_FIELD_TYPE_NOT_ALLOWED_AS_PARTITION_FIELD	Field '%s' is of a not allowed type for this type of partitioning
1660	HY000	ER_PARTITION_FIELDS_TOO_LONG	The total length of the partitioning fields is too large
1661	HY000	ER_BINLOG_ROW_ENGINE_AND_STMT_ENGINE	Cannot execute statement: impossible to write to binary log since both row-incapable engines and statement-incapable engines are involved.
1662	HY000	ER_BINLOG_ROW_MODE_AND_STMT_ENGINE	Cannot execute statement: impossible to write to binary log since BINLOG_FORMAT = ROW and at least one table uses a storage engine limited to statement-based logging.
1663	HY000	ER_BINLOG_UNSAFE_AND_STMT_ENGINE	Cannot execute statement: impossible to write to binary log since statement is unsafe, storage engin is limited to statement-based logging, and BINLOG_FORMAT = MIXED. %s
1664	HY000	ER_BINLOG_ROW_INJECTION_AND_STMT_ENGINE	Cannot execute statement: impossible to write to binary log since statement is in row format and at least one table uses a storage engine limited to statement-based logging.
1665	HY000	ER_BINLOG_STMT_MODE_AND_ROW_ENGINE	Cannot execute statement: impossible to write to binary log since BINLOG_FORMAT = STATEMEN and at least one table uses a storage engine limitec row-based logging.%s
1666	HY000	ER_BINLOG_ROW_INJECTION_AND_STMT_MODE	Cannot execute statement: impossible to write to binary log since statement is in row format and BINLOG_FORMAT = STATEMENT.
1667	HY000	ER_BINLOG_MULTIPLE_ENGINES _AND_SELF_LOGGING_ENGINE	Cannot execute statement: impossible to write to binary log since more than one engine is involved a at least one engine is self-logging.
1668	HY000	ER_BINLOG_UNSAFE_LIMIT	The statement is unsafe because it uses a LIMIT clause. This is unsafe because the set of rows included cannot be predicted.
1669	HY000	ER_BINLOG_UNSAFE_INSERT_DELAYED	The statement is unsafe because it uses INSERT DELAYED. This is unsafe because the times when rows are inserted cannot be predicted.
1670	HY000	ER_BINLOG_UNSAFE_SYSTEM_TABLE	The statement is unsafe because it uses the genera log, slow query log, or performance_schema table(s This is unsafe because system tables may differ on slaves.
1671	HY000	ER_BINLOG_UNSAFE_AUTOINC_COLUMNS	Statement is unsafe because it invokes a trigger or stored function that inserts into an AUTO_INCREMENT column. Inserted values cann be logged correctly.
1672	HY000	ER_BINLOG_UNSAFE_UDF	Statement is unsafe because it uses a UDF which may not return the same value on the slave.
1673	HY000	ER_BINLOG_UNSAFE_SYSTEM_VARIABLE	Statement is unsafe because it uses a system variable that may have a different value on the slav
1674	HY000	ER_BINLOG_UNSAFE_SYSTEM_FUNCTION	Statement is unsafe because it uses a system function that may return a different value on the sla

1675	HY000	ER_BINLOG_UNSAFE_NONTRANS_AFTER_TRANS	Statement is unsafe because it accesses a non-transactional table after accessing a transactional table within the same transaction.
1676	HY000	ER_MESSAGE_AND_STATEMENT	%s Statement: %s
1677	HY000	ER_SLAVE_CONVERSION_FAILED	Column %d of table '%s.%s' cannot be converted from type '%s' to type '%s'
1678	HY000	ER_SLAVE_CANT_CREATE_CONVERSION	Can't create conversion table for table '%s.%s'
1679	HY000	ER_INSIDE_TRANSACTION _PREVENTS_SWITCH_BINLOG_FORMAT	Cannot modify @@session.binlog_format inside a transaction
1680	HY000	ER_PATH_LENGTH	The path specified for %s is too long.
1681	HY000	ER_WARN_DEPRECATED_SYNTAX_NO_REPLACEMENT	'%s' is deprecated and will be removed in a future release.
1682	HY000	ER_WRONG_NATIVE_TABLE_STRUCTURE	Native table '%s'.'%s' has the wrong structure
1683	HY000	ER_WRONG_PERFSCHEMA_USAGE	Invalid performance_schema usage.
1684	HY000	ER_WARN_I_S_SKIPPED_TABLE	Table '%s'.'%s' was skipped since its definition is being modified by concurrent DDL statement
1685	HY000	ER_INSIDE_TRANSACTION _PREVENTS_SWITCH_BINLOG_DIRECT	Cannot modify @@session.binlog_direct_non_transactional_upda inside a transaction
1686	HY000	ER_STORED_FUNCTION_PREVENTS _SWITCH_BINLOG_DIRECT	Cannot change the binlog direct flag inside a storec function or trigger
1687	42000	ER_SPATIAL_MUST_HAVE_GEOM_COL	A SPATIAL index may only contain a geometrical ty column
1688	HY000	ER_TOO_LONG_INDEX_COMMENT	Comment for index '%s' is too long (max = %lu)
1689	HY000	ER_LOCK_ABORTED	Wait on a lock was aborted due to a pending exclusive lock
1690	22003	ER_DATA_OUT_OF_RANGE	%s value is out of range in '%s'
1691	HY000	ER_WRONG_SPVAR_TYPE_IN_LIMIT	A variable of a non-integer based type in LIMIT clause
1692	HY000	ER_BINLOG_UNSAFE_MULTIPLE_ENGINES _AND_SELF_LOGGING_ENGINE	Mixing self-logging and non-self-logging engines in statement is unsafe.
1693	HY000	ER_BINLOG_UNSAFE_MIXED_STATEMENT	Statement accesses nontransactional table as well transactional or temporary table, and writes to any them.
1694	HY000	ER_INSIDE_TRANSACTION_ PREVENTS_SWITCH_SQL_LOG_BIN	Cannot modify @@session.sql_log_bin inside a transaction
1695	HY000	ER_STORED_FUNCTION_ PREVENTS_SWITCH_SQL_LOG_BIN	Cannot change the sql_log_bin inside a stored function or trigger
1696	HY000	ER_FAILED_READ_FROM_PAR_FILE	Failed to read from the .par file
1697	HY000	ER_VALUES_IS_NOT_INT_TYPE_ERROR	VALUES value for partition '%s' must have type INT
1698	28000	ER_ACCESS_DENIED_NO_PASSWORD_ERROR	Access denied for user '%s'@'%s'
1699	HY000	ER_SET_PASSWORD_AUTH_PLUGIN	SET PASSWORD has no significance for users authenticating via plugins

Error Code	SQLSTATE	Error	Description
1700	HY000	ER_GRANT_PLUGIN_USER_EXISTS	GRANT with IDENTIFIED WITH is illegal because the user %-.*s already exists
1701	42000	ER_TRUNCATE_ILLEGAL_FK	Cannot truncate a table referenced in a for key constraint (%s)
1702	HY000	ER_PLUGIN_IS_PERMANENT	Plugin '%s' is force_plus_permanent and ca be unloaded
1703	HY000	ER_SLAVE_HEARTBEAT_VALUE_OUT_OF_RANGE_MIN	The requested value for the heartbeat perio less than 1 millisecond. The value is reset t meaning that heartbeating will effectively b disabled.

1704	HY000	ER_SLAVE_HEARTBEAT_VALUE_OUT_OF_RANGE_MAX	The requested value for the heartbeat peric exceeds the value of slave_net_timeout se A sensible value for the period should be le than the timeout.
1705	HY000	ER_STMT_CACHE_FULL	Multi-row statements required more than 'max_binlog_stmt_cache_size' bytes of sto increase this mysqld variable and try again
1706	HY000	ER_MULTI_UPDATE_KEY_CONFLICT	Primary key/partition key update is not allo since the table is updated both as '%s' and
1707	HY000	ER_TABLE_NEEDS_REBUILD	Table rebuild required. Please do "ALTER `%s` FORCE" or dump/reload to fix it!
1708	HY000	WARN_OPTION_BELOW_LIMIT	The value of '%s' should be no less than th value of '%s'
1709	HY000	ER_INDEX_COLUMN_TOO_LONG	Index column size too large. The maximum column size is %lu bytes.
1710	HY000	ER_ERROR_IN_TRIGGER_BODY	Trigger '%s' has an error in its body: '%s'
1711	HY000	ER_ERROR_IN_UNKNOWN_TRIGGER_BODY	Unknown trigger has an error in its body: '%
1712	HY000	ER_INDEX_CORRUPT	Index %s is corrupted
1713	HY000	ER_UNDO_RECORD_TOO_BIG	Undo log record is too big.
1714	HY000	ER_BINLOG_UNSAFE_INSERT_IGNORE_SELECT	INSERT IGNORE... SELECT is unsafe bec the order in which rows are retrieved by the SELECT determines which (if any) rows ar ignored. This order cannot be predicted an differ on master and the slave.
1715	HY000	ER_BINLOG_UNSAFE_INSERT_SELECT_UPDATE	INSERT... SELECT... ON DUPLICATE KE UPDATE is unsafe because the order in wh rows are retrieved by the SELECT determi which (if any) rows are updated. This order cannot be predicted and may differ on mas the slave.
1716	HY000	ER_BINLOG_UNSAFE_REPLACE_SELECT	REPLACE... SELECT is unsafe because th order in which rows are retrieved by the SE determines which (if any) rows are replace order cannot be predicted and may differ o master and the slave.
1717	HY000	ER_BINLOG_UNSAFE_CREATE_IGNORE_SELECT	CREATE... IGNORE SELECT is unsafe be the order in which rows are retrieved by the SELECT determines which (if any) rows ar ignored. This order cannot be predicted an differ on master and the slave.
1718	HY000	ER_BINLOG_UNSAFE_CREATE_REPLACE_SELECT	CREATE... REPLACE SELECT is unsafe because the order in which rows are retriev the SELECT determines which (if any) row replaced. This order cannot be predicted a may differ on master and the slave.
1719	HY000	ER_BINLOG_UNSAFE_UPDATE_IGNORE	UPDATE IGNORE is unsafe because the o in which rows are updated determines whic any) rows are ignored. This order cannot b predicted and may differ on master and the
1720		ER_UNUSED_15	You should never see it
1721		ER_UNUSED_16	You should never see it
1722	HY000	ER_BINLOG_UNSAFE_WRITE_AUTOINC_SELECT	Statements writing to a table with an auto-increment column after selecting from anot table are unsafe because the order in whic are retrieved determines what (if any) rows written. This order cannot be predicted and differ on master and the slave.
1723	HY000	ER_BINLOG_UNSAFE_CREATE_SELECT_AUTOINC	CREATE TABLE... SELECT... on a table w auto-increment column is unsafe because order in which rows are retrieved by the SE determines which (if any) rows are inserted order cannot be predicted and may differ o master and the slave.
1724	HY000	ER_BINLOG_UNSAFE_INSERT_TWO_KEYS	INSERT... ON DUPLICATE KEY UPDATE table with more than one UNIQUE KEY is u
1725	HY000	ER_TABLE_IN_FK_CHECK	Table is being used in foreign key check.
1726	HY000	ER_UNSUPPORTED_ENGINE	Storage engine '%s' does not support syste tables. [%s.%s]
1727	HY000	ER_BINLOG_UNSAFE_AUTOINC_NOT_FIRST	INSERT into autoincrement field which is n first part in the composed primary key is un
1728	HY000	ER_CANNOT_LOAD_FROM_TABLE_V2	Cannot load from %s.%s. The table is prob corrupted

1729	HY000	ER_MASTER_DELAY_VALUE_OUT_OF_RANGE	The requested value %s for the master del exceeds the maximum %u
1730	HY000	ER_ONLY_FD_AND_RBR_EVENTS_ALLOWED_IN_BINLOG_STATEMENT	Only Format_description_log_event and ro events are allowed in BINLOG statements %s was provided
1731	HY000	ER_PARTITION_EXCHANGE_DIFFERENT_OPTION	Non matching attribute '%s' between partiti and table
1732	HY000	ER_PARTITION_EXCHANGE_PART_TABLE	Table to exchange with partition is partition '%s'
1733	HY000	ER_PARTITION_EXCHANGE_TEMP_TABLE	Table to exchange with partition is tempora '%s'
1734	HY000	ER_PARTITION_INSTEAD_OF_SUBPARTITION	Subpartitioned table, use subpartition inste partition
1735	HY000	ER_UNKNOWN_PARTITION	Unknown partition '%s' in table '%s'
1736	HY000	ER_TABLES_DIFFERENT_METADATA	Tables have different definitions
1737	HY000	ER_ROW_DOES_NOT_MATCH_PARTITION	Found a row that does not match the partiti
1738	HY000	ER_BINLOG_CACHE_SIZE_GREATER_THAN_MAX	Option binlog_cache_size (%lu) is greater t max_binlog_cache_size (%lu); setting binlog_cache_size equal to max_binlog_cache_size.
1739	HY000	ER_WARN_INDEX_NOT_APPLICABLE	Cannot use %s access on index '%s' due t or collation conversion on field '%s'
1740	HY000	ER_PARTITION_EXCHANGE_FOREIGN_KEY	Table to exchange with partition has foreig references: '%s'
1741	HY000	ER_NO_SUCH_KEY_VALUE	Key value '%s' was not found in table '%s.%
1742	HY000	ER_RPL_INFO_DATA_TOO_LONG	Data for column '%s' too long
1743	HY000	ER_NETWORK_READ_EVENT_CHECKSUM_FAILURE	Replication event checksum verification fai while reading from network.
1744	HY000	ER_BINLOG_READ_EVENT_CHECKSUM_FAILURE	Replication event checksum verification fai while reading from a log file.
1745	HY000	ER_BINLOG_STMT_CACHE_SIZE_GREATER_THAN_MAX	Option binlog_stmt_cache_size (%lu) is gre than max_binlog_stmt_cache_size (%lu); s binlog_stmt_cache_size equal to max_binlog_stmt_cache_size.
1746	HY000	ER_CANT_UPDATE_TABLE_IN_CREATE_TABLE_SELECT	Can't update table '%s' while '%s' is being created.
1747	HY000	ER_PARTITION_CLAUSE_ON_NONPARTITIONED	PARTITION () clause on non partitioned ta
1748	HY000	ER_ROW_DOES_NOT_MATCH_GIVEN_PARTITION_SET	Found a row not matching the given partitio
1749	HY000	ER_NO_SUCH_PARTITION_UNUSED	partition '%s' doesn't exist
1750	HY000	ER_CHANGE_RPL_INFO_REPOSITORY_FAILURE	Failure while changing the type of replicatio repository: %s.
1751	HY000	ER_WARNING_NOT_COMPLETE_ROLLBACK_WITH_CREATED_TEMP_TABLE	The creation of some temporary tables cou be rolled back.
1752	HY000	ER_WARNING_NOT_COMPLETE_ROLLBACK_WITH_DROPPED_TEMP_TABLE	Some temporary tables were dropped, but operations could not be rolled back.
1753	HY000	ER_MTS_FEATURE_IS_NOT_SUPPORTED	%s is not supported in multi-threaded slave mode. %s
1754	HY000	ER_MTS_UPDATED_DBS_GREATER_MAX	The number of modified databases exceed maximum %d; the database names will not included in the replication event metadata.
1755	HY000	ER_MTS_CANT_PARALLEL	Cannot execute the current event group in parallel mode. Encountered event %s, rela name %s, position %s which prevents exec of this event group in parallel mode. Reaso
1756	HY000	ER_MTS_INCONSISTENT_DATA	%s
1757	HY000	ER_FULLTEXT_NOT_SUPPORTED_WITH_PARTITIONING	FULLTEXT index is not supported for partit tables.
1758	35000	ER_DA_INVALID_CONDITION_NUMBER	Invalid condition number
1759	HY000	ER_INSECURE_PLAIN_TEXT	Sending passwords in plain text without SS is extremely insecure.
1760	HY000	ER_INSECURE_CHANGE_MASTER	Storing MySQL user name or password information in the master info repository is secure and is therefore not recommended. Please consider using the USER and PASSWORD connection options for STAR SLAVE; see the 'START SLAVE Syntax' in MySQL Manual for more information.

1761	23000	ER_FOREIGN_DUPLICATE_KEY_WITH_CHILD_INFO	Foreign key constraint for table '%s', record would lead to a duplicate entry in table '%s '%s'
1762	23000	ER_FOREIGN_DUPLICATE_KEY_WITHOUT_CHILD_INFO	Foreign key constraint for table '%s', record would lead to a duplicate entry in a child ta
1763	HY000	ER_SQLTHREAD_WITH_SECURE_SLAVE	Setting authentication options is not possib when only the Slave SQL Thread is being started.
1764	HY000	ER_TABLE_HAS_NO_FT	The table does not have FULLTEXT index support this query
1765	HY000	ER_VARIABLE_NOT_SETTABLE_IN_SF_OR_TRIGGER	The system variable %s cannot be set in st functions or triggers.
1766	HY000	ER_VARIABLE_NOT_SETTABLE_IN_TRANSACTION	The system variable %s cannot be set whe there is an ongoing transaction.
1767	HY000	ER_GTID_NEXT_IS_NOT_IN_GTID_NEXT_LIST	The system variable @@SESSION.GTID_ has the value %s, which is not listed in @@SESSION.GTID_NEXT_LIST.
1768	HY000	ER_CANT_CHANGE_GTID_NEXT_IN_TRANSACTION_WHEN_GTID_NEXT_LIST_IS_NULL	The system variable @@SESSION.GTID_ cannot change inside a transaction.
1769	HY000	ER_SET_STATEMENT_CANNOT_INVOKE_FUNCTION	The statement 'SET %s' cannot invoke a st function.
1770	HY000	ER_GTID_NEXT_CANT_BE_AUTOMATIC_IF_GTID_NEXT_LIST_IS_NON_NULL	The system variable @@SESSION.GTID_ cannot be 'AUTOMATIC' when @@SESSION.GTID_NEXT_LIST is non-N
1771	HY000	ER_SKIPPING_LOGGED_TRANSACTION	Skipping transaction %s because it has alr been executed and logged.
1772	HY000	ER_MALFORMED_GTID_SET_SPECIFICATION	Malformed GTID set specification '%s'.
1773	HY000	ER_MALFORMED_GTID_SET_ENCODING	Malformed GTID set encoding.
1774	HY000	ER_MALFORMED_GTID_SPECIFICATION	Malformed GTID specification '%s'.
1775	HY000	ER_GNO_EXHAUSTED	Impossible to generate Global Transaction Identifier: the integer component reached th maximal value. Restart the server with a ne server_uuid.
1776	HY000	ER_BAD_SLAVE_AUTO_POSITION	Parameters MASTER_LOG_FILE, MASTER_LOG_POS, RELAY_LOG_FILE RELAY_LOG_POS cannot be set when MASTER_AUTO_POSITION is active.
1777	HY000	ER_AUTO_POSITION_REQUIRES_GTID_MODE_ON	CHANGE MASTER TO MASTER_AUTO_POSITION = 1 can only executed when @@GLOBAL.GTID_MODE ON.
1778	HY000	ER_CANT_DO_IMPLICIT_COMMIT_IN_TRX_WHEN_GTID_NEXT_IS_SET	Cannot execute statements with implicit co inside a transaction when @@SESSION.GTID_NEXT != AUTOMATI
1779	HY000	ER_GTID_MODE_2_OR_3_REQUIRES_DISABLE_GTID_UNSAFE_STATEMENTS_ON	GTID_MODE = ON or GTID_MODE = UPGRADE_STEP_2 requires DISABLE_GTID_UNSAFE_STATEMENTS
1779	HY000	ER_GTID_MODE_2_OR_3_REQUIRES_ENFORCE_GTID_CONSISTENCY_ON	@@GLOBAL.GTID_MODE = ON or UPGRADE_STEP_2 requires @@GLOBAL.ENFORCE_GTID_CONSIST = 1.
1780	HY000	ER_GTID_MODE_REQUIRES_BINLOG	@@GLOBAL.GTID_MODE = ON or UPGRADE_STEP_1 or UPGRADE_STEP requires --log-bin and --log-slave-updates.
1781	HY000	ER_CANT_SET_GTID_NEXT_TO_GTID_WHEN_GTID_MODE_IS_OFF	@@SESSION.GTID_NEXT cannot be set UUID:NUMBER when @@GLOBAL.GTID_MODE = OFF.
1782	HY000	ER_CANT_SET_GTID_NEXT_TO_ANONYMOUS_WHEN_GTID_MODE_IS_ON	@@SESSION.GTID_NEXT cannot be set ANONYMOUS when @@GLOBAL.GTID_ = ON.
1783	HY000	ER_CANT_SET_GTID_NEXT_LIST_TO_NON_NULL_WHEN_GTID_MODE_IS_OFF	@@SESSION.GTID_NEXT_LIST cannot b to a non-NULL value when @@GLOBAL.GTID_MODE = OFF.
1784	HY000	ER_FOUND_GTID_EVENT_WHEN_GTID_MODE_IS_OFF	Found a Gtid_log_event or Previous_gtids_log_event when @@GLOBAL.GTID_MODE = OFF.
1785	HY000	ER_GTID_UNSAFE_NON_TRANSACTIONAL_TABLE	When @@GLOBAL.ENFORCE_GTID_CONSIST = 1, updates to non-transactional tables ca be done in either autocommitted statement single-statement transactions, and never in same statement as updates to transactiona tables.

1786	HY000	ER_GTID_UNSAFE_CREATE_SELECT	CREATE TABLE ... SELECT is forbidden v @@GLOBAL.ENFORCE_GTID_CONSIST = 1.
1787	HY000	ER_GTID_UNSAFE_CREATE_DROP_TEMPORARY_TABLE_IN_TRANSACTION	When @@GLOBAL.ENFORCE_GTID_CONSIST = 1, the statements CREATE TEMPORAR TABLE and DROP TEMPORARY TABLE c executed in a non-transactional context onl require that AUTOCOMMIT = 1.
1788	HY000	ER_GTID_MODE_CAN_ONLY_CHANGE_ONE_STEP_AT_A_TIME	The value of @@GLOBAL.GTID_MODE ca only change one step at a time: OFF <-> UPGRADE_STEP_1 <-> UPGRADE_STEI > ON. Also note that this value must be ste up or down simultaneously on all servers; s Manual for instructions.
1789	HY000	ER_MASTER_HAS_PURGED_REQUIRED_GTIDS	The slave is connecting using CHANGE MASTER TO MASTER_AUTO_POSITION but the master has purged binary logs cont GTIDs that the slave requires.
1790	HY000	ER_CANT_SET_GTID_NEXT_WHEN_OWNING_GTID	@@SESSION.GTID_NEXT cannot be cha by a client that owns a GTID. The client ow %s. Ownership is released on COMMIT or ROLLBACK.
1791	HY000	ER_UNKNOWN_EXPLAIN_FORMAT	Unknown EXPLAIN format name: '%s'
1792	25006	ER_CANT_EXECUTE_IN_READ_ONLY_TRANSACTION	Cannot execute statement in a READ ONL transaction.
1793	HY000	ER_TOO_LONG_TABLE_PARTITION_COMMENT	Comment for table partition '%s' is too long = %lu
1794	HY000	ER_SLAVE_CONFIGURATION	Slave is not configured or failed to initialize properly. You must at least set --server-id t enable either a master or a slave. Additiona messages can be found in the MySQL erro
1795	HY000	ER_INNODB_FT_LIMIT	InnoDB presently supports one FULLTEXT creation at a time
1796	HY000	ER_INNODB_NO_FT_TEMP_TABLE	Cannot create FULLTEXT index on tempor InnoDB table
1797	HY000	ER_INNODB_FT_WRONG_DOCID_COLUMN	Column '%s' is of wrong type for an InnoDB FULLTEXT index
1798	HY000	ER_INNODB_FT_WRONG_DOCID_INDEX	Index '%s' is of wrong type for an InnoDB FULLTEXT index
1799	HY000	ER_INNODB_ONLINE_LOG_TOO_BIG	Creating index '%s' required more than 'innodb_online_alter_log_max_size' bytes modification log. Please try again.

Error Code	SQLSTATE	Error	Description
1800	HY000	ER_UNKNOWN_ALTER_ALGORITHM	Unknown ALGORITHM
1801	HY000	ER_UNKNOWN_ALTER_LOCK	Unknown LOCK type '
1802	HY000	ER_MTS_CHANGE_MASTER_CANT_RUN_WITH_GAPS	CHANGE MASTER ca executed when the sla stopped with an error c MTS mode. Consider u RESET SLAVE or STA SLAVE UNTIL.
1803	HY000	ER_MTS_RECOVERY_FAILURE	Cannot recover after S errored out in parallel e mode. Additional error can be found in the My log.
1804	HY000	ER_MTS_RESET_WORKERS	Cannot clean up worke tables. Additional error messages can be foun MySQL error log.
1805	HY000	ER_COL_COUNT_DOESNT_MATCH_CORRUPTED_V2	Column count of %s.% wrong. Expected %d, f The table is probably c

1806	HY000	ER_SLAVE_SILENT_RETRY_TRANSACTION	Slave must silently retr transaction
1807	HY000	ER_DISCARD_FK_CHECKS_RUNNING	There is a foreign key running on table '%s'. discard the table.
1808	HY000	ER_TABLE_SCHEMA_MISMATCH	Schema mismatch (%s
1809	HY000	ER_TABLE_IN_SYSTEM_TABLESPACE	Table '%s' in system ta
1810	HY000	ER_IO_READ_ERROR	IO Read error: (%lu, %
1811	HY000	ER_IO_WRITE_ERROR	IO Write error: (%lu, %
1812	HY000	ER_TABLESPACE_MISSING	Tablespace is missing '%s'
1813	HY000	ER_TABLESPACE_EXISTS	Tablespace for table '% Please DISCARD the t before IMPORT.
1814	HY000	ER_TABLESPACE_DISCARDED	Tablespace has been for table '%s'
1815	HY000	ER_INTERNAL_ERROR	Internal error: %s
1816	HY000	ER_INNODB_IMPORT_ERROR	ALTER TABLE '%s' IM TABLESPACE failed w %lu : '%s'
1817	HY000	ER_INNODB_INDEX_CORRUPT	Index corrupt: %s
1818	HY000	ER_INVALID_YEAR_COLUMN_LENGTH	YEAR(%lu) column typ deprecated. Creating Y column instead.
1819	HY000	ER_NOT_VALID_PASSWORD	Your password does n the current policy requi
1820	HY000	ER_MUST_CHANGE_PASSWORD	You must SET PASSW before executing this s
1821	HY000	ER_FK_NO_INDEX_CHILD	Failed to add the foreig constaint. Missing inde constraint '%s' in the fo table '%s'
1822	HY000	ER_FK_NO_INDEX_PARENT	Failed to add the foreig constaint. Missing inde constraint '%s' in the re table '%s'
1823	HY000	ER_FK_FAIL_ADD_SYSTEM	Failed to add the foreig constraint '%s' to syste
1824	HY000	ER_FK_CANNOT_OPEN_PARENT	Failed to open the refe table '%s'
1825	HY000	ER_FK_INCORRECT_OPTION	Failed to add the foreig constraint on table '%s options in FOREIGN K constraint '%s'
1826	HY000	ER_FK_DUP_NAME	Duplicate foreign key c name '%s'
1827	HY000	ER_PASSWORD_FORMAT	The password hash do the expected format. C correct password algor being used with the PASSWORD() functio
1828	HY000	ER_FK_COLUMN_CANNOT_DROP	Cannot drop column '% needed in a foreign ke constraint '%s'

1829	HY000	ER_FK_COLUMN_CANNOT_DROP_CHILD	Cannot drop column '% needed in a foreign ke constraint '%s' of table
1830	HY000	ER_FK_COLUMN_NOT_NULL	Column '%s' cannot be NULL: needed in a fore constraint '%s' SET NU
1831	HY000	ER_DUP_INDEX	Duplicate index '%s' de the table '%s.%s'. This deprecated and will be disallowed in a future r
1832	HY000	ER_FK_COLUMN_CANNOT_CHANGE	Cannot change columr used in a foreign key c '%s'
1833	HY000	ER_FK_COLUMN_CANNOT_CHANGE_CHILD	Cannot change columr used in a foreign key c '%s' of table '%s'
1834	HY000	ER_FK_CANNOT_DELETE_PARENT	Cannot delete rows fro which is parent in a for constraint '%s' of table
1835	HY000	ER_MALFORMED_PACKET	Malformed communica packet.
1836	HY000	ER_READ_ONLY_MODE	Running in read-only n
1837	HY000	ER_GTID_NEXT_TYPE_UNDEFINED_GROUP	When @@SESSION.GTID_N set to a GTID, you mus set it to a different valu COMMIT or ROLLBAC check GTID_NEXT var manual page for detail explanation. Current @@SESSION.GTID_N '%s'.
1838	HY000	ER_VARIABLE_NOT_SETTABLE_IN_SP	The system variable % be set in stored proced
1839	HY000	ER_CANT_SET_GTID_PURGED_WHEN_GTID_MODE_IS_OFF	@@GLOBAL.GTID_P can only be set when @@GLOBAL.GTID_M ON.
1840	HY000	ER_CANT_SET_GTID_PURGED_WHEN_GTID_EXECUTED_IS_NOT_EMPTY	@@GLOBAL.GTID_P can only be set when @@GLOBAL.GTID_E is empty.
1841	HY000	ER_CANT_SET_GTID_PURGED_WHEN_OWNED_GTIDS_IS_NOT_EMPTY	@@GLOBAL.GTID_P can only be set when t no ongoing transaction even in other clients).
1842	HY000	ER_GTID_PURGED_WAS_CHANGED	@@GLOBAL.GTID_P was changed from '%s
1843	HY000	ER_GTID_EXECUTED_WAS_CHANGED	@@GLOBAL.GTID_E was changed from '%s
1844	HY000	ER_BINLOG_STMT_MODE_AND_NO_REPL_TABLES	Cannot execute statem impossible to write to b since BINLOG_FORM STATEMENT, and bot replicated and non rep tables are written to.

1845	0A000	ER_ALTER_OPERATION_NOT_SUPPORTED	%s is not supported fo operation. Try %s.
1846	0A000	ER_ALTER_OPERATION_NOT_SUPPORTED_REASON	%s is not supported. R %s. Try %s.
1847	HY000	ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_COPY	COPY algorithm requir
1848	HY000	ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_PARTITION	Partition specific opera not yet support LOCK/ALGORITHM
1849	HY000	ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_FK_RENAME	Columns participating i foreign key are rename
1850	HY000	ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_COLUMN_TYPE	Cannot change columr INPLACE
1851	HY000	ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_FK_CHECK	Adding foreign keys ne foreign_key_checks=C
1852	HY000	ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_IGNORE	Creating unique indexe IGNORE requires COF algorithm to remove du rows
1853	HY000	ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_NOPK	Dropping a primary key allowed without also ac new primary key
1854	HY000	ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_AUTOINC	Adding an auto-increm column requires a lock
1855	HY000	ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_HIDDEN_FTS	Cannot replace hidden FTS_DOC_ID with a u one
1856	HY000	ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_CHANGE_FTS	Cannot drop or rename FTS_DOC_ID
1857	HY000	ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_FTS	Fulltext index creation lock
1858	HY000	ER_SQL_SLAVE_SKIP_COUNTER_NOT_SETTABLE_IN_GTID_MODE	sql_slave_skip_counte be set when the server running with @@GLOBAL.GTID_M ON. Instead, for each t that you want to skip, g an empty transaction w same GTID as the tran
1859	23000	ER_DUP_UNKNOWN_IN_INDEX	Duplicate entry for key
1860	HY000	ER_IDENT_CAUSES_TOO_LONG_PATH	Long database name a identifier for object resu path length exceeding characters. Path: '%s'.
1861	HY000	ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_NOT_NULL	cannot silently convert values, as required in t SQL_MODE
1862	HY000	ER_MUST_CHANGE_PASSWORD_LOGIN	Your password has exp log in you must change client that supports exp passwords.
1863	HY000	ER_ROW_IN_WRONG_PARTITION	Found a row in wrong p %s

1864	HY000	ER_MTS_EVENT_BIGGER_PENDING_JOBS_SIZE_MAX	Cannot schedule even relay-log name %s, po to Worker thread beca size %lu exceeds %lu slave_pending_jobs_si
1865	HY000	ER_INNODB_NO_FT_USES_PARSER	Cannot CREATE FULL INDEX WITH PARSER InnoDB table
1866	HY000	ER_BINLOG_LOGICAL_CORRUPTION	The binary log file '%s' logically corrupted: %s
1867	HY000	ER_WARN_PURGE_LOG_IN_USE	file %s was not purged it was being read by % thread(s), purged only %d files.
1868	HY000	ER_WARN_PURGE_LOG_IS_ACTIVE	file %s was not purged it is the active log file.
1869	HY000	ER_AUTO_INCREMENT_CONFLICT	Auto-increment value i UPDATE conflicts with generated values
1870	HY000	WARN_ON_BLOCKHOLE_IN_RBR	Row events are not log %s statements that mo BLACKHOLE tables in format. Table(s): '%s'
1871	HY000	ER_SLAVE_MI_INIT_REPOSITORY	Slave failed to initialize info structure from the
1872	HY000	ER_SLAVE_RLI_INIT_REPOSITORY	Slave failed to initialize info structure from the
1873	28000	ER_ACCESS_DENIED_CHANGE_USER_ERROR	Access denied trying to to user '%s'@'%s' (usin password: %s). Discon
1874	HY000	ER_INNODB_READ_ONLY	InnoDB is in read only
1875	HY000	ER_STOP_SLAVE_SQL_THREAD_TIMEOUT	STOP SLAVE comman execution is incomplete SQL thread got the sto thread is busy, SQL thr stop once the current t complete.
1876	HY000	ER_STOP_SLAVE_IO_THREAD_TIMEOUT	STOP SLAVE comman execution is incomplete IO thread got the stop thread is busy, IO threa stop once the current t complete.
1877	HY000	ER_TABLE_CORRUPT	Operation cannot be pe The table '%s.%s' is m corrupt or contains bad
1878	HY000	ER_TEMP_FILE_WRITE_FAILURE	Temporary file write fai
1879	HY000	ER_INNODB_FT_AUX_NOT_HEX_ID	Upgrade index name fa please use create inde table) algorithm copy to index.
1880		ER_LAST_MYSQL_ERROR_MESSAGE	''

MariaDB-specific error codes

Error Code	SQLSTATE	Error	Description

1900		ER_UNUSED_18	"
1901		ER_GENERATED_COLUMN_FUNCTION_IS_NOT_ALLOWED	Function or expression '%s' cannot be used in the %s clause of %`s
1902		ER_UNUSED_19	"
1903		ER_PRIMARY_KEY_BASED_ON_GENERATED_COLUMN	Primary key cannot be defined upon a generated column
1904		ER_KEY_BASED_ON_GENERATED_VIRTUAL_COLUMN	Key/Index cannot be defined on a virtual generated column
1905		ER_WRONG_FK_OPTION_FOR_GENERATED_COLUMN	Cannot define foreign key with %s clause on a generated column
1906		ER_WARNING_NON_DEFAULT_VALUE_FOR_GENERATED_COLUMN	The value specified for generated column '%s' in table '%s' has been ignored
1907		ER_UNSUPPORTED_ACTION_ON_GENERATED_COLUMN	This is not yet supported for generated columns
1908		ER_UNUSED_20	"
1909		ER_UNUSED_21	"
1910		ER_UNSUPPORTED_ENGINE_FOR_GENERATED_COLUMNS	%s storage engine does not support generated columns
1911		ER_UNKNOWN_OPTION	Unknown option '%-.64s'
1912		ER_BAD_OPTION_VALUE	Incorrect value '%-.64s' for option '%-.64s'
1913		ER_UNUSED_6	You should never see it
1914		ER_UNUSED_7	You should never see it
1915		ER_UNUSED_8	You should never see it
1916		ER_DATA_OVERFLOW 22003	Got overflow when converting '%-.128s' to %-.32s. Value truncated.
1917		ER_DATA_TRUNCATED 22003	Truncated value '%-.128s' when converting to %-.32s
1918		ER_BAD_DATA 22007	Encountered illegal value '%-.128s' when converting to %-.32s
1919		ER_DYN_COL_WRONG_FORMAT	Encountered illegal format of dynamic column string
1920		ER_DYN_COL_IMPLEMENTATION_LIMIT	Dynamic column implementation limit reached
1921		ER_DYN_COL_DATA 22007	Illegal value used as argument of dynamic column function
1922		ER_DYN_COL_WRONG_CHARSET	Dynamic column contains unknown character set
1923		ER_ILLEGAL_SUBQUERY_OPTIMIZER_SWITCHES	At least one of the 'in_to_exists' or 'materialization' optimizer_switch flags must be 'on'.
1924		ER_QUERY_CACHE_IS_DISABLED	Query cache is disabled (resize or similar command in progress); repeat this command later
1925		ER_QUERY_CACHE_IS_GLOBALY_DISABLED	Query cache is globally disabled and you can't enable it only for this session
1926		ER_VIEW_ORDERBY_IGNORED	View '%-.192s'.'%-.192s' ORDER BY clause ignored because there is other ORDER BY clause already.
1927		ER_CONNECTION_KILLED 70100	Connection was killed
1928	ER_UNUSED_11	You should never see it	
1929		ER_INSIDE_TRANSACTION_PREVENTS_SWITCH_SKIP_REPLICATION	Cannot modify @@session.skip_replication inside a transaction
1930		ER_STORED_FUNCTION_PREVENTS_SWITCH_SKIP_REPLICATION	Cannot modify @@session.skip_replication inside a stored function or trigger
1931		ER_QUERY_EXCEEDED_ROWS_EXAMINED_LIMIT	Query execution was interrupted. The query examined at least %llu rows, which exceeds LIMIT ROWS EXAMINED (%llu). The query result may be incomplete.
1932		ER_NO_SUCH_TABLE_IN_ENGINE 42S02	Table '%-.192s.%-.192s' doesn't exist in engine
1933		ER_TARGET_NOT_EXPLAINABLE	Target is not running an EXPLAINable command
1934		ER_CONNECTION_ALREADY_EXISTS	Connection '%.*s' conflicts with existing connection '%.*s'
1935		ER_MASTER_LOG_PREFIX	Master '%.*s':

1936		ER_CANT_START_STOP_SLAVE	Can't %s SLAVE '%.*s'
1937		ER_SLAVE_STARTED	SLAVE '%.*s' started
1938		ER_SLAVE_STOPPED	SLAVE '%.*s' stopped
1939		ER_SQL_DISCOVER_ERROR	Engine %s failed to discover table %`-.192s.%`-.192s with '%s'
1940		ER_FAILED_GTID_STATE_INIT	Failed initializing replication GTID state
1941		ER_INCORRECT_GTID_STATE	Could not parse GTID list
1942		ER_CANNOT_UPDATE_GTID_STATE	Could not update replication slave gtid state
1943		ER_DUPLICATE_GTID_DOMAIN	GTID %u-%u-%llu and %u-%u-%llu conflict (duplicate domain id %u)
1944		ER_GTID_OPEN_TABLE_FAILED	Failed to open %s.%s
1945		ER_GTID_POSITION_NOT_FOUND_IN_BINLOG	Connecting slave requested to start from GTID %u-%u-%llu, which is not in the master's binlog
1946		ER_CANNOT_LOAD_SLAVE_GTID_STATE	Failed to load replication slave GTID position from table %s.%s
1947		ER_MASTER_GTID_POS_CONFLICTS_WITH_BINLOG	Specified GTID %u-%u-%llu conflicts with the binary log which contains a more recent GTID %u-%u-%llu. If MASTER_GTID_POS=CURRENT_POS is used, the binlog position will override the new value of @@gtid_slave_pos.
1948		ER_MASTER_GTID_POS_MISSING_DOMAIN	Specified value for @@gtid_slave_pos contains no value for replication domain %u. This conflicts with the binary log which contains GTID %u-%u-%llu. If MASTER_GTID_POS=CURRENT_POS is used, the binlog position will override the new value of @@gtid_slave_pos.
1949		ER_UNTIL_REQUIRES_USING_GTID	START SLAVE UNTIL master_gtid_pos requires that slave is using GTID
1950		ER_GTID_STRICT_OUT_OF_ORDER	An attempt was made to binlog GTID %u-%u-%llu which would create an out-of-order sequence number with existing GTID %u-%u-%llu, and gtid strict mode is enabled.
1951		ER_GTID_START_FROM_BINLOG_HOLE	The binlog on the master is missing the GTID %u-%u-%llu requested by the slave (even though a subsequent sequence number does exist), and GTID strict mode is enabled
1952		ER_SLAVE_UNEXPECTED_MASTER_SWITCH	Unexpected GTID received from master after reconnect. This normally indicates that the master server was replaced without restarting the slave threads. %s
1953		ER_INSIDE_TRANSACTION_PREVENTS_SWITCH_GTID_DOMAIN_ID_SEQ_NO	Cannot modify @@session.gtid_domain_id or @@session.gtid_seq_no inside a transaction
1954		ER_STORED_FUNCTION_PREVENTS_SWITCH_GTID_DOMAIN_ID_SEQ_NO	Cannot modify @@session.gtid_domain_id or @@session.gtid_seq_no inside a stored function or trigger
1955		ER_GTID_POSITION_NOT_FOUND_IN_BINLOG2	Connecting slave requested to start from GTID %u-%u-%llu, which is not in the master's binlog. Since the master's binlog contains GTIDs with higher sequence numbers, it probably means that the slave has diverged due to executing extra errorneous transactions
1956		ER_BINLOG_MUST_BE_EMPTY	This operation is not allowed if any GTID has been logged to the binary log. Run RESET MASTER first to erase the log
1957		ER_NO_SUCH_QUERY	Unknown query id: %lld
1958		ER_BAD_BASE64_DATA	Bad base64 data as position %u
1959		ER_INVALID_ROLE	Invalid role specification %`s.
1960		ER_INVALID_CURRENT_USER	The current user is invalid.
1961		ER_CANNOT_GRANT_ROLE	Cannot grant role '%s' to: %s.
1962		ER_CANNOT_REVOKE_ROLE	Cannot revoke role '%s' from: %s.
1963		ER_CHANGE_SLAVE_PARALLEL_THREADS_ACTIVE	Cannot change @@slave_parallel_threads while another change is in progress

1964		ER_PRIOR_COMMIT_FAILED	Commit failed due to failure of an earlier commit on which this one depends
1965		ER_IT_IS_A_VIEW	'%-.192s' is a view
1966		ER_SLAVE_SKIP_NOT_IN_GTID	When using GTID, @@sql_slave_skip_counter can not be used. Instead, setting @@gtid_slave_pos explicitly can be used to skip to after a given GTID position.
1967		ER_TABLE_DEFINITION_TOO_BIG	The definition for table %`s is too big
1968		ER_PLUGIN_INSTALLED	Plugin '%-.192s' already installed
1969		ER_STATEMENT_TIMEOUT	Query execution was interrupted (max_statement_time exceeded)
1970		ER_SUBQUERIES_NOT_SUPPORTED	%s does not support subqueries or stored functions.
1971		ER_SET_STATEMENT_NOT_SUPPORTED	The system variable %.200s cannot be set in SET STATEMENT.
1972		ER_UNUSED_9	You should never see it
1973		ER_USER_CREATE_EXISTS	Can't create user '%-.64s'@'%-.64s'; it already exists
1974		ER_USER_DROP_EXISTS	Can't drop user '%-.64s'@'%-.64s'; it doesn't exist
1975		ER_ROLE_CREATE_EXISTS	Can't create role '%-.64s'; it already exists
1976		ER_ROLE_DROP_EXISTS	Can't drop role '%-.64s'; it doesn't exist
1977		ER_CANNOT_CONVERT_CHARACTER	Cannot convert '%s' character 0x%-.64s to '%s'
1978		ER_INVALID_DEFAULT_VALUE_FOR_FIELD	Incorrect default value '%-.128s' for column '%.192s'
1979		ER_KILL_QUERY_DENIED_ERROR	You are not owner of query %lu
1980		ER_NO_EIS_FOR_FIELD	Engine-independent statistics are not collected for column '%s'
1981		ER_WARN_AGGFUNC_DEPENDENCE	Aggregate function '%-.192s)' of SELECT #%d belongs to SELECT #%d
1982		WARN_INNODB_PARTITION_OPTION_IGNORED	<%-.64s> option ignored for InnoDB partition

Error Code	SQLSTATE	Error	Description
3000		ER_FILE_CORRUPT	File %s is corrupted
3001		ER_ERROR_ON_MASTER	Query partially completed on the maste on master: %d) and was aborted. Ther chance that your master is inconsistent point. If you are sure that your master i this query manually on the slave and th restart the slave with SET GLOBAL SQL_SLAVE_SKIP_COUNTER=1; ST SLAVE;. Query:'%s'"
3002		ER_INCONSISTENT_ERROR	Query caused different errors on maste slave. Error on master: message (form error code=%d; Error on slave:actual message='%s', error code=%d. Default database:'%s'. Query:'%s'
3003		ER_STORAGE_ENGINE_NOT_LOADED	Storage engine for table '%s'.'%s' is no
3004		ER_GET_STACKED_DA_WITHOUT_ACTIVE_HANDLER 0Z002	GET STACKED DIAGNOSTICS when not active
3005		ER_WARN_LEGACY_SYNTAX_CONVERTED	%s is no longer supported. The statem converted to %s.
3006		ER_BINLOG_UNSAFE_FULLTEXT_PLUGIN	Statement is unsafe because it uses a parser plugin which may not return the value on the slave.
3007		ER_CANNOT_DISCARD_TEMPORARY_TABLE	Cannot DISCARD/IMPORT tablespace associated with temporary table
3008		ER_FK_DEPTH_EXCEEDED	Foreign key cascade delete/update exc max depth of %d.

3009		ER_COL_COUNT_DOESNT_MATCH_PLEASE_UPDATE_V2	Column count of %s.%s is wrong. Expe %d, found %d. Created with MariaDB % running %d. Please use mysql_upgrad this error.
3010		ER_WARN_TRIGGER_DOESNT_HAVE_CREATED	Trigger %s.%s.%s does not have CRE attribute.
3011		ER_REFERENCED_TRG_DOES_NOT_EXIST_MYSQL	Referenced trigger '%s' for the given ac and event type does not exist.
3012		ER_EXPLAIN_NOT_SUPPORTED	EXPLAIN FOR CONNECTION comma supported only for SELECT/UPDATE/INSERT/DELETE/R
3013		ER_INVALID_FIELD_SIZE	Invalid size for column '%-.192s'.
3014		ER_MISSING_HA_CREATE_OPTION	Table storage engine '%-.64s' found re create option missing
3015		ER_ENGINE_OUT_OF_MEMORY	Out of memory in storage engine '%-.6
3016		ER_PASSWORD_EXPIRE_ANONYMOUS_USER	The password for anonymous user can expired.
3017		ER_SLAVE_SQL_THREAD_MUST_STOP	This operation cannot be performed wi running slave sql thread; run STOP SL SQL_THREAD first
3018		ER_NO_FT_MATERIALIZED_SUBQUERY	Cannot create FULLTEXT index on materialized subquery
3019		ER_INNODB_UNDO_LOG_FULL	Undo Log error: %s
3020		ER_INVALID_ARGUMENT_FOR_LOGARITHM	Invalid argument for logarithm
3021		ER_SLAVE_CHANNEL_IO_THREAD_MUST_STOP	This operation cannot be performed wi running slave io thread; run STOP SLA IO_THREAD FOR CHANNEL '%s' first
3022		ER_WARN_OPEN_TEMP_TABLES_MUST_BE_ZERO	This operation may not be safe when th has temporary tables. The tables will b open until the server restarts or until th are deleted by any replicated DROP st Suggest to wait until slave_open_temp 0.
3023		ER_WARN_ONLY_MASTER_LOG_FILE_NO_POS	CHANGE MASTER TO with a MASTER_LOG_FILE clause but no MASTER_LOG_POS clause may not b The old position value may not be valid new binary log file.
3024		ER_QUERY_TIMEOUT	Query execution was interrupted, maxi statement execution time exceeded
3025		ER_NON_RO_SELECT_DISABLE_TIMER	Select is not a read only statement, dis timer
3026		ER_DUP_LIST_ENTRY	Duplicate entry '%-.192s'.
3027		ER_SQL_MODE_NO_EFFECT	'%s' mode no longer has any effect. Us STRICT_ALL_TABLES or STRICT_TRANS_TABLES instead.
3028		ER_AGGREGATE_ORDER_FOR_UNION	Expression #%u of ORDER BY contain aggregate function and applies to a UN
3029		ER_AGGREGATE_ORDER_NON_AGG_QUERY	Expression #%u of ORDER BY contain aggregate function and applies to the r non-aggregated query
3030		ER_SLAVE_WORKER_STOPPED_PREVIOUS_THD_ERROR	Slave worker has stopped after at least previous worker encountered an error v slave-preserve-commit-order was enab preserve commit order, the last transac executed by this thread has not been committed. When restarting the slave a fixing any failed threads, you should fix worker as well.

3031		ER_DONT_SUPPORT_SLAVE_PRESERVE_COMMIT_ORDER	slave_preserve_commit_order is not su %s.
3032		ER_SERVER_OFFLINE_MODE	The server is currently in offline mode
3033		ER_GIS_DIFFERENT_SRIDS	Binary geometry function %s given two geometries of different srids: %u and % should have been identical.
3034		ER_GIS_UNSUPPORTED_ARGUMENT	Calling geometry function %s with unsu types of arguments.
3035		ER_GIS_UNKNOWN_ERROR	Unknown GIS error occurred in functio
3036		ER_GIS_UNKNOWN_EXCEPTION	Unknown exception caught in GIS func
3037		ER_GIS_INVALID_DATA	Invalid GIS data provided to function %
3038		ER_BOOST_GEOMETRY_EMPTY_INPUT_EXCEPTION	The geometry has no data in function %
3039		ER_BOOST_GEOMETRY_CENTROID_EXCEPTION	Unable to calculate centroid because g is empty in function %s.
3040		ER_BOOST_GEOMETRY_OVERLAY_INVALID_INPUT_EXCEPTION	Geometry overlay calculation error: geo data is invalid in function %s.
3041		ER_BOOST_GEOMETRY_TURN_INFO_EXCEPTION	Geometry turn info calculation error: ge data is invalid in function %s.
3042		ER_BOOST_GEOMETRY_SELF_INTERSECTION_POINT_EXCEPTION	Analysis procedures of intersection poi interrupted unexpectedly in function %s
3043		ER_BOOST_GEOMETRY_UNKNOWN_EXCEPTION	Unknown exception thrown in function
3044		ER_STD_BAD_ALLOC_ERROR	Memory allocation error: %-.256s in fur %s.
3045		ER_STD_DOMAIN_ERROR	Domain error: %-.256s in function %s.
3046		ER_STD_LENGTH_ERROR	Length error: %-.256s in function %s.
3047		ER_STD_INVALID_ARGUMENT	Invalid argument error: %-.256s in func
3048		ER_STD_OUT_OF_RANGE_ERROR	Out of range error: %-.256s in function
3049		ER_STD_OVERFLOW_ERROR	Overflow error error: %-.256s in functio
3050		ER_STD_RANGE_ERROR	Range error: %-.256s in function %s.
3051		ER_STD_UNDERFLOW_ERROR	Underflow error: %-.256s in function %
3052		ER_STD_LOGIC_ERROR	Logic error: %-.256s in function %s.
3053		ER_STD_RUNTIME_ERROR	Runtime error: %-.256s in function %s.
3054		ER_STD_UNKNOWN_EXCEPTION	Unknown exception: %-.384s in functio
3055		ER_GIS_DATA_WRONG_ENDIANESS	Geometry byte string must be little end
3056		ER_CHANGE_MASTER_PASSWORD_LENGTH	The password provided for the replicati exceeds the maximum length of 32 cha
3057	42000	ER_USER_LOCK_WRONG_NAME	Incorrect user-level lock name '%-.192s
3058		ER_USER_LOCK_DEADLOCK	Deadlock found when trying to get user lock; try rolling back transaction/releasi and restarting lock acquisition.
3059		ER_REPLACE_INACCESSIBLE_ROWS	REPLACE cannot be executed as it req deleting rows that are not in the view
3060		ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_GIS	Do not support online operation on tabl GIS index

Error Code	SQLSTATE	Error	Description
4000	0A000	ER_COMMULTI_BADCONTEXT	COM_MULTI can't return a result set in the given context
4001		ER_BAD_COMMAND_IN_MULTI	Command '%s' is not allowed for COM_MULTI
4002		ER_WITH_COL_WRONG_LIST	WITH column list and SELECT field list have different column counts

4003		ER_TOO_MANY_DEFINITIONS_IN_WITH_CLAUSE	Too many WITH elements in WITH cla
4004		ER_DUP_QUERY_NAME	Duplicate query name %`-.64s in WITH clause
4005		ER_RECURSIVE_WITHOUT_ANCHORS	No anchors for recursive WITH elemer '%s'
4006		ER_UNACCEPTABLE_MUTUAL_RECURSION	Unacceptable mutual recursion with anchored table '%s'
4007		ER_REF_TO_RECURSIVE_WITH_TABLE_IN_DERIVED	Reference to recursive WITH table '%s materialized derived
4008		ER_NOT_STANDARD_COMPLIANT_RECURSIVE	Restrictions imposed on recursive definitions are violated for table '%s'"R_WRONG_WINDOW_SPEC_NA
4009		ER_WRONG_WINDOW_SPEC_NAME	Window specification with name '%s' is defined
4010		ER_DUP_WINDOW_NAME	Multiple window specifications with the same name '%s'
4011		ER_PARTITION_LIST_IN_REFERENCING_WINDOW_SPEC	Window specification referencing anotl one '%s' cannot contain partition list
4012		ER_ORDER_LIST_IN_REFERENCING_WINDOW_SPEC	Referenced window specification '%s' already contains order list
4013		ER_WINDOW_FRAME_IN_REFERENCED_WINDOW_SPEC	Referenced window specification '%s' cannot contain window frame
4014		ER_BAD_COMBINATION_OF_WINDOW_FRAME_BOUND_SPECS	Unacceptable combination of window frame bound specifications
4015		ER_WRONG_PLACEMENT_OF_WINDOW_FUNCTION	Window function is allowed only in SELECT list and ORDER BY clause
4016		ER_WINDOW_FUNCTION_IN_WINDOW_SPEC	Window function is not allowed in wind specification
4017		ER_NOT_ALLOWED_WINDOW_FRAME	Window frame is not allowed with '%s'
4018		ER_NO_ORDER_LIST_IN_WINDOW_SPEC	No order list in window specification fo '%s'
4019		ER_RANGE_FRAME_NEEDS_SIMPLE_ORDERBY	RANGE-type frame requires ORDER E clause with single sort key
4020		ER_WRONG_TYPE_FOR_ROWS_FRAME	Integer is required for ROWS-type fran
4021		ER_WRONG_TYPE_FOR_RANGE_FRAME	Numeric datatype is required for RANG type frame
4022		ER_FRAME_EXCLUSION_NOT_SUPPORTED	Frame exclusion is not supported yet
4023		ER_WINDOW_FUNCTION_DONT_HAVE_FRAME	This window function may not have a window frame
4024		ER_INVALID_NTILE_ARGUMENT	Argument of NTILE must be greater tha
4025	23000	ER_CONSTRAINT_FAILED	CONSTRAINT %`s failed for %`-.192s. .192s
4026		ER_EXPRESSION_IS_TOO_BIG	Expression in the %s clause is too big
4027		ER_ERROR_EVALUATING_EXPRESSION	Got an error evaluating stored express %s
4028		ER_CALCULATING_DEFAULT_VALUE	Got an error when calculating default value for %`s
4029		ER_EXPRESSION_REFERS_TO_UNINIT_FIELD	Expression for field %`-.64s is referring uninitialized field %`s
4030		ER_PARTITION_DEFAULT_ERROR	Only one DEFAULT partition allowed
4031		ER_REFERENCED_TRG_DOES_NOT_EXIST	Referenced trigger '%s' for the given action time and event type does not ex
4032		ER_INVALID_DEFAULT_PARAM	Default/ignore value is not supported fo such parameter usage

4033		ER_BINLOG_NON_SUPPORTED_BULK	Only row based replication supported f bulk operations
4034		ER_BINLOG_UNCOMPRESS_ERROR	Uncompress the compressed binlog fa
4035		ER_JSON_BAD_CHR	Broken JSON string in argument %d to function '%s' at position %d
4036		ER_JSON_NOT_JSON_CHR	Character disallowed in JSON in argun %d to function '%s' at position %d
4037		ER_JSON_EOS	Unexpected end of JSON text in argum %d to function '%s'
4038		ER_JSON_SYNTAX	Syntax error in JSON text in argument to function '%s' at position %d
4039		ER_JSON_ESCAPING	Incorrect escaping in JSON text in argument %d to function '%s' at positio %d
4040		ER_JSON_DEPTH	Limit of %d on JSON nested strucures depth is reached in argument %d to function '%s' at position %d
4041		ER_JSON_PATH_EOS	Unexpected end of JSON path in argu %d to function '%s'
4042		ER_JSON_PATH_SYNTAX	Syntax error in JSON path in argument to function '%s' at position %d
4043		ER_JSON_PATH_DEPTH	Limit of %d on JSON path depth is reached in argument %d to function '% position %d
4044		ER_JSON_PATH_NO_WILDCARD	Wildcards in JSON path not allowed in argument %d to function '%s'
4045		ER_JSON_PATH_ARRAY	JSON path should end with an array identifier in argument %d to function '%
4046		ER_JSON_ONE_OR_ALL	Argument 2 to function '%s' must be "o or "all".
4047		ER_UNSUPPORT_COMPRESSED_TEMPORARY_TABLE	CREATE TEMPORARY TABLE is not allowed with ROW_FORMAT=COMPRESSED or KEY_BLOCK_SIZE.
4048		ER_GEOJSON_INCORRECT	Incorrect GeoJSON format specified fo st_geomfromgeojson function.
4049		ER_GEOJSON_TOO_FEW_POINTS	Incorrect GeoJSON format - too few po for linestring specified.
4050		ER_GEOJSON_NOT_CLOSED	Incorrect GeoJSON format - polygon n closed.
4051		ER_JSON_PATH_EMPTY	Path expression '$' is not allowed in argument %d to function '%s'.
4052		ER_SLAVE_SAME_ID	A slave with the same server_uuid/server_id as this slave has connected to the master
4053		ER_FLASHBACK_NOT_SUPPORTED	Flashback does not support %s %s
4054		ER_KEYS_OUT_OF_ORDER	Keys are out order during bulk load
4055		ER_OVERLAPPING_KEYS	Bulk load rows overlap existing rows
4056		ER_REQUIRE_ROW_BINLOG_FORMAT	Can't execute updates on master with binlog_format != ROW.
4057		ER_ISOLATION_MODE_NOT_SUPPORTED	MyRocks supports only READ COMMITTED and REPEATABLE REA isolation levels. Please change from current isolation level %s

4058		ER_ON_DUPLICATE_DISABLED	When unique checking is disabled in MyRocks, INSERT,UPDATE,LOAD statements with clauses that update or replace the key (i.e. INSERT ON DUPLICATE KEY UPDATE, REPLACE are not allowed. Query: %s
4059		ER_UPDATES_WITH_CONSISTENT_SNAPSHOT	Can't execute updates when you starte transaction with START TRANSACTIO WITH CONSISTENT [ROCKSDB] SNAPSHOT.
4060		ER_ROLLBACK_ONLY	This transaction was rolled back and cannot be committed. Only supported operation is to roll it back, so all pendir changes will be discarded. Please rest another transaction.
4061		ER_ROLLBACK_TO_SAVEPOINT	MyRocks currently does not support ROLLBACK TO SAVEPOINT if modify rows.
4062		ER_ISOLATION_LEVEL_WITH_CONSISTENT_SNAPSHOT	Only REPEATABLE READ isolation le is supported for START TRANSACTIO WITH CONSISTENT SNAPSHOT in RocksDB Storage Engine.
4063		ER_UNSUPPORTED_COLLATION	Unsupported collation on string indexe column %s.%s Use binary collation (%
4064		ER_METADATA_INCONSISTENCY	Table '%s' does not exist, but metadata information exists inside MyRocks. Thi a sign of data inconsistency. Please ch if '%s.frm' exists, and try to restore it if does not exist.
4065		ER_CF_DIFFERENT	Column family ('%s') flag (%d) is differe from an existing flag (%d). Assign a ne CF flag, or do not change existing CF f
4066		ER_RDB_TTL_DURATION_FORMAT	TTL duration (%s) in MyRocks must be unsigned non-null 64-bit integer.
4067		ER_RDB_STATUS_GENERAL	Status error %d received from RocksD %s
4068		ER_RDB_STATUS_MSG	%s, Status error %d received from RocksDB: %s
4069		ER_RDB_TTL_UNSUPPORTED	TTL support is currently disabled when table has a hidden PK.
4070		ER_RDB_TTL_COL_FORMAT	TTL column (%s) in MyRocks must be unsigned non-null 64-bit integer, exist inside the table, and have an accompanying ttl duration.
4071		ER_PER_INDEX_CF_DEPRECATED	The per-index column family option has been deprecated
4072		ER_KEY_CREATE_DURING_ALTER	MyRocks failed creating new key definitions during alter.
4073		ER_SK_POPULATE_DURING_ALTER	MyRocks failed populating secondary k during alter.
4074		ER_SUM_FUNC_WITH_WINDOW_FUNC_AS_ARG	Window functions can not be used as arguments to group functions.
4075		ER_NET_OK_PACKET_TOO_LARGE	OK packet too large
4076		ER_GEOJSON_EMPTY_COORDINATES	Incorrect GeoJSON format - empty 'coordinates' array.
4077		ER_MYROCKS_CANT_NOPAD_COLLATION	MyRocks doesn't currently support collations with \"No pad\" attribute.
4078		ER_ILLEGAL_PARAMETER_DATA_TYPES2_FOR_OPERATION	Illegal parameter data types %s and % for operation '%s'

4079		ER_ILLEGAL_PARAMETER_DATA_TYPE_FOR_OPERATION	Illegal parameter data type %s for operation '%s'
4080		ER_WRONG_PARAMCOUNT_TO_CURSOR	Incorrect parameter count to cursor '%-.192s'
4081		ER_UNKNOWN_STRUCTURED_VARIABLE	Unknown structured system variable or ROW routine variable '%-.*s'
4082		ER_ROW_VARIABLE_DOES_NOT_HAVE_FIELD	Row variable '%-.192s' does not have a field '%-.192s'
4083		ER_END_IDENTIFIER_DOES_NOT_MATCH	END identifier '%-.192s' does not match '%-.192s'
4084		ER_SEQUENCE_RUN_OUT	Sequence '%-.64s.%-.64s' has run out
4085		ER_SEQUENCE_INVALID_DATA	Sequence '%-.64s.%-.64s' values are conflicting
4086		ER_SEQUENCE_INVALID_TABLE_STRUCTURE	Sequence '%-.64s.%-.64s' table structure is invalid (%s)
4087		ER_SEQUENCE_ACCESS_ERROR	Sequence '%-.64s.%-.64s' access error
4088		ER_SEQUENCE_BINLOG_FORMAT	Sequences requires binlog_format mixed or row
4089		ER_NOT_SEQUENCE	'%-.64s.%-.64s' is not a SEQUENCE
4090		ER_NOT_SEQUENCE2	'%-.192s' is not a SEQUENCE
4091		ER_UNKNOWN_SEQUENCES	Unknown SEQUENCE: '%-.300s'
4092		ER_UNKNOWN_VIEW	Unknown VIEW: '%-.300s'
4093		ER_WRONG_INSERT_INTO_SEQUENCE	Wrong INSERT into a SEQUENCE. One can only do single table INSERT into a sequence object (like with mysqldump) If you want to change the SEQUENCE, use ALTER SEQUENCE instead.
4094		ER_SP_STACK_TRACE	At line %u in %s
4095		ER_PACKAGE_ROUTINE_IN_SPEC_NOT_DEFINED_IN_BODY	Subroutine '%-.192s' is declared in the package specification but is not defined in the package body
4096		ER_PACKAGE_ROUTINE_FORWARD_DECLARATION_NOT_DEFINED	Subroutine '%-.192s' has a forward declaration but is not defined
4097		ER_COMPRESSED_COLUMN_USED_AS_KEY	Compressed column '%-.192s' can't be used in key specification
4098		ER_UNKNOWN_COMPRESSION_METHOD	Unknown compression method: %s
4099		ER_WRONG_NUMBER_OF_VALUES_IN_TVC	The used table value constructor has a different number of values
4100		ER_FIELD_REFERENCE_IN_TVC	Field reference '%-.192s' can't be used in table value constructor
4101		ER_WRONG_TYPE_FOR_PERCENTILE_FUNC	Numeric datatype is required for %s function
4102		ER_ARGUMENT_NOT_CONSTANT	Argument to the %s function is not a constant for a partition
4103		ER_ARGUMENT_OUT_OF_RANGE	Argument to the %s function does not belong to the range [0,1]
4104		ER_WRONG_TYPE_OF_ARGUMENT	%s function only accepts arguments that can be converted to numerical types
4105		ER_NOT_AGGREGATE_FUNCTION	Aggregate specific instruction (FETCH GROUP NEXT ROW) used in a wrong context
4106		ER_INVALID_AGGREGATE_FUNCTION	Aggregate specific instruction(FETCH GROUP NEXT ROW) missing from the aggregate function
4107		ER_INVALID_VALUE_TO_LIMIT	Limit only accepts integer values

4108		ER_INVISIBLE_NOT_NULL_WITHOUT_DEFAULT	Invisible column %`s must have a defa value
4109		ER_UPDATE_INFO_WITH_SYSTEM_VERSIONING	Rows matched: %ld Changed: %ld Inserted: %ld Warnings: %ld
4110		ER_VERS_FIELD_WRONG_TYPE	%`s must be of type %s for system-versioned table %`s
4111		ER_VERS_ENGINE_UNSUPPORTED	Transaction-precise system versioning %`s is not supported
4112		ER_UNUSED_23	You should never see it
4113		ER_PARTITION_WRONG_TYPE	Wrong partitioning type, expected type %`s
4114		WARN_VERS_PART_FULL	Versioned table %`s.%`s: last HISTOR partition (%`s) is out of %s, need more HISTORY partitions
4115		WARN_VERS_PARAMETERS	Maybe missing parameters: %s
4116		ER_VERS_DROP_PARTITION_INTERVAL	Can only drop oldest partitions when rotating by INTERVAL
4117		ER_UNUSED_25	You should never see it
4118		WARN_VERS_PART_NON_HISTORICAL	Partition %`s contains non-historical da
4119		ER_VERS_ALTER_NOT_ALLOWED	Not allowed for system-versioned %`s.%`s. Change @@system_versioning_alter_history t proceed with ALTER.
4120		ER_VERS_ALTER_ENGINE_PROHIBITED	Not allowed for system-versioned %`s.%`s. Change to/from native syster versioning engine is not supported.
4121		ER_VERS_RANGE_PROHIBITED	SYSTEM_TIME range selector is not allowed
4122		ER_CONFLICTING_FOR_SYSTEM_TIME	Conflicting FOR SYSTEM_TIME claus in WITH RECURSIVE
4123		ER_VERS_TABLE_MUST_HAVE_COLUMNS	Table %`s must have at least one versioned column
4124		ER_VERS_NOT_VERSIONED	Table %`s is not system-versioned
4125		ER_MISSING	Wrong parameters for %`s: missing '%
4126		ER_VERS_PERIOD_COLUMNS	PERIOD FOR SYSTEM_TIME must us columns %`s and %`s
4127		ER_PART_WRONG_VALUE	Wrong parameters for partitioned %`s: wrong value for '%s'
4128		ER_VERS_WRONG_PARTS	Wrong partitions for %`s: must have at least one HISTORY and exactly one la CURRENT
4129		ER_VERS_NO_TRX_ID	TRX_ID %llu not found in `mysql.transaction_registry`
4130		ER_VERS_ALTER_SYSTEM_FIELD	Can not change system versioning fiel %`s
4131		ER_DROP_VERSIONING_SYSTEM_TIME_PARTITION	Can not DROP SYSTEM VERSIONIN for table %`s partitioned BY SYSTEM_TIME
4132		ER_VERS_DB_NOT_SUPPORTED	System-versioned tables in the %`s database are not supported
4133		ER_VERS_TRT_IS_DISABLED	Transaction registry is disabled
4134		ER_VERS_DUPLICATE_ROW_START_END	Duplicate ROW %s column %`s
4135		ER_VERS_ALREADY_VERSIONED	Table %`s is already system-versioned
4136		ER_UNUSED_24	You should never see it
4137		ER_VERS_NOT_SUPPORTED	System-versioned tables do not suppo %s

4138		ER_VERS_TRX_PART_HISTORIC_ROW_NOT_SUPPORTED	Transaction-precise system-versioned tables do not support partitioning by R(START or ROW END
4139		ER_INDEX_FILE_FULL	The index file for table '%-.192s' is full
4140		ER_UPDATED_COLUMN_ONLY_ONCE	The column %`s.%`s cannot be change more than once in a single UPDATE statement
4141		ER_EMPTY_ROW_IN_TVC	Row with no elements is not allowed in table value constructor in this context
4142		ER_VERS_QUERY_IN_PARTITION	SYSTEM_TIME partitions in table %`s does not support historical query
4143		ER_KEY_DOESNT_SUPPORT	%s index %`s does not support this operation
4144		ER_ALTER_OPERATION_TABLE_OPTIONS_NEED_REBUILD	Changing table options requires the tal to be rebuilt
4145		ER_BACKUP_LOCK_IS_ACTIVE	Can't execute the command as you ha BACKUP STAGE active
4146		ER_BACKUP_NOT_RUNNING	You must start backup with \"BACKUP STAGE START\"
4147		ER_BACKUP_WRONG_STAGE	Backup stage '%s' is same or before current backup stage '%s'
4148		ER_BACKUP_STAGE_FAILED	Backup stage '%s' failed
4149		ER_BACKUP_UNKNOWN_STAGE	Unknown backup stage: '%s'. Stage should be one of START, FLUSH, BLOCK_DDL, BLOCK_COMMIT or EN
4150		ER_USER_IS_BLOCKED	User is blocked because of too many credential errors; unblock with 'FLUSH PRIVILEGES'
4151		ER_ACCOUNT_HAS_BEEN_LOCKED	Access denied, this account is locked
4152		ER_PERIOD_TEMPORARY_NOT_ALLOWED	Application-time period table cannot be temporary
4153		ER_PERIOD_TYPES_MISMATCH	Fields of PERIOD FOR %`s have differ types
4154		ER_MORE_THAN_ONE_PERIOD	Cannot specify more than one applicat time period
4155		ER_PERIOD_FIELD_WRONG_ATTRIBUTES	Period field %`s cannot be %s
4156		ER_PERIOD_NOT_FOUND	Period %`s is not found in table
4157		ER_PERIOD_COLUMNS_UPDATED	Column %`s used in period %`s specifi in update SET list
4158		ER_PERIOD_CONSTRAINT_DROP	Can't DROP CONSTRAINT `%s`. Use DROP PERIOD `%s` for this
4159	42000 S1009	ER_TOO_LONG_KEYPART	Specified key part was too long; max k part length is %u bytes
4160		ER_TOO_LONG_DATABASE_COMMENT	Comment for database '%-.64s' is too l (max = %u)
4161		ER_UNKNOWN_DATA_TYPE	Unknown data type: '%-.64s'
4162		ER_UNKNOWN_OPERATOR	Operator does not exists: '%-.128s'
4163		ER_WARN_HISTORY_ROW_START_TIME	Table `%s.%s` history row start '%s' is later than row end '%s'
4164		ER_PART_STARTS_BEYOND_INTERVAL	%`s: STARTS is later than query time, history partition may exceed INTERVA value
4165		ER_GALERA_REPLICATION_NOT_SUPPORTED	DDL-statement is forbidden as table storage engine does not support Galer replication

4166	HY000	ER_LOAD_INFILE_CAPABILITY_DISABLED	The used command is not allowed because the MariaDB server or client h disabled the local infile capability
4167		ER_NO_SECURE_TRANSPORTS_CONFIGURED	No secure transports are configured, unable to set --require_secure_transport=ON
4168		ER_SLAVE_IGNORED_SHARED_TABLE	Slave SQL thread ignored the '%s' because table is shared
4169		ER_NO_AUTOINCREMENT_WITH_UNIQUE	AUTO_INCREMENT column %`s cann be used in the UNIQUE index %`s
4170		ER_KEY_CONTAINS_PERIOD_FIELDS	Key %`s cannot explicitly include colun %`s
4171		ER_KEY_CANT_HAVE_WITHOUT_OVERLAPS	Key %`s cannot have WITHOUT OVERLAPS
4172		ER_NOT_ALLOWED_IN_THIS_CONTEXT	'%-.128s' is not allowed in this context
4173		ER_DATA_WAS_COMMITED_UNDER_ROLLBACK	Engine %s does not support rollback. Changes where commited during rollba call
4174		ER_PK_INDEX_CANT_BE_IGNORED	A primary key cannot be marked as IGNORE
4175		ER_BINLOG_UNSAFE_SKIP_LOCKED	SKIP LOCKED makes this statement unsafe
4176		ER_JSON_TABLE_ERROR_ON_FIELD	Field '%s' can't be set for JSON_TABL '%s'.
4177		ER_JSON_TABLE_ALIAS_REQUIRED	Every table function must have an alias
4178		ER_JSON_TABLE_SCALAR_EXPECTED	Can't store an array or an object in the scalar column '%s' of JSON_TABLE '%
4179		ER_JSON_TABLE_MULTIPLE_MATCHES	Can't store multiple matches of the pat the column '%s' of JSON_TABLE '%s'.
4180		ER_WITH_TIES_NEEDS_ORDER	FETCH ... WITH TIES requires ORDEF BY clause to be present
4181		ER_REMOVED_ORPHAN_TRIGGER	Dropped orphan trigger '%-.64s', origin created for table: '%-.192s'
4182		ER_STORAGE_ENGINE_DISABLED	Storage engine %s is disabled

1.1.2.10 Numeric Literals

Numeric literals are written as a sequence of digits from `0` to `9`. Initial zeros are ignored. A sign can always precede the digits, but it is optional for positive numbers. In decimal numbers, the integer part and the decimal part are divided with a dot (`.`).

If the integer part is zero, it can be omitted, but the literal must begin with a dot.

The notation with exponent can be used. The exponent is preceded by an `E` or `e` character. The exponent can be preceded by a sign and must be an integer. A number `N` with an exponent part `X`, is calculated as `N * POW(10, X)`.

In some cases, adding zeroes at the end of a decimal number can increment the precision of the expression where the number is used. For example, `PI()` by default returns a number with 6 decimal digits. But the `PI()+0.0000000000` expression (with 10 zeroes) returns a number with 10 decimal digits.

Hexadecimal literals are interpreted as numbers when used in numeric contexts.

Examples

```
10
+10
-10
```

All these literals are equivalent:

```
0.1
.1
+0.1
+.1
```

With exponents:

```
0.2E3 -- 0.2 * POW(10, 3) = 200
.2e3
.2e+2
1.1e-10 -- 0.00000000011
-1.1e10 -- -11000000000
```

1.1.2.11 Reserved Words

Contents

The following is a list of all reserved words in MariaDB.

Reserved words cannot be used as Identifiers, unless they are quoted.

The definitive list of reserved words for each version can be found by examining the `sql/lex.h` and `sql/sql_yacc.yy` files.

Reserved Words

Keyword	Notes
ACCESSIBLE	
ADD	
ALL	
ALTER	
ANALYZE	
AND	
AS	
ASC	
ASENSITIVE	
BEFORE	
BETWEEN	
BIGINT	
BINARY	
BLOB	
BOTH	
BY	
CALL	
CASCADE	
CASE	
CHANGE	
CHAR	

CHARACTER	
CHECK	
COLLATE	
COLUMN	
CONDITION	
CONSTRAINT	
CONTINUE	
CONVERT	
CREATE	
CROSS	
CURRENT_DATE	
CURRENT_ROLE	
CURRENT_TIME	
CURRENT_TIMESTAMP	
CURRENT_USER	
CURSOR	
DATABASE	
DATABASES	
DAY_HOUR	
DAY_MICROSECOND	
DAY_MINUTE	
DAY_SECOND	
DEC	
DECIMAL	
DECLARE	
DEFAULT	
DELAYED	
DELETE	
DELETE_DOMAIN_ID	
DESC	
DESCRIBE	
DETERMINISTIC	
DISTINCT	
DISTINCTROW	
DIV	
DO_DOMAIN_IDS	
DOUBLE	
DROP	
DUAL	
EACH	
ELSE	
ELSEIF	
ENCLOSED	

ESCAPED	
EXCEPT	Added in MariaDB 10.3.0
EXISTS	
EXIT	
EXPLAIN	
FALSE	
FETCH	
FLOAT	
FLOAT4	
FLOAT8	
FOR	
FORCE	
FOREIGN	
FROM	
FULLTEXT	
GENERAL	
GRANT	
GROUP	
HAVING	
HIGH_PRIORITY	
HOUR_MICROSECOND	
HOUR_MINUTE	
HOUR_SECOND	
IF	
IGNORE	
IGNORE_DOMAIN_IDS	
IGNORE_SERVER_IDS	
IN	
INDEX	
INFILE	
INNER	
INOUT	
INSENSITIVE	
INSERT	
INT	
INT1	
INT2	
INT3	
INT4	
INT8	
INTEGER	
INTERSECT	Added in MariaDB 10.3.0
INTERVAL	

INTO	
IS	
ITERATE	
JOIN	
KEY	
KEYS	
KILL	
LEADING	
LEAVE	
LEFT	
LIKE	
LIMIT	
LINEAR	
LINES	
LOAD	
LOCALTIME	
LOCALTIMESTAMP	
LOCK	
LONG	
LONGBLOB	
LONGTEXT	
LOOP	
LOW_PRIORITY	
MASTER_HEARTBEAT_PERIOD	
MASTER_SSL_VERIFY_SERVER_CERT	
MATCH	
MAXVALUE	
MEDIUMBLOB	
MEDIUMINT	
MEDIUMTEXT	
MIDDLEINT	
MINUTE_MICROSECOND	
MINUTE_SECOND	
MOD	
MODIFIES	
NATURAL	
NOT	
NO_WRITE_TO_BINLOG	
NULL	
NUMERIC	
OFFSET	Added in MariaDB 10.6.0
ON	
OPTIMIZE	

OPTION	
OPTIONALLY	
OR	
ORDER	
OUT	
OUTER	
OUTFILE	
OVER	
PAGE_CHECKSUM	
PARSE_VCOL_EXPR	
PARTITION	
POSITION	
PRECISION	
PRIMARY	
PROCEDURE	
PURGE	
RANGE	
READ	
READS	
READ_WRITE	
REAL	
RECURSIVE	
REF_SYSTEM_ID	
REFERENCES	
REGEXP	
RELEASE	
RENAME	
REPEAT	
REPLACE	
REQUIRE	
RESIGNAL	
RESTRICT	
RETURN	
RETURNING	
REVOKE	
RIGHT	
RLIKE	
ROW_NUMBER	From MariaDB 10.7
ROWS	
SCHEMA	
SCHEMAS	
SECOND_MICROSECOND	
SELECT	

SENSITIVE	
SEPARATOR	
SET	
SHOW	
SIGNAL	
SLOW	
SMALLINT	
SPATIAL	
SPECIFIC	
SQL	
SQLEXCEPTION	
SQLSTATE	
SQLWARNING	
SQL_BIG_RESULT	
SQL_CALC_FOUND_ROWS	
SQL_SMALL_RESULT	
SSL	
STARTING	
STATS_AUTO_RECALC	
STATS_PERSISTENT	
STATS_SAMPLE_PAGES	
STRAIGHT_JOIN	
TABLE	
TERMINATED	
THEN	
TINYBLOB	
TINYINT	
TINYTEXT	
TO	
TRAILING	
TRIGGER	
TRUE	
UNDO	
UNION	
UNIQUE	
UNLOCK	
UNSIGNED	
UPDATE	
USAGE	
USE	
USING	
UTC_DATE	
UTC_TIME	

UTC_TIMESTAMP	
VALUES	
VARBINARY	
VARCHAR	
VARCHARACTER	
VARYING	
WHEN	
WHERE	
WHILE	
WINDOW	Only disallowed for table aliases.
WITH	
WRITE	
XOR	
YEAR_MONTH	
ZEROFILL	

Exceptions

Some keywords are exceptions for historical reasons, and are permitted as unquoted identifiers. These include:

Keyword
ACTION
BIT
DATE
ENUM
NO
TEXT
TIME
TIMESTAMP

Oracle Mode

In Oracle mode, from MariaDB 10.3, there are a number of extra reserved words:

Keyword	Notes
BODY	
ELSIF	
GOTO	
HISTORY	<= MariaDB 10.3.6 only
MINUS	From MariaDB 10.6.1
OTHERS	
PACKAGE	
PERIOD	<= MariaDB 10.3.6 only
RAISE	
ROWNUM	From MariaDB 10.6.1
ROWTYPE	

SYSDATE	From MariaDB 10.6.1
SYSTEM	<= MariaDB 10.3.6 only. Note however that SYSTEM sometimes needs to be quoted to avoid confusion with System-versioned tables.
SYSTEM_TIME	<= MariaDB 10.3.6 only
VERSIONING	<= MariaDB 10.3.6 only
WITHOUT	<= MariaDB 10.3.6 only

Function Names

If the `IGNORE_SPACE` SQL_MODE flag is set, function names become reserved words.

6.4.1.1.2.1.3 SQLSTATE

1.1.2.13 String Literals

Strings are sequences of characters and are enclosed with quotes.

The syntax is:

```
[_charset_name]'string' [COLLATE collation_name]
```

For example:

```
'The MariaDB Foundation'
_utf8 'Foundation' COLLATE utf8_unicode_ci;
```

Strings can either be enclosed in single quotes or in double quotes (the same character must be used to both open and close the string).

The ANSI SQL-standard does not permit double quotes for enclosing strings, and although MariaDB does by default, if the MariaDB server has enabled the ANSI_QUOTES_SQL SQL_MODE, double quotes will be treated as being used for identifiers instead of strings.

Strings that are next to each other are automatically concatenated. For example:

```
'The ' 'MariaDB ' 'Foundation'
```

and

```
'The MariaDB Foundation'
```

are equivalent.

The `\` (backslash character) is used to escape characters (unless the SQL_MODE hasn't been set to NO_BACKSLASH_ESCAPES). For example:

```
'MariaDB's new features'
```

is not a valid string because of the single quote in the middle of the string, which is treated as if it closes the string, but is actually meant as part of the string, an apostrophe. The backslash character helps in situations like this:

```
'MariaDB\'s new features'
```

is now a valid string, and if displayed, will appear without the backslash.

```
SELECT 'MariaDB\'s new features';
+------------------------+
| MariaDB's new features |
+------------------------+
| MariaDB's new features |
+------------------------+
```

Another way to escape the quoting character is repeating it twice:

```
SELECT 'I''m here', """Double""";
+----------+----------+
| I'm here | "Double" |
+----------+----------+
| I'm here | "Double" |
+----------+----------+
```

Escape Sequences

There are other escape sequences also. Here is a full list:

Escape sequence	Character
`\0`	ASCII NUL (0x00).
`\'`	Single quote (“'”).
`\"`	Double quote (“"”).
`\b`	Backspace.
`\n`	Newline, or linefeed,.
`\r`	Carriage return.
`\t`	Tab.
`\Z`	ASCII 26 (Control+Z). See note following the table.
`\\`	Backslash (“\”).
`\%`	“%” character. See note following the table.
`\_`	A “_” character. See note following the table.

Escaping the `%` and `_` characters can be necessary when using the LIKE operator, which treats them as special characters.

The ASCII 26 character (`\Z`) needs to be escaped when included in a batch file which needs to be executed in Windows. The reason is that ASCII 26, in Windows, is the end of file (EOF).

Backslash (`\`), if not used as an escape character, must always be escaped. When followed by a character that is not in the above table, backslashes will simply be ignored.

1.1.2.14 Table Value Constructors

MariaDB starting with 10.3.3
Table Value Constructors were introduced in MariaDB 10.3.3

Syntax

```
VALUES ( row_value[, row_value...]), (...)...
```

Description

Contents

In Unions, Views, and sub-queries, a Table Value Constructor (TVC) allows you to inject arbitrary values into the result-set. The given values must have the same number of columns as the result-set, otherwise it returns Error 1222.

Examples

Using TVC's with `UNION` operations:

```
CREATE TABLE test.t1 (val1 INT, val2 INT);
INSERT INTO test.t1 VALUES(5, 8), (3, 4), (1, 2);

SELECT * FROM test.t1
UNION
VALUES (70, 90), (100, 110);

+------+------+
| val1 | val2 |
+------+------+
|    5 |    8 |
|    3 |    4 |
|    1 |    2 |
|   70 |   90 |
|  100 |  110 |
+------+------+
```

Using TVC's with a `CREATE VIEW` statement:

```
CREATE VIEW v1 AS VALUES (7, 9), (9, 10);

SELECT * FROM v1;
+---+----+
| 7 |  9 |
+---+----+
| 7 |  9 |
| 9 | 10 |
+---+----+
```

Using TVC with an `ORDER BY` clause:

```
SELECT * FROM test.t1
UNION
VALUES (10, 20), (30, 40), (50, 60), (70, 80)
ORDER BY val1 DESC;
```

Using TVC with `LIMIT` clause:

```
SELECT * FROM test.t1
UNION
VALUES (10, 20), (30, 40), (50, 60), (70, 80)
LIMIT 2 OFFSET 4;

+------+------+
| val1 | val2 |
+------+------+
|   30 |   40 |
|   50 |   60 |
+------+------+
```

1.1.2.15 User-Defined Variables

Contents

User-defined variables are variables which can be created by the user and exist in the session. This means that no one can access user-defined variables that have been set by another user, and when the session is closed these variables expire. However, these variables can be shared between several queries and stored programs.

User-defined variables names must be preceded by a single *at* character (`@`). While it is safe to use a reserved word as a user-variable name, the only allowed characters are ASCII letters, digits, dollar sign (`$`), underscore (`_`) and dot (`.`). If other characters are used, the name can be quoted in one of the following ways:

- @`var_name`
- @'var_name'

- @"var_name"

These characters can be escaped as usual.

User-variables names are case insensitive, though they were case sensitive in MySQL 4.1 and older versions.

User-defined variables cannot be declared. They can be read even if no value has been set yet; in that case, they are NULL. To set a value for a user-defined variable you can use:

- SET statement;
- := operator within a SQL statement;
- SELECT ... INTO.

Since user-defined variables type cannot be declared, the only way to force their type is using CAST() or CONVERT():

```
SET @str = CAST(123 AS CHAR(5));
```

If a variable has not been used yet, its value is NULL:

```
SELECT @x IS NULL;
+------------+
| @x IS NULL |
+------------+
|          1 |
+------------+
```

It is unsafe to read a user-defined variable and set its value in the same statement (unless the command is SET), because the order of these actions is undefined.

User-defined variables can be used in most MariaDB's statements and clauses which accept an SQL expression. However there are some exceptions, like the LIMIT clause.

They must be used to PREPARE a prepared statement:

```
@sql = 'DELETE FROM my_table WHERE c>1;';
PREPARE stmt FROM @sql;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;
```

Another common use is to include a counter in a query:

```
SET @var = 0;
SELECT a, b, c, (@var:=@var+1) AS counter FROM my_table;
```

Information Schema

User-defined variables can be viewed in the Information Schema USER_VARIABLES Table (as part of the User Variables plugin) from MariaDB 10.2.

Flushing User-Defined Variables

User-defined variables are reset and the Information Schema table emptied with the FLUSH USER_VARIABLES statement.

```
SET @str = CAST(123 AS CHAR(5));

SELECT * FROM information_schema.USER_VARIABLES ORDER BY VARIABLE_NAME;
+---------------+----------------+---------------+--------------------+
| VARIABLE_NAME | VARIABLE_VALUE | VARIABLE_TYPE | CHARACTER_SET_NAME |
+---------------+----------------+---------------+--------------------+
| str           | 123            | VARCHAR       | utf8mb3            |
+---------------+----------------+---------------+--------------------+

FLUSH USER_VARIABLES;

SELECT * FROM information_schema.USER_VARIABLES ORDER BY VARIABLE_NAME;
Empty set (0.000 sec)
```

1.1.3 Geographic & Geometric Features

MariaDB supports spatial extensions that enable the creation, storage and analysis of geographic features. These can be used in the Aria, MyISAM, InnoDB/XtraDB and ARCHIVE engines in MariaDB.

Partitioned tables do not support geometric types.

GIS Resources

Resources for those interested in GIS

GIS features in 5.3.3

Basic information about the existing spatial features can be found in the G...

Geometry Types

Supported geometry types.

Geometry Hierarchy

The base Geometry class has subclasses for Point, Curve, Surface and GeometryCollection

Geometry Constructors

Geometry constructors

Geometry Properties

Geometry properties

Geometry Relations

Geometry relations

LineString Properties

LineString properties

MBR (Minimum Bounding Rectangle)

Point Properties

Point properties

Polygon Properties

Polygon properties

WKB

Well-Known Binary format for geometric data

WKT

Well-Known Text geometry representation

MySQL/MariaDB Spatial Support Matrix

Table comparing when different spatial features were introduced into MySQL and MariaDB

SPATIAL INDEX

An index type used for geometric columns.

MariaDB Plans - GIS

Old GIS plans

The maria/5.3-gis tree on Launchpad.

Note: This page is obsolete. The information is old, outdated, or otherwise...

GeoJSON

GeoJSON functions

There are 5 related questions.

1.1.3.1 GIS Resources

Here are a few resources for those interested in GIS in MariaDB.

- OGC Simple Feature Access - the Open Geospatial Consortium's OpenGIS Simple Features Specifications For SQL.
- Geo/Spatial Search with MySQL - a presentation by Alexander Rubin, from the MySQL Conference in 2006.

There are currently no differences between GIS in stable versions of MariaDB and GIS in MySQL. There are, however, some extensions and enhancements being worked on. See "MariaDB Plans - GIS" for more information.

1.1.3.2 GIS features in 5.3.3

Basic information about the existing spatial features can be found in the Geographic Features section of the Knowlegebase. The Spatial Extensions page of the MySQL manual also applies to MariaDB.

The MariaDB 5.3.3 release , contains code improving the spatial functionality in MariaDB.

MySQL operates on spatial data based on the OpenGIS standards, particularly the OpenGIS SFS (Simple feature access, SQL option).

Initial support was based on version 05-134 of the standard. MariaDB implements a subset of the 'SQL with Geometry Types' environment proposed by the OGC. And the SQL environment was extended with a set of geometry types.

MariaDB supports spatial extensions to operate on spatial features. These features are available for Aria, MyISAM, InnoDB, NDB, and ARCHIVE tables.

For spatial columns, Aria and MyISAM supports both SPATIAL and non-SPATIAL indexes. Other storage engines support non-SPATIAL indexes.

The most recent changes in the code are aimed at meeting the OpenGIS requirements. One thing missed in previous versions is that the functions which check spatial relations didn't consider the actual shape of an object, instead they operate only on their bounding rectangles. These legacy functions have been left as they are and new, properly-working functions are named with an ' `ST_` ' prefix, in accordance with the latest OpenGIS requirements. Also, operations over geometry features were added.

The list of new functions:

Spatial operators. They produce new geometries.

Name	Description
ST_UNION(A, B)	union of A and B
ST_INTERSECTION(A, B)	intersection of A and B
ST_SYMDIFFERENCE(A, B)	symdifference, notintersecting parts of A and B
ST_BUFFER(A, radius)	returns the shape of the area that lies in 'radius' distance from the shape A.

Predicates, return boolean result of the relationship

Name	Description
ST_INTERSECTS(A, B)	if A and B have an intersection
ST_CROSSES(A, B)	if A and B cross
ST_EQUALS(A, B)	if A and B are equal
ST_WITHIN(A, B)	if A lies within B
ST_CONTAINS(A,B)	if B lies within A
ST_DISJOINT(A,B)	if A and B have no intersection
ST_TOUCHES(A,B)	if A touches B

1.1.3.3 Geometry Types

Contents

Description

MariaDB provides a standard way of creating spatial columns for geometry types, for example, with CREATE TABLE or ALTER TABLE. Currently, spatial columns are supported for MyISAM, InnoDB and ARCHIVE tables. See also SPATIAL INDEX.

The basic geometry type is `GEOMETRY`. But the type can be more specific. The following types are supported:

Geometry Types
POINT
LINESTRING
POLYGON
MULTIPOINT
MULTILINESTRING
MULTIPOLYGON
GEOMETRYCOLLECTION
GEOMETRY

Examples

Note: For clarity, only one type is listed per table in the examples below, but a table row can contain multiple types. For example:

```
CREATE TABLE object (shapeA POLYGON, shapeB LINESTRING);
```

POINT

```
CREATE TABLE gis_point  (g POINT);
SHOW FIELDS FROM gis_point;
INSERT INTO gis_point VALUES
    (PointFromText('POINT(10 10)')),
    (PointFromText('POINT(20 10)')),
    (PointFromText('POINT(20 20)')),
    (PointFromWKB(AsWKB(PointFromText('POINT(10 20)'))));
```

LINESTRING

```
CREATE TABLE gis_line  (g LINESTRING);
SHOW FIELDS FROM gis_line;
INSERT INTO gis_line VALUES
    (LineFromText('LINESTRING(0 0,0 10,10 0)')),
    (LineStringFromText('LINESTRING(10 10,20 10,20 20,10 20,10 10)')),
    (LineStringFromWKB(AsWKB(LineString(Point(10, 10), Point(40, 10)))));
```

POLYGON

```
CREATE TABLE gis_polygon    (g POLYGON);
SHOW FIELDS FROM gis_polygon;
INSERT INTO gis_polygon VALUES
    (PolygonFromText('POLYGON((10 10,20 10,20 20,10 20,10 10))')),
    (PolyFromText('POLYGON((0 0,50 0,50 50,0 50,0 0), (10 10,20 10,20 20,10 20,10 10))')),
    (PolyFromWKB(AsWKB(Polygon(LineString(Point(0, 0), Point(30, 0), Point(30, 30), Point(0,
0))))));
```

MULTIPOINT

```
CREATE TABLE gis_multi_point (g MULTIPOINT);
SHOW FIELDS FROM gis_multi_point;
INSERT INTO gis_multi_point VALUES
    (MultiPointFromText('MULTIPOINT(0 0,10 10,10 20,20 20)')),
    (MPointFromText('MULTIPOINT(1 1,11 11,11 21,21 21)')),
    (MPointFromWKB(AsWKB(MultiPoint(Point(3, 6), Point(4, 10)))));
```

MULTILINESTRING

```
CREATE TABLE gis_multi_line (g MULTILINESTRING);
SHOW FIELDS FROM gis_multi_line;
INSERT INTO gis_multi_line VALUES
    (MultiLineStringFromText('MULTILINESTRING((10 48,10 21,10 0),(16 0,16 23,16 48))')),
    (MLineFromText('MULTILINESTRING((10 48,10 21,10 0))')),
    (MLineFromWKB(AsWKB(MultiLineString(LineString(Point(1, 2), Point(3, 5)),
LineString(Point(2, 5), Point(5, 8), Point(21, 7))))));
```

MULTIPOLYGON

```
CREATE TABLE gis_multi_polygon  (g MULTIPOLYGON);
SHOW FIELDS FROM gis_multi_polygon;
INSERT INTO gis_multi_polygon VALUES
    (MultiPolygonFromText('MULTIPOLYGON(((28 26,28 0,84 0,84 42,28 26),(52 18,66 23,73 9,48
6,52 18)),((59 18,67 18,67 13,59 13,59 18)))')),
    (MPolyFromText('MULTIPOLYGON(((28 26,28 0,84 0,84 42,28 26),(52 18,66 23,73 9,48 6,52 18)),
((59 18,67 18,67 13,59 13,59 18)))')),
    (MPolyFromWKB(AsWKB(MultiPolygon(Polygon(LineString(Point(0, 3), Point(3, 3), Point(3, 0),
Point(0, 3)))))));
```

GEOMETRYCOLLECTION

```
CREATE TABLE gis_geometrycollection  (g GEOMETRYCOLLECTION);
SHOW FIELDS FROM gis_geometrycollection;
INSERT INTO gis_geometrycollection VALUES
    (GeomCollFromText('GEOMETRYCOLLECTION(POINT(0 0), LINESTRING(0 0,10 10))')),
    (GeometryFromWKB(AsWKB(GeometryCollection(Point(44, 6), LineString(Point(3, 6), Point(7,
9))))),
    (GeomFromText('GeometryCollection()')),
    (GeomFromText('GeometryCollection EMPTY'));
```

GEOMETRY

```
CREATE TABLE gis_geometry (g GEOMETRY);
SHOW FIELDS FROM gis_geometry;
INSERT into gis_geometry SELECT * FROM gis_point;
INSERT into gis_geometry SELECT * FROM gis_line;
INSERT into gis_geometry SELECT * FROM gis_polygon;
INSERT into gis_geometry SELECT * FROM gis_multi_point;
INSERT into gis_geometry SELECT * FROM gis_multi_line;
INSERT into gis_geometry SELECT * FROM gis_multi_polygon;
INSERT into gis_geometry SELECT * FROM gis_geometrycollection;
```

1.1.3.4 Geometry Hierarchy

Description

Geometry is the base class. It is an abstract class. The instantiable subclasses of Geometry are restricted to zero-, one-, and two-dimensional geometric objects that exist in two-dimensional coordinate space. All instantiable geometry classes are defined so that valid instances of a geometry class are topologically closed (that is, all defined geometries include their boundary).

The base Geometry class has subclasses for Point, Curve, Surface, and GeometryCollection:

- Point represents zero-dimensional objects.
- Curve represents one-dimensional objects, and has subclass LineString, with sub-subclasses Line and LinearRing.
- Surface is designed for two-dimensional objects and has subclass Polygon.
- GeometryCollection has specialized zero-, one-, and two-dimensional collection classes named MultiPoint, MultiLineString, and MultiPolygon for modeling geometries corresponding to collections of Points, LineStrings, and Polygons, respectively. MultiCurve and MultiSurface are introduced as abstract superclasses that generalize the collection interfaces to handle Curves and Surfaces.

Geometry, Curve, Surface, MultiCurve, and MultiSurface are defined as non-instantiable classes. They define a common set of methods for their subclasses and are included for extensibility.

Point, LineString, Polygon, GeometryCollection, MultiPoint, MultiLineString, and MultiPolygon are instantiable classes.

1.2.9.3.1 Geometry Constructors

1.2.9.3.2 Geometry Properties

1.2.9.3.3 Geometry Relations

1.2.9.3.4 LineString Properties

1.2.9.3.5 MBR (Minimum Bounding Rectangle)

1.2.9.3.6 Point Properties

1.2.9.3.7 Polygon Properties

1.2.9.3.8 WKB

1.2.9.3.9 WKT

1.1.3.14 MySQL/MariaDB Spatial Support Matrix

This table shows when different spatial features were introduced into MySQL and MariaDB.

My	MySQL
MDB	MariaDB
x	This feature is supported.
MBR	This feature is present, but operates on the Minimum Bounding Rectangle instead of the actual shape.
d	This feature is present, but has been deprecated and will be removed in a future version.
*****	This feature is present, but may not work the way you expect.
-	This feature is not supported.

	My 5.4.2	My 5.5	My 5.6.1	My 5.7.4	My 5.7.5	My 5.7.6	MDB 5.1	MDB 5.3.3	MDB 10.1.2	MDB 10.2

InnoDB Spatial Indexes	-	-	-	-	x	x	-	-	-	x
MyISAM Spatial Indexes	x	x	x	x	x	x	x	x	x	x
Aria Spatial Indexes	-	-	-	-	-	-	x	x	x	x
Area	x	x	x	x	x	d	x	x	x	x
AsBinary	x	x	x	x	x	d	x	x	x	x
AsText	x	x	x	x	x	d	x	x	x	x
AsWKB	x	x	x	x	x	d	x	x	x	x
AsWKT	x	x	x	x	x	d	x	x	x	x
Boundary	-	-	-	-	-	-	-	-	x	x
Buffer	-	-	x	x	x	d	x	x	x	x
Centroid	-	x	x	x	x	d	x	x	x	x
Contains	MBR	MBR	MBR	MBR	MBR	d	MBR	MBR	MBR	MBR
ConvexHull	-	-	-	-	x	d	-	-	x	x
Crosses	MBR	x	x	x	x	d	MBR	MBR	MBR	MBR
Dimension	x	x	x	x	x	d	x	x	x	x
Disjoint	MBR	MBR	MBR	MBR	MBR	d	MBR	MBR	MBR	MBR
Distance	MBR	-	-	x	x	d	-	-	-	-
EndPoint	x	x	x	x	x	d	x	x	x	x
Envelope	x	x	x	x	x	d	x	x	x	x
Equals	MBR	MBR	MBR	MBR	MBR	d	MBR	MBR	MBR	MBR
	My 5.4.2	**My 5.5**	**My 5.6.1**	**My 5.7.4**	**My 5.7.5**	**My 5.7.6**	**MDB 5.1**	**MDB 5.3.3**	**MDB 10.1.2**	**MDB 10.2**
ExteriorRing	x	x	x	x	x	d	x	x	x	x
GeomCollFromText	x	x	x	x	x	d	x	x	x	x
GeomCollFromWKB	x	x	x	x	x	d	x	x	x	x
GeometryCollection	x	x	x	x	x	x	x	x	x	x
GeometryCollectionFromText	x	x	x	x	x	d	x	x	x	x
GeometryCollectionFromWKB	x	x	x	x	x	d	x	x	x	x
GeometryFromText	x	x	x	x	x	d	x	x	x	x
GeometryFromWKB	x	x	x	x	x	d	x	x	x	x
GeometryN	x	x	x	x	x	d	x	x	x	x
GeometryType	x	x	x	x	x	d	x	x	x	x
GeomFromText	x	x	x	x	x	d	x	x	x	x
GeomFromWKB	x	x	x	x	x	d	x	x	x	x
GLength	x	x	x	x	x	d	x	x	x	x
InteriorRingN	x	x	x	x	x	d	x	x	x	x
Intersects	MBR	MBR	MBR	MBR	MBR	d	MBR	MBR	MBR	MBR
IsClosed	x	x	x	x	x	d	x	x	x	x
IsEmpty	-	*	*	*	*	d	x	x	x	x
IsRing	-	-	-	-	-	-	-	-	x	x
IsSimple	-	*	*	x	x	d	-	x	x	x
LineFromText	x	x	x	x	x	d	x	x	x	x
	My 5.4.2	**My 5.5**	**My 5.6.1**	**My 5.7.4**	**My 5.7.5**	**My 5.7.6**	**MDB 5.1**	**MDB 5.3.3**	**MDB 10.1.2**	**MDB 10.2**

LineFromWKB	x	x	x	x	x	d	x	x	x	x
LineString	x	x	x	x	x	x	x	x	x	x
LineStringFromText	x	x	x	x	x	d	x	x	x	x
LineStringFromWKB	x	x	x	x	x	d	x	x	x	x
MBRContains	MBR	MBR	MBR	MBR	MBR	MBR	MBR	MBR	MBR	MBR
MBRCoveredBy	-	-	-	MBR	MBR	MBR	-	-	-	-
MBRDisjoint	MBR	MBR	MBR	MBR	MBR	MBR	MBR	MBR	MBR	MBR
MBREqual	MBR	MBR	MBR	MBR	MBR	MBR	MBR	MBR	MBR	MBR
MBREquals	-	-	-	MBR	MBR	MBR	-	-	-	MBR
MBRIntersects	MBR	MBR	MBR	MBR	MBR	MBR	MBR	MBR	MBR	MBR
MBROverlaps	MBR	MBR	MBR	MBR	MBR	MBR	MBR	MBR	MBR	MBR
MBRTouches	MBR	MBR	MBR	MBR	MBR	MBR	MBR	MBR	MBR	MBR
MBRWithin	MBR	MBR	MBR	MBR	MBR	MBR	MBR	MBR	MBR	MBR
MLineFromText	x	x	x	x	x	d	x	x	x	x
MLineFromWKB	x	x	x	x	x	d	x	x	x	x
MPointFromText	x	x	x	x	x	d	x	x	x	x
MPointFromWKB	x	x	x	x	x	d	x	x	x	x
MPolyFromText	x	x	x	x	x	d	x	x	x	x
MPolyFromWKB	x	x	x	x	x	d	x	x	x	x
MultiLineString	x	x	x	x	x	x	x	x	x	x
	My 5.4.2	**My 5.5**	**My 5.6.1**	**My 5.7.4**	**My 5.7.5**	**My 5.7.6**	**MDB 5.1**	**MDB 5.3.3**	**MDB 10.1.2**	**MDB 10.2**
MultiLineStringFromText	x	x	x	x	x	d	x	x	x	x
MultiLineStringFromWKB	x	x	x	x	x	d	x	x	x	x
MultiPoint	x	x	x	x	x	x	x	x	x	x
MultiPointFromText	x	x	x	x	x	d	x	x	x	x
MultiPointFromWKB	x	x	x	x	x	d	x	x	x	x
MultiPolygon	x	x	x	x	x	x	x	x	x	x
MultiPolygonFromText	x	x	x	x	x	d	x	x	x	x
MultiPolygonFromWKB	x	x	x	x	x	d	x	x	x	x
NumGeometries	x	x	x	x	x	d	x	x	x	x
NumInteriorRings	x	x	x	x	x	d	x	x	x	x
NumPoints	x	x	x	x	x	d	x	x	x	x
Overlaps	MBR	MBR	MBR	MBR	MBR	d	MBR	MBR	MBR	MBR
Point	x	x	x	x	x	x	x	x	x	x
PointFromText	x	x	x	x	x	d	x	x	x	x
PointFromWKB	x	x	x	x	x	d	x	x	x	x
PointOnSurface	-	-	-	-	-	-	-	-	x	x
PointN	x	x	x	x	x	d	x	x	x	x
PolyFromText	x	x	x	x	x	d	x	x	x	x
PolyFromWKB	x	x	x	x	x	d	x	x	x	x
Polygon	x	x	x	x	x	x	x	x	x	x
	My 5.4.2	**My 5.5**	**My 5.6.1**	**My 5.7.4**	**My 5.7.5**	**My 5.7.6**	**MDB 5.1**	**MDB 5.3.3**	**MDB 10.1.2**	**MDB 10.2**

PolygonFromText	x	x	x	x	x	d	x	x	x	x
PolygonFromWKB	x	x	x	x	x	d	x	x	x	x
SRID	x	x	x	x	x	d	x	x	x	x
ST_Area	-	-	x	x	x	x	-	x	x	x
ST_AsBinary	-	-	x	x	x	x	-	x	x	x
ST_AsGeoJSON	-	-	-	x	x	x	-	-	-	x
ST_AsText	-	-	x	x	x	x	-	x	x	x
ST_AsWKB	-	-	x	x	x	x	-	x	x	x
ST_AsWKT	-	-	x	x	x	x	-	x	x	x
ST_Boundary	-	-	-	-	-	-	-	-	x	x
ST_Buffer	-	-	x	x	x	x	-	x	x	x
ST_Buffer_Strategy	-	-	-	x	x	x	-	-	-	-
ST_Centroid	-	-	x	x	x	x	-	x	x	x
ST_Contains	-	-	x	x	x	x	-	x	x	x
ST_ConvexHull	-	-	-	-	x	x	-	-	x	x
ST_Crosses	-	-	x	x	x	x	-	x	x	x
ST_Difference	-	-	x	x	x	x	-	x	x	x
ST_Dimension	-	-	x	x	x	x	-	x	x	x
ST_Disjoint	-	-	x	x	x	x	-	x	x	x
ST_Distance	-	-	x	x	x	x	-	x	x	x

	My 5.4.2	My 5.5	My 5.6.1	My 5.7.4	My 5.7.5	My 5.7.6	MDB 5.1	MDB 5.3.3	MDB 10.1.2	MDB 10.2
ST_Distance_Sphere	-	-	-	-	-	x	-	-	-	-
ST_EndPoint	-	-	x	x	x	x	-	x	x	x
ST_Envelope	-	-	x	x	x	x	-	x	x	x
ST_Equals	-	-	x	x	x	x	-	x	x	x
ST_ExteriorRing	-	-	x	x	x	x	-	x	x	x
ST_GeoHash	-	-	-	-	x	x	-	-	-	-
ST_GeomCollFromText	-	-	x	x	x	x	-	x	x	x
ST_GeomCollFromWKB	-	-	x	x	x	x	-	x	x	x
ST_GeometryCollectionFromText	-	-	x	x	x	x	-	x	x	x
ST_GeometryCollectionFromWKB	-	-	x	x	x	x	-	x	x	x
ST_GeometryFromText	-	-	x	x	x	x	-	x	x	x
ST_GeometryFromWKB	-	-	x	x	x	x	-	x	x	x
ST_GeometryN	-	-	x	x	x	x	-	x	x	x
ST_GeometryType	-	-	x	x	x	x	-	x	x	x
ST_GeomFromGeoJSON	-	-	-	-	x	x	-	-	-	x
ST_GeomFromText	-	-	x	x	x	x	-	x	x	x
ST_GeomFromWKB	-	-	x	x	x	x	-	x	x	x
ST_InteriorRingN	-	-	x	x	x	x	-	x	x	x
ST_Intersection	-	-	x	x	x	x	-	x	x	x
ST_Intersects	-	-	x	x	x	x	-	x	x	x

	My 5.4.2	My 5.5	My 5.6.1	My 5.7.4	My 5.7.5	My 5.7.6	MDB 5.1	MDB 5.3.3	MDB 10.1.2	MDB 10.2

ST_IsClosed	-	-	x	x	x	x	-	x	x	x
ST_IsEmpty	-	-	x	x	x	x	-	x	x	x
ST_IsRing	-	-	-	-	-	-	-	-	x	x
ST_IsSimple	-	-	x	x	x	x	-	x	x	x
ST_IsValid	-	-	-	-	-	x	-	-	-	-
ST_LatFromGeoHash	-	-	-	-	x	x	-	-	-	-
ST_Length	-	-	-	-	-	x	-	x	x	x
ST_LineFromText	-	-	x	x	x	x	-	x	x	x
ST_LineFromWKB	-	-	x	x	x	x	-	x	x	x
ST_LineStringFromText	-	-	x	x	x	x	-	x	x	x
ST_LineStringFromWKB	-	-	x	x	x	x	-	x	x	x
ST_LongFromGeoHash	-	-	-	-	x	x	-	-	-	-
ST_NumGeometries	-	-	x	x	x	x	-	x	x	x
ST_NumInteriorRings	-	-	x	x	x	x	-	x	x	x
ST_NumPoints	-	-	x	x	x	x	-	x	x	x
ST_Overlaps	-	-	x	x	x	x	-	x	x	x
ST_PointFromGeoHash	-	-	-	-	x	x	-	-	-	-
ST_PointFromText	-	-	x	x	x	x	-	x	x	x
ST_PointFromWKB	-	-	x	x	x	x	-	x	x	x
ST_PointOnSurface	-	-	-	-	-	-	-	-	x	x
	My 5.4.2	My 5.5	My 5.6.1	My 5.7.4	My 5.7.5	My 5.7.6	MDB 5.1	MDB 5.3.3	MDB 10.1.2	MDB 10.2
ST_PointN	-	-	x	x	x	x	-	x	x	x
ST_PolyFromText	-	-	x	x	x	x	-	x	x	x
ST_PolyFromWKB	-	-	x	x	x	x	-	x	x	x
ST_PolygonFromText	-	-	x	x	x	x	-	x	x	x
ST_PolygonFromWKB	-	-	x	x	x	x	-	x	x	x
ST_Relate	-	-	-	-	-	-	-	-	x	x
ST_Simplify	-	-	-	-	-	x	-	-	-	-
ST_SRID	-	-	x	x	x	x	-	x	x	x
ST_StartPoint	-	-	x	x	x	x	-	x	x	x
ST_SymDifference	-	-	x	x	x	x	-	x	x	x
ST_Touches	-	-	x	x	x	x	-	x	x	x
ST_Union	-	-	x	x	x	x	-	x	x	x
ST_Validate	-	-	-	-	-	x	-	-	-	-
ST_Within	-	-	x	x	x	x	-	x	x	x
ST_X	-	-	x	x	x	x	-	x	x	x
ST_Y	-	-	x	x	x	x	-	x	x	x
StartPoint	x	x	x	x	x	d	x	x	x	x
Touches	MBR	x	x	x	x	d	MBR	MBR	MBR	MBR
Within	MBR	MBR	MBR	MBR	MBR	d	MBR	MBR	MBR	MBR
X	x	x	x	x	x	d	x	x	x	x
Y	x	x	x	x	x	d	x	x	x	x

1.1.3.15 SPATIAL INDEX

Contents

Description

On MyISAM, Aria and InnoDB tables, MariaDB can create spatial indexes (an R-tree index) using syntax similar to that for creating regular indexes, but extended with the `SPATIAL` keyword. Currently, columns in spatial indexes must be declared `NOT NULL`.

Spatial indexes can be created when the table is created, or added after the fact like so:

- with CREATE TABLE:

```
CREATE TABLE geom (g GEOMETRY NOT NULL, SPATIAL INDEX(g));
```

- with ALTER TABLE:

```
ALTER TABLE geom ADD SPATIAL INDEX(g);
```

- with CREATE INDEX:

```
CREATE SPATIAL INDEX sp_index ON geom (g);
```

`SPATIAL INDEX` creates an `R-tree` index. For storage engines that support non-spatial indexing of spatial columns, the engine creates a `B-tree` index. A `B-tree` index on spatial values is useful for exact-value lookups, but not for range scans.

For more information on indexing spatial columns, see CREATE INDEX.

To drop spatial indexes, use ALTER TABLE or DROP INDEX:

- with ALTER TABLE:

```
ALTER TABLE geom DROP INDEX g;
```

- with DROP INDEX:

```
DROP INDEX sp_index ON geom;
```

Data-at-Rest Encyption

Before MariaDB 10.4.3, InnoDB's spatial indexes could not be encrypted. If an InnoDB table was encrypted and if it contained spatial indexes, then those indexes would be unencrypted.

In MariaDB 10.4.3 and later, if `innodb_checksum_algorithm` is set to `full_crc32` or `strict_full_crc32`, and if the table does not use `ROW_FORMAT=COMPRESSED`, then InnoDB spatial indexes will be encrypted if the table is encrypted.

See MDEV-12026 for more information.

1.1.3.16 GeoJSON

GeoJSON is a format for encoding various geographic data structures.

ST_AsGeoJSON

Returns the given geometry as a GeoJSON element.

ST_GeomFromGeoJSON

Given a GeoJSON input, returns a geometry object

1.1.3.16.1 ST_AsGeoJSON

Syntax

```
ST_AsGeoJSON(g[, max_decimals[, options]])
```

Description

Returns the given geometry *g* as a GeoJSON element. The optional *max_decimals* limits the maximum number of decimals displayed.

The optional *options* flag can be set to `1` to add a bounding box to the output.

Examples

```
SELECT ST_AsGeoJSON(ST_GeomFromText('POINT(5.3 7.2)'));
+-------------------------------------------------+
| ST_AsGeoJSON(ST_GeomFromText('POINT(5.3 7.2)')) |
+-------------------------------------------------+
| {"type": "Point", "coordinates": [5.3, 7.2]}    |
+-------------------------------------------------+
```

1.1.3.16.2 ST_GeomFromGeoJSON

MariaDB starting with 10.2.4
ST_GeomFromGeoJSON was added in MariaDB 10.2.4

Syntax

```
ST_GeomFromGeoJSON(g[, option])
```

Description

Given a GeoJSON input *g*, returns a geometry object. The *option* specifies what to do if *g* contains geometries with coordinate dimensions higher than 2.

Option	Description
1	Return an error (the default)
2 - 4	The document is accepted, but the coordinates for higher coordinate dimensions are stripped off.

Note that this function did not work correctly before MariaDB 10.2.8 - see MDEV-12180.

Examples

```
SET @j = '{ "type": "Point", "coordinates": [5.3, 15.0]}';

SELECT ST_AsText(ST_GeomFromGeoJSON(@j));
+-----------------------------------+
| ST_AsText(ST_GeomFromGeoJSON(@j)) |
+-----------------------------------+
| POINT(5.3 15)                     |
+-----------------------------------+
```

1.1.4 NoSQL

MariaDB supports a lot of commands and interfaces that are closer to NoSQL than to SQL.

CONNECT

The CONNECT storage engine enables MariaDB to access external local or remote data.

HANDLER

Direct access to reading rows from the storage engine.

HandlerSocket

A NoSQL plugin giving you direct access to InnoDB and SPIDER.

Dynamic Columns

Dynamic columns allow one to store different sets of columns for each row in a table

Dynamic Columns from MariaDB 10

Improvements to Dynamic Columns from MariaDB 10.

Dynamic Column API

Client-side API for reading and writing Dynamic Columns blobs

Dynamic Columns API

MariaDB 10.0 API for reading and writing dynamic-columns blobs

JSON Functions

Built-in functions related to JSON.

LOAD_FILE

Returns file contents as a string.

Cassandra Storage Engine

A storage engine interface to Cassandra.

There are 1 related questions.

5.3.7 CONNECT

1.1.4.2 HANDLER

The `HANDLER` statements give you direct access to reading rows from the storage engine. This is much faster than normal access through SELECT as there is less parsing involved and no optimizer involved.

You can use prepared statements for `HANDLER READ`, which should give you a speed comparable to HandlerSocket. Also see Yoshinori Matsunobu's blog post Using MySQL as a NoSQL - A story for exceeding 750,000 qps on a commodity server.

HANDLER Commands

Direct access to table storage engine interfaces for key lookups and key or table scans.

HANDLER for MEMORY Tables

Using HANDLER commands efficiently with MEMORY/HEAP tables

1.1.4.2.1 HANDLER Commands

Syntax

```
HANDLER tbl_name OPEN [ [AS] alias]
HANDLER tbl_name READ index_name { = | >= | <= | < } (value1,value2,...)
    [ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name READ index_name { FIRST | NEXT | PREV | LAST }
    [ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name READ { FIRST | NEXT }
    [ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name CLOSE
```

Contents

Description

The `HANDLER` statement provides direct access to table storage engine interfaces for key lookups and key or table scans. It is available for at least Aria, Memory, MyISAM and InnoDB tables (and should work with most 'normal' storage engines, but not with system tables, MERGE or views).

`HANDLER ... OPEN` opens a table, allowing it to be accessible to subsequent `HANDLER ... READ` statements. The table can either be opened using an alias (which must then be used by `HANDLER ... READ`, or a table name.

The table object is only closed when `HANDLER ... CLOSE` is called by the session, and is not shared by other sessions.

Prepared statements work with `HANDLER READ`, which gives a much higher performance (50% speedup) as there is no parsing and all data is transformed in binary (without conversions to text, as with the normal protocol).

The HANDLER command does not work with partitioned tables.

Key Lookup

A key lookup is started with:

```
HANDLER tbl_name READ index_name { = | >= | <= | < }  (value,value) [LIMIT...]
```

The values stands for the value of each of the key columns. For most key types (except for HASH keys in MEMORY storage engine) you can use a prefix subset of it's columns.

If you are using LIMIT, then in case of >= or > then there is an implicit NEXT implied, while if you are using <= or < then there is an implicit PREV implied.

After the initial read, you can use

```
HANDLER tbl_name READ index_name NEXT [ LIMIT ... ]
or
HANDLER tbl_name READ index_name PREV [ LIMIT ... ]
```

to scan the rows in key order.

Note that the row order is not defined for keys with duplicated values and will vary from engine to engine.

Key Scans

You can scan a table in key order by doing:

```
HANDLER tbl_name READ index_name FIRST [ LIMIT ... ]
HANDLER tbl_name READ index_name NEXT  [ LIMIT ... ]
```

or, if the handler supports backwards key scans (most do):

```
HANDLER tbl_name READ index_name LAST [ LIMIT ... ]
HANDLER tbl_name READ index_name PREV [ LIMIT ... ]
```

Table Scans

You can scan a table in row order by doing:

```
HANDLER tbl_name READ FIRST [ LIMIT ... ]
HANDLER tbl_name READ NEXT  [ LIMIT ... ]
```

Limitations

As this is a direct interface to the storage engine, some limitations may apply for what you can do and what happens if the table changes. Here follows some of the common limitations:

Finding 'Old Rows'

HANDLER READ is not transaction safe, consistent or atomic. It's ok for the storage engine to returns rows that existed when you started the scan but that were later deleted. This can happen as the storage engine may cache rows as part of the scan from a previous read.

You may also find rows committed since the scan originally started.

Invisible Columns

`HANDLER ... READ` also reads the data of invisible-columns.

System-Versioned Tables

`HANDLER ... READ` reads everything from system-versioned tables, and so includes `row_start` and `row_end` fields, as well as all rows that have since been deleted or changed, including when history partitions are used.

Other Limitations

- If you do an ALTER TABLE, all your HANDLERs for that table are automatically closed.
- If you do an ALTER TABLE for a table that is used by some other connection with HANDLER, the ALTER TABLE will wait for the HANDLER to be closed.
- For HASH keys, you must use all key parts when searching for a row.
- For HASH keys, you can't do a key scan of all values. You can only find all rows with the same key value.
- While each HANDLER READ command is atomic, if you do a scan in many steps, then some engines may give you error 1020 if the table changed between the commands. Please refer to the specific engine handler page if this happens.

Error Codes

- Error 1031 (ER_ILLEGAL_HA) Table storage engine for 't1' doesn't have this option
  - If you get this for HANDLER OPEN it means the storage engine doesn't support HANDLER calls.
  - If you get this for HANDLER READ it means you are trying to use an incomplete HASH key.
- Error 1020 (ER_CHECKREAD) Record has changed since last read in table '...'
  - This means that the table changed between two reads and the handler can't handle this case for the given scan.

1.1.4.2.2 HANDLER for MEMORY Tables

This article explains how to use HANDLER commands efficiently with MEMORY/HEAP tables.

If you want to scan a table for over different key values, not just search for exact key values, you should create your keys with 'USING BTREE':

```
CREATE TABLE t1 (a INT, b INT, KEY(a), KEY b USING BTREE (b)) engine=memory;
```

In the above table, `a` is a HASH key that only supports exact matches (=) while `b` is a BTREE key that you can use to scan the table in key order, starting from start or from a given key value.

The limitations for HANDLER READ with Memory|HEAP tables are:

Limitations for HASH keys

- You must use all key parts when searching for a row.
- You can't do a key scan of all values. You can only find all rows with the same key value.
- READ NEXT gives error 1031 if the tables changed since last read.

Limitations for BTREE keys

- READ NEXT gives error 1031 if the tables changed since last read. This limitation can be lifted in the future.

Limitations for table scans

- READ NEXT gives error 1031 if the table was truncated since last READ call.

1.1.4.3 HandlerSocket

HandlerSocket gives you direct access to InnoDB and SPIDER. It is included in MariaDB as a ready-to use plugin.

HandlerSocket is a NoSQL plugin for MariaDB. It works as a daemon inside the mysqld process, accepting TCP connections, and executing requests from clients. HandlerSocket does not support SQL queries. Instead, it supports simple CRUD operations on tables.

HandlerSocket can be much faster than mysqld/libmysql in some cases because it has lower CPU, disk, and network overhead:

1. To lower CPU usage it does not parse SQL.
2. Next, it batch-processes requests where possible, which further reduces CPU usage and lowers disk usage.
3. Lastly, the client/server protocol is very compact compared to mysql/libmysql, which reduces network usage.

HandlerSocket Installation

Installing the HandlerSocket plugin.

HandlerSocket Configuration Options

HandlerSocket configuration options.

HandlerSocket Client Libraries

Available HandlerSocket Client Libraries

Testing HandlerSocket in a Source Distribution

Testing HandlerSocket in a source distribution.

HandlerSocket External Resources

HandlerSocket external resources and documentation

1.1.4.3.1 HandlerSocket Installation

After MariaDB is installed, use the INSTALL PLUGIN command (as the root user) to install the HandlerSocket plugin. This command only needs to be run once, like so:

```
INSTALL PLUGIN handlersocket SONAME 'handlersocket.so';
```

After installing the plugin, SHOW PROCESSLIST you first need to configure some settings. All HandlerSocket configuration options are placed in the [mysqld] section of your my.cnf file.

At least the handlersocket_address, handlersocket_port and handlersocket_port_wr options need to be set. For example:

```
handlersocket_address="127.0.0.1"
handlersocket_port="9998"
handlersocket_port_wr="9999"
```

After updating the configuration options, restart MariaDB.

On the client side, to make use of the plugin you will need to install the appropriate client library (i.e. libhsclient for C++ applications and perl-Net-HandlerSocket for perl applications).

1.1.4.3.2 HandlerSocket Configuration Options

Contents

The HandlerSocket plugin has the following options.

See also the Full list of MariaDB options, system and status variables.

Add the options to the [mysqld] section of your my.cnf file.

---

handlersocket_accept_balance

- **Description:** When set to a value other than zero (' `0` '), handlersocket will try to balance accepted connections among threads. Default is `0` but if you use persistent connections (for example if you use client-side connection pooling) then a non-zero value is recommended.
- **Commandline:** `--handlersocket-accept-balance="value"`
- **Scope:** Global
- **Dynamic:** No
- **Type:** number
- **Range:** `0` to `10000`
- **Default Value:** `0`

---

handlersocket_address

- **Description:** Specify the IP address to bind to.
- **Commandline:** `--handlersocket-address="value"`
- **Scope:** Global
- **Dynamic:** No
- **Type:** IP Address
- **Default Value:** Empty, previously `0.0.0.0`

---

handlersocket_backlog

- **Description:** Specify the listen backlog length.
- **Commandline:** `--handlersocket-backlog="value"`
- **Scope:** Global
- **Dynamic:** No
- **Type:** number
- **Range:** `5` to `1000000`
- **Default Value:** `32768`

---

handlersocket_epoll

- **Description:** Specify whether to use epoll for I/O multiplexing.
- **Commandline:** `--handlersocket-epoll="value"`
- **Scope:** Global
- **Dynamic:** No
- **Type:** number
- **Valid values:**
  - **Min:** `0`
  - **Max:** `1`
- **Default Value:** `1`

---

handlersocket_plain_secret

- **Description:** When set, enables plain-text authentication for the listener for read requests, with the value of the option specifying the secret authentication key.
- **Commandline:** `--handlersocket-plain-secret="value"`
- **Dynamic:** No
- **Type:** string
- **Default Value:** Empty

---

handlersocket_plain_secret_wr

- **Description:** When set, enables plain-text authentication for the listener for write requests, with the value of the option specifying the secret authentication key.
- **Commandline:** `--handlersocket-plain-secret-wr="value"`
- **Dynamic:** No
- **Type:** string
- **Default Value:** Empty

---

handlersocket_port

- **Description:** Specify the port to bind to for reads. An empty value disables the listener.
- **Commandline:** `--handlersocket-port="value"`
- **Scope:** Global
- **Dynamic:** No
- **Type:** number
- **Default Value:** Empty, previously `9998`

---

handlersocket_port_wr

- **Description:** Specify the port to bind to for writes. An empty value disables the listener.
- **Commandline:** `--handlersocket-port-wr="value"`
- **Scope:** Global
- **Dynamic:** No
- **Type:** number
- **Default Value:** Empty, previously `9999`

---

handlersocket_rcvbuf

- **Description:** Specify the maximum socket receive buffer (in bytes). If '0' then the system default is used.
- **Commandline:** `--handlersocket-rcvbuf="value"`
- **Scope:** Global
- **Dynamic:** No
- **Type:** number
- **Range:** `0` to `1677216`
- **Default Value:** `0`

---

handlersocket_readsize

- **Description:** Specify the minimum length of the request buffer. Larger values consume available memory but can make handlersocket faster for large requests.
- **Commandline:** `--handlersocket-readsize="value"`
- **Scope:** Global
- **Dynamic:** No
- **Type:** number
- **Range:** `0` to `1677216`
- **Default Value:** `0` (possibly `4096`)

---

handlersocket_sndbuf

- **Description:** Specify the maximum socket send buffer (in bytes). If '0' then the system default is used.
- **Commandline:** `--handlersocket-sndbuf="value"`
- **Scope:** Global
- **Dynamic:** No
- **Type:** number
- **Range:** `0` to `1677216`
- **Default Value:** `0`

---

handlersocket_threads

- **Description:** Specify the number of worker threads for reads. Recommended value = ((# CPU cores) * 2).
- **Commandline:** `--handlersocket-threads="value"`
- **Scope:** Global
- **Dynamic:** No
- **Type:** number
- **Range:** `1` to `3000`
- **Default Value:** `16`

---

handlersocket_threads_wr

- **Description:** Specify the number of worker threads for writes. Recommended value = 1.
- **Commandline:** `--handlersocket-threads-wr="value"`
- **Scope:** Global
- **Dynamic:** No
- **Type:** number
- **Range:** `1` to `3000`
- **Default Value:** `1`

---

handlersocket_timeout

- **Description:** Specify the socket timeout in seconds.
- **Commandline:** `--handlersocket-timeout="value"`
- **Scope:** Global
- **Dynamic:** No
- **Type:** number
- **Range:** `30` to `3600`
- **Default Value:** `300`

---

handlersocket_verbose

- **Description:** Specify the logging verbosity.
- **Commandline:** `--handlersocket-verbose="value"`
- **Scope:** Global
- **Dynamic:** No
- **Type:** number

- **Valid values:**
  - **Min:** `0`
  - **Max:** `10000`
- **Default Value:** `10`

`handlersocket_wrlock_timeout`

- **Description:** The write lock timeout in seconds. When acting on write requests, handlersocket locks an advisory lock named 'handlersocket_wr' and this option sets the timeout for it.
- **Commandline:** `--handlersocket-wrlock-timeout="value"`
- **Scope:** Global
- **Dynamic:** No
- **Type:** number
- **Range:** `0` to `3600`

1.1.4.3.3 HandlerSocket Client Libraries

In order to make use of the HandlerSocket plugin in your applications, you will need to use the appropriate client library. The following client libraries are available:

- C++
  - libhsclient *(included with the HandlerSocket plugin source)*
- Perl
  - perl-Net-HandlerSocket *(included with the HandlerSocket plugin source)*
- PHP
  - Net_HandlerSocket
  - HSPHP
  - php-ext-handlersocketi
- Java
  - hs4j
  - handlersocketforjava
- Python
  - python-handler-socket
  - pyhandlersocket
- Ruby
  - ruby-handlersocket
  - handlersocket
- JavaScript
  - node-handlersocket
- Scala
  - hs2client
- Haskell
  - HandlerSocket-Haskell-Client

1.1.4.3.4 Testing HandlerSocket in a Source Distribution

Contents

MariaDB 5.5

In MariaDB 5.5, which is built using `cmake`, `Makefile.PL` is not generated automatically. If you want to run the perl tests, you will need to create it manually from `Makefile.PL.in`. It is fairly easy to do by replacing `LIB` and `INC` values with the correct ones. Also, `libhsclient.so` is not built by default; `libhsclient.a` can be found in `plugin/handler_socket` folder.

MariaDB 5.3

If you want to test or use handlersocket with a source installation of MariaDB 5.3, here is one way to do this:

1. Compile with one of the build scripts that has the `-max` option, like `BUILD/compile-pentium64-max` or `BUILD/compile-pentium64-debug-max`
2. Start mysqld with the test framework

```
cd mysql-test
LD_LIBRARY_PATH=../plugin/handler_socket/libhsclient/.libs \
MTR_VERSION=1 perl mysql-test-run.pl --start-and-exit 1st \
--mysqld=--plugin-dir=../plugin/handler_socket/handlersocket/.libs \
--mysqld=--loose-handlersocket_port=9998 \
--mysqld=--loose-handlersocket_port_wr=9999 \
--master_port=9306 --mysqld=--innodb
```

3. This will end with:

```
Servers started, exiting
```

4. Load handlersocket

```
client/mysql -uroot --protocol=tcp --port=9306 \
-e 'INSTALL PLUGIN handlersocket soname "handlersocket.so"'
```

5. Configure and compile the handlersocket perl module

```
cd plugin/handler_socket/perl-Net-HandlerSocket
perl Makefile.PL
make
```

6. If you would like to install the handlersocket perl module permanently, you should do:

```
make install
```

   If you do this, you don't have to set `PERL5LIB` below.
7. Run the handlersocket test suite

```
cd plugin/handler_socket/regtest/test_01_lib
MYHOST=127.0.0.1 MYPORT=9306 LD_LIBRARY_PATH=../../libhsclient/.libs/ \
PERL5LIB=../common:../../perl-Net-HandlerSocket/lib:../../perl-Net-
HandlerSocket/blib/arch/auto/Net/HandlerSocket/ ./run.sh
```

1.1.4.3.5 HandlerSocket External Resources

Some resources and documentation about HandlerSocket.

- The home of HandlerSocket is here.
- The story of handlersocket can be found here.
- Comparison of HANDLER and HandlerSocket can be found here.
- HandlerSocket plugin for MySQL presentation by Akira Higuchi of DeNA - June 29 2010 - DeNA Technology Seminar
- HandlerSocket plugin for MySQL presentation by Akira Higuchi of DeNA - June 29 2011 - in Japanese

1.1.4.4 Dynamic Columns

Contents

Dynamic columns allow one to store different sets of columns for each row in a table. It works by storing a set of columns in a blob and having a small set of functions to manipulate it.

Dynamic columns should be used when it is not possible to use regular columns.

A typical use case is when one needs to store items that may have many different attributes (like size, color, weight, etc), and the set of possible attributes is very large and/or unknown in advance. In that case, attributes can be put into dynamic columns.

Dynamic Columns Basics

The table should have a blob column which will be used as storage for dynamic columns:

```
create table assets (
  item_name varchar(32) primary key, -- A common attribute for all items
  dynamic_cols  blob  -- Dynamic columns will be stored here
);
```

Once created, one can access dynamic columns via dynamic column functions:

Insert a row with two dynamic columns: color=blue, size=XL

```
INSERT INTO assets VALUES
  ('MariaDB T-shirt', COLUMN_CREATE('color', 'blue', 'size', 'XL'));
```

Insert another row with dynamic columns: color=black, price=500

```
INSERT INTO assets VALUES
  ('Thinkpad Laptop', COLUMN_CREATE('color', 'black', 'price', 500));
```

Select dynamic column 'color' for all items:

```
SELECT item_name, COLUMN_GET(dynamic_cols, 'color' as char)
  AS color FROM assets;
+-----------------+-------+
| item_name       | color |
+-----------------+-------+
| MariaDB T-shirt | blue  |
| Thinkpad Laptop | black |
+-----------------+-------+
```

It is possible to add and remove dynamic columns from a row:

```
-- Remove a column:
UPDATE assets SET dynamic_cols=COLUMN_DELETE(dynamic_cols, "price")
WHERE COLUMN_GET(dynamic_cols, 'color' as char)='black';

-- Add a column:
UPDATE assets SET dynamic_cols=COLUMN_ADD(dynamic_cols, 'warranty', '3 years')
WHERE item_name='Thinkpad Laptop';
```

You can also list all columns, or get them together with their values in JSON format:

```
SELECT item_name, column_list(dynamic_cols) FROM assets;
+-----------------+---------------------------+
| item_name       | column_list(dynamic_cols) |
+-----------------+---------------------------+
| MariaDB T-shirt | `size`,`color`            |
| Thinkpad Laptop | `color`,`warranty`        |
+-----------------+---------------------------+

SELECT item_name, COLUMN_JSON(dynamic_cols) FROM assets;
+-----------------+-----------------------------------------+
| item_name       | COLUMN_JSON(dynamic_cols)               |
+-----------------+-----------------------------------------+
| MariaDB T-shirt | {"size":"XL","color":"blue"}            |
| Thinkpad Laptop | {"color":"black","warranty":"3 years"} |
+-----------------+-----------------------------------------+
```

Dynamic Columns Reference

The rest of this page is a complete reference of dynamic columns in MariaDB

Dynamic Columns Functions

COLUMN_CREATE

```
COLUMN_CREATE(column_nr, value [as type], [column_nr, value
  [as type]]...);
COLUMN_CREATE(column_name, value [as type], [column_name, value
  [as type]]...);
```

Return a dynamic columns blob that stores the specified columns with values.

The return value is suitable for

- storing in a table
- further modification with other dynamic columns functions

The `as type` part allows one to specify the value type. In most cases, this is redundant because MariaDB will be able to deduce the type of the value. Explicit type specification may be needed when the type of the value is not apparent. For example, a literal `'2012-12-01'` has a CHAR type by default, one will need to specify `'2012-12-01' AS DATE` to have it stored as a date. See the Datatypes section for further details. Note also MDEV-597.

Typical usage:

```
-- MariaDB 5.3+:
INSERT INTO tbl SET dyncol_blob=COLUMN_CREATE(1 /*column id*/, "value");
-- MariaDB 10.0.1+:
INSERT INTO tbl SET dyncol_blob=COLUMN_CREATE("column_name", "value");
```

COLUMN_ADD

```
COLUMN_ADD(dyncol_blob, column_nr, value [as type],
  [column_nr, value [as type]]...);
COLUMN_ADD(dyncol_blob, column_name, value [as type],
  [column_name, value [as type]]...);
```

Adds or updates dynamic columns.

- `dyncol_blob` must be either a valid dynamic columns blob (for example, `COLUMN_CREATE` returns such blob), or an empty string.
- `column_name` specifies the name of the column to be added. If `dyncol_blob` already has a column with this name, it will be overwritten.
- `value` specifies the new value for the column. Passing a NULL value will cause the column to be deleted.
- `as type` is optional. See #datatypes section for a discussion about types.

The return value is a dynamic column blob after the modifications.

Typical usage:

```
-- MariaDB 5.3+:
UPDATE tbl SET dyncol_blob=COLUMN_ADD(dyncol_blob, 1 /*column id*/, "value")
  WHERE id=1;
-- MariaDB 10.0.1+:
UPDATE t1 SET dyncol_blob=COLUMN_ADD(dyncol_blob, "column_name", "value")
  WHERE id=1;
```

Note: `COLUMN_ADD()` is a regular function (just like `CONCAT()`), hence, in order to update the value in the table you have to use the `UPDATE ... SET dynamic_col=COLUMN_ADD(dynamic_col, ....)` pattern.

COLUMN_GET

```
COLUMN_GET(dyncol_blob, column_nr as type);
COLUMN_GET(dyncol_blob, column_name as type);
```

Get the value of a dynamic column by its name. If no column with the given name exists, `NULL` will be returned.

`column_name as type` requires that one specify the datatype of the dynamic column they are reading.

This may seem counter-intuitive: why would one need to specify which datatype they're retrieving? Can't the dynamic columns system figure the datatype from the data being stored?

The answer is: SQL is a statically-typed language. The SQL interpreter needs to know the datatypes of all expressions before the query is run (for example, when one is using prepared statements and runs `"select COLUMN_GET(...)"`, the prepared statement API requires the server to inform the client about the datatype of the column being read before the query is executed and the server can see what datatype the column actually has).

See the Datatypes section for more information about datatypes.

COLUMN_DELETE

```
COLUMN_DELETE(dyncol_blob, column_nr, column_nr...);
COLUMN_DELETE(dyncol_blob, column_name, column_name...);
```

Delete a dynamic column with the specified name. Multiple names can be given.

The return value is a dynamic column blob after the modification.

COLUMN_EXISTS

```
COLUMN_EXISTS(dyncol_blob, column_nr);
COLUMN_EXISTS(dyncol_blob, column_name);
```

Check if a column with name `column_name` exists in `dyncol_blob`. If yes, return `1`, otherwise return `0`.

COLUMN_LIST

```
COLUMN_LIST(dyncol_blob);
```

Return a comma-separated list of column names. The names are quoted with backticks.

```
SELECT column_list(column_create('col1','val1','col2','val2'));
+-----------------------------------------------------+
| column_list(column_create('col1','val1','col2','val2')) |
+-----------------------------------------------------+
| `col1`,`col2`                                       |
+-----------------------------------------------------+
```

COLUMN_CHECK

```
COLUMN_CHECK(dyncol_blob);
```

Check if `dyncol_blob` is a valid packed dynamic columns blob. Return value of 1 means the blob is valid, return value of 0 means it is not.

Rationale: Normally, one works with valid dynamic column blobs. Functions like `COLUMN_CREATE`, `COLUMN_ADD`, `COLUMN_DELETE` always return valid dynamic column blobs. However, if a dynamic column blob is accidentally truncated, or transcoded from one character set to another, it will be corrupted. This function can be used to check if a value in a blob field is a valid dynamic column blob.

Note: It is possible that a truncation cut a Dynamic Column "clearly" so that COLUMN_CHECK will not notice the corruption, but in any case of truncation a warning is issued during value storing.

COLUMN_JSON

```
COLUMN_JSON(dyncol_blob);
```

Return a JSON representation of data in `dyncol_blob`.

Example:

```
SELECT item_name, COLUMN_JSON(dynamic_cols) FROM assets;
+-----------------+-----------------------------------------+
| item_name       | COLUMN_JSON(dynamic_cols)               |
+-----------------+-----------------------------------------+
| MariaDB T-shirt | {"size":"XL","color":"blue"}            |
| Thinkpad Laptop | {"color":"black","warranty":"3 years"} |
+-----------------+-----------------------------------------+
```

Limitation: `COLUMN_JSON` will decode nested dynamic columns at a nesting level of not more than 10 levels deep. Dynamic columns that are nested deeper than 10 levels will be shown as BINARY string, without encoding.

Nesting Dynamic Columns

It is possible to use nested dynamic columns by putting one dynamic column blob inside another. The `COLUMN_JSON` function will display nested columns.

```
SET @tmp= column_create('parent_column',
  column_create('child_column', 12345));
Query OK, 0 rows affected (0.00 sec)

SELECT column_json(@tmp);
+-----------------------------------------+
| column_json(@tmp)                       |
+-----------------------------------------+
| {"parent_column":{"child_column":12345}} |
+-----------------------------------------+

SELECT column_get(column_get(@tmp, 'parent_column' AS char),
  'child_column' AS int);
+-------------------------------------------------------------------------------+
| column_get(column_get(@tmp, 'parent_column' as char), 'child_column' as int) |
+-------------------------------------------------------------------------------+
|                                                                         12345 |
+-------------------------------------------------------------------------------+
```

If you are trying to get a nested dynamic column as a string use 'as BINARY' as the last argument of COLUMN_GET (otherwise problems with character set conversion and illegal symbols are possible):

```
select column_json( column_get(
  column_create('test1',
    column_create('key1','value1','key2','value2','key3','value3')),
  'test1' as BINARY));
```

Datatypes

In SQL, one needs to define the type of each column in a table. Dynamic columns do not provide any way to declare a type in advance ("whenever there is a column 'weight', it should be integer" is not possible). However, each particular dynamic column value is stored together with its datatype.

The set of possible datatypes is mostly the same as that used by the SQL `CAST` and `CONVERT` functions. However, note that there are currently some differences - see MDEV-597.

type	dynamic column internal type	description
`BINARY[(N)]`	`DYN_COL_STRING`	(variable length string with binary charset)
`CHAR[(N)]`	`DYN_COL_STRING`	(variable length string with charset)
`DATE`	`DYN_COL_DATE`	(date - 3 bytes)
`DATETIME[(D)]`	`DYN_COL_DATETIME`	(date and time (with microseconds) - 9 bytes)
`DECIMAL[(M[,D])]`	`DYN_COL_DECIMAL`	(variable length binary decimal representation with MariaDB limitation)
`DOUBLE[(M,D)]`	`DYN_COL_DOUBLE`	(64 bit double-precision floating point)
`INTEGER`	`DYN_COL_INT`	(variable length, up to 64 bit signed integer)
`SIGNED [INTEGER]`	`DYN_COL_INT`	(variable length, up to 64 bit signed integer)
`TIME[(D)]`	`DYN_COL_TIME`	(time (with microseconds, may be negative) - 6 bytes)
`UNSIGNED [INTEGER]`	`DYN_COL_UINT`	(variable length, up to 64bit unsigned integer)

A Note About Lengths

If you're running queries like

```
SELECT COLUMN_GET(blob, 'colname' as CHAR) ...
```

without specifying a maximum length (i.e. using #as CHAR#, not `as CHAR(n)`), MariaDB will report the maximum length of the resultset column to be `53,6870,911` (bytes or characters?) for MariaDB 5.3-10.0.0 and `16,777,216` for MariaDB 10.0.1+. This may cause excessive memory usage in some client libraries, because they try to pre-allocate a buffer of maximum resultset width. If you suspect you're hitting this problem, use `CHAR(n)` whenever you're using `COLUMN_GET` in the select list.

MariaDB 5.3 vs MariaDB 10.0

The dynamic columns feature was introduced into MariaDB in two steps:

1. MariaDB 5.3 was the first version to support dynamic columns. Only numbers could be used as column names in this version.
2. In MariaDB 10.0.1, column names can be either numbers or strings. Also, the `COLUMN_JSON` and `COLUMN_CHECK` functions were added.

See also Dynamic Columns in MariaDB 10.

Client-side API

It is also possible to create or parse dynamic columns blobs on the client side. `libmysql` client library now includes an API for writing/reading dynamic column blobs. See dynamic-columns-api for details.

Limitations

Description	Limit
Max number of columns	65535
Max total length of packed dynamic column	max_allowed_packet (1G)

1.1.4.5 Dynamic Columns from MariaDB 10

Contents

MariaDB starting with 10.0.1

MariaDB 10.0.1 introduced the following improvements to the dynamic columns feature.

Column Name Support

It is possible to refer to column by names. Names can be used everywhere where in MariaDB 5.3 one could use only strings:

- Create a dynamic column blob:

```
COLUMN_CREATE('int_col', 123 as int, 'double_col', 3.14 as double, 'string_col', 'text-
data' as char);
```

- Set a column value:

```
COLUMN_ADD(dyncol_blob, 'intcol', 1234);
```

- Get a column value:

```
COLUMN_GET(dynstr, 'column1' as char(10));
```

- Check whether a column exists

```
COLUMN_EXISTS(dyncol_blob, 'column_name');
```

Changes in Behavior

- Column list output now includes quoting:

```
select column_list(column_create(1, 22, 2, 23));
+----------------------------------------+
| column_list(column_create(1, 22, 2, 23)) |
+----------------------------------------+
| `1`,`2`                                |
+----------------------------------------+
select column_list(column_create('column1', 22, 'column2', 23));
+-------------------------------------------------------+
| column_list(column_create('column1', 22, 'column2', 23)) |
+-------------------------------------------------------+
| `column1`,`column2`                                   |
+-------------------------------------------------------+
```

- Column name interpretation has been changed so that the string now is not converted to a number. So some "magic" tricks will not work any more, for example, "1test" and "1" now become different column names:

```
select column_list(column_add(column_create('1a', 22), '1b', 23));
+-----------------------------------------------------------+
| column_list(column_add(column_create('1a', 22), '1b', 23)) |
+-----------------------------------------------------------+
| `1a`,`1b`                                                 |
+-----------------------------------------------------------+
```

Old behavior:

```
select column_list(column_add(column_create('1a', 22), '1b', 23));
+-----------------------------------------------------------+
| column_list(column_add(column_create('1a', 22), '1b', 23)) |
+-----------------------------------------------------------+
| 1                                                         |
+-----------------------------------------------------------+
```

New Functions

The following new functions have been added to dynamic columns in MariaDB 10

COLUMN_CHECK

COLUMN_CHECK is used to check a column's integrity. When it encounters an error it does not return illegal format errors but returns false instead. It also checks integrity more thoroughly and finds errors in the dynamic column internal structures which might not be found by other functions.

```
select column_check(column_create('column1', 22));
+-----------------------------------------+
| column_check(column_create('column1', 22)) |
+-----------------------------------------+
|                                       1 |
+-----------------------------------------+
select column_check('abracadabra');
+----------------------------+
| column_check('abracadabra') |
+----------------------------+
|                          0 |
+----------------------------+
```

COLUMN_JSON

COLUMN_JSON converts all dynamic column record content to a JSON object.

```
select column_json(column_create('column1', 1, 'column2', "two"));
+-----------------------------------------------------------+
| column_json(column_create('column1', 1, 'column2', "two")) |
+-----------------------------------------------------------+
| {"column1":1,"column2":"two"}                             |
+-----------------------------------------------------------+
```

Other Changes

- All API functions has prefix mariadb_dyncol_ (old prefix dynamic_column_ is depricated
- API changed to be able to work with the new format (*_named functions).
- Removed 'delete' function because deleting could be done by adding NULL value.
- 'Time' and 'datetime' in the new format are stored without microseconds if they are 0.
- New function added to API (except that two which are representing SQL level functions):
  - 'Unpack' the dynamic columns content to an arrays of values and names.
  - 3 functions to get any column value as string, integer (long long) or floating point (double).
- New type of "dynamic column" row added on the API level (in SQL level output it is a string but if you use dynamic column functions to construct object it will be added as dynamic column value) which allow to add dynamic columns inside dynamic columns. JSON function represent such recursive constructions correctly but limit depth of representation as current implementation limit (internally depth of dynamic columns embedding is not limited).

Interface with Cassandra

> CassandraSE is no longer actively being developed and has been removed in MariaDB 10.6. See MDEV-23024.

Some internal changes were added to dynamic columns to allow them to serve as an interface to Apache Cassandra dynamic columns. The Cassandra engine may pack all columns which were not mentioned in the MariaDB interface table definition and even bring changes in the dynamic column contents back to the cassandra columns family (the table analog in cassandra).

1.1.4.6 Dynamic Column API

This page describes the client-side API for reading and writing Dynamic Columns blobs.

Normally, you should use Dynamic column functions which are run inside the MariaDB server and allow one to access Dynamic Columns content without any client-side libraries.

If you need to read/write dynamic column blobs **on the client** for some reason, this API enables that.

Contents

Where to get it

The API is a part of `libmysql` C client library. In order to use it, you need to include this header file

```
#include <mysql/ma_dyncol.h>
```

and link against `libmysql` .

Data structures

DYNAMIC_COLUMN

`DYNAMIC_COLUMN` represents a packed dynamic column blob. It is essentially a string-with-length and is defined as follows:

```
/* A generic-purpose arbitrary-length string defined in MySQL Client API */
typedef struct st_dynamic_string
{
  char *str;
  size_t length,max_length,alloc_increment;
} DYNAMIC_STRING;

...

typedef DYNAMIC_STRING DYNAMIC_COLUMN;
```

DYNAMIC_COLUMN_VALUE

Dynamic columns blob stores {name, value} pairs. `DYNAMIC_COLUMN_VALUE` structure is used to represent the value in accessible form.

```
struct st_dynamic_column_value
{
  DYNAMIC_COLUMN_TYPE type;
  union
  {
    long long long_value;
    unsigned long long ulong_value;
    double double_value;
    struct {
      MYSQL_LEX_STRING value;
      CHARSET_INFO *charset;
    } string;
    struct {
      decimal_digit_t buffer[DECIMAL_BUFF_LENGTH];
      decimal_t value;
    } decimal;
    MYSQL_TIME time_value;
  } x;
};
typedef struct st_dynamic_column_value DYNAMIC_COLUMN_VALUE;
```

Every value has a type, which is determined by the `type` member.

type	structure field
DYN_COL_NULL	-
DYN_COL_INT	value.x.long_value
DYN_COL_UINT	value.x.ulong_value
DYN_COL_DOUBLE	value.x.double_value
DYN_COL_STRING	value.x.string.value , value.x.string.charset
DYN_COL_DECIMAL	value.x.decimal.value
DYN_COL_DATETIME	value.x.time_value
DYN_COL_DATE	value.x.time_value
DYN_COL_TIME	value.x.time_value
DYN_COL_DYNCOL	value.x.string.value

Notes

- Values with type `DYN_COL_NULL` do not ever occur in dynamic columns blobs.
- Type `DYN_COL_DYNCOL` means that the value is a packed dynamic blob. This is how nested dynamic columns are done.
- Before storing a value to `value.x.decimal.value` , one must call `mariadb_dyncol_prepare_decimal()` to initialize the space for storage.

enum_dyncol_func_result

`enum enum_dyncol_func_result` is used as return value.

value	name	meaning
0	ER_DYNCOL_OK	OK
0	ER_DYNCOL_NO	(the same as ER_DYNCOL_OK but for functions which return a YES/NO)
1	ER_DYNCOL_YES	YES response or success
2	ER_DYNCOL_TRUNCATED	Operation succeeded but the data was truncated
-1	ER_DYNCOL_FORMAT	Wrong format of the encoded string
-2	ER_DYNCOL_LIMIT	A limit of implementation reached
-3	ER_DYNCOL_RESOURCE	Out of resources
-4	ER_DYNCOL_DATA	Incorrect input data
-5	ER_DYNCOL_UNKNOWN_CHARSET	Unknown character set

Result codes that are less than zero represent error conditions.

Function reference

Functions come in pairs:

- `xxx()` operates on the old (pre-MariaDB-10.0.1) dynamic column blob format where columns were identified by numbers.
- `xxx_named()` can operate on both old or new data format. If it modifies the blob, it will convert it to the new data format.

You should use `xxx_named()` functions, unless you need to keep the data compatible with MariaDB versions before 10.0.1.

mariadb_dyncol_create_many

Create a packed dynamic blob from arrays of values and names.

```
enum enum_dyncol_func_result
mariadb_dyncol_create_many(DYNAMIC_COLUMN *str,
                           uint column_count,
                           uint *column_numbers,
                           DYNAMIC_COLUMN_VALUE *values,
                           my_bool new_string);
enum enum_dyncol_func_result
mariadb_dyncol_create_many_named(DYNAMIC_COLUMN *str,
                                 uint column_count,
                                 MYSQL_LEX_STRING *column_keys,
                                 DYNAMIC_COLUMN_VALUE *values,
                                 my_bool new_string);
```

where

`str`	`OUT`	Packed dynamic blob will be put here
`column_count`	`IN`	Number of columns
`column_numbers`	`IN`	Column numbers array (old format)
`column_keys`	`IN`	Column names array (new format)
`values`	`IN`	Column values array
`new_string`	`IN`	If TRUE then the `str` will be reinitialized (not freed) before usage

mariadb_dyncol_update_many

Add or update columns in a dynamic columns blob. To delete a column, update its value to a "non-value" of type `DYN_COL_NULL`

```
enum enum_dyncol_func_result
mariadb_dyncol_update_many(DYNAMIC_COLUMN *str,
                           uint column_count,
                           uint *column_numbers,
                           DYNAMIC_COLUMN_VALUE *values);
enum enum_dyncol_func_result
mariadb_dyncol_update_many_named(DYNAMIC_COLUMN *str,
                                 uint column_count,
                                 MYSQL_LEX_STRING *column_keys,
                                 DYNAMIC_COLUMN_VALUE *values);
```

str	IN/OUT	Dynamic columns blob to be modified.
column_count	IN	Number of columns in following arrays
column_numbers	IN	Column numbers array (old format)
column_keys	IN	Column names array (new format)
values	IN	Column values array

mariadb_dyncol_exists

Check if column with given name exists in the blob

```
enum enum_dyncol_func_result
mariadb_dyncol_exists(DYNAMIC_COLUMN *str, uint column_number);
enum enum_dyncol_func_result
mariadb_dyncol_exists_named(DYNAMIC_COLUMN *str, MYSQL_LEX_STRING *column_key);
```

str	IN	Packed dynamic columns string.
column_number	IN	Column number (old format)
column_key	IN	Column name (new format)

The function returns YES/NO or Error code

mariadb_dyncol_column_count

Get number of columns in a dynamic column blob

```
enum enum_dyncol_func_result
mariadb_dyncol_column_count(DYNAMIC_COLUMN *str, uint *column_count);
```

str	IN	Packed dynamic columns string.
column_count	OUT	Number of not NULL columns in the dynamic columns string

mariadb_dyncol_list

List columns in a dynamic column blob.

```
enum enum_dyncol_func_result
mariadb_dyncol_list(DYNAMIC_COLUMN *str, uint *column_count, uint **column_numbers);
enum enum_dyncol_func_result
mariadb_dyncol_list_named(DYNAMIC_COLUMN *str, uint *column_count,
                          MYSQL_LEX_STRING **column_keys);
```

str	IN	Packed dynamic columns string.
column_count	OUT	Number of columns in following arrays
column_numbers	OUT	Column numbers array (old format). Caller should free this array.
column_keys	OUT	Column names array (new format). Caller should free this array.

mariadb_dyncol_get

Get a value of one column

```
enum enum_dyncol_func_result
mariadb_dyncol_get(DYNAMIC_COLUMN *org, uint column_number,
                   DYNAMIC_COLUMN_VALUE *value);
enum enum_dyncol_func_result
mariadb_dyncol_get_named(DYNAMIC_COLUMN *str, MYSQL_LEX_STRING *column_key,
                         DYNAMIC_COLUMN_VALUE *value);
```

str	IN	Packed dynamic columns string.
column_number	IN	Column numbers array (old format)
column_key	IN	Column names array (new format)
value	OUT	Value of the column

If the column is not found NULL returned as a value of the column.

mariadb_dyncol_unpack

Get value of all columns

```
enum enum_dyncol_func_result
mariadb_dyncol_unpack(DYNAMIC_COLUMN *str,
                      uint *column_count,
                      MYSQL_LEX_STRING **column_keys,
                      DYNAMIC_COLUMN_VALUE **values);
```

str	IN	Packed dynamic columns string to unpack.
column_count	OUT	Number of columns in following arrays
column_keys	OUT	Column names array (should be free by caller)
values	OUT	Values of the columns array (should be free by caller)

mariadb_dyncol_has_names

Check whether the dynamic columns blob uses new data format (the one where columns are identified by names)

```
my_bool mariadb_dyncol_has_names(DYNAMIC_COLUMN *str);
```

str	IN	Packed dynamic columns string.

mariadb_dyncol_check

Check whether dynamic column blob has correct data format.

```
enum enum_dyncol_func_result
mariadb_dyncol_check(DYNAMIC_COLUMN *str);
```

str	IN	Packed dynamic columns string.

mariadb_dyncol_json

Get contents od a dynamic columns blob in a JSON form

```
enum enum_dyncol_func_result
mariadb_dyncol_json(DYNAMIC_COLUMN *str, DYNAMIC_STRING *json);
```

str	IN	Packed dynamic columns string.

`json` `OUT` JSON representation

mariadb_dyncol_val_TYPE

Get dynamic column value as one of the base types

```
enum enum_dyncol_func_result
mariadb_dyncol_val_str(DYNAMIC_STRING *str, DYNAMIC_COLUMN_VALUE *val,
                       CHARSET_INFO *cs, my_bool quote);
enum enum_dyncol_func_result
mariadb_dyncol_val_long(longlong *ll, DYNAMIC_COLUMN_VALUE *val);
enum enum_dyncol_func_result
mariadb_dyncol_val_double(double *dbl, DYNAMIC_COLUMN_VALUE *val);
```

`str` or `ll` or `dbl` `OUT` value of the column

`val` `IN` Value

mariadb_dyncol_prepare_decimal

Initialize `DYNAMIC_COLUMN_VALUE` before value of `value.x.decimal.value` can be set

```
void mariadb_dyncol_prepare_decimal(DYNAMIC_COLUMN_VALUE *value);
```

`value` `OUT` Value of the column

This function links `value.x.decimal.value` to `value.x.decimal.buffer` .

mariadb_dyncol_value_init

Initialize a `DYNAMIC_COLUMN_VALUE` structure to a safe default.

```
#define mariadb_dyncol_value_init(V) (V)->type= DYN_COL_NULL
```

mariadb_dyncol_column_cmp_named

Compare two column names (currently, column names are compared with memcmp())

```
int mariadb_dyncol_column_cmp_named(const MYSQL_LEX_STRING *s1,
                                    const MYSQL_LEX_STRING *s2);
```

1.1.4.7 Dynamic Columns from MariaDB 10

Contents

MariaDB starting with 10.0.1

MariaDB 10.0.1 introduced the following improvements to the dynamic columns feature.

Column Name Support

It is possible to refer to column by names. Names can be used everywhere where in MariaDB 5.3 one could use only strings:

- Create a dynamic column blob:

```
COLUMN_CREATE('int_col', 123 as int, 'double_col', 3.14 as double, 'string_col', 'text-
data' as char);
```

- Set a column value:

```
COLUMN_ADD(dyncol_blob, 'intcol', 1234);
```

- Get a column value:

```
COLUMN_GET(dynstr, 'column1' as char(10));
```

- Check whether a column exists

```
COLUMN_EXISTS(dyncol_blob, 'column_name');
```

Changes in Behavior

- Column list output now includes quoting:

```
select column_list(column_create(1, 22, 2, 23));
+----------------------------------------+
| column_list(column_create(1, 22, 2, 23)) |
+----------------------------------------+
| `1`,`2`                                |
+----------------------------------------+
select column_list(column_create('column1', 22, 'column2', 23));
+-------------------------------------------------------+
| column_list(column_create('column1', 22, 'column2', 23)) |
+-------------------------------------------------------+
| `column1`,`column2`                                   |
+-------------------------------------------------------+
```

- Column name interpretation has been changed so that the string now is not converted to a number. So some "magic" tricks will not work any more, for example, "1test" and "1" now become different column names:

```
select column_list(column_add(column_create('1a', 22), '1b', 23));
+-----------------------------------------------------------+
| column_list(column_add(column_create('1a', 22), '1b', 23)) |
+-----------------------------------------------------------+
| `1a`,`1b`                                                 |
+-----------------------------------------------------------+
```

  Old behavior:

```
select column_list(column_add(column_create('1a', 22), '1b', 23));
+-----------------------------------------------------------+
| column_list(column_add(column_create('1a', 22), '1b', 23)) |
+-----------------------------------------------------------+
| 1                                                         |
+-----------------------------------------------------------+
```

New Functions

The following new functions have been added to dynamic columns in MariaDB 10

COLUMN_CHECK

COLUMN_CHECK is used to check a column's integrity. When it encounters an error it does not return illegal format errors but returns false instead. It also checks integrity more thoroughly and finds errors in the dynamic column internal structures which might not be found by other functions.

```
select column_check(column_create('column1', 22));
+-----------------------------------------+
| column_check(column_create('column1', 22)) |
+-----------------------------------------+
|                                       1 |
+-----------------------------------------+
select column_check('abracadabra');
+----------------------------+
| column_check('abracadabra') |
+----------------------------+
|                          0 |
+----------------------------+
```

COLUMN_JSON

COLUMN_JSON converts all dynamic column record content to a JSON object.

```
select column_json(column_create('column1', 1, 'column2', "two"));
+-------------------------------------------------------------+
| column_json(column_create('column1', 1, 'column2', "two")) |
+-------------------------------------------------------------+
| {"column1":1,"column2":"two"}                               |
+-------------------------------------------------------------+
```

Other Changes

- All API functions has prefix mariadb_dyncol_ (old prefix dynamic_column_ is depricated
- API changed to be able to work with the new format (*_named functions).
- Removed 'delete' function because deleting could be done by adding NULL value.
- 'Time' and 'datetime' in the new format are stored without microseconds if they are 0.
- New function added to API (except that two which are representing SQL level functions):
  - 'Unpack' the dynamic columns content to an arrays of values and names.
  - 3 functions to get any column value as string, integer (long long) or floating point (double).
- New type of "dynamic column" row added on the API level (in SQL level output it is a string but if you use dynamic column functions to construct object it will be added as dynamic column value) which allow to add dynamic columns inside dynamic columns. JSON function represent such recursive constructions correctly but limit depth of representation as current implementation limit (internally depth of dynamic columns embedding is not limited).

Interface with Cassandra

> CassandraSE is no longer actively being developed and has been removed in MariaDB 10.6. See MDEV-23024.

Some internal changes were added to dynamic columns to allow them to serve as an interface to Apache Cassandra dynamic columns. The Cassandra engine may pack all columns which were not mentioned in the MariaDB interface table definition and even bring changes in the dynamic column contents back to the cassandra columns family (the table analog in cassandra).

1.2.9.4 JSON Functions

1.1.1.4.2.4.4 LOAD_FILE

1.1.5 Operators

Operators can be used for comparing values or for assigning values. There are several operators and they may be used in different SQL statements and clauses. Some can be used somewhat on their own, not within an SQL statement clause.

For comparing values—string or numeric—you can use symbols such as the equal-sign (i.e., `=`) or the exclamation point and the equal-sign together (i.e., `!=`). You might use these in `WHERE` clauses or within a flow-control statement or function (e.g., IF( )). You can also use basic regular expressions with the `LIKE` operator.

For assigning values, you can also use the equal-sign or other arithmetic symbols (e.g. plus-sign). You might do this with the SET statement or in a `SET` clause in an UPDATE statement.

Arithmetic Operators

Addition Operator (+)

Addition.

DIV

Integer division.

Division Operator (/)

Division.

MOD

Modulo operation. Remainder of N divided by M.

Modulo Operator (%)

Modulo operator. Returns the remainder of N divided by M.

Multiplication Operator (*)

Multiplication.

Subtraction Operator (-)

Subtraction and unary minus.

Assignment Operators

Assignment Operator (:=)

Assignment operator for assigning a value.

Assignment Operator (=)

The equal sign as an assignment operator.

Bit Functions and Operators

Operator Precedence

Precedence of SQL operators

&

Bitwise AND

<<

Left shift

>>

Shift right

BIT_COUNT

Returns the number of set bits

^

Bitwise XOR

|

Bitwise OR

~

Bitwise NOT

Parentheses

Parentheses modify the precedence of other operators in an expression

TRUE FALSE

TRUE and FALSE evaluate to 1 and 0

Comparison Operators

!=

Not equal operator.

<

Less than operator.

<=

Less than or equal operator.

<=>

NULL-safe equal operator.

=

Equal operator.

>

Greater than operator.

>=

Greater than or equal operator.

BETWEEN AND

True if expression between two values.

COALESCE

Returns the first non-NULL parameter

GREATEST

Returns the largest argument.

IN

True if expression equals any of the values in the list.

INTERVAL

Index of the argument that is less than the first argument

IS

Tests whether a boolean is TRUE, FALSE, or UNKNOWN.

IS NOT

Tests whether a boolean value is not TRUE, FALSE, or UNKNOWN

IS NOT NULL

Tests whether a value is not NULL

IS NULL

Tests whether a value is NULL

ISNULL

Checks if an expression is NULL

LEAST

Returns the smallest argument.

NOT BETWEEN

Same as NOT (expr BETWEEN min AND max)

NOT IN

Same as NOT (expr IN (value,...))

Logical Operators

!

Logical NOT.

&&

Logical AND.

XOR

Logical XOR.

||

Logical OR.

Other Operators Articles

Operator Precedence

Precedence of SQL operators

There are 3 related questions.

1.1.5.1 Arithmetic Operators

Arithmetic operators for addition, subtraction, multiplication, division and the modulo operator

Addition Operator (+)

Addition.

DIV

Integer division.

Division Operator (/)

Division.

MOD

Modulo operation. Remainder of N divided by M.

Modulo Operator (%)

Modulo operator. Returns the remainder of N divided by M.

Multiplication Operator (*)

Multiplication.

Subtraction Operator (-)

Subtraction and unary minus.

1.1.5.1.1 Addition Operator (+)

Syntax

```
+
```

Contents

Description

Addition.

If both operands are integers, the result is calculated with BIGINT precision. If either integer is unsigned, the result is also an unsigned integer.

For real or string operands, the operand with the highest precision determines the result precision.

Examples

```
SELECT 3+5;
+-----+
| 3+5 |
+-----+
|   8 |
+-----+
```

1.2.5.6 DIV

1.1.5.1.3 Division Operator (/)

Syntax

```
/
```

Contents

Description

Division operator. Dividing by zero will return NULL. By default, returns four digits after the decimal. This is determined by the server system variable div_precision_increment which by default is four. It can be set from 0 to 30.

Dividing by zero returns `NULL`. If the `ERROR_ON_DIVISION_BY_ZERO` SQL_MODE is used (the default since MariaDB 10.2.4), a division by zero also produces a warning.

Examples

```
SELECT 4/5;
+--------+
| 4/5    |
+--------+
| 0.8000 |
+--------+

SELECT 300/(2-2);
+-----------+
| 300/(2-2) |
+-----------+
|      NULL |
+-----------+

SELECT 300/7;
+---------+
| 300/7   |
+---------+
| 42.8571 |
+---------+
```

Changing div_precision_increment for the session from the default of four to six:

```
SET div_precision_increment = 6;

SELECT 300/7;
+-----------+
| 300/7     |
+-----------+
| 42.857143 |
+-----------+

SELECT 300/7;
+-----------+
| 300/7     |
+-----------+
| 42.857143 |
+-----------+
```

1.2.5.28 MOD

1.1.5.1.5 Modulo Operator (%)

Syntax

```
N % M
```

Description

Modulo operator. Returns the remainder of `N` divided by `M`. See also MOD.

Examples

```
SELECT 1042 % 50;
+-----------+
| 1042 % 50 |
+-----------+
|        42 |
+-----------+
```

1.1.5.1.6 Multiplication Operator (*)

Syntax

```
*
```

Contents

Description

Multiplication operator.

Examples

```
SELECT 7*6;
+-----+
| 7*6 |
+-----+
|  42 |
+-----+

SELECT 1234567890*9876543210;
+----------------------+
| 1234567890*9876543210 |
+----------------------+
|  -6253480962446024716 |
+----------------------+

SELECT 18014398509481984*18014398509481984.0;
+---------------------------------------+
| 18014398509481984*18014398509481984.0 |
+---------------------------------------+
|   324518553658426726783156020576256.0 |
+---------------------------------------+

SELECT 18014398509481984*18014398509481984;
+-------------------------------------+
| 18014398509481984*18014398509481984 |
+-------------------------------------+
|                                   0 |
+-------------------------------------+
```

1.1.5.1.7 Subtraction Operator (-)

Syntax

```
-
```

Description

Subtraction. The operator is also used as the unary minus for changing sign.

If both operands are integers, the result is calculated with BIGINT precision. If either integer is unsigned, the result is also an unsigned integer, unless the NO_UNSIGNED_SUBTRACTION SQL_MODE is enabled, in which case the result is always signed.

For real or string operands, the operand with the highest precision determines the result precision.

Examples

```
SELECT 96-9;
+------+
| 96-9 |
+------+
|   87 |
+------+

SELECT 15-17;
+-------+
| 15-17 |
+-------+
|    -2 |
+-------+

SELECT 3.66 + 1.333;
+--------------+
| 3.66 + 1.333 |
+--------------+
|        4.993 |
+--------------+
```

Unary minus:

```
 SELECT - (3+5);
+---------+
| - (3+5) |
+---------+
|      -8 |
+---------+
```

1.1.5.2 Assignment Operators

Operators for assigning a value

Assignment Operator (:=)

Assignment operator for assigning a value.

Assignment Operator (=)

The equal sign as an assignment operator.

1.1.5.2.1 Assignment Operator (:=)

Syntax

```
var_name := expr
```

Description

Assignment operator for assigning a value. The value on the right is assigned to the variable on left.

Unlike the = operator, `:=` can always be used to assign a value to a variable.

This operator works with both user-defined variables and local variables.

When assigning the same value to several variables, LAST_VALUE() can be useful.

Examples

```
 SELECT @x := 10;
+----------+
| @x := 10 |
+----------+
|       10 |
+----------+

SELECT @x, @y := @x;
+------+----------+
| @x   | @y := @x |
+------+----------+
|   10 |       10 |
+------+----------+
```

1.1.5.2.2 Assignment Operator (=)

Syntax

```
identifier = expr
```

Contents

Description

The equal sign is used as both an assignment operator in certain contexts, and as a comparison operator. When used as assignment operator, the value on the right is assigned to the variable (or column, in some contexts) on the left.

Since its use can be ambiguous, unlike the := assignment operator, the = assignment operator cannot be used in all contexts, and is only valid as part of a SET statement, or the SET clause of an UPDATE statement

This operator works with both user-defined variables and local variables.

Examples

```
UPDATE table_name SET x = 2 WHERE x > 100;
```

```
SET @x = 1, @y := 2;
```

1.2.8.1 Bit Functions and Operators

1.1.5.4 Comparison Operators

The comparison operators include: !=, <, <=, <=>, >=, >, etc...

!=

Not equal operator.

<

Less than operator.

<=

Less than or equal operator.

<=>

NULL-safe equal operator.

=

Equal operator.

>

Greater than operator.

>=

Greater than or equal operator.

BETWEEN AND

True if expression between two values.

COALESCE

Returns the first non-NULL parameter

GREATEST

Returns the largest argument.

IN

True if expression equals any of the values in the list.

INTERVAL

Index of the argument that is less than the first argument

IS

Tests whether a boolean is TRUE, FALSE, or UNKNOWN.

IS NOT

Tests whether a boolean value is not TRUE, FALSE, or UNKNOWN

IS NOT NULL

Tests whether a value is not NULL

IS NULL

Tests whether a value is NULL

ISNULL

Checks if an expression is NULL

LEAST

Returns the smallest argument.

NOT BETWEEN

Same as NOT (expr BETWEEN min AND max)

NOT IN

Same as NOT (expr IN (value,...))

1.1.5.4.1 Not Equal Operator: !=

Syntax

```
<>, !=
```

Contents

Description

Not equal operator. Evaluates both SQL expressions and returns 1 if they are not equal and 0 if they are equal, or `NULL` if either expression is NULL. If the expressions return different data types, (for instance, a number and a string), performs type conversion.

When used in row comparisons these two queries return the same results:

```
SELECT (t1.a, t1.b) != (t2.x, t2.y)
FROM t1 INNER JOIN t2;

SELECT (t1.a != t2.x) OR (t1.b != t2.y)
FROM t1 INNER JOIN t2;
```

Examples

```
SELECT '.01' <> '0.01';
+-----------------+
| '.01' <> '0.01' |
+-----------------+
|               1 |
+-----------------+

SELECT .01 <> '0.01';
+---------------+
| .01 <> '0.01' |
+---------------+
|             0 |
+---------------+

SELECT 'zapp' <> 'zappp';
+-------------------+
| 'zapp' <> 'zappp' |
+-------------------+
|                 1 |
+-------------------+
```

1.1.5.4.2 <

Syntax

```
<
```

Contents

Description

Less than operator. Evaluates both SQL expressions and returns 1 if the left value is less than the right value and 0 if it is not, or `NULL` if either expression is NULL. If the expressions return different data types, (for instance, a number and a string), performs type conversion.

When used in row comparisons these two queries return the same results:

```
SELECT (t1.a, t1.b) < (t2.x, t2.y)
FROM t1 INNER JOIN t2;

SELECT (t1.a < t2.x) OR ((t1.a = t2.x) AND (t1.b < t2.y))
FROM t1 INNER JOIN t2;
```

Examples

```
SELECT 2 < 2;
+-------+
| 2 < 2 |
+-------+
|     0 |
+-------+
```

Type conversion:

```
SELECT 3<'4';
+-------+
| 3<'4' |
+-------+
|     1 |
+-------+
```

Case insensitivity - see Character Sets and Collations:

```
SELECT 'a'<'A';
+---------+
| 'a'<'A' |
+---------+
|       0 |
+---------+
```

1.1.5.4.3 <=

Syntax

```
<=
```

Contents

Description

Less than or equal operator. Evaluates both SQL expressions and returns 1 if the left value is less than or equal to the right value and 0 if it is not, or `NULL` if either expression is NULL. If the expressions return different data types, (for instance, a number and a string), performs type conversion.

When used in row comparisons these two queries return the same results:

```
SELECT (t1.a, t1.b) <= (t2.x, t2.y)
FROM t1 INNER JOIN t2;

SELECT (t1.a < t2.x) OR ((t1.a = t2.x) AND (t1.b <= t2.y))
FROM t1 INNER JOIN t2;
```

Examples

```
SELECT 0.1 <= 2;
+----------+
| 0.1 <= 2 |
+----------+
|        1 |
+----------+
```

```
SELECT 'a'<='A';
+----------+
| 'a'<='A' |
+----------+
|        1 |
+----------+
```

1.1.5.4.4 <=>

Syntax

```
<=>
```

Description

NULL-safe equal operator. It performs an equality comparison like the = operator, but returns 1 rather than NULL if both operands are NULL, and 0 rather than NULL if one operand is NULL.

`a <=> b` is equivalent to `a = b OR (a IS NULL AND b IS NULL)`.

When used in row comparisons these two queries return the same results:

```
SELECT (t1.a, t1.b) <=> (t2.x, t2.y)
FROM t1 INNER JOIN t2;

SELECT (t1.a <=> t2.x) AND (t1.b <=> t2.y)
FROM t1 INNER JOIN t2;
```

See also NULL Values in MariaDB.

Examples

```
SELECT 1 <=> 1, NULL <=> NULL, 1 <=> NULL;
+---------+---------------+-------------+
| 1 <=> 1 | NULL <=> NULL | 1 <=> NULL |
+---------+---------------+-------------+
|       1 |             1 |           0 |
+---------+---------------+-------------+

SELECT 1 = 1, NULL = NULL, 1 = NULL;
+-------+-------------+----------+
| 1 = 1 | NULL = NULL | 1 = NULL |
+-------+-------------+----------+
|     1 |        NULL |     NULL |
+-------+-------------+----------+
```

1.1.5.4.5 =

Syntax

```
left_expr = right_expr
```

Contents

Description

Equal operator. Evaluates both SQL expressions and returns 1 if they are equal, 0 if they are not equal, or `NULL` if either expression is NULL. If the expressions return different data types (for example, a number and a string), a type conversion is performed.

When used in row comparisons these two queries are synonymous and return the same results:

```
SELECT (t1.a, t1.b) = (t2.x, t2.y) FROM t1 INNER JOIN t2;

SELECT (t1.a = t2.x) AND (t1.b = t2.y) FROM t1 INNER JOIN t2;
```

To perform a NULL-safe comparison, use the `<=>` operator.

`=` can also be used as an assignment operator.

Examples

```
SELECT 1 = 0;
+-------+
| 1 = 0 |
+-------+
|     0 |
+-------+

SELECT '0' = 0;
+---------+
| '0' = 0 |
+---------+
|       1 |
+---------+

SELECT '0.0' = 0;
+-----------+
| '0.0' = 0 |
+-----------+
|         1 |
+-----------+

SELECT '0.01' = 0;
+------------+
| '0.01' = 0 |
+------------+
|          0 |
+------------+

SELECT '.01' = 0.01;
+--------------+
| '.01' = 0.01 |
+--------------+
|            1 |
+--------------+

SELECT (5 * 2) = CONCAT('1', '0');
+----------------------------+
| (5 * 2) = CONCAT('1', '0') |
+----------------------------+
|                          1 |
+----------------------------+

SELECT 1 = NULL;
+----------+
| 1 = NULL |
+----------+
|     NULL |
+----------+

SELECT NULL = NULL;
+-------------+
| NULL = NULL |
+-------------+
|        NULL |
+-------------+
```

1.1.5.4.6 >

Syntax

```
>
```

Contents

Description

Greater than operator. Evaluates both SQL expressions and returns 1 if the left value is greater than the right value and 0 if it is not, or `NULL` if either expression is NULL. If the expressions return different data types, (for instance, a number and a string), performs type conversion.

When used in row comparisons these two queries return the same results:

```
SELECT (t1.a, t1.b) > (t2.x, t2.y)
FROM t1 INNER JOIN t2;

SELECT (t1.a > t2.x) OR ((t1.a = t2.x) AND (t1.b > t2.y))
FROM t1 INNER JOIN t2;
```

Examples

```
SELECT 2 > 2;
+-------+
| 2 > 2 |
+-------+
|     0 |
+-------+

SELECT 'b' > 'a';
+-----------+
| 'b' > 'a' |
+-----------+
|         1 |
+-----------+
```

1.1.5.4.7 >=

Syntax

```
>=
```

Contents

Description

Greater than or equal operator. Evaluates both SQL expressions and returns 1 if the left value is greater than or equal to the right value and 0 if it is not, or `NULL` if either expression is NULL. If the expressions return different data types, (for instance, a number and a string), performs type conversion.

When used in row comparisons these two queries return the same results:

```
SELECT (t1.a, t1.b) >= (t2.x, t2.y)
FROM t1 INNER JOIN t2;

SELECT (t1.a > t2.x) OR ((t1.a = t2.x) AND (t1.b >= t2.y))
FROM t1 INNER JOIN t2;
```

Examples

```
SELECT 2 >= 2;
+--------+
| 2 >= 2 |
+--------+
|      1 |
+--------+

SELECT 'A' >= 'a';
+------------+
| 'A' >= 'a' |
+------------+
|          1 |
+------------+
```

1.1.5.4.8 BETWEEN AND

Syntax

```
expr BETWEEN min AND max
```

Contents

Description

If expr is greater than or equal to min and expr is less than or equal to max, BETWEEN returns 1, otherwise it returns 0. This is equivalent to the expression (min <= expr AND expr <= max) if all the arguments are of the same type. Otherwise type conversion takes place according to the rules described at Type Conversion, but applied to all the three arguments.

Examples

```
SELECT 1 BETWEEN 2 AND 3;
+-------------------+
| 1 BETWEEN 2 AND 3 |
+-------------------+
|                 0 |
+-------------------+
```

```
SELECT 'b' BETWEEN 'a' AND 'c';
+-------------------------+
| 'b' BETWEEN 'a' AND 'c' |
+-------------------------+
|                       1 |
+-------------------------+
```

```
SELECT 2 BETWEEN 2 AND '3';
+--------------------+
| 2 BETWEEN 2 AND '3' |
+--------------------+
|                  1 |
+--------------------+
```

```
SELECT 2 BETWEEN 2 AND 'x-3';
+----------------------+
| 2 BETWEEN 2 AND 'x-3' |
+----------------------+
|                    0 |
+----------------------+
1 row in set, 1 warning (0.00 sec)

Warning (Code 1292): Truncated incorrect DOUBLE value: 'x-3'
```

NULL:

```
SELECT 1 BETWEEN 1 AND NULL;
+----------------------+
| 1 BETWEEN 1 AND NULL |
+----------------------+
|                 NULL |
+----------------------+
```

DATE, DATETIME and TIMESTAMP examples. Omitting the time component compares against `00:00`, so later times on the same date are not returned:

```
CREATE TABLE `x` (
  a date ,
  b datetime,
  c timestamp
)

INSERT INTO x VALUES
 ('2018-11-11', '2018-11-11 05:15', '2018-11-11 05:15'),
 ('2018-11-12', '2018-11-12 05:15', '2018-11-12 05:15');

SELECT * FROM x WHERE a BETWEEN '2018-11-11' AND '2018-11-12';
+------------+---------------------+---------------------+
| a          | b                   | c                   |
+------------+---------------------+---------------------+
| 2018-11-11 | 2018-11-11 05:15:00 | 2018-11-11 05:15:00 |
| 2018-11-12 | 2018-11-12 05:15:00 | 2018-11-12 05:15:00 |
+------------+---------------------+---------------------+

SELECT * FROM x WHERE b BETWEEN '2018-11-11' AND '2018-11-12';
+------------+---------------------+---------------------+
| a          | b                   | c                   |
+------------+---------------------+---------------------+
| 2018-11-11 | 2018-11-11 05:15:00 | 2018-11-11 05:15:00 |
+------------+---------------------+---------------------+

SELECT * FROM x WHERE c BETWEEN '2018-11-11' AND '2018-11-12';
+------------+---------------------+---------------------+
| a          | b                   | c                   |
+------------+---------------------+---------------------+
| 2018-11-11 | 2018-11-11 05:15:00 | 2018-11-11 05:15:00 |
+------------+---------------------+---------------------+
```

1.1.5.4.9 COALESCE

Syntax

```
COALESCE(value,...)
```

Description

Returns the first non-NULL value in the list, or NULL if there are no non-NULL values. At least one parameter must be passed.

The function is useful when substituting a default value for null values when displaying data.

See also NULL Values in MariaDB.

Examples

```
SELECT COALESCE(NULL,1);
+------------------+
| COALESCE(NULL,1) |
+------------------+
|                1 |
+------------------+
```

```
SELECT COALESCE(NULL,NULL,NULL);
+--------------------------+
| COALESCE(NULL,NULL,NULL) |
+--------------------------+
|                     NULL |
+--------------------------+
```

When two arguments are given, COALESCE() is the same as IFNULL():

```
SET @a=NULL, @b=1;

SELECT COALESCE(@a, @b), IFNULL(@a, @b);
+-------------------+----------------+
| COALESCE(@a, @b) | IFNULL(@a, @b) |
+-------------------+----------------+
|                1 |              1 |
+-------------------+----------------+
```

Hex type confusion:

```
CREATE TABLE t1 (a INT, b VARCHAR(10));
INSERT INTO t1 VALUES (0x31, 0x61),(COALESCE(0x31), COALESCE(0x61));

SELECT * FROM t1;
+------+------+
| a    | b    |
+------+------+
|   49 | a    |
|    1 | a    |
+------+------+
```

The reason for the differing results above is that when 0x31 is inserted directly to the column, it's treated as a number (see Hexadecimal Literals), while when 0x31 is passed to COALESCE(), it's treated as a string, because:

- HEX values have a string data type by default.
- COALESCE() has the same data type as the argument.

Substituting zero for NULL (in this case when the aggregate function returns NULL after finding no rows):

```
SELECT SUM(score) FROM student;
+------------+
| SUM(score) |
+------------+
|       NULL |
+------------+

SELECT COALESCE(SUM(score),0) FROM student;
+-------------------------+
| COALESCE(SUM(score),0) |
+-------------------------+
|                       0 |
+-------------------------+
```

1.1.5.4.10 GREATEST

Syntax

```
GREATEST(value1,value2,...)
```

Description

With two or more arguments, returns the largest (maximum-valued) argument. The arguments are compared using the same rules as for LEAST().

Examples

```
SELECT GREATEST(2,0);
+---------------+
| GREATEST(2,0) |
+---------------+
|             2 |
+---------------+
```

```
SELECT GREATEST(34.0,3.0,5.0,767.0);
+-----------------------------+
| GREATEST(34.0,3.0,5.0,767.0) |
+-----------------------------+
|                       767.0 |
+-----------------------------+
```

```
SELECT GREATEST('B','A','C');
+-----------------------+
| GREATEST('B','A','C') |
+-----------------------+
| C                     |
+-----------------------+
```

1.1.5.4.11 IN

Syntax

```
expr IN (value,...)
```

Description

Returns 1 if *expr* is equal to any of the values in the IN list, else returns 0. If all values are constants, they are evaluated according to the type of *expr* and sorted. The search for the item then is done using a binary search. This means IN is very quick if the IN value list consists entirely of constants. Otherwise, type conversion takes place according to the rules described at Type Conversion, but applied to all the arguments.

If *expr* is NULL, IN always returns NULL. If at least one of the values in the list is NULL, and one of the comparisons is true, the result is 1. If at least one of the values in the list is NULL and none of the comparisons is true, the result is NULL.

Examples

```
SELECT 2 IN (0,3,5,7);
+---------------+
| 2 IN (0,3,5,7) |
+---------------+
|              0 |
+---------------+
```

```
SELECT 'wefwf' IN ('wee','wefwf','weg');
+---------------------------------+
| 'wefwf' IN ('wee','wefwf','weg') |
+---------------------------------+
|                                1 |
+---------------------------------+
```

Type conversion:

```
SELECT 1 IN ('1', '2', '3');
+----------------------+
| 1 IN ('1', '2', '3') |
+----------------------+
|                    1 |
+----------------------+
```

```
SELECT NULL IN (1, 2, 3);
+------------------+
| NULL IN (1, 2, 3) |
+------------------+
|             NULL |
+------------------+

SELECT 1 IN (1, 2, NULL);
+------------------+
| 1 IN (1, 2, NULL) |
+------------------+
|                1 |
+------------------+

SELECT 5 IN (1, 2, NULL);
+------------------+
| 5 IN (1, 2, NULL) |
+------------------+
|             NULL |
+------------------+
```

1.1.5.4.12 INTERVAL

Syntax

```
INTERVAL(N,N1,N2,N3,...)
```

Description

Returns the index of the last argument that is less than the first argument or is NULL.

Returns 0 if N < N1, 1 if N < N2, 2 if N < N3 and so on or -1 if N is NULL. All arguments are treated as integers. It is required that N1 < N2 < N3 < ... < Nn for this function to work correctly. This is because a fast binary search is used.

Examples

```
SELECT INTERVAL(23, 1, 15, 17, 30, 44, 200);
+-------------------------------------+
| INTERVAL(23, 1, 15, 17, 30, 44, 200) |
+-------------------------------------+
|                                   3 |
+-------------------------------------+

SELECT INTERVAL(10, 1, 10, 100, 1000);
+---------------------------------+
| INTERVAL(10, 1, 10, 100, 1000) |
+---------------------------------+
|                               2 |
+---------------------------------+

SELECT INTERVAL(22, 23, 30, 44, 200);
+--------------------------------+
| INTERVAL(22, 23, 30, 44, 200) |
+--------------------------------+
|                              0 |
+--------------------------------+

SELECT INTERVAL(10, 2, NULL);
+-----------------------+
| INTERVAL(10, 2, NULL) |
+-----------------------+
|                     2 |
+-----------------------+
```

1.1.5.4.13 IS

Syntax

```
IS boolean_value
```

Description

Tests a value against a boolean value, where `boolean_value` can be TRUE, FALSE, or UNKNOWN.

There is an important difference between using IS TRUE or comparing a value with TRUE using `=`. When using `=`, only `1` equals to TRUE. But when using IS TRUE, all values which are logically true (like a number > 1) return TRUE.

Examples

```
SELECT 1 IS TRUE, 0 IS FALSE, NULL IS UNKNOWN;
+-----------+------------+----------------+
| 1 IS TRUE | 0 IS FALSE | NULL IS UNKNOWN |
+-----------+------------+----------------+
|         1 |          1 |               1 |
+-----------+------------+----------------+
```

Difference between `=` and `IS TRUE`:

```
SELECT 2 = TRUE, 2 IS TRUE;
+----------+-----------+
| 2 = TRUE | 2 IS TRUE |
+----------+-----------+
|        0 |         1 |
+----------+-----------+
```

1.1.5.4.14 IS NOT

Syntax

```
IS NOT boolean_value
```

Description

Tests a value against a boolean value, where boolean_value can be TRUE, FALSE, or UNKNOWN.

Examples

```
SELECT 1 IS NOT UNKNOWN, 0 IS NOT UNKNOWN, NULL IS NOT UNKNOWN;
+------------------+------------------+----------------------+
| 1 IS NOT UNKNOWN | 0 IS NOT UNKNOWN | NULL IS NOT UNKNOWN |
+------------------+------------------+----------------------+
|                1 |                1 |                    0 |
+------------------+------------------+----------------------+
```

```
SELECT NULL IS NOT TRUE, NULL IS NOT FALSE;
+------------------+-------------------+
| NULL IS NOT TRUE | NULL IS NOT FALSE |
+------------------+-------------------+
|                1 |                 1 |
+------------------+-------------------+
```

1.1.5.4.15 IS NOT NULL

Syntax

```
IS NOT NULL
```

Description

Tests whether a value is not NULL. See also NULL Values in MariaDB.

Examples

```
SELECT 1 IS NOT NULL, 0 IS NOT NULL, NULL IS NOT NULL;
+--------------+--------------+------------------+
| 1 IS NOT NULL | 0 IS NOT NULL | NULL IS NOT NULL |
+--------------+--------------+------------------+
|             1 |             1 |                0 |
+--------------+--------------+------------------+
```

1.1.5.4.16 IS NULL

Syntax

```
IS NULL
```

Description

Tests whether a value is NULL. See also NULL Values in MariaDB.

Examples

```
SELECT 1 IS NULL, 0 IS NULL, NULL IS NULL;
+-----------+-----------+--------------+
| 1 IS NULL | 0 IS NULL | NULL IS NULL |
+-----------+-----------+--------------+
|         0 |         0 |            1 |
+-----------+-----------+--------------+
```

Compatibility

Some `ODBC` applications use the syntax `auto_increment_field IS NOT NULL` to find the latest row that was inserted with an autogenerated key value. If your applications need this, you can set the sql_auto_is_null variable to 1.

```
SET @@sql_auto_is_null=1;
CREATE TABLE t1 (auto_increment_column INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
INSERT INTO t1 VALUES (NULL);
SELECT * FROM t1 WHERE auto_increment_column IS NULL;

+-----------------------+
| auto_increment_column |
+-----------------------+
|                     1 |
+-----------------------+
```

1.1.5.4.17 ISNULL

Syntax

```
ISNULL(expr)
```

Description

If *`expr`* is NULL, ISNULL() returns 1, otherwise it returns 0.

See also NULL Values in MariaDB.

Examples

```
SELECT ISNULL(1+1);
+-------------+
| ISNULL(1+1) |
+-------------+
|           0 |
+-------------+

SELECT ISNULL(1/0);
+-------------+
| ISNULL(1/0) |
+-------------+
|           1 |
+-------------+
```

1.1.5.4.18 LEAST

Syntax

```
LEAST(value1,value2,...)
```

Description

With two or more arguments, returns the smallest (minimum-valued) argument. The arguments are compared using the following rules:

- If the return value is used in an INTEGER context or all arguments are integer-valued, they are compared as integers.
- If the return value is used in a REAL context or all arguments are real-valued, they are compared as reals.
- If any argument is a case-sensitive string, the arguments are compared as case-sensitive strings.
- In all other cases, the arguments are compared as case-insensitive strings.

LEAST() returns NULL if any argument is NULL.

Examples

```
SELECT LEAST(2,0);
+------------+
| LEAST(2,0) |
+------------+
|          0 |
+------------+
```

```
SELECT LEAST(34.0,3.0,5.0,767.0);
+--------------------------+
| LEAST(34.0,3.0,5.0,767.0) |
+--------------------------+
|                      3.0 |
+--------------------------+
```

```
SELECT LEAST('B','A','C');
+--------------------+
| LEAST('B','A','C') |
+--------------------+
| A                  |
+--------------------+
```

1.1.5.4.19 NOT BETWEEN

Syntax

```
expr NOT BETWEEN min AND max
```

Description

This is the same as NOT (expr BETWEEN min AND max).

Note that the meaning of the alternative form `NOT expr BETWEEN min AND max` is affected by the `HIGH_NOT_PRECEDENCE` SQL_MODE flag.

Examples

```
SELECT 1 NOT BETWEEN 2 AND 3;
+----------------------+
| 1 NOT BETWEEN 2 AND 3 |
+----------------------+
|                    1 |
+----------------------+
```

```
SELECT 'b' NOT BETWEEN 'a' AND 'c';
+-----------------------------+
| 'b' NOT BETWEEN 'a' AND 'c' |
+-----------------------------+
|                           0 |
+-----------------------------+
```

NULL:

```
SELECT 1 NOT BETWEEN 1 AND NULL;
+--------------------------+
| 1 NOT BETWEEN 1 AND NULL |
+--------------------------+
|                     NULL |
+--------------------------+
```

1.1.5.4.20 NOT IN

Syntax

```
expr NOT IN (value,...)
```

Description

This is the same as NOT (expr IN (value,...)).

Examples

```
SELECT 2 NOT IN (0,3,5,7);
+--------------------+
| 2 NOT IN (0,3,5,7) |
+--------------------+
|                  1 |
+--------------------+
```

```
SELECT 'wefwf' NOT IN ('wee','wefwf','weg');
+-------------------------------------+
| 'wefwf' NOT IN ('wee','wefwf','weg') |
+-------------------------------------+
|                                   0 |
+-------------------------------------+
```

```
SELECT 1 NOT IN ('1', '2', '3');
+--------------------------+
| 1 NOT IN ('1', '2', '3') |
+--------------------------+
|                        0 |
+--------------------------+
```

NULL:

```
SELECT NULL NOT IN (1, 2, 3);
+-----------------------+
| NULL NOT IN (1, 2, 3) |
+-----------------------+
|                  NULL |
+-----------------------+

SELECT 1 NOT IN (1, 2, NULL);
+-----------------------+
| 1 NOT IN (1, 2, NULL) |
+-----------------------+
|                     0 |
+-----------------------+

SELECT 5 NOT IN (1, 2, NULL);
+-----------------------+
| 5 NOT IN (1, 2, NULL) |
+-----------------------+
|                  NULL |
+-----------------------+
```

1.1.5.5 Logical Operators

NOT, AND, Exclusive OR and OR

!

Logical NOT.

&&

Logical AND.

XOR

Logical XOR.

||

Logical OR.

1.1.5.5.1 !

Syntax

```
NOT, !
```

Contents

Description

Logical NOT. Evaluates to 1 if the operand is 0, to 0 if the operand is non-zero, and NOT NULL returns NULL.

By default, the `!` operator has a higher precedence. If the `HIGH_NOT_PRECEDENCE` SQL_MODE flag is set, `NOT` and `!` have the same precedence.

Examples

```
SELECT NOT 10;
+--------+
| NOT 10 |
+--------+
|      0 |
+--------+

SELECT NOT 0;
+-------+
| NOT 0 |
+-------+
|     1 |
+-------+

SELECT NOT NULL;
+----------+
| NOT NULL |
+----------+
|     NULL |
+----------+

SELECT ! (1+1);
+---------+
| ! (1+1) |
+---------+
|       0 |
+---------+

SELECT ! 1+1;
+-------+
| ! 1+1 |
+-------+
|     1 |
+-------+
```

1.1.5.5.2 &&

Syntax

```
AND, &&
```

Contents

Description

Logical AND. Evaluates to 1 if all operands are non-zero and not NULL, to 0 if one or more operands are 0, otherwise NULL is returned.

For this operator, short-circuit evaluation can be used.

Examples

```
SELECT 1 && 1;
+--------+
| 1 && 1 |
+--------+
|      1 |
+--------+

SELECT 1 && 0;
+--------+
| 1 && 0 |
+--------+
|      0 |
+--------+

SELECT 1 && NULL;
+-----------+
| 1 && NULL |
+-----------+
|      NULL |
+-----------+

SELECT 0 && NULL;
+-----------+
| 0 && NULL |
+-----------+
|         0 |
+-----------+

SELECT NULL && 0;
+-----------+
| NULL && 0 |
+-----------+
|         0 |
+-----------+
```

1.1.5.5.3 XOR

Syntax

```
XOR
```

Contents

Description

XOR stands for eXclusive OR. Returns NULL if either operand is NULL. For non-NULL operands, evaluates to 1 if an odd number of operands is non-zero, otherwise 0 is returned.

Examples

```
SELECT 1 XOR 1;
+---------+
| 1 XOR 1 |
+---------+
|       0 |
+---------+

SELECT 1 XOR 0;
+---------+
| 1 XOR 0 |
+---------+
|       1 |
+---------+

SELECT 1 XOR NULL;
+------------+
| 1 XOR NULL |
+------------+
|       NULL |
+------------+
```

In the following example, the right `1 XOR 1` is evaluated first, and returns `0`. Then, `1 XOR 0` is evaluated, and `1` is returned.

```
SELECT 1 XOR 1 XOR 1;
+---------------+
| 1 XOR 1 XOR 1 |
+---------------+
|             1 |
+---------------+
```

1.1.5.5.4 ||

Syntax

```
OR, ||
```

Contents

Description

Logical OR. When both operands are non-NULL, the result is 1 if any operand is non-zero, and 0 otherwise. With a NULL operand, the result is 1 if the other operand is non-zero, and NULL otherwise. If both operands are NULL, the result is NULL.

For this operator, short-circuit evaluation can be used.

Note that, if the `PIPES_AS_CONCAT` SQL_MODE is set, `||` is used as a string concatenation operator. This means that `a || b` is the same as `CONCAT(a,b)`. See `CONCAT()` for details.

Oracle Mode

MariaDB starting with 10.3

In Oracle mode from MariaDB 10.3, `||` ignores NULL.

Examples

```
SELECT 1 || 1;
+--------+
| 1 || 1 |
+--------+
|      1 |
+--------+

SELECT 1 || 0;
+--------+
| 1 || 0 |
+--------+
|      1 |
+--------+

SELECT 0 || 0;
+--------+
| 0 || 0 |
+--------+
|      0 |
+--------+

SELECT 0 || NULL;
+-----------+
| 0 || NULL |
+-----------+
|      NULL |
+-----------+

SELECT 1 || NULL;
+-----------+
| 1 || NULL |
+-----------+
|         1 |
+-----------+
```

In Oracle mode, from MariaDB 10.3:

```
SELECT 0 || NULL;
+-----------+
| 0 || NULL |
+-----------+
| 0         |
+-----------+
```

1.1.5.6 Operator Precedence

The precedence is the order in which the SQL operators are evaluated.

The following list shows the SQL operator precedence. Operators that appear first in the list have a higher precedence. Operators which are listed together have the same precedence.

- `INTERVAL`
- `BINARY`, `COLLATE`
- `!`
- `-` (unary minus), [[bitwise-not|]] (unary bit inversion)
- `||` (string concatenation)
- `^`
- `*`, `/`, `DIV`, `%`, `MOD`
- `-`, `+`
- `<<`, `>>`
- `&`
- `|`
- `=` (comparison), `<=>`, `>=`, `>`, `<=`, `<`, `<>`, `!=`, `IS`, `LIKE`, `REGEXP`, `IN`
- `BETWEEN`, `CASE`, `WHEN`, `THEN`, `ELSE`, `END`
- `NOT`
- `&&`, `AND`
- `XOR`
- `||` (logical or), `OR`

- `=` (assignment), `:=`

Functions precedence is always higher than operators precedence.

In this page `CASE` refers to the CASE operator, not to the CASE statement .

If the `HIGH_NOT_PRECEDENCE` SQL_MODE is set, `NOT` has the same precedence as `!` .

The `||` operator's precedence, as well as its meaning, depends on the `PIPES_AS_CONCAT` SQL_MODE flag: if it is on, `||` can be used to concatenate strings (like the CONCAT() function) and has a higher precedence.

The `=` operator's precedence depends on the context - it is higher when `=` is used as a comparison operator.

Parenthesis can be used to modify the operators precedence in an expression.

Short-circuit evaluation

The `AND` , `OR` , `&&` and `||` operators support short-circuit evaluation. This means that, in some cases, the expression on the right of those operators is not evaluated, because its result cannot affect the result. In the following cases, short-circuit evaluation is used and `x()` is not evaluated:

- `FALSE AND x()`
- `FALSE && x()`
- `TRUE OR x()`
- `TRUE || x()`
- `NULL BETWEEN x() AND x()`

Note however that the short-circuit evaluation does *not* apply to `NULL AND x()` . Also, `BETWEEN` 's right operands are not evaluated if the left operand is `NULL` , but in all other cases all the operands are evaluated.

This is a speed optimization. Also, since functions can have side-effects, this behavior can be used to choose whether execute them or not using a concise syntax:

```
SELECT some_function() OR log_error();
```

1.1.6 Sequences

This section is about sequence objects. For details about the storage engine, see Sequence Storage Engine.

A sequence is an object that generates a sequence of numeric values, as specified by the CREATE SEQUENCE statement. Sequences are an alternative to AUTO_INCREMENT when you want more control over how sequence numbers are generated.

Since a `SEQUENCE` caches values, it can sometimes be faster. Also, you can access the last value generated by all used sequences; it's not subjected to limitations of LAST_INSERT_ID( ).

See also Sequence Storage Engine.

Sequence Overview

Object that generates a sequence of numeric values.

CREATE SEQUENCE

Creates a sequence that generates new values when called with NEXT VALUE FOR.

SHOW CREATE SEQUENCE

Shows the CREATE SEQUENCE statement that created the sequence.

ALTER SEQUENCE

Change options for a SEQUENCE.

DROP SEQUENCE

Deleting a SEQUENCE.

SEQUENCE Functions

Functions that can be used on SEQUENCEs.

SHOW TABLES

List of non-temporary tables, views or sequences.

1.1.6.1 Sequence Overview

This page is about sequence objects. For details about the storage engine, see Sequence Storage Engine.

Introduction

A sequence is an object that generates a sequence of numeric values, as specified by the CREATE SEQUENCE statement.

CREATE SEQUENCE will create a sequence that generates new values when called with NEXT VALUE FOR sequence_name. It's an alternative to AUTO INCREMENT when one wants to have more control of how the numbers are generated. As the SEQUENCE caches values (up to the CACHE value in the CREATE SEQUENCE statement, by default 1000) it can in some cases be much faster than AUTO INCREMENT. Another benefit is that one can access the last value generated by all used sequences, which solves one of the limitations with LAST_INSERT_ID().

Creating a Sequence

The CREATE SEQUENCE statement is used to create a sequence. Here is an example of a sequence starting at 100, incrementing by 10 each time:

```
CREATE SEQUENCE s START WITH 100 INCREMENT BY 10;
```

The CREATE SEQUENCE statement, along with defaults, can be viewd with the SHOW CREATE SEQUENCE STATEMENT, for example:

```
SHOW CREATE SEQUENCE s\G
*************************** 1. row ***************************
       Table: s
Create Table: CREATE SEQUENCE `s` start with 100 minvalue 1 maxvalue 9223372036854775806
  increment by 10 cache 1000 nocycle ENGINE=InnoDB
```

Using Sequence Objects

To get the next value from a sequence, use

```
NEXT VALUE FOR sequence_name
```

or

```
NEXTVAL(sequence_name)
```

or in Oracle mode (SQL_MODE=ORACLE)

```
sequence_name.nextval
```

For retrieving the last value used by the current connection from a sequence use:

```
PREVIOUS VALUE FOR sequence_name
```

or

```
LASTVAL(sequence_name)
```

or in Oracle mode (SQL_MODE=ORACLE)

```
sequence_name.currval
```

For example:

```
SELECT NEXTVAL(s);
+------------+
| NEXTVAL(s) |
+------------+
|        100 |
+------------+

SELECT NEXTVAL(s);
+------------+
| NEXTVAL(s) |
+------------+
|        110 |
+------------+

SELECT LASTVAL(s);
+------------+
| LASTVAL(s) |
+------------+
|        110 |
+------------+
```

Using Sequences in DEFAULT

Sequences can be used in DEFAULT:

```
create sequence s1;
create table t1 (a int primary key default (next value for s1), b int);
insert into t1 (b) values (1),(2);
select * from t1;
+---+------+
| a | b    |
+---+------+
| 1 |    1 |
| 2 |    2 |
+---+------+
```

Changing a Sequence

The ALTER SEQUENCE statement is used for changing sequences. For example, to restart the sequence at another value:

```
ALTER SEQUENCE s RESTART 50;

SELECT NEXTVAL(s);
+------------+
| NEXTVAL(s) |
+------------+
|         50 |
+------------+
```

The SETVAL function can also be used to set the next value to be returned for a SEQUENCE, for example:

```
SELECT SETVAL(s, 100);
+----------------+
| SETVAL(s, 100) |
+----------------+
|            100 |
+----------------+
```

`SETVAL` can only be used to increase the sequence value. Attempting to set a lower value will fail, returning NULL:

```
SELECT SETVAL(s, 50);
+---------------+
| SETVAL(s, 50) |
+---------------+
|          NULL |
+---------------+
```

Dropping a Sequence

The DROP SEQUENCE statement is used to drop a sequence, for example:

```
DROP SEQUENCE s;
```

Replication

If one wants to use Sequences in a master-master setup or with Galera one should use `INCREMENT=0`. This will tell the Sequence to use auto_increment_increment and auto_increment_offset to generate unique values for each server.

Standards Compliance

MariaDB supports both ANSI SQL and Oracle syntax for sequences.

However as `SEQUENCE` is implemented as a special kind of table, it uses the same namespace as tables. The benefits are that sequences show up in SHOW TABLES, and one can also create a sequence with CREATE TABLE and drop it with DROP TABLE. One can SELECT from it as from any other table. This ensures that all old tools that work with tables should work with sequences.

Since sequence objects act as regular tables in many contexts, they will be affected by LOCK TABLES. This is not the case in other DBMS, such as Oracle, where LOCK TABLE does not affect sequences.

Notes

One of the goals with the Sequence implementation is that all old tools, such as mariadb-dump (previously mysqldump), should work unchanged, while still keeping the normal usage of sequence standard compatibly.

To make this possible, `sequence` is currently implemented as a table with a few exclusive properties.

The special properties for sequence tables are:

- A sequence table has always one row.
- When one creates a sequence, either with CREATE TABLE or CREATE SEQUENCE, one row will be inserted.
- If one tries to insert into a sequence table, the single row will be updated. This allows mariadb-dump to work but also gives the additional benefit that one can change all properties of a sequence with a single insert. New applications should of course also use `ALTER SEQUENCE`.
- UPDATE or DELETE can't be performed on Sequence objects.
- Doing a select on the sequence shows the current state of the sequence, except the values that are reserved in the cache. The `next_value` column shows the next value not reserved by the cache.
- FLUSH TABLES will close the sequence and the next sequence number generated will be according to what's stored in the Sequence object. In effect, this will discard the cached values.
- A number of normal table operations work on Sequence tables. See next section.

Table Operations that Work with Sequences

- SHOW CREATE TABLE sequence_name. This shows the table structure that is behind the `SEQUENCE` including the field names that can be used with SELECT or even CREATE TABLE.
- CREATE TABLE sequence-structure ... SEQUENCE=1
- ALTER TABLE sequence RENAME TO sequence2
- RENAME TABLE sequence_name TO new_sequence_name
- DROP TABLE sequence_name. This is allowed mainly to get old tools like mariadb-dump to work with sequence tables.
- SHOW TABLES

Implementation

Internally, sequence tables are created as a normal table without rollback (the InnoDB, Aria and MySAM engines support this), wrapped by a sequence engine object. This allowed us to create sequences with almost no performance impact for normal tables. (The cost is one 'if' per insert if the binary log is enabled).

Underlying Table Structure

The following example shows the table structure of sequences and how it can be used as a table. (Output of results are

slightly edited to make them easier to read)

```
create sequence t1;
show create sequence t1\G
*************************** 1. row ***************************
  CREATE SEQUENCE `t1` start with 1 minvalue 1 maxvalue 9223372036854775806
  increment by 1 cache 1000 nocycle ENGINE=InnoDB

show create table t1\G
*************************** 1. row ***************************
Create Table: CREATE TABLE `t1` (
  `next_not_cached_value` bigint(21) NOT NULL,
  `minimum_value` bigint(21) NOT NULL,
  `maximum_value` bigint(21) NOT NULL,
  `start_value` bigint(21) NOT NULL COMMENT 'start value when sequences is created or value if
RESTART is used',
  `increment` bigint(21) NOT NULL COMMENT 'increment value',
  `cache_size` bigint(21) unsigned NOT NULL,
  `cycle_option` tinyint(1) unsigned NOT NULL COMMENT '0 if no cycles are allowed, 1 if the
sequence should begin a new cycle when maximum_value is passed',
  `cycle_count` bigint(21) NOT NULL COMMENT 'How many cycles have been done'
) ENGINE=InnoDB SEQUENCE=1


select * from t1\G
next_not_cached_value: 1
 minimum_value: 1
 maximum_value: 9223372036854775806
  start_value: 1
  increment: 1
  cache_size: 1000
  cycle_option: 0
  cycle_count: 0
```

The `cycle_count` column is incremented every time the sequence wraps around.

Credits

- Thanks to Jianwe Zhao from Aliyun for his work on SEQUENCE in AliSQL, which gave ideas and inspiration for this work.
- Thanks to Peter Gulutzan,who helped test and gave useful comments about the implementation.

1.1.6.2 CREATE SEQUENCE

Syntax

```
CREATE [OR REPLACE] [TEMPORARY] SEQUENCE [IF NOT EXISTS] sequence_name
[ INCREMENT [ BY | = ] increment ]
[ MINVALUE [=] minvalue | NO MINVALUE | NOMINVALUE ]
[ MAXVALUE [=] maxvalue | NO MAXVALUE | NOMAXVALUE ]
[ START [ WITH | = ] start ]
[ CACHE [=] cache | NOCACHE ] [ CYCLE | NOCYCLE]
[table_options]
```

Contents

The options for `CREATE SEQUENCE` can be given in any order, optionally followed by `table_options`.

table_options can be any of the normal table options in CREATE TABLE but the most usable ones are `ENGINE=...` and `COMMENT=`.

`NOMAXVALUE` and `NOMINVALUE` are there to allow one to create SEQUENCEs using the Oracle syntax.

Description

CREATE SEQUENCE will create a sequence that generates new values when called with `NEXT VALUE FOR sequence_name`. It's an alternative to AUTO INCREMENT when one wants to have more control of how the numbers are generated. As the SEQUENCE caches values (up to `CACHE`) it can in some cases be much faster than AUTO INCREMENT. Another benefit is that one can access the last value generated by all used sequences, which solves one of the limitations with LAST_INSERT_ID().

CREATE SEQUENCE requires the CREATE privilege.

DROP SEQUENCE can be used to drop a sequence, and ALTER SEQUENCE to change it.

Arguments to Create

The following options may be used:

Option	Default value	Description
INCREMENT	1	Increment to use for values. May be negative. Setting an increment of 0 causes the sequence to use the value of the auto_increment_increment system variable at the time of creation, which is always a positive number. (see MDEV-16035).
MINVALUE	1 if INCREMENT > 0 and -9223372036854775807 if INCREMENT < 0	Minimum value for the sequence
MAXVALUE	9223372036854775806 if INCREMENT > 0 and -1 if INCREMENT < 0	Max value for sequence
START	MINVALUE if INCREMENT > 0 and MAX_VALUE if INCREMENT< 0	First value that the sequence will generate
CACHE	1000	Number of values that should be cached. 0 if no CACHE. The underlying table will be updated first time a new sequence number is generated and each time the cache runs out.

If `CYCLE` is used then the sequence should start again from `MINVALUE` after it has run out of values. Default value is `NOCYCLE`.

Constraints on Create Arguments

To be able to create a legal sequence, the following must hold:

- MAXVALUE >= start
- MAXVALUE > MINVALUE
- START >= MINVALUE
- MAXVALUE <= 9223372036854775806 (LONGLONG_MAX-1)
- MINVALUE >= -9223372036854775807 (LONGLONG_MIN+1)

Note that sequences can't generate the maximum/minimum 64 bit number because of the constraint of `MINVALUE` and `MAXVALUE`.

Atomic DDL

MariaDB starting with 10.6.1

MariaDB 10.6.1 supports Atomic DDL and `CREATE SEQUENCE` is atomic.

Examples

```
CREATE SEQUENCE s START WITH 100 INCREMENT BY 10;

CREATE SEQUENCE s2 START WITH -100 INCREMENT BY -10;
```

The following statement fails, as the increment conflicts with the defaults

```
CREATE SEQUENCE s3 START WITH -100 INCREMENT BY 10;
ERROR 4082 (HY000): Sequence 'test.s3' values are conflicting
```

The sequence can be created by specifying workable minimum and maximum values:

```
CREATE SEQUENCE s3 START WITH -100 INCREMENT BY 10 MINVALUE=-100 MAXVALUE=1000;
```

1.1.1.2.8.17 SHOW CREATE SEQUENCE

1.1.6.4 ALTER SEQUENCE

Syntax

```
ALTER SEQUENCE [IF EXISTS] sequence_name
[ INCREMENT [ BY | = ] increment ]
[ MINVALUE [=] minvalue | NO MINVALUE | NOMINVALUE ]
[ MAXVALUE [=] maxvalue | NO MAXVALUE | NOMAXVALUE ]
[ START [ WITH | = ] start ] [ CACHE [=] cache ] [ [ NO ] CYCLE ]
[ RESTART [[WITH | =] restart]
```

Contents

`ALTER SEQUENCE` allows one to change any values for a `SEQUENCE` created with CREATE SEQUENCE.

The options for `ALTER SEQUENCE` can be given in any order.

Description

`ALTER SEQUENCE` changes the parameters of an existing sequence generator. Any parameters not specifically set in the `ALTER SEQUENCE` command retain their prior settings.

`ALTER SEQUENCE` requires the ALTER privilege.

Arguments to `ALTER SEQUENCE`

The following options may be used:

Option	Default value	Description
`INCREMENT`	1	Increment to use for values. May be negative.
`MINVALUE`	1 if `INCREMENT` > 0 and -9223372036854775807 if `INCREMENT` < 0	Minimum value for the sequence.
`MAXVALUE`	9223372036854775806 if `INCREMENT` > 0 and -1 if `INCREMENT` < 0	Max value for sequence.

START	MINVALUE if INCREMENT > 0 and MAX_VALUE if INCREMENT < 0	First value that the sequence will generate.
CACHE	1000	Number of values that should be cached. 0 if no CACHE . The underlying table will be updated first time a new sequence number is generated and each time the cache runs out.
CYCLE	0 (= NO CYCLE)	1 if the sequence should start again from MINVALUE # after it has run out of values.
RESTART	START if restart value not is given	If RESTART option is used, NEXT VALUE will return the restart value.

The optional clause RESTART [WITH restart] sets the next value for the sequence. This is equivalent to calling the SETVAL() function with the is_used argument as 0. The specified value will be returned by the next call of nextval. Using RESTART with no restart value is equivalent to supplying the start value that was recorded by CREATE SEQUENCE or last set by ALTER SEQUENCE START WITH .

ALTER SEQUENCE will not allow you to change the sequence so that it's inconsistent. For example:

```
CREATE SEQUENCE s1;
ALTER SEQUENCE s1 MINVALUE 10;
ERROR 4061 (HY000): Sequence 'test.t1' values are conflicting

ALTER SEQUENCE s1 MINVALUE 10 RESTART 10;
ERROR 4061 (HY000): Sequence 'test.t1' values are conflicting

ALTER SEQUENCE s1 MINVALUE 10 START 10 RESTART 10;
```

INSERT

To allow SEQUENCE objects to be backed up by old tools, like mariadb-dump, one can use SELECT to read the current state of a SEQUENCE object and use an INSERT to update the SEQUENCE object. INSERT is only allowed if all fields are specified:

```
CREATE SEQUENCE s1;
INSERT INTO s1 VALUES(1000,10,2000,1005,1,1000,0,0);
SELECT * FROM s1;

+------------+-----------+-----------+-------+-----------+-------+-------+-------+
| next_value | min_value | max_value | start | increment | cache | cycle | round |
+------------+-----------+-----------+-------+-----------+-------+-------+-------+
|       1000 |        10 |      2000 |  1005 |         1 |  1000 |     0 |     0 |
+------------+-----------+-----------+-------+-----------+-------+-------+-------+

SHOW CREATE SEQUENCE s1;
+-------+-----------------------------------------------------------------------------------------
--------------------------+
| Table | Create Table
|
+-------+-----------------------------------------------------------------------------------------
--------------------------+
| s1    | CREATE SEQUENCE `s1` start with 1005 minvalue 10 maxvalue 2000 increment by 1 cache
1000 nocycle ENGINE=Aria |
+-------+-----------------------------------------------------------------------------------------
--------------------------+
```

Notes

ALTER SEQUENCE will instantly affect all future SEQUENCE operations. This is in contrast to some other databases where the changes requested by ALTER SEQUENCE will not be seen until the sequence cache has run out.

ALTER SEQUENCE will take a full table lock of the sequence object during its (brief) operation. This ensures that ALTER SEQUENCE is replicated correctly. If you only want to set the next sequence value to a higher value than current, then you should use SETVAL() instead, as this is not blocking.

If you want to change storage engine, sequence comment or rename the sequence, you can use ALTER TABLE for this.

1.1.6.5 DROP SEQUENCE

Syntax

```
DROP [TEMPORARY] SEQUENCE [IF EXISTS] [/*COMMENT TO SAVE*/]
    sequence_name [, sequence_name] ...
```

Contents

Description

`DROP SEQUENCE` removes one or more sequences created with CREATE SEQUENCE. You must have the `DROP` privilege for each sequence. MariaDB returns an error indicating by name which non-existing tables it was unable to drop, but it also drops all of the tables in the list that do exist.

Important: When a table is dropped, user privileges on the table are not automatically dropped. See GRANT.

If another connection is using the sequence, a metadata lock is active, and this statement will wait until the lock is released. This is also true for non-transactional tables.

For each referenced sequence, DROP SEQUENCE drops a temporary sequence with that name, if it exists. If it does not exist, and the `TEMPORARY` keyword is not used, it drops a non-temporary sequence with the same name, if it exists. The `TEMPORARY` keyword ensures that a non-temporary sequence will not accidentally be dropped.

Use `IF EXISTS` to prevent an error from occurring for sequences that do not exist. A NOTE is generated for each non-existent sequence when using `IF EXISTS`. See SHOW WARNINGS.

DROP SEQUENCE requires the DROP privilege.

Notes

DROP SEQUENCE only removes sequences, not tables. However, DROP TABLE can remove both sequences and tables.

1.1.6.6 SEQUENCE Functions

Functions that can be used on SEQUENCEs

LASTVAL

Synonym for PREVIOUS VALUE for sequence_name.

NEXT VALUE for sequence_name

Generate next value for a SEQUENCE. Same as NEXTVAL().

NEXTVAL

Synonym for NEXT VALUE for sequence_name.

PREVIOUS VALUE FOR sequence_name

Get last value generated from a SEQUENCE. Same as LASTVAL().

SETVAL

Set the next value to be returned from a SEQUENCE.

1.1.6.6.1 LASTVAL

`LASTVAL` is a synonym for PREVIOUS VALUE for sequence_name.

1.1.6.6.2 NEXT VALUE for sequence_name

Syntax

```
NEXT VALUE FOR sequence
```

or

```
NEXTVAL(sequence_name)
```

or in Oracle mode (SQL_MODE=ORACLE)

```
sequence_name.nextval
```

Contents

`NEXT VALUE FOR` is ANSI SQL syntax while `NEXTVAL()` is PostgreSQL syntax.

Description

Generate next value for a `SEQUENCE`.

- You can greatly speed up `NEXT VALUE` by creating the sequence with the `CACHE` option. If not, every `NEXT VALUE` usage will cause changes in the stored `SEQUENCE` table.
- When using `NEXT VALUE` the value will be reserved at once and will not be reused, except if the `SEQUENCE` was created with `CYCLE`. This means that when you are using `SEQUENCE`s you have to expect gaps in the generated sequence numbers.
- If one updates the `SEQUENCE` with SETVAL() or ALTER SEQUENCE ... RESTART, `NEXT VALUE FOR` will notice this and start from the next requested value.
- FLUSH TABLES will close the sequence and the next sequence number generated will be according to what's stored in the `SEQUENCE` object. In effect, this will discard the cached values.
- A server restart (or closing the current connection) also causes a drop of all cached values. The cached sequence numbers are reserved only for the current connection.
- `NEXT VALUE` requires the `INSERT` privilege.
- You can also use `NEXT VALUE FOR sequence` for column `DEFAULT`.

1.1.6.6.3 NEXTVAL

`NEXTVAL` is a synonym for NEXT VALUE for sequence_name.

1.1.6.6.4 PREVIOUS VALUE FOR sequence_name

Syntax

```
PREVIOUS VALUE FOR sequence_name
```

or

```
LASTVAL(sequence_name)
```

or in Oracle mode (SQL_MODE=ORACLE)

```
sequence_name.currval
```

Contents

`PREVIOUS VALUE FOR` is IBM DB2 syntax while `LASTVAL()` is PostgreSQL syntax.

Description

Get last value in the current connection generated from a sequence.

- If the sequence has not yet been used by the connection, `PREVIOUS VALUE FOR` returns `NULL` (the same thing applies with a new connection which doesn't see a last value for an existing sequence).
- If a `SEQUENCE` has been dropped and re-created then it's treated as a new `SEQUENCE` and `PREVIOUS VALUE FOR` will return `NULL`.
- `FLUSH TABLES` has no effect on `PREVIOUS VALUE FOR`.
- Previous values for all used sequences are stored per connection until connection ends.
- `PREVIOUS VALUE FOR` requires the SELECT privilege.

Example

```
CREATE SEQUENCE s START WITH 100 INCREMENT BY 10;

SELECT PREVIOUS VALUE FOR s;
+----------------------+
| PREVIOUS VALUE FOR s |
+----------------------+
|                 NULL |
+----------------------+

# The function works for sequences only, if the table is used an error is generated
SELECT PREVIOUS VALUE FOR t;
ERROR 4089 (42S02): 'test.t' is not a SEQUENCE

# Call the NEXT VALUE FOR s:
SELECT NEXT VALUE FOR s;
+------------------+
| NEXT VALUE FOR s |
+------------------+
|              100 |
+------------------+

SELECT PREVIOUS VALUE FOR s;
+----------------------+
| PREVIOUS VALUE FOR s |
+----------------------+
|                  100 |
+----------------------+
```

Now try to start the new connection and check that the last value is still NULL, before updating the value in the new connection after the output of the new connection gets current value (110 in the example below). Note that first connection cannot see this change and the result of last value still remains the same (100 in the example above).

```
$ .mysql -uroot test -e"SELECT PREVIOUS VALUE FOR s; SELECT NEXT VALUE FOR s; SELECT PREVIOUS
VALUE FOR s;"
+----------------------+
| PREVIOUS VALUE FOR s |
+----------------------+
|                 NULL |
+----------------------+
+------------------+
| NEXT VALUE FOR s |
+------------------+
|              110 |
+------------------+
+----------------------+
| PREVIOUS VALUE FOR s |
+----------------------+
|                  110 |
+----------------------+
```

1.1.6.6.5 SETVAL

Syntax

```
SETVAL(sequence_name, next_value, [is_used, [round]])
```

Contents

Description

Set the next value to be returned for a `SEQUENCE`.

This function is compatible with PostgreSQL syntax, extended with the `round` argument.

If the `is_used` argument is not given or is `1` or `true`, then the next used value will one after the given value. If `is_used` is `0` or `false` then the next generated value will be the given value.

If `round` is used then it will set the `round` value (or the internal cycle count, starting at zero) for the sequence. If `round` is not used, it's assumed to be 0.

`next_value` must be an integer literal.

For `SEQUENCE` tables defined with `CYCLE` (see CREATE SEQUENCE) one should use both `next_value` and `round` to define the next value. In this case the current sequence value is defined to be `round`, `next_value`.

The result returned by `SETVAL()` is `next_value` or NULL if the given `next_value` and `round` is smaller than the current value.

`SETVAL()` will not set the `SEQUENCE` value to a something that is less than its current value. This is needed to ensure that `SETVAL()` is replication safe. If you want to set the SEQUENCE to a smaller number use ALTER SEQUENCE.

If `CYCLE` is used, first `round` and then `next_value` are compared to see if the value is bigger than the current value.

Internally, in the MariaDB server, `SETVAL()` is used to inform slaves that a `SEQUENCE` has changed value. The slave may get `SETVAL()` statements out of order, but this is ok as only the biggest one will have an effect.

`SETVAL` requires the INSERT privilege.

Examples

```
SELECT setval(foo, 42);           -- Next nextval will return 43
SELECT setval(foo, 42, true);     -- Same as above
SELECT setval(foo, 42, false);    -- Next nextval will return 42
```

SETVAL setting higher and lower values on a sequence with an increment of 10:

```
SELECT NEXTVAL(s);
+------------+
| NEXTVAL(s) |
+------------+
|         50 |
+------------+

SELECT SETVAL(s, 100);
+----------------+
| SETVAL(s, 100) |
+----------------+
|            100 |
+----------------+

SELECT NEXTVAL(s);
+------------+
| NEXTVAL(s) |
+------------+
|        110 |
+------------+

SELECT SETVAL(s, 50);
+---------------+
| SETVAL(s, 50) |
+---------------+
|          NULL |
+---------------+

SELECT NEXTVAL(s);
+------------+
| NEXTVAL(s) |
+------------+
|        120 |
+------------+
```

Example demonstrating `round`:

```
CREATE OR REPLACE SEQUENCE s1
  START WITH 1
  MINVALUE 1
  MAXVALUE 99
  INCREMENT BY 1
  CACHE 20
  CYCLE;

SELECT SETVAL(s1, 99, 1, 0);
+----------------------+
| SETVAL(s1, 99, 1, 0) |
+----------------------+
|                   99 |
+----------------------+

SELECT NEXTVAL(s1);
+-------------+
| NEXTVAL(s1) |
+-------------+
|           1 |
+-------------+
```

The following statement returns NULL, as the given `next_value` and `round` is smaller than the current value.

```
SELECT SETVAL(s1, 99, 1, 0);
+----------------------+
| SETVAL(s1, 99, 1, 0) |
+----------------------+
|                 NULL |
+----------------------+

SELECT NEXTVAL(s1);
+-------------+
| NEXTVAL(s1) |
+-------------+
|           2 |
+-------------+
```

Increasing the round from zero to 1 will allow `next_value` to be returned.

```
SELECT SETVAL(s1, 99, 1, 1);
+----------------------+
| SETVAL(s1, 99, 1, 1) |
+----------------------+
|                   99 |
+----------------------+

SELECT NEXTVAL(s1);
+-------------+
| NEXTVAL(s1) |
+-------------+
|           1 |
+-------------+
```

1.1.1.2.8.52 SHOW TABLES

1.1.7 Temporal Tables

MariaDB supports temporal data tables in the form of system-versioning tables (allowing you to query and operate on historic data), application-time periods (allow you to query and operate on a temporal range of data), and bitemporal tables (which combine both system-versioning and application-time periods).

System-Versioned Tables

System-versioned tables record the history of all changes to table data.

Application-Time Periods

Application-time period tables, defined by a range between two temporal columns.

Bitemporal Tables

Bitemporal tables use versioning both at the system and application-time period levels.

1.1.7.1 System-Versioned Tables

Contents

MariaDB supports temporal data tables in the form of system-versioning tables (allowing you to query and operate on historic data, discussed below), application-time periods (allow you to query and operate on a temporal range of data), and bitemporal tables (which combine both system-versioning and application-time periods).

System-Versioned Tables

System-versioned tables store the history of all changes, not only data which is currently valid. This allows data analysis for any point in time, auditing of changes and comparison of data from different points in time. Typical uses cases are:

- Forensic analysis & legal requirements to store data for N years.
- Data analytics (retrospective, trends etc.), e.g. to get your staff information as of one year ago.
- Point-in-time recovery - recover a table state as of particular point in time.

System-versioned tables were first introduced in the SQL:2011 standard.

Creating a System-Versioned Table

The CREATE TABLE syntax has been extended to permit creating a system-versioned table. To be system-versioned, according to SQL:2011, a table must have two generated columns, a period, and a special table option clause:

```
CREATE TABLE t(
   x INT,
   start_timestamp TIMESTAMP(6) GENERATED ALWAYS AS ROW START,
   end_timestamp TIMESTAMP(6) GENERATED ALWAYS AS ROW END,
   PERIOD FOR SYSTEM_TIME(start_timestamp, end_timestamp)
) WITH SYSTEM VERSIONING;
```

In MariaDB one can also use a simplified syntax:

```
CREATE TABLE t (
   x INT
) WITH SYSTEM VERSIONING;
```

In the latter case no extra columns will be created and they won't clutter the output of, say, `SELECT * FROM t`. The versioning information will still be stored, and it can be accessed via the pseudo-columns `ROW_START` and `ROW_END`:

```
SELECT x, ROW_START, ROW_END FROM t;
```

Adding or Removing System Versioning To/From a Table

An existing table can be altered to enable system versioning for it.

```
CREATE TABLE t(
  x INT
);
```

```
ALTER TABLE t ADD SYSTEM VERSIONING;
```

```
SHOW CREATE TABLE t\G
*************************** 1. row ***************************
       Table: t
Create Table: CREATE TABLE `t` (
  `x` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1 WITH SYSTEM VERSIONING
```

Similarly, system versioning can be removed from a table:

```
ALTER TABLE t DROP SYSTEM VERSIONING;
```

```
SHOW CREATE TABLE t\G
*************************** 1. row ***************************
       Table: t
Create Table: CREATE TABLE `t` (
  `x` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

One can also add system versioning with all columns created explicitly:

```
ALTER TABLE t ADD COLUMN ts TIMESTAMP(6) GENERATED ALWAYS AS ROW START,
              ADD COLUMN te TIMESTAMP(6) GENERATED ALWAYS AS ROW END,
              ADD PERIOD FOR SYSTEM_TIME(ts, te),
              ADD SYSTEM VERSIONING;
```

```
SHOW CREATE TABLE t\G
*************************** 1. row ***************************
       Table: t
Create Table: CREATE TABLE `t` (
  `x` int(11) DEFAULT NULL,
  `ts` timestamp(6) GENERATED ALWAYS AS ROW START,
  `te` timestamp(6) GENERATED ALWAYS AS ROW END,
  PERIOD FOR SYSTEM_TIME (`ts`, `te`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 WITH SYSTEM VERSIONING
```

Inserting Data

When data is inserted into a system-versioned table, it is given a *row_start* value of the current timestamp, and a *row_end* value of FROM_UNIXTIME(2147483647.999999). The current timestamp can be adjusted by setting the timestamp system variable, for example:

```
SELECT NOW();
+---------------------+
| NOW()               |
+---------------------+
| 2022-10-24 23:09:38 |
+---------------------+

INSERT INTO t VALUES(1);

SET @@timestamp = UNIX_TIMESTAMP('2033-10-24');

INSERT INTO t VALUES(2);

SET @@timestamp = default;

INSERT INTO t VALUES(3);

SELECT a,row_start,row_end FROM t;
+------+----------------------------+----------------------------+
| a    | row_start                  | row_end                    |
+------+----------------------------+----------------------------+
|    1 | 2022-10-24 23:09:38.951347 | 2038-01-19 05:14:07.999999 |
|    2 | 2033-10-24 00:00:00.000000 | 2038-01-19 05:14:07.999999 |
|    3 | 2022-10-24 23:09:38.961857 | 2038-01-19 05:14:07.999999 |
+------+----------------------------+----------------------------+
```

Querying Historical Data

`SELECT`

To query the historical data one uses the clause `FOR SYSTEM_TIME` directly after the table name (before the table alias, if any). SQL:2011 provides three syntactic extensions:

- `AS OF` is used to see the table as it was at a specific point in time in the past:

```
SELECT * FROM t FOR SYSTEM_TIME AS OF TIMESTAMP'2016-10-09 08:07:06';
```

- `BETWEEN start AND end` will show all rows that were visible at any point between two specified points in time. It works inclusively, a row visible exactly at *start* or exactly at *end* will be shown too.

```
SELECT * FROM t FOR SYSTEM_TIME BETWEEN (NOW() - INTERVAL 1 YEAR) AND NOW();
```

- `FROM start TO end` will also show all rows that were visible at any point between two specified points in time, including *start*, but **excluding** *end*.

```
SELECT * FROM t FOR SYSTEM_TIME FROM '2016-01-01 00:00:00' TO '2017-01-01 00:00:00';
```

Additionally MariaDB implements a non-standard extension:

- `ALL` will show all rows, historical and current.

```
SELECT * FROM t FOR SYSTEM_TIME ALL;
```

If the `FOR SYSTEM_TIME` clause is not used, the table will show the *current* data. This is usually the same as if one had specified `FOR SYSTEM_TIME AS OF CURRENT_TIMESTAMP`, unless one has adjusted the *row_start* value (until MariaDB 10.11, only possible by setting the secure_timestamp variable). For example:

```
CREATE OR REPLACE TABLE t (a int) WITH SYSTEM VERSIONING;

SELECT NOW();
+---------------------+
| NOW()               |
+---------------------+
| 2022-10-24 23:43:37 |
+---------------------+

INSERT INTO t VALUES (1);

SET @@timestamp = UNIX_TIMESTAMP('2033-03-03');

INSERT INTO t VALUES (2);

DELETE FROM t;

SET @@timestamp = default;


SELECT a, row_start, row_end FROM t FOR SYSTEM_TIME ALL;
+------+----------------------------+----------------------------+
| a    | row_start                  | row_end                    |
+------+----------------------------+----------------------------+
|    1 | 2022-10-24 23:43:37.192725 | 2033-03-03 00:00:00.000000 |
|    2 | 2033-03-03 00:00:00.000000 | 2033-03-03 00:00:00.000000 |
+------+----------------------------+----------------------------+
2 rows in set (0.000 sec)


SELECT a, row_start, row_end FROM t FOR SYSTEM_TIME AS OF CURRENT_TIMESTAMP;
+------+----------------------------+----------------------------+
| a    | row_start                  | row_end                    |
+------+----------------------------+----------------------------+
|    1 | 2022-10-24 23:43:37.192725 | 2033-03-03 00:00:00.000000 |
+------+----------------------------+----------------------------+
1 row in set (0.000 sec)


SELECT a, row_start, row_end FROM t;
Empty set (0.001 sec)
```

Views and Subqueries

When a system-versioned tables is used in a view or in a subquery in the from clause, `FOR SYSTEM_TIME` can be used directly in the view or subquery body, or (non-standard) applied to the whole view when it's being used in a `SELECT`:

```
CREATE VIEW v1 AS SELECT * FROM t FOR SYSTEM_TIME AS OF TIMESTAMP'2016-10-09 08:07:06';
```

Or

```
CREATE VIEW v1 AS SELECT * FROM t;
SELECT * FROM v1 FOR SYSTEM_TIME AS OF TIMESTAMP'2016-10-09 08:07:06';
```

Use in Replication and Binary Logs

Tables that use system-versioning implicitly add the `row_end` column to the Primary Key. While this is generally not an issue for most use cases, it can lead to problems when re-applying write statements from the binary log or in replication environments, where a primary retries an SQL statement on the replica.

Specifically, these writes include a value on the `row_end` column containing the timestamp from when the write was initially made. The re-occurrence of the Primary Key with the old system-versioning columns raises an error due to the duplication.

To mitigate this with MariaDB Replication, set the secure_timestamp system variable to `YES` on the replica. When set, the replica uses its own system clock when applying to the row log, meaning that the primary can retry as many times as needed without causing a conflict. The retries generate new historical rows with new values for the `row_start` and `row_end` columns.

Transaction-Precise History in InnoDB

A point in time when a row was inserted or deleted does not necessarily mean that a change became visible at the same moment. With transactional tables, a row might have been inserted in a long transaction, and became visible hours after it was inserted.

For some applications — for example, when doing data analytics on one-year-old data — this distinction does not matter much. For others — forensic analysis — it might be crucial.

MariaDB supports transaction-precise history (only for the InnoDB storage engine) that allows seeing the data exactly as it would've been seen by a new connection doing a `SELECT` at the specified point in time — rows inserted *before* that point, but committed *after* will not be shown.

To use transaction-precise history, InnoDB needs to remember not timestamps, but transaction identifier per row. This is done by creating generated columns as `BIGINT UNSIGNED`, not `TIMESTAMP(6)`:

```
CREATE TABLE t(
   x INT,
   start_trxid BIGINT UNSIGNED GENERATED ALWAYS AS ROW START,
   end_trxid BIGINT UNSIGNED GENERATED ALWAYS AS ROW END,
   PERIOD FOR SYSTEM_TIME(start_trxid, end_trxid)
) WITH SYSTEM VERSIONING;
```

These columns must be specified explicitly, but they can be made INVISIBLE to avoid cluttering `SELECT *` output.

When one uses transaction-precise history, one can optionally use transaction identifiers in the `FOR SYSTEM_TIME` clause:

```
SELECT * FROM t FOR SYSTEM_TIME AS OF TRANSACTION 12345;
```

This will show the data, exactly as it was seen by the transaction with the identifier 12345.

Storing the History Separately

When the history is stored together with the current data, it increases the size of the table, so current data queries — table scans and index searches — will take more time, because they will need to skip over historical data. If most queries on that table use only current data, it might make sense to store the history separately, to reduce the overhead from versioning.

This is done by partitioning the table by `SYSTEM_TIME`. Because of the partition pruning optimization, all current data queries will only access one partition, the one that stores current data.

This example shows how to create such a partitioned table:

```
CREATE TABLE t (x INT) WITH SYSTEM VERSIONING
  PARTITION BY SYSTEM_TIME (
    PARTITION p_hist HISTORY,
    PARTITION p_cur CURRENT
  );
```

In this example all history will be stored in the partition `p_hist` while all current data will be in the partition `p_cur`. The table must have exactly one current partition and at least one historical partition.

Partitioning by `SYSTEM_TIME` also supports automatic partition rotation. One can rotate historical partitions by time or by size. This example shows how to rotate partitions by size:

```
CREATE TABLE t (x INT) WITH SYSTEM VERSIONING
  PARTITION BY SYSTEM_TIME LIMIT 100000 (
    PARTITION p0 HISTORY,
    PARTITION p1 HISTORY,
    PARTITION pcur CURRENT
  );
```

MariaDB will start writing history rows into partition `p0`, and when it reaches a size of 100000 rows, MariaDB will switch to partition `p1`. There are only two historical partitions, so when `p1` overflows, MariaDB will issue a warning, but will continue writing into it.

Similarly, one can rotate partitions by time:

```
CREATE TABLE t (x INT) WITH SYSTEM VERSIONING
  PARTITION BY SYSTEM_TIME INTERVAL 1 WEEK (
    PARTITION p0 HISTORY,
    PARTITION p1 HISTORY,
    PARTITION p2 HISTORY,
    PARTITION pcur CURRENT
  );
```

This means that the history for the first week after the table was created will be stored in `p0`. The history for the second week — in `p1`, and all later history will go into `p2`. One can see the exact rotation time for each partition in the INFORMATION_SCHEMA.PARTITIONS table.

It is possible to combine partitioning by `SYSTEM_TIME` and subpartitions:

```
CREATE TABLE t (x INT) WITH SYSTEM VERSIONING
  PARTITION BY SYSTEM_TIME
    SUBPARTITION BY KEY (x)
    SUBPARTITIONS 4 (
    PARTITION ph HISTORY,
    PARTITION pc CURRENT
  );
```

Default Partitions

MariaDB starting with 10.5.0

Since partitioning by current and historical data is such a typical usecase, from MariaDB 10.5, it is possible to use a simplified statement to do so. For example, instead of

```
CREATE TABLE t (x INT) WITH SYSTEM VERSIONING
  PARTITION BY SYSTEM_TIME (
    PARTITION p0 HISTORY,
    PARTITION pn CURRENT
);
```

you can use

```
CREATE TABLE t (x INT) WITH SYSTEM VERSIONING
  PARTITION BY SYSTEM_TIME;
```

You can also specify the number of partitions, which is useful if you want to rotate history by time, for example:

```
CREATE TABLE t (x INT) WITH SYSTEM VERSIONING
  PARTITION BY SYSTEM_TIME
    INTERVAL 1 MONTH
    PARTITIONS 12;
```

Specifying the number of partitions without specifying a rotation condition will result in a warning:

```
CREATE OR REPLACE TABLE t (x INT) WITH SYSTEM VERSIONING
  PARTITION BY SYSTEM_TIME PARTITIONS 12;
Query OK, 0 rows affected, 1 warning (0.518 sec)

Warning (Code 4115): Maybe missing parameters: no rotation condition for multiple HISTORY
partitions.
```

while specifying only 1 partition will result in an error:

```
CREATE OR REPLACE TABLE t (x INT) WITH SYSTEM VERSIONING
  PARTITION BY SYSTEM_TIME PARTITIONS 1;
ERROR 4128 (HY000): Wrong partitions for `t`: must have at least one HISTORY and exactly
one last CURRENT
```

Automatically Creating Partitions

MariaDB starting with 10.9.1

From MariaDB 10.9.1, the `AUTO` keyword can be used to automatically create history partitions.

For example

```
CREATE TABLE t1 (x int) WITH SYSTEM VERSIONING
    PARTITION BY SYSTEM_TIME INTERVAL 1 HOUR AUTO;

CREATE TABLE t1 (x int) WITH SYSTEM VERSIONING
   PARTITION BY SYSTEM_TIME INTERVAL 1 MONTH
   STARTS '2021-01-01 00:00:00' AUTO PARTITIONS 12;

CREATE TABLE t1 (x int) WITH SYSTEM VERSIONING
  PARTITION BY SYSTEM_TIME LIMIT 1000 AUTO;
```

Or with explicit partitions:

```
CREATE TABLE t1 (x int) WITH SYSTEM VERSIONING
  PARTITION BY SYSTEM_TIME INTERVAL 1 HOUR AUTO
  (PARTITION p0 HISTORY, PARTITION pn CURRENT);
```

To disable or enable auto-creation one can use ALTER TABLE by adding or removing AUTO from the partitioning specification:

```
CREATE TABLE t1 (x int) WITH SYSTEM VERSIONING
  PARTITION BY SYSTEM_TIME INTERVAL 1 HOUR AUTO;

# Disables auto-creation:
ALTER TABLE t1 PARTITION BY SYSTEM_TIME INTERVAL 1 HOUR;

# Enables auto-creation:
ALTER TABLE t1 PARTITION BY SYSTEM_TIME INTERVAL 1 HOUR AUTO;
```

If the rest of the partitioning specification is identical to CREATE TABLE, no repartitioning will be done (for details see MDEV-27328).

Removing Old History

Because it stores all the history, a system-versioned table might grow very large over time. There are many options to trim down the space and remove the old history.

One can completely drop the versioning from the table and add it back again, this will delete all the history:

```
ALTER TABLE t DROP SYSTEM VERSIONING;
ALTER TABLE t ADD SYSTEM VERSIONING;
```

It might be a rather time-consuming operation, though, as the table will need to be rebuilt, possibly twice (depending on the storage engine).

Another option would be to use partitioning and drop some of historical partitions:

```
ALTER TABLE t DROP PARTITION p0;
```

Note, that one cannot drop a current partition or the only historical partition.

And the third option; one can use a variant of the DELETE statement to prune the history:

```
DELETE HISTORY FROM t;
```

or only old history up to a specific point in time:

```
DELETE HISTORY FROM t BEFORE SYSTEM_TIME '2016-10-09 08:07:06';
```

or to a specific transaction (with `BEFORE SYSTEM_TIME TRANSACTION xxx`).

To protect the integrity of the history, this statement requires a special DELETE HISTORY privilege.

Currently, using the DELETE HISTORY statement with a BEFORE SYSTEM_TIME greater than the ROW_END of the active records (as a TIMESTAMP, this has a maximum value of '2038-01-19 03:14:07' UTC) will result in the historical records being dropped, and the active records being deleted and moved to history. See MDEV-25468.

Prior to MariaDB 10.4.5, the TRUNCATE TABLE statement drops all historical records from a system-versioned-table.

From MariaDB 10.4.5, historic data is protected from TRUNCATE statements, as per the SQL standard, and an Error 4137 is instead raised:

```
TRUNCATE t;
ERROR 4137 (HY000): System-versioned tables do not support TRUNCATE TABLE
```

Excluding Columns From Versioning

Another MariaDB extension allows one to version only a subset of columns in a table. This is useful, for example, if you have a table with user information that should be versioned, but one column is, let's say, a login counter that is incremented often and is not interesting to version. Such a column can be excluded from versioning by declaring it `WITHOUT VERSIONING`

```
CREATE TABLE t (
   x INT,
   y INT WITHOUT SYSTEM VERSIONING
) WITH SYSTEM VERSIONING;
```

A column can also be declared `WITH VERSIONING`, that will automatically make the table versioned. The statement below is equivalent to the one above:

```
CREATE TABLE t (
   x INT WITH SYSTEM VERSIONING,
   y INT
);
```

System Variables

There are a number of system variables related to system-versioned tables:

system_versioning_alter_history

- **Description:** SQL:2011 does not allow ALTER TABLE on system-versioned tables. When this variable is set to `ERROR`, an attempt to alter a system-versioned table will result in an error. When this variable is set to `KEEP`, ALTER TABLE will be allowed, but the history will become incorrect — querying historical data will show the new table structure. This mode is still useful, for example, when adding new columns to a table. Note that if historical data contains or would contain nulls, attempting to ALTER these columns to be `NOT NULL` will return an error (or warning if strict_mode is not set).
- **Commandline:** `--system-versioning-alter-history=value`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Type:** Enum
- **Default Value:** `ERROR`
- **Valid Values:** `ERROR`, `KEEP`

---

system_versioning_asof

- **Description:** If set to a specific timestamp value, an implicit `FOR SYSTEM_TIME AS OF` clause will be applied to all queries. This is useful if one wants to do many queries for history at the specific point in time. Set it to `DEFAULT` to restore the default behavior. Has no effect on DML, so queries such as INSERT .. SELECT and REPLACE .. SELECT need to state AS OF explicitly.
- **Commandline:** None
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Type:** Varchar
- **Default Value:** `DEFAULT`

---

system_versioning_innodb_algorithm_simple

- **Description:** Never fully implemented and removed in the following release.
- **Commandline:** `--system-versioning-innodb-algorithm-simple[={0|1}]`

- **Scope:** Global, Session
- **Dynamic:** Yes
- **Type:** Boolean
- **Default Value:** `ON`
- **Introduced:** MariaDB 10.3.4
- **Removed:** MariaDB 10.3.5

---

system_versioning_insert_history

- **Description:** Allows direct inserts into ROW_START and ROW_END columns if secure_timestamp allows changing timestamp.
- **Commandline:** `--system-versioning-insert-history[={0|1}]`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Type:** Boolean
- **Default Value:** `OFF`
- **Introduced:** MariaDB 10.11.0

Limitations

- Versioning clauses can not be applied to generated (virtual and persistent) columns.
- Before MariaDB 10.11, mariadb-dump did not read historical rows from versioned tables, and so historical data would not be backed up. Also, a restore of the timestamps would not be possible as they cannot be defined by an insert/a user. From MariaDB 10.11, use the `-H` or `--dump-history` options to include the history.

1.1.7.2 Application-Time Periods

Contents

MariaDB starting with 10.4.3

Support for application-time period-versioning was added in MariaDB 10.4.3.

Extending system-versioned tables, MariaDB 10.4 supports application-time period tables. Time periods are defined by a range between two temporal columns. The columns must be of the same temporal data type, i.e. DATE, TIMESTAMP or DATETIME (TIME and YEAR are not supported), and of the same width.

Using time periods implicitly defines the two columns as `NOT NULL`. It also adds a constraint to check whether the first value is less than the second value. The constraint is invisible to SHOW CREATE TABLE statements. The name of this constraint is prefixed by the time period name, to avoid conflict with other constraints.

Creating Tables with Time Periods

To create a table with a time period, use a CREATE TABLE statement with the `PERIOD` table option.

```
CREATE TABLE t1(
   name VARCHAR(50),
   date_1 DATE,
   date_2 DATE,
   PERIOD FOR date_period(date_1, date_2));
```

This creates a table with a `time_period` period and populates the table with some basic temporal values.

Examples are available in the MariaDB Server source code, at `mysql-test/suite/period/r/create.result`.

Adding and Removing Time Periods

The ALTER TABLE statement now supports syntax for adding and removing time periods from a table. To add a period, use the `ADD PERIOD` clause.

For example:

```
CREATE OR REPLACE TABLE rooms (
 room_number INT,
 guest_name VARCHAR(255),
 checkin DATE,
 checkout DATE
 );

ALTER TABLE rooms ADD PERIOD FOR p(checkin,checkout);
```

To remove a period, use the `DROP PERIOD` clause:

```
ALTER TABLE rooms DROP PERIOD FOR p;
```

Both `ADD PERIOD` and `DROP PERIOD` clauses include an option to handle whether the period already exists:

```
ALTER TABLE rooms ADD PERIOD IF NOT EXISTS FOR p(checkin,checkout);

ALTER TABLE rooms DROP PERIOD IF EXISTS FOR p;
```

Deletion by Portion

You can also remove rows that fall within certain time periods.

When MariaDB executes a `DELETE FOR PORTION` statement, it removes the row:

- When the row period falls completely within the delete period, it removes the row.
- When the row period overlaps the delete period, it shrinks the row, removing the overlap from the first or second row period value.
- When the delete period falls completely within the row period, it splits the row into two rows. The first row runs from the starting row period to the starting delete period. The second runs from the ending delete period to the ending row period.

To test this, first populate the table with some data to operate on:

```
CREATE TABLE t1(
   name VARCHAR(50),
   date_1 DATE,
   date_2 DATE,
   PERIOD FOR date_period(date_1, date_2));

INSERT INTO t1 (name, date_1, date_2) VALUES
    ('a', '1999-01-01', '2000-01-01'),
    ('b', '1999-01-01', '2018-12-12'),
    ('c', '1999-01-01', '2017-01-01'),
    ('d', '2017-01-01', '2019-01-01');

SELECT * FROM t1;
+------+------------+------------+
| name | date_1     | date_2     |
+------+------------+------------+
| a    | 1999-01-01 | 2000-01-01 |
| b    | 1999-01-01 | 2018-12-12 |
| c    | 1999-01-01 | 2017-01-01 |
| d    | 2017-01-01 | 2019-01-01 |
+------+------------+------------+
```

Then, run the `DELETE FOR PORTION` statement:

```
DELETE FROM t1
FOR PORTION OF date_period
    FROM '2001-01-01' TO '2018-01-01';
Query OK, 3 rows affected (0.028 sec)

SELECT * FROM t1 ORDER BY name;
+------+------------+------------+
| name | date_1     | date_2     |
+------+------------+------------+
| a    | 1999-01-01 | 2000-01-01 |
| b    | 1999-01-01 | 2001-01-01 |
| b    | 2018-01-01 | 2018-12-12 |
| c    | 1999-01-01 | 2001-01-01 |
| d    | 2018-01-01 | 2019-01-01 |
+------+------------+------------+
```

Here:

- *a* is unchanged, as the range falls entirely out of the specified portion to be deleted.
- *b*, with values ranging from 1999 to 2018, is split into two rows, 1999 to 2000 and 2018-01 to 2018-12.
- *c*, with values ranging from 1999 to 2017, where only the upper value falls within the portion to be deleted, has been shrunk to 1999 to 2001.
- *d*, with values ranging from 2017 to 2019, where only the lower value falls within the portion to be deleted, has been shrunk to 2018 to 2019.

The `DELETE FOR PORTION` statement has the following restrictions

- The `FROM...TO` clause must be constant
- Multi-delete is not supported

If there are `DELETE` or `INSERT` triggers, it works as follows: any matched row is deleted, and then one or two rows are inserted. If the record is deleted completely, nothing is inserted.

Updating by Portion

The UPDATE syntax now supports `UPDATE FOR PORTION`, which modifies rows based on their occurrence in a range:

To test it, first populate the table with some data:

```
TRUNCATE t1;

INSERT INTO t1 (name, date_1, date_2) VALUES
    ('a', '1999-01-01', '2000-01-01'),
    ('b', '1999-01-01', '2018-12-12'),
    ('c', '1999-01-01', '2017-01-01'),
    ('d', '2017-01-01', '2019-01-01');

SELECT * FROM t1;
+------+------------+------------+
| name | date_1     | date_2     |
+------+------------+------------+
| a    | 1999-01-01 | 2000-01-01 |
| b    | 1999-01-01 | 2018-12-12 |
| c    | 1999-01-01 | 2017-01-01 |
| d    | 2017-01-01 | 2019-01-01 |
+------+------------+------------+
```

Then run the update:

```
UPDATE t1 FOR PORTION OF date_period
  FROM '2000-01-01' TO '2018-01-01'
SET name = CONCAT(name,'_original');

SELECT * FROM t1 ORDER BY name;
+------------+------------+------------+
| name       | date_1     | date_2     |
+------------+------------+------------+
| a          | 1999-01-01 | 2000-01-01 |
| b          | 1999-01-01 | 2000-01-01 |
| b          | 2018-01-01 | 2018-12-12 |
| b_original | 2000-01-01 | 2018-01-01 |
| c          | 1999-01-01 | 2000-01-01 |
| c_original | 2000-01-01 | 2017-01-01 |
| d          | 2018-01-01 | 2019-01-01 |
| d_original | 2017-01-01 | 2018-01-01 |
+------------+------------+------------+
```

- *a* is unchanged, as the range falls entirely out of the specified portion to be deleted.
- *b*, with values ranging from 1999 to 2018, is split into two rows, 1999 to 2000 and 2018-01 to 2018-12.
- *c*, with values ranging from 1999 to 2017, where only the upper value falls within the portion to be deleted, has been shrunk to 1999 to 2001.
- *d*, with values ranging from 2017 to 2019, where only the lower value falls within the portion to be deleted, has been shrunk to 2018 to 2019.
- Original rows affected by the update have "_original" appended to the name.

The `UPDATE FOR PORTION` statement has the following limitations:

- The operation cannot modify the two temporal columns used by the time period
- The operation cannot reference period values in the `SET` expression
- `FROM...TO` expressions must be constant

WITHOUT OVERLAPS

MariaDB starting with 10.5.3

MariaDB 10.5 introduced a new clause, `WITHOUT OVERLAPS`, which allows one to create an index specifying that application time periods should not overlap.

An index constrained by `WITHOUT OVERLAPS` is required to be either a primary key or a unique index.

Take the following example, an application time period table for a booking system:

```
CREATE OR REPLACE TABLE rooms (
 room_number INT,
 guest_name VARCHAR(255),
 checkin DATE,
 checkout DATE,
 PERIOD FOR p(checkin,checkout)
 );

INSERT INTO rooms VALUES
 (1, 'Regina', '2020-10-01', '2020-10-03'),
 (2, 'Cochise', '2020-10-02', '2020-10-05'),
 (1, 'Nowell', '2020-10-03', '2020-10-07'),
 (2, 'Eusebius', '2020-10-04', '2020-10-06');
```

Our system is not intended to permit overlapping bookings, so the fourth record above should not have been inserted. Using `WITHOUT OVERLAPS` in a unique index (in this case based on a combination of room number and the application time period) allows us to specify this constraint in the table definition.

```
CREATE OR REPLACE TABLE rooms (
 room_number INT,
 guest_name VARCHAR(255),
 checkin DATE,
 checkout DATE,
 PERIOD FOR p(checkin,checkout),
 UNIQUE (room_number, p WITHOUT OVERLAPS)
 );

INSERT INTO rooms VALUES
 (1, 'Regina', '2020-10-01', '2020-10-03'),
 (2, 'Cochise', '2020-10-02', '2020-10-05'),
 (1, 'Nowell', '2020-10-03', '2020-10-07'),
 (2, 'Eusebius', '2020-10-04', '2020-10-06');
ERROR 1062 (23000): Duplicate entry '2-2020-10-06-2020-10-04' for key 'room_number'
```

Further Examples

The implicit change from NULL to NOT NULL:

```
CREATE TABLE `t2` (
  `id` int(11) DEFAULT NULL,
  `d1` datetime DEFAULT NULL,
  `d2` datetime DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

ALTER TABLE t2 ADD PERIOD FOR p(d1,d2);

SHOW CREATE TABLE t2\G
*************************** 1. row ***************************
       Table: t2
Create Table: CREATE TABLE `t2` (
  `id` int(11) DEFAULT NULL,
  `d1` datetime NOT NULL,
  `d2` datetime NOT NULL,
  PERIOD FOR `p` (`d1`, `d2`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

Due to this constraint, trying to add a time period where null data already exists will fail.

```
CREATE OR REPLACE TABLE `t2` (
  `id` int(11) DEFAULT NULL,
  `d1` datetime DEFAULT NULL,
  `d2` datetime DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

INSERT INTO t2(id) VALUES(1);

ALTER TABLE t2 ADD PERIOD FOR p(d1,d2);
ERROR 1265 (01000): Data truncated for column 'd1' at row 1
```

1.1.7.3 Bitemporal Tables

MariaDB starting with 10.4.3

Bitemporal tables are tables that use versioning both at the system and application-time period levels.

Contents

Using Bitemporal Tables

To create a bitemporal table, use:

```
CREATE TABLE test.t3 (
   date_1 DATE,
   date_2 DATE,
   row_start TIMESTAMP(6) AS ROW START INVISIBLE,
   row_end TIMESTAMP(6) AS ROW END INVISIBLE,
   PERIOD FOR application_time(date_1, date_2),
   PERIOD FOR system_time(row_start, row_end))
WITH SYSTEM VERSIONING;
```

Note that, while `system_time` here is also a time period, it cannot be used in `DELETE FOR PORTION` or `UPDATE FOR PORTION` statements.

```
DELETE FROM test.t3
FOR PORTION OF system_time
    FROM '2000-01-01' TO '2018-01-01';
ERROR 42000: You have an error in your SQL syntax; check the manual that corresponds
  to your MariaDB server version for the right syntax to use near
  'of system_time from '2000-01-01' to '2018-01-01'' at line 1
```

1.2 Built-in Functions

Functions and procedures in MariaDB

Function and Operator Reference

A complete list of MariaDB functions and operators in alphabetical order.

String Functions

Built-In functions for the handling of strings and columns containing string values.

Date & Time Functions

Built-In functions for the handling of dates and times.

Aggregate Functions

Aggregate functions used with GROUP BY clauses.

Numeric Functions

Numeric and arithmetic related functions in MariaDB.

Control Flow Functions

Built-In functions for assessing data to determine what results to return.

Pseudo Columns

MariaDB has pseudo columns that can be used for different purposes.

Secondary Functions

Bit Functions and Operators

Operators for comparison and setting of values, and related functions.

Encryption, Hashing and Compression Functions

Functions used for encryption, hashing and compression.

Information Functions

Functions which return information on the server, the user, or a given query.

Miscellaneous Functions

Functions for very singular and specific needs.

Special Functions

Dynamic Columns Functions

Functions for storing key/value pairs of data within a column.

Galera Functions

Built-in functions related to Galera.

Geographic Functions

Geographic, as well as geometric functions.

JSON Functions

Built-in functions related to JSON.

SEQUENCE Functions

Functions that can be used on SEQUENCEs.

Spider Functions

User-defined functions available with the Spider storage engine.

Window Functions

Window functions for performing calculations on a set of rows related to the current row.

There are 11 related questions.

1.2.1 Function and Operator Reference

Name	Description	Added
+	Addition operator	
/	Division operator	
*	Multiplication operator	
%	Modulo operator. Returns the remainder of N divided by M	
-	Subtraction operator	
!=	Not equals	
<	Less than	
<=	Less than or equal	
<=>	NULL-safe equal	
=	Equal	
>	Greater than	
>=	Greater than or equal	
&	Bitwise AND	
<<	Shift left	
>>	Shift right	
^	Bitwise XOR	
!	Logical NOT	
&&	Logical AND	
XOR	Logical XOR	
||	Logical OR	
|	Bitwise OR	
:=	Assignment operator	
=	Assignment and comparison operator	
~	Bitwise NOT	
ABS	Returns an absolute value	
ACOS	Returns an arc cosine	

ADD_MONTHS	Add months to a date	
ADDDATE	Add days or another interval to a date	
ADDTIME	Adds a time to a time or datetime	
AES_DECRYPT	Decryption data encrypted with AES_ENCRYPT	
AES_ENCRYPT	Encrypts a string with the AES algorithm	
AREA	Synonym for ST_AREA	
AsBinary	Synonym for ST_AsBinary	
ASCII	Numeric ASCII value of leftmost character	
ASIN	Returns the arc sine	
AsText	Synonym for ST_AsText	
AsWKB	Synonym for ST_AsBinary	
AsWKT	Synonym for ST_AsText	
ATAN	Returns the arc tangent	
ATAN2	Returns the arc tangent of two variables	
AVG	Returns the average value	
BENCHMARK	Executes an expression repeatedly	
BETWEEN AND	True if expression between two values	
BIN	Returns binary value	
BINARY OPERATOR	Casts to a binary string	
BINLOG_GTID_POS	Returns a string representation of the corresponding GTID position	
BIT_AND	Bitwise AND	
BIT_COUNT	Returns the number of set bits	
BIT_LENGTH	Returns the length of a string in bits	
BIT_OR	Bitwise OR	
BIT_XOR	Bitwise XOR	
BOUNDARY	Synonym for ST_BOUNDARY	
BUFFER	Synonym for ST_BUFFER	
CASE	Returns the result where value=compare_value or for the first condition that is true	
CAST	Casts a value of one type to another type	
CEIL	Synonym for CEILING()	
CEILING	Returns the smallest integer not less than X	
CENTROID	Synonym for ST_CENTROID	
CHAR Function	Returns string based on the integer values for the individual characters	
CHARACTER_LENGTH	Synonym for CHAR_LENGTH()	
CHAR_LENGTH	Length of the string in characters	
CHARSET	Returns the character set	
CHR	Returns a string consisting of the character given by the code values of the integer	

COALESCE	Returns the first non-NULL parameter	
COERCIBILITY	Returns the collation coercibility value	
COLLATION	Collation of the string argument	
COLUMN_ADD	Adds or updates dynamic columns	
COLUMN_CHECK	Checks if a dynamic column blob is valid	
COLUMN_CREATE	Returns a dynamic columns blob	
COLUMN_DELETE	Deletes a dynamic column	
COLUMN_EXISTS	Checks is a column exists	
COLUMN_GET	Gets a dynamic column value by name	
COLUMN_JSON	Returns a JSON representation of dynamic column blob data	
COLUMN_LIST	Returns comma-separated list	
COMPRESS	Returns a binary, compressed string	
CONCAT	Returns concatenated string	
CONCAT_WS	Concatenate with separator	
CONNECTION_ID	Connection thread ID	
CONTAINS	Whether one geometry contains another	
CONVERT	Convert a value from one type to another type	
CONV	Converts numbers between different number bases	
CONVERT_TZ	Converts a datetime from on time zone to another	
CONVEXHULL	Synonym for ST_CONVEXHULL	
COS	Returns the cosine	
COT	Returns the cotangent	
COUNT	Returns count of non-null values	
COUNT DISTINCT	Returns count of number of different non-NULL values	
CRC32	Computes a cyclic redundancy check value	
CRC32C	Computes a cyclic redundancy check value	MariaDB 10.8
CROSSES	Whether two geometries spatially cross	
CUME_DIST	Window function that returns the cumulative distribution of a given row	
CURDATE	Returns the current date	
CURRENT_DATE	Synonym for CURDATE()	
CURRENT_ROLE	Current role name	
CURRENT_TIME	Synonym for CURTIME()	
CURRENT_TIMESTAMP	Synonym for NOW()	
CURRENT_USER	Username/host that authenicated the current client	
CURTIME	Returns the current time	
DATABASE	Current default database	

DATE FUNCTION	Extracts the date portion of a datetime	
DATEDIFF	Difference in days between two date/time values	
DATE_ADD	Date arithmetic - addition	
DATE_FORMAT	Formats the date value according to the format string	
DATE_SUB	Date arithmetic - subtraction	
DAY	Synonym for DAYOFMONTH()	
DAYNAME	Return the name of the weekday	
DAYOFMONTH	Returns the day of the month	
DAYOFWEEK	Returns the day of the week index	
DAYOFYEAR	Returns the day of the year	
DECODE	Decrypts a string encoded with ENCODE()	
DECODE_HISTOGRAM	Returns comma separated numerics corresponding to a probability distribution represented by a histogram	
DEFAULT	Returns column default	
DEGREES	Converts from radians to degrees	
DENSE_RANK	Rank of a given row with identical values receiving the same result, no skipping	
DES_DECRYPT	Decrypts a string encrypted with DES_ENCRYPT()	
DES_ENCRYPT	Encrypts a string using the Triple-DES algorithm	
DIMENSION	Synonym for ST_DIMENSION	
DISJOINT	Whether the two elements do not intersect	
DIV	Integer division	
ELT	Returns the N'th element from a set of strings	
ENCODE	Encrypts a string	
ENCRYPT	Encrypts a string with Unix crypt()	
ENDPOINT	Synonym for ST_ENDPOINT	
ENVELOPE	Synonym for ST_ENVELOPE	
EQUALS	Indicates whether two geometries are spatially equal	
EXP	e raised to the power of the argument	
EXPORT_SET	Returns an on string for every bit set, an off string for every bit not set	
ExteriorRing	Synonym for ST_ExteriorRing	
EXTRACT	Extracts a portion of the date	
EXTRACTVALUE	Returns the text of the first text node matched by the XPath expression	
FIELD	Returns the index position of a string in a list	
FIND_IN_SET	Returns the position of a string in a set of strings	

FLOOR	Largest integer value not greater than the argument	
FORMAT	Formats a number	
FORMAT_PICO_TIME	Given a time in picoseconds, returns a human-readable time value and unit indicator	MariaDB 11.0.2
FOUND_ROWS	Number of (potentially) returned rows	
FROM_BASE64	Given a base-64 encoded string, returns the decoded result as a binary string	
FROM_DAYS	Returns a date given a day	
FROM_UNIXTIME	Returns a datetime from a Unix timestamp	
GeomCollFromText	Synonym for ST_GeomCollFromText	
GeomCollFromWKB	Synonym for ST_GeomCollFromWKB	
GeometryCollectionFromText	Synonym for ST_GeomCollFromText	
GeometryCollectionFromWKB	Synonym for ST_GeomCollFromWKB	
GeometryFromText	Synonym for ST_GeomFromText	
GeometryFromWKB	Synonym for ST_GeomFromWKB	
GeomFromText	Synonym for ST_GeomFromText	
GeomFromWKB	Synonym for ST_GeomFromWKB	
GeometryN	Synonym for ST_GeometryN	
GEOMETRYCOLLECTION	Constructs a WKB GeometryCollection	
GeometryType	Synonym for ST_GeometryType	
GET_FORMAT	Returns a format string	
GET_LOCK	Obtain LOCK	
GLENGTH	Length of a LineString value	
GREATEST	Returns the largest argument	
GROUP_CONCAT	Returns string with concatenated values from a group	
HEX	Returns hexadecimal value	
HOUR	Returns the hour	
IF	If expr1 is TRUE, returns expr2; otherwise returns expr3	
IFNULL	Check whether an expression is NULL	
IN	True if expression equals any of the values in the list	
INTERVAL	Index of the argument that is less than the first argument	
INET6_ATON	Given an IPv6 or IPv4 network address, returns a VARBINARY numeric value	
INET6_NTOA	Given an IPv6 or IPv4 network address, returns the address as a nonbinary string	
INET_ATON	Returns numeric value of IPv4 address	

INET_NTOA	Returns dotted-quad representation of IPv4 address	
INSERT Function	Replaces a part of a string with another string	
INSTR	Returns the position of a string withing a string	
InteriorRingN	Synonym for ST_InteriorRingN	
INTERSECTS	Indicates whether two geometries spatially intersect	
IS	Tests whether a boolean is TRUE, FALSE, or UNKNOWN	
IsClosed	Synonym for ST_IsClosed	
IsEmpty	Synonym for ST_IsEmpty	
IS_FREE_LOCK	Checks whether lock is free to use	
IS_IPV4	Whether or not an expression is a valid IPv4 address	
IS_IPV4_COMPAT	Whether or not an IPv6 address is IPv4-compatible	
IS_IPV4_MAPPED	Whether an IPv6 address is a valid IPv4-mapped address	
IS_IPV6	Whether or not an expression is a valid IPv6 address	
IS NOT	Tests whether a boolean value is not TRUE, FALSE, or UNKNOWN	
IS NOT NULL	Tests whether a value is not NULL	
IS NULL	Tests whether a value is NULL	
ISNULL	Checks if an expression is NULL	
IsRing	Synonym for ST_IsRing	
IsSimple	Synonym for ST_IsSimple	
IS_USED_LOCK	Check if lock is in use	
JSON_ARRAY	Returns a JSON array containing the listed values	
JSON_ARRAY_INTERSECT		MariaDB 11.2.0
JSON_ARRAY_APPEND	Appends values to the end of the given arrays within a JSON document	
JSON_ARRAY_INSERT	Inserts a value into a JSON document	
JSON_COMPACT	Removes all unnecessary spaces so the json document is as short as possible	
JSON_CONTAINS	Whether a value is found in a given JSON document or at a specified path within the document	
JSON_CONTAINS_PATH	Indicates whether the given JSON document contains data at the specified path or paths	
JSON_DEPTH	Maximum depth of a JSON document	
JSON_DETAILED	Represents JSON in the most understandable way emphasizing nested structures	
JSON_EQUALS	Check for equality between JSON objects.	

JSON_EXISTS	Determines whether a specified JSON value exists in the given data	
JSON_EXTRACT	Extracts data from a JSON document.	
JSON_INSERT	Inserts data into a JSON document	
JSON_KEYS	Returns keys from top-level value of a JSON object or top-level keys from the path	
JSON_LENGTH	Returns the length of a JSON document, or the length of a value within the document	
JSON_LOOSE	Adds spaces to a JSON document to make it look more readable	
JSON_MERGE	Merges the given JSON documents	
JSON_MERGE_PATCH	RFC 7396-compliant merge of the given JSON documents	
JSON_MERGE_PRESERVE	Synonym for JSON_MERGE_PATCH.	
JSON_NORMALIZE	Recursively sorts keys and removes spaces, allowing comparison of json documents for equality	
JSON_OBJECT	Returns a JSON object containing the given key/value pairs	
JSON_OBJECT_FILTER_KEYS		MariaDB 11.2.0
JSON_OBJECT_TO_ARRAY		MariaDB 11.2.0
JSON_OBJECTAGG	Returns a JSON object containing key-value pairs	
JSON_OVERLAPS	Compares two json documents for overlaps	
JSON_PRETTY	Alias for json_detailed	MariaDB 10.10.3, MariaDB 10.9.5, MariaDB 10.8.7, MariaDB 10.7.8, MariaDB 10.6.12, MariaDB 10.5.19 and MariaDB 10.4.28
JSON_QUERY	Given a JSON document, returns an object or array specified by the path	
JSON_QUOTE	Quotes a string as a JSON value	
JSON_REMOVE	Removes data from a JSON document	
JSON_REPLACE	Replaces existing values in a JSON document	
JSON_SCHEMA_VALID	Validates a JSON schema	MariaDB 11.1.0
JSON_SEARCH	Returns the path to the given string within a JSON document	
JSON_SET	Updates or inserts data into a JSON document	
JSON_TABLE	Returns a representation of a JSON document as a relational table	
JSON_TYPE	Returns the type of a JSON value	
JSON_UNQUOTE	Unquotes a JSON value, returning a string	
JSON_VALID	Whether a value is a valid JSON document or not	
JSON_VALUE	Given a JSON document, returns the specified scalar	
KDF	Key derivation function	MariaDB 11.3.0

LAST_DAY	Returns the last day of the month	
LAST_INSERT_ID	Last inserted autoinc value	
LAST_VALUE	Returns the last value in a list	
LASTVAL	Get last value generated from a sequence	
LCASE	Synonym for [LOWER()	
LEAST	Returns the smallest argument	
LEFT	Returns the leftmost characters from a string	
LENGTH	Length of the string in bytes	
LIKE	Whether expression matches a pattern	
LineFromText	Synonym for ST_LineFromText	
LineFromWKB	Synonym for ST_LineFromWKB	
LINESTRING	Constructs a WKB LineString value from a number of WKB Point arguments	
LineStringFromText	Synonym for ST_LineFromText	
LineStringFromWKB	Synonym for ST_LineFromWKB	
LN	Returns natural logarithm	
LOAD_FILE	Returns file contents as a string	
LOCALTIME	Synonym for NOW()	
LOCALTIMESTAMP	Synonym for NOW()	
LOCATE	Returns the position of a substring in a string	
LOG	Returns the natural logarithm	
LOG10	Returns the base-10 logarithm	
LOG2	Returns the base-2 logarithm	
LOWER	Returns a string with all characters changed to lowercase	
LPAD	Returns the string left-padded with another string to a given length	
LTRIM	Returns the string with leading space characters removed	
MAKE_SET	Make a set of strings that matches a bitmask	
MAKEDATE	Returns a date given a year and day	
MAKETIME	Returns a time	
MASTER_GTID_WAIT	Wait until slave reaches the GTID position	
MASTER_POS_WAIT	Blocks until the slave has applied all specified updates	
MATCH AGAINST	Perform a fulltext search on a fulltext index	
MAX	Returns the maximum value	
MBRContains	Indicates one Minimum Bounding Rectangle contains another	
MBRDisjoint	Indicates whether the Minimum Bounding Rectangles of two geometries are disjoint	
MBREqual	Whether the Minimum Bounding Rectangles of two geometries are the same.	

MBRIntersects	Indicates whether the Minimum Bounding Rectangles of the two geometries intersect	
MBROverlaps	Whether the Minimum Bounding Rectangles of two geometries overlap	
MBRTouches	Whether the Minimum Bounding Rectangles of two geometries touch.	
MBRWithin	Indicates whether one Minimum Bounding Rectangle is within another	
MD5	MD5 checksum	
MEDIAN	Window function that returns the median value of a range of values	
MICROSECOND	Returns microseconds from a date or datetime	
MID	Synonym for SUBSTRING(str,pos,len)	
MIN	Returns the minimum value	
MINUTE	Returns a minute from 0 to 59	
MLineFromText	Constructs MULTILINESTRING using its WKT representation and SRID	
MLineFromWKB	Constructs a MULTILINESTRING	
MOD	Modulo operation. Remainder of N divided by M	
MONTH	Returns a month from 1 to 12	
MONTHNAME	Returns the full name of the month	
MPointFromText	Constructs a MULTIPOINT value using its WKT and SRID	
MPointFromWKB	Constructs a MULTIPOINT value using its WKB representation and SRID	
MPolyFromText	Constructs a MULTIPOLYGON value	
MPolyFromWKB	Constructs a MULTIPOLYGON value using its WKB representation and SRID	
MultiLineStringFromText	Synonym for MLineFromText	
MultiLineStringFromWKB	A synonym for MLineFromWKB	
MULTIPOINT	Constructs a WKB MultiPoint value	
MultiPointFromText	Synonym for MPointFromText	
MultiPointFromWKB	Synonym for MPointFromWKB	
MULTIPOLYGON	Constructs a WKB MultiPolygon	
MultiPolygonFromText	Synonym for MPolyFromText	
MultiPolygonFromWKB	Synonym for MPolyFromWKB	
MULTILINESTRING	Constructs a MultiLineString value	
NAME_CONST	Returns the given value	
NATURAL_SORT_KEY	Sorting that is more more similar to natural human sorting	
NOT LIKE	Same as NOT(expr LIKE pat [ESCAPE 'escape_char'])	
NOT REGEXP	Same as NOT (expr REGEXP pat)	
NULLIF	Returns NULL if expr1 = expr2	

NEXTVAL	Generate next value for sequence	
NOT BETWEEN	Same as NOT (expr BETWEEN min AND max)	
NOT IN	Same as NOT (expr IN (value,...))	
NOW	Returns the current date and time	
NTILE	Returns an integer indicating which group a given row falls into	
NumGeometries	Synonym for ST_NumGeometries	
NumInteriorRings	Synonym for NumInteriorRings	
NumPoints	Synonym for ST_NumPoints	
OCT	Returns octal value	
OCTET_LENGTH	Synonym for LENGTH()	
OLD_PASSWORD	Pre MySQL 4.1 password implementation	
ORD	Return ASCII or character code	
OVERLAPS	Indicates whether two elements spatially overlap	
PASSWORD	Calculates a password string	
PERCENT_RANK	Window function that returns the relative percent rank of a given row	
PERCENTILE_CONT	Returns a value which corresponds to the given fraction in the sort order.	
PERCENTILE_DISC	Returns the first value in the set whose ordered position is the same or more than the specified fraction.	
PERIOD_ADD	Add months to a period	
PERIOD_DIFF	Number of months between two periods	
PI	Returns the value of π (pi)	
POINT	Constructs a WKB Point	
PointFromText	Synonym for ST_PointFromText	
PointFromWKB	Synonym for PointFromWKB	
PointN	Synonym for PointN	
PointOnSurface	Synonym for ST_PointOnSurface	
POLYGON	Constructs a WKB Polygon value from a number of WKB LineString arguments	
PolyFromText	Synonym for ST_PolyFromText	
PolyFromWKB	Synonym for ST_PolyFromWKB	
PolygonFromText	Synonym for ST_PolyFromText	
PolygonFromWKB	Synonym for ST_PolyFromWKB	
POSITION	Returns the position of a substring in a string	
POW	Returns X raised to the power of Y	
POWER	Synonym for POW()	
QUARTER	Returns year quarter from 1 to 4	
QUOTE	Returns quoted, properly escaped string	
RADIANS	Converts from degrees to radians	
RAND	Random floating-point value	

RANK	Rank of a given row with identical values receiving the same result	
REGEXP	Performs pattern matching	
REGEXP_INSTR	Position of the first appearance of a regex	
REGEXP_REPLACE	Replaces all occurrences of a pattern	
REGEXP_SUBSTR	Returns the matching part of a string	
RELEASE_LOCK	Releases lock obtained with GET_LOCK()	
REPEAT Function	Returns a string repeated a number of times	
REPLACE Function	Replace occurrences of a string	
REVERSE	Reverses the order of a string	
RIGHT	Returns the rightmost N characters from a string	
RLIKE	Synonym for REGEXP()	
RPAD	Returns the string right-padded with another string to a given length	
ROUND	Rounds a number	
ROW_COUNT	Number of rows affected by previous statement	
ROW_NUMBER	Row number of a given row with identical values receiving a different result	
RTRIM	Returns the string with trailing space characters removed	
SCHEMA	Synonym for DATABASE()	
SECOND	Returns the second of a time	
SEC_TO_TIME	Converts a second to a time	
SETVAL	Set the next value to be returned by a sequence	
SESSION_USER	Synonym for USER()	
SHA	Synonym for SHA1()	
SHA1	Calculates an SHA-1 checksum	
SHA2	Calculates an SHA-2 checksum	
SIGN	Returns 1, 0 or -1	
SIN	Returns the sine	
SLEEP	Pauses for the given number of seconds	
SOUNDEX	Returns a string based on how the string sounds	
SOUNDS LIKE	SOUNDEX(expr1) = SOUNDEX(expr2)	
SPACE	Returns a string of space characters	
SPIDER_BG_DIRECT_SQL	Background SQL execution	
SPIDER_COPY_TABLES	Copy table data	
SPIDER_DIRECT_SQL	Execute SQL on the remote server	
SPIDER_FLUSH_TABLE_MON_CACHE	Refreshing Spider monitoring server information	
SQRT	Square root	
SRID	Synonym for ST_SRID	

ST_AREA	Area of a Polygon	
ST_AsBinary	Converts a value to its WKB representation	
ST_AsText	Converts a value to its WKT-Definition	
ST_AsWKB	Synonym for ST_AsBinary	
ST_ASWKT	Synonym for ST_ASTEXT()	
ST_BOUNDARY	Returns a geometry that is the closure of a combinatorial boundary	
ST_BUFFER	A new geometry with a buffer added to the original geometry	
ST_CENTROID	The mathematical centroid (geometric center) for a MultiPolygon	
ST_CONTAINS	Whether one geometry is contained by another	
ST_CONVEXHULL	The minimum convex geometry enclosing all geometries within the set	
ST_CROSSES	Whether two geometries spatially cross	
ST_DIFFERENCE	Point set difference	
ST_DIMENSION	Inherent dimension of a geometry value	
ST_DISJOINT	Whether one geometry is spatially disjoint from another	
ST_DISTANCE	The distance between two geometries	
ST_DISTANCE_SPHERE	The spherical distance between two geometries	
ST_ENDPOINT	Returns the endpoint of a LineString	
ST_ENVELOPE	Returns the Minimum Bounding Rectangle for a geometry value	
ST_EQUALS	Whether two geometries are spatoially equal	
ST_ExteriorRing	Returns the exterior ring of a Polygon as a LineString	
ST_GeomCollFromText	Constructs a GEOMETRYCOLLECTION value	
ST_GeomCollFromWKB	Constructs a GEOMETRYCOLLECTION value from a WKB	
ST_GeometryCollectionFromText	Synonym for ST_GeomCollFromText	
ST_GeometryCollectionFromWKB	Synonym for ST_GeomCollFromWKB	
ST_GeometryFromText	Synonym for ST_GeomFromText	
ST_GeometryFromWKB	Synonym for ST_GeomFromWKB	
ST_GEOMETRYN	Returns the N-th geometry in a GeometryCollection	
ST_GEOMETRYTYPE	Returns name of the geometry type of which a given geometry instance is a member	
ST_GeomFromText	Constructs a geometry value using its WKT and SRID	

ST_GeomFromWKB	Constructs a geometry value using its WKB representation and SRID	
ST_InteriorRingN	Returns the N-th interior ring for a Polygon	
ST_INTERSECTION	The intersection, or shared portion, of two geometries	
ST_INTERSECTS	Whether two geometries spatially intersect	
ST_ISCLOSED	Returns true if a given LINESTRING's start and end points are the same	
ST_ISEMPTY	Indicated validity of geometry value	
ST_IsRing	Returns true if a given LINESTRING is both ST_IsClosed and ST_IsSimple	
ST_IsSimple	Returns true if the given Geometry has no anomalous geometric points	
ST_LENGTH	Length of a LineString value	
ST_LineFromText	Creates a linestring value	
ST_LineFromWKB	Constructs a LINESTRING using its WKB and SRID	
ST_LineStringFromText	Synonym for ST_LineFromText	
ST_LineStringFromWKB	Synonym for ST_LineFromWKB	
ST_NUMGEOMETRIES	Number of geometries in a GeometryCollection	
ST_NumInteriorRings	Number of interior rings in a Polygon	
ST_NUMPOINTS	Returns the number of Point objects in a LineString	
ST_OVERLAPS	Whether two geometries overlap	
ST_PointFromText	Constructs a POINT value	
ST_PointFromWKB	Constructs POINT using its WKB and SRID	
ST_POINTN	Returns the N-th Point in the LineString	
ST_POINTONSURFACE	Returns a POINT guaranteed to intersect a surface	
ST_PolyFromText	Constructs a POLYGON value	
ST_PolyFromWKB	Constructs POLYGON value using its WKB representation and SRID	
ST_PolygonFromText	Synonym for ST_PolyFromText	
ST_PolygonFromWKB	Synonym for ST_PolyFromWKB	
ST_RELATE	Returns true if two geometries are related	
ST_SRID	Returns a Spatial Reference System ID	
ST_STARTPOINT	Returns the start point of a LineString	
ST_SYMDIFFERENCE	Portions of two geometries that don't intersect	
ST_TOUCHES	Whether one geometry g1 spatially touches another	
ST_UNION	Union of two geometries	
ST_WITHIN	Whether one geometry is within another	

ST_X	X-coordinate value for a point	
ST_Y	Y-coordinate for a point	
STARTPOINT	Synonym for ST_StartPoint	
STD	Population standard deviation	
STDDEV	Population standard deviation	
STDDEV_POP	Returns the population standard deviation	
STDDEV_SAMP	Standard deviation	
STR_TO_DATE	Converts a string to date	
STRCMP	Compares two strings in sort order	
SUBDATE	Subtract a date unit or number of days	
SUBSTR	Returns a substring from string starting at a given position	
SUBSTRING	Returns a substring from string starting at a given position	
SUBSTRING_INDEX	Returns the substring from string before count occurrences of a delimiter	
SUBTIME	Subtracts a time from a date/time	
SUM	Sum total	
SYS.EXTRACT_SCHEMA_FROM_FILE_NAME	Given a file path, returns the schema (database) name	MariaDB 10.6
SYS.EXTRACT_TABLE_FROM_FILE_NAME	Given a file path, returns the table name	MariaDB 10.6
SYS.FORMAT_BYTES	Returns a string consisting of a value and the units in a human-readable format	MariaDB 10.6
SYS.FORMAT_PATH	Returns a modified path after replacing subpaths matching the values of various system variables with the variable name	MariaDB 10.6
SYS.FORMAT_STATEMENT	Returns a reduced length string	MariaDB 10.6
SYS.FORMAT_TIME	Returns a human-readable time value and unit indicator	MariaDB 10.6
SYS.LIST_ADD	Adds a value to a given list	MariaDB 10.6
SYS.LIST_DROP	Drops a value from a given list	MariaDB 10.6
SYS.PS_IS_ACCOUNT_ENABLED	Whether Performance Schema instrumentation for the given account is enabled	MariaDB 10.6
SYS.PS_IS_CONSUMER_ENABLED	Whether Performance Schema instrumentation for the given consumer is enabled	MariaDB 10.6
SYS.PS_IS_INSTRUMENT_DEFAULT_ENABLED	Whether a given Performance Schema instrument is enabled by default	MariaDB 10.6
SYS.PS_IS_INSTRUMENT_DEFAULT_TIMED	Returns whether a given Performance Schema instrument is timed by default	MariaDB 10.6
SYS.PS_IS_THREAD_INSTRUMENTED	Returns whether or not Performance Schema instrumentation for the given connection_id is enabled	MariaDB 10.6
SYS.PS_THREAD_ACCOUNT	Returns the account (username@hostname) associated with the given thread_id	MariaDB 10.6

SYS.PS_THREAD_ID	Returns the thread_id associated with the given connection_id	MariaDB 10.6
SYS.PS_THREAD_STACK	Returns all statements, stages, and events within the Performance Schema for a given thread_id	MariaDB 10.6
SYS.PS_THREAD_TRX_INFO	Returns a JSON object with information about the thread specified by the given thread_id	MariaDB 10.6
SYS.QUOTE_IDENTIFIER	Quotes a string to produce a result that can be used as an identifier in an SQL statement	MariaDB 10.6
SYS.SYS_GET_CONFIG	Returns a configuration option value from the sys_config table	MariaDB 10.6
SYS.VERSION_MAJOR	Returns the MariaDB Server major release version	MariaDB 10.6
SYS.VERSION_MINOR	Returns the MariaDB Server minor release version	MariaDB 10.6
SYS.VERSION_PATCH	Returns the MariaDB Server patch release version	MariaDB 10.6
SYS_GUID	Generates a globally unique identifier	
SYSDATE	Returns the current date and time	
SYSTEM_USER	Synonym for USER()	
TAN	Returns the tangent	
TIME function	Extracts the time	
TIMEDIFF	Returns the difference between two date/times	
TIMESTAMP FUNCTION	Return the datetime, or add a time to a date/time	
TIMESTAMPADD	Add interval to a date or datetime	
TIMESTAMPDIFF	Difference between two datetimes	
TIME_FORMAT	Formats the time value according to the format string	
TIME_TO_SEC	Returns the time argument, converted to seconds	
TO_BASE64	Converts a string to its base-64 encoded form	
TO_CHAR	Converts a date/time type to a char	
TO_DAYS	Number of days since year 0	
TO_SECONDS	Number of seconds since year 0	
TOUCHES	Whether two geometries spatially touch	
TRIM	Returns a string with all given prefixes or suffixes removed	
TRUNCATE	Truncates X to D decimal places	
UCASE	Synonym for UPPER]]()	
UNHEX	Interprets pairs of hex digits as a number and converts to the character represented by the number	
UNCOMPRESS	Uncompresses string compressed with COMPRESS()	
UNCOMPRESSED_LENGTH	Returns length of a string before being compressed with COMPRESS()	

UNIX_TIMESTAMP	Returns a Unix timestamp	
UPDATEXML	Replace XML	
UPPER	Changes string to uppercase	
USER	Current user/host	
UTC_DATE	Returns the current UTC date	
UTC_TIME	Returns the current UTC time	
UTC_TIMESTAMP	Returns the current UTC date and time	
UUID	Returns a Universal Unique Identifier	
UUID_SHORT	Return short universal identifier	
VALUES or VALUE	Refer to columns in INSERT ... ON DUPLICATE KEY UPDATE	
VAR_POP	Population standard variance	
VAR_SAMP	Returns the sample variance	
VARIANCE	Population standard variance	
VERSION	MariaDB server version	
WEEK	Returns the week number	
WEEKDAY	Returns the weekday index	
WEEKOFYEAR	Returns the calendar week of the date as a number in the range from 1 to 53	
WEIGHT_STRING	Weight of the input string	
WITHIN	Indicate whether a geographic element is spacially within another	
WSREP_LAST_SEEN_GTID	Returns the Global Transaction ID of the most recent write transaction observed by the client.	
WSREP_LAST_WRITTEN_GTID	Returns the Global Transaction ID of the most recent write transaction performed by the client.	
WSREP_SYNC_WAIT_UPTO_GTID	Blocks the client until the transaction specified by the given Global Transaction ID is applied and committed by the node	
X	Synonym for ST_X	
Y	Synonym for ST_Y	
YEAR	Returns the year for the given date	
YEARWEEK	Returns year and week for a date	

1.2.2 String Functions

Functions dealing with strings, such as CHAR, CONVERT, CONCAT, PAD, REGEXP, TRIM, etc.

Regular Expressions Functions

Functions for dealing with regular expressions

Dynamic Columns Functions

Functions for storing key/value pairs of data within a column.

ASCII

Numeric ASCII value of leftmost character.

BIN

Returns binary value.

BINARY Operator

Casts to a binary string.

BIT_LENGTH

Returns the length of a string in bits.

CAST

Casts a value of one type to another type.

CHAR Function

Returns string based on the integer values for the individual characters.

CHARACTER_LENGTH

Synonym for CHAR_LENGTH().

CHAR_LENGTH

Length of the string in characters.

CHR

Returns string based on integer values of the individual characters.

CONCAT

Returns concatenated string.

CONCAT_WS

Concatenate with separator.

CONVERT

Convert a value from one type to another type.

ELT

Returns the N'th element from a set of strings.

EXPORT_SET

Returns an on string for every bit set, an off string for every bit not set.

EXTRACTVALUE

Returns the text of the first text node matched by the XPath expression.

FIELD

Returns the index position of a string in a list.

FIND_IN_SET

Returns the position of a string in a set of strings.

FORMAT

Formats a number.

FROM_BASE64

Given a base-64 encoded string, returns the decoded result as a binary string.

HEX

Returns hexadecimal value.

INSERT Function

Replaces a part of a string with another string.

INSTR

Returns the position of a string within a string.

LCASE

Synonym for LOWER().

LEFT

Returns the leftmost characters from a string.

LENGTH

Length of the string in bytes.

LENGTHB

Length of the given string, in bytes.

LIKE

Whether expression matches a pattern.

LOAD_FILE

Returns file contents as a string.

LOCATE

Returns the position of a substring in a string.

LOWER

Returns a string with all characters changed to lowercase.

LPAD

Returns the string left-padded with another string to a given length.

LTRIM

Returns the string with leading space characters removed.

MAKE_SET

Make a set of strings that matches a bitmask.

MATCH AGAINST

Perform a fulltext search on a fulltext index.

Full-Text Index Stopwords

Default list of full-text stopwords used by MATCH...AGAINST.

MID

Synonym for SUBSTRING(str,pos,len).

NATURAL_SORT_KEY

Sorting that is closer to natural human sorting.

NOT LIKE

Same as NOT(expr LIKE pat [ESCAPE 'escape_char']).

NOT REGEXP

Same as NOT (expr REGEXP pat).

OCTET_LENGTH

Returns the length of the given string, in bytes.

ORD

Return ASCII or character code.

POSITION

Returns the position of a substring in a string.

QUOTE

Returns quoted, properly escaped string.

REPEAT Function

Returns a string repeated a number of times.

REPLACE Function

Replace occurrences of a string.

REVERSE

Reverses the order of a string.

RIGHT

Returns the rightmost N characters from a string.

RPAD

Returns the string right-padded with another string to a given length.

RTRIM

Returns the string with trailing space characters removed.

SFORMAT

Given a string and a formatting specification, returns a formatted string.

SOUNDEX

Returns a string based on how the string sounds.

SOUNDS LIKE

SOUNDEX(expr1) = SOUNDEX(expr2).

SPACE

Returns a string of space characters.

STRCMP

Compares two strings in sort order.

SUBSTR

Returns a substring from string starting at a given position.

SUBSTRING

Returns a substring from string starting at a given position.

SUBSTRING_INDEX

Returns the substring from string before count occurrences of a delimiter.

TO_BASE64

Converts a string to its base-64 encoded form.

TO_CHAR

Converts a date/time/timestamp type expression to a string.

TRIM

Returns a string with all given prefixes or suffixes removed.

TRIM_ORACLE

Synonym for the Oracle mode version of TRIM().

UCASE

Synonym for UPPER().

UNCOMPRESS

Uncompresses string compressed with COMPRESS().

UNCOMPRESSED_LENGTH

Returns length of a string before being compressed with COMPRESS().

UNHEX

Interprets pairs of hex digits as numbers and converts to the character represented by the number.

UPDATEXML
Replace XML.

UPPER
Changes string to uppercase.

WEIGHT_STRING
Weight of the input string.

Type Conversion
When implicit type conversion takes place.

There are 3 related questions.

1.2.2.1 Regular Expressions Functions

MariaDB includes a number of functions for dealing with regular expressions.

Regular Expressions Overview
Regular Expressions allow MariaDB to perform complex pattern matching on a string.

PCRE - Perl Compatible Regular Expressions
PCRE (Perl Compatible Regular Expressions) for enhanced regular expressions.

NOT REGEXP
Same as NOT (expr REGEXP pat).

REGEXP
Performs pattern matching

REGEXP_INSTR
Position of the first appearance of a regex.

REGEXP_REPLACE
Replaces all occurrences of a pattern.

REGEXP_SUBSTR
Returns the matching part of a string.

RLIKE
Synonym for REGEXP

There are 1 related questions.

1.2.2.1.1 Regular Expressions Overview

Contents

Regular Expressions allow MariaDB to perform complex pattern matching on a string. In many cases, the simple pattern matching provided by LIKE is sufficient. `LIKE` performs two kinds of matches:

- `_` - the underscore, matching a single character
- `%` - the percentage sign, matching any number of characters.

In other cases you may need more control over the returned matches, and will need to use regular expressions.

Until MariaDB 10.0.5, MariaDB used the POSIX 1003.2 compliant regular expression library. The current PCRE library is mostly backwards compatible with what is described below - see the PCRE Regular Expressions article for the enhancements made in 10.0.5.

Regular expression matches are performed with the REGEXP function. `RLIKE` is a synonym for `REGEXP`.

> Comparisons are performed on the byte value, so characters that are treated as equivalent by a collation, but do not have the same byte-value, such as accented characters, could evaluate as unequal.

Without any special characters, a regular expression match is true if the characters match. The match is case-insensitive, except in the case of BINARY strings.

```
SELECT 'Maria' REGEXP 'Maria';
+------------------------+
| 'Maria' REGEXP 'Maria' |
+------------------------+
|                      1 |
+------------------------+

SELECT 'Maria' REGEXP 'maria';
+------------------------+
| 'Maria' REGEXP 'maria' |
+------------------------+
|                      1 |
+------------------------+

SELECT BINARY 'Maria' REGEXP 'maria';
+-------------------------------+
| BINARY 'Maria' REGEXP 'maria' |
+-------------------------------+
|                             0 |
+-------------------------------+
```

Note that the word being matched must match the whole pattern:

```
SELECT 'Maria' REGEXP 'Mari';
+-----------------------+
| 'Maria' REGEXP 'Mari' |
+-----------------------+
|                     1 |
+-----------------------+

SELECT 'Mari' REGEXP 'Maria';
+-----------------------+
| 'Mari' REGEXP 'Maria' |
+-----------------------+
|                     0 |
+-----------------------+
```

The first returns true because the pattern "Mari" exists in the expression "Maria". When the order is reversed, the result is false, as the pattern "Maria" does not exist in the expression "Mari"

A match can be performed against more than one word with the `|` character. For example:

```
SELECT 'Maria' REGEXP 'Monty|Maria';
+------------------------------+
| 'Maria' REGEXP 'Monty|Maria' |
+------------------------------+
|                            1 |
+------------------------------+
```

Special Characters

The above examples introduce the syntax, but are not very useful on their own. It's the special characters that give regular expressions their power.

^

`^` matches the beginning of a string (inside square brackets it can also mean NOT - see below):

```
SELECT 'Maria' REGEXP '^Ma';
+----------------------+
| 'Maria' REGEXP '^Ma' |
+----------------------+
|                    1 |
+----------------------+
```

$

`$` matches the end of a string:

```
SELECT 'Maria' REGEXP 'ia$';
+----------------------+
| 'Maria' REGEXP 'ia$' |
+----------------------+
|                    1 |
+----------------------+
```

.

`.` matches any single character:

```
SELECT 'Maria' REGEXP 'Ma.ia';
+------------------------+
| 'Maria' REGEXP 'Ma.ia' |
+------------------------+
|                      1 |
+------------------------+

SELECT 'Maria' REGEXP 'Ma..ia';
+-------------------------+
| 'Maria' REGEXP 'Ma..ia' |
+-------------------------+
|                       0 |
+-------------------------+
```

*

`x*` matches zero or more of a character `x`. In the examples below, it's the `r` character.

```
SELECT 'Maria' REGEXP 'Mar*ia';
+-------------------------+
| 'Maria' REGEXP 'Mar*ia' |
+-------------------------+
|                       1 |
+-------------------------+

SELECT 'Maia' REGEXP 'Mar*ia';
+------------------------+
| 'Maia' REGEXP 'Mar*ia' |
+------------------------+
|                      1 |
+------------------------+

SELECT 'Marrria' REGEXP 'Mar*ia';
+---------------------------+
| 'Marrria' REGEXP 'Mar*ia' |
+---------------------------+
|                         1 |
+---------------------------+
```

+

`x+` matches one or more of a character `x`. In the examples below, it's the `r` character.

```
SELECT 'Maria' REGEXP 'Mar+ia';
+-------------------------+
| 'Maria' REGEXP 'Mar+ia' |
+-------------------------+
|                       1 |
+-------------------------+

SELECT 'Maia' REGEXP 'Mar+ia';
+------------------------+
| 'Maia' REGEXP 'Mar+ia' |
+------------------------+
|                      0 |
+------------------------+

SELECT 'Marrria' REGEXP 'Mar+ia';
+---------------------------+
| 'Marrria' REGEXP 'Mar+ia' |
+---------------------------+
|                         1 |
+---------------------------+
```

?

`x?` matches zero or one of a character `x`. In the examples below, it's the `r` character.

```
SELECT 'Maria' REGEXP 'Mar?ia';
+-------------------------+
| 'Maria' REGEXP 'Mar?ia' |
+-------------------------+
|                       1 |
+-------------------------+

SELECT 'Maia' REGEXP 'Mar?ia';
+------------------------+
| 'Maia' REGEXP 'Mar?ia' |
+------------------------+
|                      1 |
+------------------------+

SELECT 'Marrria' REGEXP 'Mar?ia';
+---------------------------+
| 'Marrria' REGEXP 'Mar?ia' |
+---------------------------+
|                         0 |
+---------------------------+
```

()

`(xyz)` - combine a sequence, for example `(xyz)+` or `(xyz)*`

```
SELECT 'Maria' REGEXP '(ari)+';
+-------------------------+
| 'Maria' REGEXP '(ari)+' |
+-------------------------+
|                       1 |
+-------------------------+
```

{}

`x{n}` and `x{m,n}` This notation is used to match many instances of the `x` . In the case of `x{n}` the match must be exactly that many times. In the case of `x{m,n}` , the match can occur from `m` to `n` times. For example, to match zero or one instance of the string `ari` (which is identical to `(ari)?`), the following can be used:

```
SELECT 'Maria' REGEXP '(ari){0,1}';
+-----------------------------+
| 'Maria' REGEXP '(ari){0,1}' |
+-----------------------------+
|                           1 |
+-----------------------------+
```

[]

`[xy]` groups characters for matching purposes. For example, to match either the `p` or the `r` character:

```
SELECT 'Maria' REGEXP 'Ma[pr]ia';
+---------------------------+
| 'Maria' REGEXP 'Ma[pr]ia' |
+---------------------------+
|                         1 |
+---------------------------+
```

The square brackets also permit a range match, for example, to match any character from a-z, `[a-z]` is used. Numeric ranges are also permitted.

```
SELECT 'Maria' REGEXP 'Ma[a-z]ia';
+---------------------------+
| 'Maria' REGEXP 'Ma[a-z]ia' |
+---------------------------+
|                          1 |
+---------------------------+
```

The following does not match, as `r` falls outside of the range `a-p` .

```
SELECT 'Maria' REGEXP 'Ma[a-p]ia';
+----------------------------+
| 'Maria' REGEXP 'Ma[a-p]ia' |
+----------------------------+
|                          0 |
+----------------------------+
```

^

The `^` character means does `NOT` match, for example:

```
SELECT 'Maria' REGEXP 'Ma[^p]ia';
+----------------------------+
| 'Maria' REGEXP 'Ma[^p]ia' |
+----------------------------+
|                          1 |
+----------------------------+

SELECT 'Maria' REGEXP 'Ma[^r]ia';
+----------------------------+
| 'Maria' REGEXP 'Ma[^r]ia' |
+----------------------------+
|                          0 |
+----------------------------+
```

The `[` and `]` characters on their own can be literally matched inside a `[]` block, without escaping, as long as they immediately match the opening bracket:

```
SELECT '[Maria' REGEXP '[[]';
+-----------------------+
| '[Maria' REGEXP '[[]' |
+-----------------------+
|                     1 |
+-----------------------+

SELECT '[Maria' REGEXP '[]]';
+-----------------------+
| '[Maria' REGEXP '[]]' |
+-----------------------+
|                     0 |
+-----------------------+

SELECT ']Maria' REGEXP '[]]';
+-----------------------+
| ']Maria' REGEXP '[]]' |
+-----------------------+
|                     1 |
+-----------------------+

SELECT ']Maria' REGEXP '[]a]';
+------------------------+
| ']Maria' REGEXP '[]a]' |
+------------------------+
|                      1 |
+------------------------+
```

Incorrect order, so no match:

```
SELECT ']Maria' REGEXP '[a]]';
+------------------------+
| ']Maria' REGEXP '[a]]' |
+------------------------+
|                      0 |
+------------------------+
```

The `-` character can also be matched in the same way:

```
SELECT '-Maria' REGEXP '[1-10]';
+--------------------------+
| '-Maria' REGEXP '[1-10]' |
+--------------------------+
|                        0 |
+--------------------------+

SELECT '-Maria' REGEXP '[-1-10]';
+---------------------------+
| '-Maria' REGEXP '[-1-10]' |
+---------------------------+
|                         1 |
+---------------------------+
```

Word boundaries

The `:<:` and `:>:` patterns match the beginning and the end of a word respectively. For example:

```
SELECT 'How do I upgrade MariaDB?' REGEXP '[[:<:]]MariaDB[[:>:]]';
+-----------------------------------------------------------+
| 'How do I upgrade MariaDB?' REGEXP '[[:<:]]MariaDB[[:>:]]' |
+-----------------------------------------------------------+
|                                                         1 |
+-----------------------------------------------------------+

SELECT 'How do I upgrade MariaDB?' REGEXP '[[:<:]]Maria[[:>:]]';
+-----------------------------------------------------------+
| 'How do I upgrade MariaDB?' REGEXP '[[:<:]]Maria[[:>:]]' |
+-----------------------------------------------------------+
|                                                         0 |
+-----------------------------------------------------------+
```

Character Classes

There are a number of shortcuts to match particular preset character classes. These are matched with the `[:character_class:]` pattern (inside a `[]` set). The following character classes exist:

Character Class	Description
alnum	Alphanumeric
alpha	Alphabetic
blank	Whitespace
cntrl	Control characters
digit	Digits
graph	Graphic characters
lower	Lowercase alphabetic
print	Graphic or space characters
punct	Punctuation
space	Space, tab, newline, and carriage return
upper	Uppercase alphabetic
xdigit	Hexadecimal digit

For example:

```
SELECT 'Maria' REGEXP 'Mar[[:alnum:]]*';
+---------------------------------+
| 'Maria' REGEXP 'Mar[:alnum:]*' |
+---------------------------------+
|                               1 |
+---------------------------------+
```

Remember that matches are by default case-insensitive, unless a binary string is used, so the following example, specifically looking for an uppercase, counter-intuitively matches a lowercase character:

```
SELECT 'Mari' REGEXP 'Mar[[:upper:]]+';
+---------------------------------+
| 'Mari' REGEXP 'Mar[[:upper:]]+' |
+---------------------------------+
|                               1 |
+---------------------------------+

SELECT BINARY 'Mari' REGEXP 'Mar[[:upper:]]+';
+----------------------------------------+
| BINARY 'Mari' REGEXP 'Mar[[:upper:]]+' |
+----------------------------------------+
|                                      0 |
+----------------------------------------+
```

Character Names

There are also number of shortcuts to match particular preset character names. These are matched with the `[.character.]` pattern (inside a `[]` set). The following character classes exist:

Name	Character
NUL	0
SOH	001
STX	002
ETX	003
EOT	004
ENQ	005
ACK	006
BEL	007
alert	007
BS	010
backspace	'\b'
HT	011
tab	'\t'
LF	012
newline	'\n'
VT	013
vertical-tab	'\v'
FF	014
form-feed	'\f'
CR	015
carriage-return	'\r'
SO	016
SI	017
DLE	020
DC1	021
DC2	022
DC3	023
DC4	024
NAK	025
SYN	026

ETB	027
CAN	030
EM	031
SUB	032
ESC	033
IS4	034
FS	034
IS3	035
GS	035
IS2	036
RS	036
IS1	037
US	037
space	' '
exclamation-mark	'!'
quotation-mark	'"'
number-sign	'#'
dollar-sign	'$'
percent-sign	'%'
ampersand	'&'
apostrophe	'\''
left-parenthesis	'('
right-parenthesis	')'
asterisk	'*'
plus-sign	'+'
comma	','
hyphen	'-'
hyphen-minus	'-'
period	'.'
full-stop	'.'
slash	'/'
solidus	'/'
zero	'0'
one	'1'
two	'2'
three	'3'
four	'4'
five	'5'
six	'6'
seven	'7'
eight	'8'
nine	'9'
colon	':'

semicolon	';'
less-than-sign	'<'
equals-sign	'='
greater-than-sign	'>'
question-mark	'?'
commercial-at	'@'
left-square-bracket	'['
backslash	' '
reverse-solidus	' '
right-square-bracket	']'
circumflex	'^'
circumflex-accent	'^'
underscore	'_'
low-line	'_'
grave-accent	'`'
left-brace	'{'
left-curly-bracket	'{'
vertical-line	'|'
right-brace	'}'
right-curly-bracket	'}'
tilde	''
DEL	177

For example:

```
SELECT '|' REGEXP '[[.vertical-line.]]';
+----------------------------------+
| '|' REGEXP '[[.vertical-line.]]' |
+----------------------------------+
|                                1 |
+----------------------------------+
```

Combining

The true power of regular expressions is unleashed when the above is combined, to form more complex examples. Regular expression's reputation for complexity stems from the seeming complexity of multiple combined regular expressions, when in reality, it's simply a matter of understanding the characters and how they apply:

The first example fails to match, as while the `Ma` matches, either `i` or `r` only matches once before the `ia` characters at the end.

```
SELECT 'Maria' REGEXP 'Ma[ir]{2}ia';
+------------------------------+
| 'Maria' REGEXP 'Ma[ir]{2}ia' |
+------------------------------+
|                            0 |
+------------------------------+
```

This example matches, as either `i` or `r` match exactly twice after the `Ma`, in this case one `r` and one `i`.

```
SELECT 'Maria' REGEXP 'Ma[ir]{2}';
+----------------------------+
| 'Maria' REGEXP 'Ma[ir]{2}' |
+----------------------------+
|                          1 |
+----------------------------+
```

Escaping

With the large number of special characters, care needs to be taken to properly escape characters. Two backslash characters,
(one for the MariaDB parser, one for the regex library), are required to properly escape a character. For example:

To match the literal `(Ma` :

```
SELECT '(Maria)' REGEXP '(Ma';
ERROR 1139 (42000): Got error 'parentheses not balanced' from regexp

SELECT '(Maria)' REGEXP '\(Ma';
ERROR 1139 (42000): Got error 'parentheses not balanced' from regexp

SELECT '(Maria)' REGEXP '\\(Ma';
+--------------------------+
| '(Maria)' REGEXP '\\(Ma' |
+--------------------------+
|                        1 |
+--------------------------+
```

To match `r+` : The first two examples are incorrect, as they match `r` one or more times, not `r+` :

```
SELECT 'Mar+ia' REGEXP 'r+';
+----------------------+
| 'Mar+ia' REGEXP 'r+' |
+----------------------+
|                    1 |
+----------------------+

SELECT 'Maria' REGEXP 'r+';
+--------------------+
| 'Maria' REGEXP 'r+' |
+--------------------+
|                  1 |
+--------------------+

SELECT 'Maria' REGEXP 'r\\+';
+----------------------+
| 'Maria' REGEXP 'r\\+' |
+----------------------+
|                    0 |
+----------------------+

SELECT 'Maria' REGEXP 'r+';
+--------------------+
| 'Maria' REGEXP 'r+' |
+--------------------+
|                  1 |
+--------------------+
```

1.2.2.1.2 Perl Compatible Regular Expressions (PCRE) Documentation

Contents

PCRE Versions

PCRE Version	Introduced	Maturity
PCRE2 10.34	MariaDB 10.5.1	Stable
PCRE 8.43	MariaDB 10.1.39	Stable
PCRE 8.42	MariaDB 10.2.15, MariaDB 10.1.33, MariaDB 10.0.35	Stable
PCRE 8.41	MariaDB 10.2.8, MariaDB 10.1.26, MariaDB 10.0.32	Stable
PCRE 8.40	MariaDB 10.2.5, MariaDB 10.1.22, MariaDB 10.0.30	Stable
PCRE 8.39	MariaDB 10.1.15, MariaDB 10.0.26	Stable
PCRE 8.38	MariaDB 10.1.10, MariaDB 10.0.23	Stable
PCRE 8.37	MariaDB 10.1.5, MariaDB 10.0.18	Stable
PCRE 8.36	MariaDB 10.1.2, MariaDB 10.0.15	Stable
PCRE 8.35	MariaDB 10.1.0, MariaDB 10.0.12	Stable
PCRE 8.34	MariaDB 10.0.8	Stable

PCRE Enhancements

MariaDB 10.0.5 switched to the PCRE library, which significantly improved the power of the REGEXP/RLIKE operator.

The switch to PCRE added a number of features, including recursive patterns, named capture, look-ahead and look-behind assertions, non-capturing groups, non-greedy quantifiers, Unicode character properties, extended syntax for characters and character classes, multi-line matching, and many other.

Additionally, MariaDB 10.0.5 introduced three new functions that work with regular expressions: REGEXP_REPLACE(), REGEXP_INSTR() and REGEXP_SUBSTR().

Also, REGEXP/RLIKE, and the new functions, now work correctly with all multi-byte character sets supported by MariaDB, including East-Asian character sets (big5, gb2313, gbk, eucjp, eucjpms, cp932, ujis, euckr), and Unicode character sets (utf8, utf8mb4, ucs2, utf16, utf16le, utf32). In earlier versions of MariaDB (and all MySQL versions) REGEXP/RLIKE works correctly only with 8-bit character sets.

New Regular Expression Functions

- REGEXP_REPLACE(subject, pattern, replace) - Replaces all occurrences of a pattern.
- REGEXP_INSTR(subject, pattern) - Position of the first appearance of a regex .
- REGEXP_SUBSTR(subject,pattern) - Returns the matching part of a string.

See the individual articles for more details and examples.

PCRE Syntax

In most cases PCRE is backward compatible with the old POSIX 1003.2 compliant regexp library (see Regular Expressions Overview), so you won't need to change your applications that use SQL queries with the REGEXP/RLIKE predicate.

MariaDB 10.0.11 introduced the default_regex_flags variable to address the remaining compatibilities between PCRE and the old regex library.

This section briefly describes the most important extended PCRE features. For more details please refer to the documentation on the PCRE site, or to the documentation which is bundled in the /pcre/doc/html/ directory of a MariaDB sources distribution. The pages pcresyntax.html and pcrepattern.html should be a good start. Regular-Expressions.Info is another good resource to learn about PCRE and regular expressions generally.

Special Characters

PCRE supports the following escape sequences to match special characters:

Sequence	Description
\a	0x07 (BEL)
\cx	"control-x", where x is any ASCII character
\e	0x1B (escape)
\f	0x0C (form feed)
\n	0x0A (newline)
\r	0x0D (carriage return)
\t	0x09 (TAB)
\ddd	character with octal code ddd
\xhh	character with hex code hh
\x{hhh..}	character with hex code hhh..

Note, the backslash characters (here, and in all examples in the sections below) must be escaped with another backslash, unless you're using the SQL_MODE `NO_BACKSLASH_ESCAPES` .

This example tests if a character has hex code 0x61:

```
SELECT 'a' RLIKE '\\x{61}';
-> 1
```

Character Classes

PCRE supports the standard POSIX character classes such as `alnum` , `alpha` , `blank` , `cntrl` , `digit` , `graph` , `lower` , `print` , `punct` , `space` , `upper` , `xdigit` , with the following additional classes:

Class	Description
ascii	any ASCII character (0x00..0x7F)

word	any "word" character (a letter, a digit, or an underscore)

This example checks if the string consists of ASCII characters only:

```
SELECT 'abc' RLIKE '^[[:ascii:]]+$';
-> 1
```

Generic Character Types

Generic character types complement the POSIX character classes and serve to simplify writing patterns:

Class	Description
\d	a decimal digit (same as [:digit:])
\D	a character that is not a decimal digit
\h	a horizontal white space character
\H	a character that is not a horizontal white space character
\N	a character that is not a new line
\R	a newline sequence
\s	a white space character
\S	a character that is not a white space character
\v	a vertical white space character
\V	a character that is not a vertical white space character
\w	a "word" character (same as [:word:])
\W	a "non-word" character

This example checks if the string consists of "word" characters only:

```
SELECT 'abc' RLIKE '^\\w+$';
-> 1
```

Unicode Character Properties

`\p{xx}` is a character with the `xx` property, and `\P{xx}` is a character without the `xx` property.

The property names represented by `xx` above are limited to the Unicode script names, the general category properties, and "Any", which matches any character (including newline). Those that are not part of an identified script are lumped together as "Common".

General Category Properties For \p and \P

Property	Description
C	Other
Cc	Control
Cf	Format
Cn	Unassigned
Co	Private use
Cs	Surrogate
L	Letter
Ll	Lower case letter
Lm	Modifier letter
Lo	Other letter
Lt	Title case letter

Lu	Upper case letter
L&	Ll, Lu, or Lt
M	Mark
Mc	Spacing mark
Me	Enclosing mark
Mn	Non-spacing mark
N	Number
Nd	Decimal number
Nl	Letter number
No	Other number
P	Punctuation
Pc	Connector punctuation
Pd	Dash punctuation
Pe	Close punctuation
Pf	Final punctuation
Pi	Initial punctuation
Po	Other punctuation
Ps	Open punctuation
S	Symbol
Sc	Currency symbol
Sk	Modifier symbol
Sm	Mathematical symbol
So	Other symbol
Z	Separator
Zl	Line separator
Zp	Paragraph separator
Zs	Space separator

This example checks if the string consists only of characters with property N (number):

```
SELECT '1¼①' RLIKE '^\\p{N}+$';
-> 1
```

Special Category Properties For \p and \P

Property	Description
Xan	Alphanumeric: union of properties L and N
Xps	POSIX space: property Z or tab, NL, VT, FF, CR
Xsp	Perl space: property Z or tab, NL, FF, CR
Xuc	A character than can be represented by a Universal Character Name
Xwd	Perl word: property Xan or underscore

The property `Xuc` matches any character that can be represented by a Universal Character Name (in C++ and other programming languages). These include `$`, `@`, `` ` ``, and all characters with Unicode code points greater than `U+00A0`, excluding the surrogates `U+D800` .. `U+DFFF`.

Script Names For \p and \P

Arabic, Armenian, Avestan, Balinese, Bamum, Batak, Bengali, Bopomofo, Brahmi, Braille, Buginese, Buhid,

Canadian_Aboriginal, Carian, Chakma, Cham, Cherokee, Common, Coptic, Cuneiform, Cypriot, Cyrillic, Deseret, Devanagari, Egyptian_Hieroglyphs, Ethiopic, Georgian, Glagolitic, Gothic, Greek, Gujarati, Gurmukhi, Han, Hangul, Hanunoo, Hebrew, Hiragana, Imperial_Aramaic, Inherited, Inscriptional_Pahlavi, Inscriptional_Parthian, Javanese, Kaithi, Kannada, Katakana, Kayah_Li, Kharoshthi, Khmer, Lao, Latin, Lepcha, Limbu, Linear_B, Lisu, Lycian, Lydian, Malayalam, Mandaic, Meetei_Mayek, Meroitic_Cursive, Meroitic_Hieroglyphs, Miao, Mongolian, Myanmar, New_Tai_Lue, Nko, Ogham, Old_Italic, Old_Persian, Old_South_Arabian, Old_Turkic, Ol_Chiki, Oriya, Osmanya, Phags_Pa, Phoenician, Rejang, Runic, Samaritan, Saurashtra, Sharada, Shavian, Sinhala, Sora_Sompeng, Sundanese, Syloti_Nagri, Syriac, Tagalog, Tagbanwa, Tai_Le, Tai_Tham, Tai_Viet, Takri, Tamil, Telugu, Thaana, Thai, Tibetan, Tifinagh, Ugaritic, Vai, Yi.

This example checks if the string consists only of Greek characters:

```
SELECT 'ΣΦΩ' RLIKE '^\\p{Greek}+$';
-> 1
```

Extended Unicode Grapheme Sequence

The `\X` escape sequence matches a character sequence that makes an "extended grapheme cluster", i.e. a composite character that consists of multiple Unicode code points.

One of the examples of a composite character can be a letter followed by non-spacing accent marks. This example demonstrates that `U+0045 LATIN CAPITAL LETTER E` followed by `U+0302 COMBINING CIRCUMFLEX ACCENT` followed by `U+0323 COMBINING DOT BELOW` together form an extended grapheme cluster:

```
SELECT _ucs2 0x004503020323 RLIKE '^\\X$';
-> 1
```

See the PCRE documentation for the other types of extended grapheme clusters.

Simple Assertions

An assertion specifies a certain condition that must match at a particular point, but without consuming characters from the subject string. In addition to the standard POSIX simple assertions `^` (that matches at the beginning of a line) and `$` (that matches at the end of a line), PCRE supports a number of other assertions:

Assertion	Description
\b	matches at a word boundary
\B	matches when not at a word boundary
\A	matches at the start of the subject
\Z	matches at the end of the subject, also matches before a newline at the end of the subject
\z	matches only at the end of the subject
\G	matches at the first matching position in the subject

This example cuts a word that consists only of 3 characters from a string:

```
SELECT REGEXP_SUBSTR('---abcd---xyz---', '\\b\\w{3}\\b');
-> xyz
```

Notice that the two `\b` assertions checked the word boundaries but did not get into the matching pattern.

The `\b` assertions work well in the beginning and the end of the subject string:

```
SELECT REGEXP_SUBSTR('xyz', '\\b\\w{3}\\b');
-> xyz
```

By default, the `^` and `$` assertions have the same meaning with `\A`, `\Z`, and `\z`. However, the meanings of `^` and `$` can change in multiline mode (see below). By contrast, the meanings of `\A`, `\Z`, and `\z` are always the same; they are independent of the multiline mode.

Option Setting

A number of options that control the default match behavior can be changed within the pattern by a sequence of option letters enclosed between `(?` and `)`.

Option	Description
(?i)	case insensitive match
(?m)	multiline mode
(?s)	dotall mode (dot matches newline characters)
(?x)	extended (ignore white space)
(?U)	ungreedy (lazy) match
(?J)	allow named subpatterns with duplicate names
(?X)	extra PCRE functionality (e.g. force error on unknown escaped character)
(?-...)	unset option(s)

For example, `(?im)` sets case insensitive multiline matching.

A hyphen followed by the option letters unset the options. For example, `(?-im)` means case sensitive single line match.

A combined setting and unsetting is also possible, e.g. `(?im-sx)`.

If an option is set outside of subpattern parentheses, the option applies to the remainder of the pattern that follows the option. If an option is set inside a subpattern, it applies to the part of this subpattern that follows the option.

In this example the pattern `(?i)m((?-i)aria)db` matches the words `MariaDB`, `Mariadb`, `mariadb`, but not `MARIADB`:

```
SELECT 'MariaDB' RLIKE '(?i)m((?-i)aria)db';
-> 1

SELECT 'mariaDB' RLIKE '(?i)m((?-i)aria)db';
-> 1

SELECT 'Mariadb' RLIKE '(?i)m((?-i)aria)db';
-> 1

SELECT 'MARIADB' RLIKE '(?i)m((?-i)aria)db';
-> 0
```

This example demonstrates that the `(?x)` option makes the regexp engine ignore all white spaces in the pattern (other than in a class).

```
SELECT 'ab' RLIKE '(?x)a b';
-> 1
```

Note, putting spaces into a pattern in combination with the (?x) option can be useful to split different logical parts of a complex pattern, to make it more readable.

Multiline Matching

Multiline matching changes the meaning of `^` and `$` from "the beginning of the subject string" and "the end of the subject string" to "the beginning of any line in the subject string" and "the end of any line in the subject string" respectively.

This example checks if the subject string contains two consequent lines that fully consist of digits:

```
SELECT 'abc\n123\n456\nxyz\n' RLIKE '(?m)^\\d+\\R\\d+$';
-> 1
```

Notice the `(?m)` option in the beginning of the pattern, which switches to the multiline matching mode.

Newline Conventions

PCRE supports five line break conventions:

- `CR (\r)` - a single carriage return character
- `LF (\n)` - a single linefeed character
- `CRLF (\r\n)` - a carriage return followed by a linefeed
- any of the previous three

- any Unicode newline sequence

By default, the newline convention is set to any Unicode newline sequence, which includes:

Sequence	Description
LF	(U+000A, carriage return)
CR	(U+000D, carriage return)
CRLF	(a carriage return followed by a linefeed)
VT	(U+000B, vertical tab)
FF	(U+000C, form feed)
NEL	(U+0085, next line)
LS	(U+2028, line separator)
PS	(U+2029, paragraph separator)

The newline convention can be set by starting a pattern with one of the following sequences:

Sequence	Description
(*CR)	carriage return
(*LF)	linefeed
(*CRLF)	carriage return followed by linefeed
(*ANYCRLF)	any of the previous three
(*ANY)	all Unicode newline sequences

The newline conversion affects the `^` and `$` assertions, the interpretation of the dot metacharacter, and the behavior of `\N`.

Note, the new line convention does not affect the meaning of `\R`.

This example demonstrates that the dot metacharacter matches `\n`, because it is not a newline sequence anymore:

```
SELECT 'a\nb' RLIKE '(*CR)a.b';
-> 1
```

Newline Sequences

By default, the escape sequence `\R` matches any Unicode newline sequences.

The meaning of `\R` can be set by starting a pattern with one of the following sequences:

Sequence	Description
(*BSR_ANYCRLF)	any of CR, LF or CRLF
(*BSR_UNICODE)	any Unicode newline sequence

Comments

It's possible to include comments inside a pattern. Comments do not participate in the pattern matching. Comments start at the `(?#` sequence and continue up to the next closing parenthesis:

```
SELECT 'ab12' RLIKE 'ab(?#expect digits)12';
-> 1
```

Quoting

POSIX uses the backslash to remove a special meaning from a character. PCRE introduces a syntax to remove special meaning from a sequence of characters. The characters inside `\Q` ... `\E` are treated literally, without their special meaning.

This example checks if the string matches a dollar sign followed by a parenthesized name (a variable reference in some languages):

```
SELECT '$(abc)' RLIKE '^\\Q$(\\E\\w+\\Q)\\E$';
-> 1
```

Note that the leftmost dollar sign and the parentheses are used literally, while the rightmost dollar sign is still used to match the end of the string.

Resetting the Match Start

The escape sequence `\K` causes any previously matched characters to be excluded from the final matched sequence. For example, the pattern: `(foo)\Kbar` matches `foobar`, but reports that it has matched `bar`. This feature is similar to a look-behind assertion. However, in this case, the part of the subject before the real match does not have to be of fixed length:

```
SELECT REGEXP_SUBSTR('aaa123', '[a-z]*\\K[0-9]*');
-> 123
```

Non-Capturing Groups

The question mark and the colon after the opening parenthesis create a non-capturing group: `(?:...)`.

This example removes an optional article from a word, for example for better sorting of the results.

```
SELECT REGEXP_REPLACE('The King','(?:the|an|a)[^a-z]([a-z]+)','\\1');
-> King
```

Note that the articles are listed inside the left parentheses using the alternation operator `|` but they do not produce a captured subpattern, so the word followed by the article is referenced by `'\1'` in the third argument to the function. Using non-capturing groups can be useful to save numbers on the sup-patterns that won't be used in the third argument of REGEXP_REPLACE(), as well as for performance purposes.

Non-Greedy Quantifiers

By default, the repetition quantifiers `?`, `*`, `+` and `{n,m}` are "greedy", that is, they try to match as much as possible. Adding a question mark after a repetition quantifier makes it "non-greedy", so the pattern matches the minimum number of times possible.

This example cuts C comments from a line:

```
SELECT REGEXP_REPLACE('/* Comment1 */ i+= 1; /* Comment2 */', '/[*].*?[*]/','');
->  i+= 1;
```

The pattern without the non-greedy flag to the quantifier `/[*].*[*]/` would match the entire string between the leftmost `/*` and the rightmost `*/`.

Atomic Groups

A sequence inside `(?>...)` makes an atomic group. Backtracking inside an atomic group is prevented once it has matched; however, backtracking past to the previous items works normally.

Consider the pattern `\d+foo` applied to the subject string `123bar`. Once the engine scans `123` and fails on the letter `b`, it would normally backtrack to `2` and try to match again, then fail and backtrack to `1` and try to match and fail again, and finally fail the entire pattern. In case of an atomic group `(?>\d+)foo` with the same subject string `123bar`, the engine gives up immediately after the first failure to match `foo`. An atomic group with a quantifier can match all or nothing.

Atomic groups produce faster false results (i.e. in case when a long subject string does not match the pattern), because the regexp engine saves performance on backtracking. However, don't hurry to put everything into atomic groups. This example demonstrates the difference between atomic and non-atomic match:

```
SELECT 'abcc' RLIKE 'a(?>bc|b)c' AS atomic1;
-> 1

SELECT 'abc' RLIKE 'a(?>bc|b)c' AS atomic2;
-> 0

SELECT 'abcc' RLIKE 'a(bc|b)c' AS non_atomic1;
-> 1

SELECT 'abc' RLIKE 'a(bc|b)c' AS non_atomic2;
-> 1
```

The non-atomic pattern matches both `abbc` and `abc`, while the atomic pattern matches `abbc` only.

The atomic group `(?>bc|b)` in the above example can be "translated" as "if there is `bc`, then don't try to match as `b`". So `b` can match only if `bc` is not found.

Atomic groups are not capturing. To make an atomic group capturing, put it into parentheses:

```
SELECT REGEXP_REPLACE('abcc','a((?>bc|b))c','\\1');
-> bc
```

Possessive quantifiers

An atomic group which ends with a quantifier can be rewritten using a so called "possessive quantifier" syntax by putting an additional `+` sign following the quantifier.

The pattern `(?>\d+)foo` from the previous section's example can be rewritten as `\d++foo`.

Absolute and Relative Numeric Backreferences

Backreferences match the same text as previously matched by a capturing group. Backreferences can be written using:

- a backslash followed by a digit
- the `\g` escape sequence followed by a positive or negative number
- the `\g` escape sequence followed by a positive or negative number enclosed in braces

The following backreferences are identical and refer to the first capturing group:

- `\1`
- `\g1`
- `\g{1}`

This example demonstrates a pattern that matches "sense and sensibility" and "response and responsibility", but not "sense and responsibility":

```
SELECT 'sense and sensibility' RLIKE '(sens|respons)e and \\1ibility';
-> 1
```

This example removes doubled words that can unintentionally creep in when you edit a text in a text editor:

```
SELECT REGEXP_REPLACE('using using the the regexp regexp',
 '\\b(\\w+)\\s+\\1\\b','\\1');
-> using the regexp
```

Note that all double words were removed, in the beginning, in the middle and in the end of the subject string.

A negative number in a `\g` sequence means a relative reference. Relative references can be helpful in long patterns, and also in patterns that are created by joining fragments together that contain references within themselves. The sequence `\g{-1}` is a reference to the most recently started capturing subpattern before `\g`.

In this example `\g{-1}` is equivalent to `\2`:

```
SELECT 'abc123def123' RLIKE '(abc(123)def)\\g{-1}';
-> 1

SELECT 'abc123def123' RLIKE '(abc(123)def)\\2';
-> 1
```

Named Subpatterns and Backreferences

Using numeric backreferences for capturing groups can be hard to track in a complicated regular expression. Also, the numbers can change if an expression is modified. To overcome these difficulties, PCRE supports named subpatterns.

A subpattern can be named in one of three ways: `(?<name> ... )` or `(?'name' ... )` as in Perl, or `(?P<name> ... )` as in Python. References to capturing subpatterns from other parts of the pattern, can be made by name as well as by number.

Backreferences to a named subpattern can be written using the .NET syntax `\k{name}`, the Perl syntax `\k<name>` or `\k'name'` or `\g{name}`, or the Python syntax `(?P=name)`.

This example tests if the string is a correct HTML tag:

```
SELECT '<a href="../">Up</a>' RLIKE '<(?<tag>[a-z][a-z0-9]*)[^>]*>[^<]*</(?P=tag)>';
-> 1
```

Positive and Negative Look-Ahead and Look-Behind Assertions

Look-ahead and look-behind assertions serve to specify the context for the searched regular expression pattern. Note that the assertions only check the context, they do not capture anything themselves!

This example finds the letter which is not followed by another letter (negative look-ahead):

```
SELECT REGEXP_SUBSTR('ab1','[a-z](?![a-z])');
-> b
```

This example finds the letter which is followed by a digit (positive look-ahead):

```
SELECT REGEXP_SUBSTR('ab1','[a-z](?=[0-9])');
-> b
```

This example finds the letter which does not follow a digit character (negative look-behind):

```
SELECT REGEXP_SUBSTR('1ab','(?<![0-9])[a-z]');
-> b
```

This example finds the letter which follows another letter character (positive look-behind):

```
SELECT REGEXP_SUBSTR('1ab','(?<=[a-z])[a-z]');
-> b
```

Note that look-behind assertions can only be of fixed length; you cannot have repetition operators or alternations with different lengths:

```
SELECT 'aaa' RLIKE '(?<=(a|bc))a';
ERROR 1139 (42000): Got error 'lookbehind assertion is not fixed length at offset 10' from regexp
```

Subroutine Reference and Recursive Patterns

PCRE supports a special syntax to recourse the entire pattern or its individual subpatterns:

Syntax	Description
(?R)	Recourse the entire pattern
(?n)	call subpattern by absolute number
(?+n)	call subpattern by relative number
(?-n)	call subpattern by relative number
(?&name)	call subpattern by name (Perl)
(?P>name)	call subpattern by name (Python)
\g<name>	call subpattern by name (Oniguruma)
\g'name'	call subpattern by name (Oniguruma)

\g<n>	call subpattern by absolute number (Oniguruma)
\g'n'	call subpattern by absolute number (Oniguruma)
\g<+n>	call subpattern by relative number
\g<-n>	call subpattern by relative number
\g'+n'	call subpattern by relative number
\g'-n'	call subpattern by relative number

This example checks for a correct additive arithmetic expression consisting of numbers, unary plus and minus, binary plus and minus, and parentheses:

```
SELECT '1+2-3+(+(4-1)+(-2)+(+1))' RLIKE  '^(([+-]?(\\d+|[(](?1)[)]))(([+-](?1))*))$';
-> 1
```

The recursion is done using `(?1)` to call for the first parenthesized subpattern, which includes everything except the leading `^` and the trailing `$`.

The regular expression in the above example implements the following BNF grammar:

1. `<expression> ::= <term> [(<sign> <term>)...]`
2. `<term> ::= [ <sign> ] <primary>`
3. `<primary> ::= <number> | <left paren> <expression> <right paren>`
4. `<sign> ::= <plus sign> | <minus sign>`

Defining Subpatterns For Use By Reference

Use the `(?(DEFINE) ... )` syntax to define subpatterns that can be referenced from elsewhere.

This example defines a subpattern with the name `letters` that matches one or more letters, which is further reused two times:

```
SELECT 'abc123xyz' RLIKE '^(?(DEFINE)(?<letters>[a-z]+))(?&letters)[0-9]+(?&letters)$';
-> 1
```

The above example can also be rewritten to define the digit part as a subpattern as well:

```
SELECT 'abc123xyz' RLIKE
 '^(?(DEFINE)(?<letters>[a-z]+)(?<digits>[0-9]+))(?&letters)(?&digits)(?&letters)$';
-> 1
```

Conditional Subpatterns

There are two forms of conditional subpatterns:

```
(?(condition)yes-pattern)
(?(condition)yes-pattern|no-pattern)
```

The `yes-pattern` is used if the condition is satisfied, otherwise the `no-pattern` (if any) is used.

Conditions With Subpattern References

If a condition consists of a number, it makes a condition with a subpattern reference. Such a condition is true if a capturing subpattern corresponding to the number has previously matched.

This example finds an optionally parenthesized number in a string:

```
SELECT REGEXP_SUBSTR('a(123)b', '([(])?[0-9]+(?(1)[)])');
-> (123)
```

The `([(])?` part makes a capturing subpattern that matches an optional opening parenthesis; the `[0-9]+` part matches a number, and the `(?(1)[)])` part matches a closing parenthesis, but only if the opening parenthesis has been previously found.

Other Kinds of Conditions

The other possible condition kinds are: recursion references and assertions. See the PCRE documentation for details.

Matching Zero Bytes (0x00)

PCRE correctly works with zero bytes in the subject strings:

```
SELECT 'a\0b' RLIKE '^a.b$';
-> 1
```

Zero bytes, however, are not supported literally in the pattern strings and should be escaped using the `\xhh` or `\x{hh}` syntax:

```
SELECT 'a\0b' RLIKE '^a\\x{00}b$';
-> 1
```

Other PCRE Features

PCRE provides other extended features that were not covered in this document, such as duplicate subpattern numbers, backtracking control, breaking utf-8 sequences into individual bytes, setting the match limit, setting the recursion limit, optimization control, recursion conditions, assertion conditions and more types of extended grapheme clusters. Please refer to the PCRE documentation for details.

Enhanced regex was implemented as a GSoC 2013 project by Sudheera Palihakkara.

default_regex_flags Examples

The default_regex_flags variable was introduced to address the remaining incompatibilities between PCRE and the old regex library. Here are some examples of its usage:

The default behaviour (multiline match is off)

```
SELECT 'a\nb\nc' RLIKE '^b$';
+----------------------------+
| '(?m)a\nb\nc' RLIKE '^b$' |
+----------------------------+
|                          0 |
+----------------------------+
```

Enabling the multiline option using the PCRE option syntax:

```
SELECT 'a\nb\nc' RLIKE '(?m)^b$';
+----------------------------+
| 'a\nb\nc' RLIKE '(?m)^b$' |
+----------------------------+
|                          1 |
+----------------------------+
```

Enabling the miltiline option using default_regex_flags

```
SET default_regex_flags='MULTILINE';
SELECT 'a\nb\nc' RLIKE '^b$';
+-----------------------+
| 'a\nb\nc' RLIKE '^b$' |
+-----------------------+
|                     1 |
+-----------------------+
```

1.2.2.1.3 NOT REGEXP

Syntax

```
expr NOT REGEXP pat, expr NOT RLIKE pat
```

Description

This is the same as NOT (expr REGEXP pat).

1.2.2.1.4 REGEXP

Syntax

```
expr REGEXP pat, expr RLIKE pat
```

Description

Performs a pattern match of a string expression `expr` against a pattern `pat`. The pattern can be an extended regular expression. See Regular Expressions Overview for details on the syntax for regular expressions (see also PCRE Regular Expressions).

Returns `1` if `expr` matches `pat` or `0` if it doesn't match. If either `expr` or `pat` are NULL, the result is NULL.

The negative form NOT REGEXP also exists, as an alias for `NOT (string REGEXP pattern)`. RLIKE and NOT RLIKE are synonyms for REGEXP and NOT REGEXP, originally provided for mSQL compatibility.

The pattern need not be a literal string. For example, it can be specified as a string expression or table column.

Note: Because MariaDB uses the C escape syntax in strings (for example, "\n" to represent the newline character), you must double any "\" that you use in your REGEXP strings.

REGEXP is not case sensitive, except when used with binary strings.

MariaDB 10.0.5 moved to the PCRE regex library - see PCRE Regular Expressions for enhancements to REGEXP introduced in MariaDB 10.0.5.

The default_regex_flags variable addresses the remaining compatibilities between PCRE and the old regex library.

Examples

```
SELECT 'Monty!' REGEXP 'm%y%%';
+-------------------------+
| 'Monty!' REGEXP 'm%y%%' |
+-------------------------+
|                       0 |
+-------------------------+

SELECT 'Monty!' REGEXP '.*';
+----------------------+
| 'Monty!' REGEXP '.*' |
+----------------------+
|                    1 |
+----------------------+

SELECT 'new*\n*line' REGEXP 'new\\*.\\*line';
+---------------------------------------+
| 'new*\n*line' REGEXP 'new\\*.\\*line' |
+---------------------------------------+
|                                     1 |
+---------------------------------------+

SELECT 'a' REGEXP 'A', 'a' REGEXP BINARY 'A';
+----------------+-----------------------+
| 'a' REGEXP 'A' | 'a' REGEXP BINARY 'A' |
+----------------+-----------------------+
|              1 |                     0 |
+----------------+-----------------------+

SELECT 'a' REGEXP '^[a-d]';
+---------------------+
| 'a' REGEXP '^[a-d]' |
+---------------------+
|                   1 |
+---------------------+
```

default_regex_flags examples

MariaDB 10.0.11 introduced the default_regex_flags variable to address the remaining compatibilities between PCRE and the old regex library.

The default behaviour (multiline match is off)

```
SELECT 'a\nb\nc' RLIKE '^b$';
+---------------------------+
| '(?m)a\nb\nc' RLIKE '^b$' |
+---------------------------+
|                         0 |
+---------------------------+
```

Enabling the multiline option using the PCRE option syntax:

```
SELECT 'a\nb\nc' RLIKE '(?m)^b$';
+---------------------------+
| 'a\nb\nc' RLIKE '(?m)^b$' |
+---------------------------+
|                         1 |
+---------------------------+
```

Enabling the multiline option using default_regex_flags

```
SET default_regex_flags='MULTILINE';
SELECT 'a\nb\nc' RLIKE '^b$';
+-----------------------+
| 'a\nb\nc' RLIKE '^b$' |
+-----------------------+
|                     1 |
+-----------------------+
```

1.2.2.1.5 REGEXP_INSTR

Syntax

```
REGEXP_INSTR(subject, pattern)
```

Returns the position of the first occurrence of the regular expression `pattern` in the string `subject`, or 0 if pattern was not found.

The positions start with 1 and are measured in characters (i.e. not in bytes), which is important for multi-byte character sets. You can cast a multi-byte character set to BINARY to get offsets in bytes.

The function follows the case sensitivity rules of the effective collation. Matching is performed case insensitively for case insensitive collations, and case sensitively for case sensitive collations and for binary data.

The collation case sensitivity can be overwritten using the (?i) and (?-i) PCRE flags.

MariaDB uses the PCRE regular expression library for enhanced regular expression performance, and REGEXP_INSTR was introduced as part of this enhancement.

Examples

```
SELECT REGEXP_INSTR('abc','b');
-> 2

SELECT REGEXP_INSTR('abc','x');
-> 0

SELECT REGEXP_INSTR('BJÖRN','N');
-> 5
```

Casting a multi-byte character set as BINARY to get offsets in bytes:

```
SELECT REGEXP_INSTR(BINARY 'BJÖRN','N') AS cast_utf8_to_binary;
-> 6
```

Case sensitivity:

```
SELECT REGEXP_INSTR('ABC','b');
-> 2

SELECT REGEXP_INSTR('ABC' COLLATE utf8_bin,'b');
-> 0

SELECT REGEXP_INSTR(BINARY'ABC','b');
-> 0

SELECT REGEXP_INSTR('ABC','(?-i)b');
-> 0

SELECT REGEXP_INSTR('ABC' COLLATE utf8_bin,'(?i)b');
-> 2
```

1.2.2.1.6 REGEXP_REPLACE

Syntax

```
REGEXP_REPLACE(subject, pattern, replace)
```

Description

`REGEXP_REPLACE` returns the string `subject` with all occurrences of the regular expression `pattern` replaced by the string `replace`. If no occurrences are found, then `subject` is returned as is.

The replace string can have backreferences to the subexpressions in the form \N, where N is a number from 1 to 9.

The function follows the case sensitivity rules of the effective collation. Matching is performed case insensitively for case insensitive collations, and case sensitively for case sensitive collations and for binary data.

The collation case sensitivity can be overwritten using the (?i) and (?-i) PCRE flags.

MariaDB uses the PCRE regular expression library for enhanced regular expression performance, and `REGEXP_REPLACE` was introduced as part of this enhancement.

The default_regex_flags variable addresses the remaining compatibilities between PCRE and the old regex library.

Examples

```
SELECT REGEXP_REPLACE('ab12cd','[0-9]','') AS remove_digits;
-> abcd

SELECT REGEXP_REPLACE('<html><head><title>title</title><body>body</body></htm>', '<.+?>',' ')
AS strip_html;
-> title  body
```

Backreferences to the subexpressions in the form `\N`, where N is a number from 1 to 9:

```
SELECT REGEXP_REPLACE('James Bond','^(.*) (.*)$','\\2, \\1') AS reorder_name;
-> Bond, James
```

Case insensitive and case sensitive matches:

```
SELECT REGEXP_REPLACE('ABC','b','-') AS case_insensitive;
-> A-C

SELECT REGEXP_REPLACE('ABC' COLLATE utf8_bin,'b','-') AS case_sensitive;
-> ABC

SELECT REGEXP_REPLACE(BINARY 'ABC','b','-') AS binary_data;
-> ABC
```

Overwriting the collation case sensitivity using the (?i) and (?-i) PCRE flags.

```
SELECT REGEXP_REPLACE('ABC','(?-i)b','-') AS force_case_sensitive;
-> ABC

SELECT REGEXP_REPLACE(BINARY 'ABC','(?i)b','-') AS force_case_insensitive;
-> A-C
```

1.2.2.1.7 REGEXP_SUBSTR

Syntax

```
REGEXP_SUBSTR(subject,pattern)
```

Description

Returns the part of the string `subject` that matches the regular expression `pattern`, or an empty string if `pattern` was not found.

The function follows the case sensitivity rules of the effective collation. Matching is performed case insensitively for case insensitive collations, and case sensitively for case sensitive collations and for binary data.

The collation case sensitivity can be overwritten using the (?i) and (?-i) PCRE flags.

MariaDB uses the PCRE regular expression library for enhanced regular expression performance, and `REGEXP_SUBSTR`

was introduced as part of this enhancement.

The default_regex_flags variable addresses the remaining compatibilities between PCRE and the old regex library.

Examples

```
SELECT REGEXP_SUBSTR('ab12cd','[0-9]+');
-> 12

SELECT REGEXP_SUBSTR(
  'See https://mariadb.org/en/foundation/ for details',
  'https?://[^/]*');
-> https://mariadb.org
```

```
SELECT REGEXP_SUBSTR('ABC','b');
-> B

SELECT REGEXP_SUBSTR('ABC' COLLATE utf8_bin,'b');
->

SELECT REGEXP_SUBSTR(BINARY'ABC','b');
->

SELECT REGEXP_SUBSTR('ABC','(?i)b');
-> B

SELECT REGEXP_SUBSTR('ABC' COLLATE utf8_bin,'(?+i)b');
-> B
```

1.2.2.1.8 RLIKE

Syntax

```
expr REGEXP pat, expr RLIKE pat
```

Description

`RLIKE` is a synonym for REGEXP.

1.2.2.2 Dynamic Columns Functions

Dynamic columns is a feature that allows one to store different sets of columns for each row in a table. It works by storing a set of columns in a blob and having a small set of functions to manipulate it.

COLUMN_ADD

Adds or updates dynamic columns.

COLUMN_CHECK

Checks if a dynamic column blob is valid

COLUMN_CREATE

Returns a dynamic columns blob.

COLUMN_DELETE

Deletes a dynamic column.

COLUMN_EXISTS

Checks is a column exists.

COLUMN_GET

Gets a dynamic column value by name.

COLUMN_JSON

Returns a JSON representation of dynamic column blob data

COLUMN_LIST

Returns comma-separated list of columns names.

1.2.2.2.1 COLUMN_ADD

Syntax

```
COLUMN_ADD(dyncol_blob, column_nr, value [as type], [column_nr, value [as type]]...);
COLUMN_ADD(dyncol_blob, column_name, value [as type], [column_name, value [as type]]...);
```

Description

Adds or updates dynamic columns.

- **`dyncol_blob`** must be either a valid dynamic columns blob (for example, `COLUMN_CREATE` returns such blob), or an empty string.
- **`column_name`** specifies the name of the column to be added. If `dyncol_blob` already has a column with this name, it will be overwritten.
- **`value`** specifies the new value for the column. Passing a NULL value will cause the column to be deleted.
- **`as type`** is optional. See #datatypes section for a discussion about types.

The return value is a dynamic column blob after the modifications.

Examples

```
UPDATE t1 SET dyncol_blob=COLUMN_ADD(dyncol_blob, "column_name", "value") WHERE id=1;
```

Note: `COLUMN_ADD()` is a regular function (just like `CONCAT()`), hence, in order to update the value in the table you have to use the `UPDATE ... SET dynamic_col=COLUMN_ADD(dynamic_col, ....)` pattern.

1.2.2.2.2 COLUMN_CHECK

Syntax

```
COLUMN_CHECK(dyncol_blob);
```

Description

Check if `dyncol_blob` is a valid packed dynamic columns blob. Return value of 1 means the blob is valid, return value of 0 means it is not.

Rationale: Normally, one works with valid dynamic column blobs. Functions like COLUMN_CREATE, COLUMN_ADD, COLUMN_DELETE always return valid dynamic column blobs. However, if a dynamic column blob is accidentally truncated, or transcoded from one character set to another, it will be corrupted. This function can be used to check if a value in a blob field is a valid dynamic column blob.

1.2.2.2.3 COLUMN_CREATE

Syntax

```
COLUMN_CREATE(column_nr, value [as type], [column_nr, value [as type]]...);
COLUMN_CREATE(column_name, value [as type], [column_name, value [as type]]...);
```

Description

Returns a dynamic columns blob that stores the specified columns with values.

The return value is suitable for

- storing in a table
- further modification with other dynamic columns functions

The `as type` part allows one to specify the value type. In most cases, this is redundant because MariaDB will be able to deduce the type of the value. Explicit type specification may be needed when the type of the value is not apparent. For example, a literal `'2012-12-01'` has a CHAR type by default, one will need to specify `'2012-12-01' AS DATE` to have it stored as a date. See Dynamic Columns:Datatypes for further details.

Examples

```
INSERT INTO tbl SET dyncol_blob=COLUMN_CREATE("column_name", "value");
```

1.2.2.2.4 COLUMN_DELETE

Syntax

```
COLUMN_DELETE(dyncol_blob, column_nr, column_nr...);
COLUMN_DELETE(dyncol_blob, column_name, column_name...);
```

Description

Deletes a dynamic column with the specified name. Multiple names can be given. The return value is a dynamic column blob after the modification.

1.2.2.2.5 COLUMN_EXISTS

Syntax

```
COLUMN_EXISTS(dyncol_blob, column_nr);
COLUMN_EXISTS(dyncol_blob, column_name);
```

Description

Checks if a column with name `column_name` exists in `dyncol_blob`. If yes, return `1`, otherwise return `0`. See dynamic columns for more information.

1.2.2.2.6 COLUMN_GET

Syntax

```
COLUMN_GET(dyncol_blob, column_nr as type);
COLUMN_GET(dyncol_blob, column_name as type);
```

Description

Gets the value of a dynamic column by its name. If no column with the given name exists, `NULL` will be returned.

`column_name as type` requires that one specify the datatype of the dynamic column they are reading.

This may seem counter-intuitive: why would one need to specify which datatype they're retrieving? Can't the dynamic columns system figure the datatype from the data being stored?

The answer is: SQL is a statically-typed language. The SQL interpreter needs to know the datatypes of all expressions before the query is run (for example, when one is using prepared statements and runs `"select COLUMN_GET(...)"`, the prepared statement API requires the server to inform the client about the datatype of the column being read before the query is executed and the server can see what datatype the column actually has).

Lengths

If you're running queries like:

```
SELECT COLUMN_GET(blob, 'colname' as CHAR) ...
```

without specifying a maximum length (i.e. using `as CHAR`, not `as CHAR(n)`), MariaDB will report the maximum length of the resultset column to be 16,777,216. This may cause excessive memory usage in some client libraries, because they try to pre-allocate a buffer of maximum resultset width. To avoid this problem, use CHAR(n) whenever you're using COLUMN_GET in the select list.

See Dynamic Columns:Datatypes for more information about datatypes.

1.2.2.2.7 COLUMN_JSON

Syntax

```
COLUMN_JSON(dyncol_blob)
```

Description

Returns a JSON representation of data in `dyncol_blob`. Can also be used to display nested columns. See dynamic columns for more information.

Example

```
select item_name, COLUMN_JSON(dynamic_cols) from assets;
+-----------------+-----------------------------------------+
| item_name       | COLUMN_JSON(dynamic_cols)               |
+-----------------+-----------------------------------------+
| MariaDB T-shirt | {"size":"XL","color":"blue"}            |
| Thinkpad Laptop | {"color":"black","warranty":"3 years"} |
+-----------------+-----------------------------------------+
```

Limitation: `COLUMN_JSON` will decode nested dynamic columns at a nesting level of not more than 10 levels deep. Dynamic columns that are nested deeper than 10 levels will be shown as BINARY string, without encoding.

1.2.2.2.8 COLUMN_LIST

Syntax

```
COLUMN_LIST(dyncol_blob);
```

Description

Returns a comma-separated list of column names. The names are quoted with backticks.

See dynamic columns for more information.

1.2.2.3 ASCII

Syntax

```
ASCII(str)
```

Description

Returns the numeric ASCII value of the leftmost character of the string argument. Returns `0` if the given string is empty and `NULL` if it is `NULL`.

`ASCII()` works for 8-bit characters.

Examples

```
SELECT ASCII(9);
+----------+
| ASCII(9) |
+----------+
|       57 |
+----------+

SELECT ASCII('9');
+------------+
| ASCII('9') |
+------------+
|         57 |
+------------+

SELECT ASCII('abc');
+--------------+
| ASCII('abc') |
+--------------+
|           97 |
+--------------+
```

1.2.2.4 BIN

Syntax

```
BIN(N)
```

Description

Returns a string representation of the binary value of the given longlong (that is, `BIGINT`) number. This is equivalent to `CONV(N,10,2)`. The argument should be positive. If it is a `FLOAT`, it will be truncated. Returns `NULL` if the argument is `NULL`.

Examples

```
SELECT BIN(12);
+---------+
| BIN(12) |
+---------+
| 1100    |
+---------+
```

1.2.2.5 BINARY Operator

This page describes the BINARY operator. For details about the data type, see Binary Data Type.

Syntax

```
BINARY
```

Description

The `BINARY` operator casts the string following it to a binary string. This is an easy way to force a column comparison to be done byte by byte rather than character by character. This causes the comparison to be case sensitive even if the column isn't defined as `BINARY` or `BLOB`.

`BINARY` also causes trailing spaces to be significant.

Examples

```
SELECT 'a' = 'A';
+-----------+
| 'a' = 'A' |
+-----------+
|         1 |
+-----------+

SELECT BINARY 'a' = 'A';
+------------------+
| BINARY 'a' = 'A' |
+------------------+
|                0 |
+------------------+

SELECT 'a' = 'a ';
+------------+
| 'a' = 'a ' |
+------------+
|          1 |
+------------+

SELECT BINARY 'a' = 'a ';
+-------------------+
| BINARY 'a' = 'a ' |
+-------------------+
|                 0 |
+-------------------+
```

1.2.2.6 BIT_LENGTH

Syntax

```
BIT_LENGTH(str)
```

Contents

Description

Returns the length of the given string argument in bits. If the argument is not a string, it will be converted to string. If the argument is `NULL`, it returns `NULL`.

Examples

```
SELECT BIT_LENGTH('text');
+--------------------+
| BIT_LENGTH('text') |
+--------------------+
|                 32 |
+--------------------+
```

```
SELECT BIT_LENGTH('');
+----------------+
| BIT_LENGTH('') |
+----------------+
|              0 |
+----------------+
```

Compatibility

PostgreSQL and Sybase support BIT_LENGTH().

1.2.2.7 CAST

Syntax

```
CAST(expr AS type)
```

Contents

Description

The `CAST()` function takes a value of one type and produces a value of another type, similar to the CONVERT() function.

The type can be one of the following values:

- BINARY
- CHAR
- DATE
- DATETIME
- DECIMAL[(M[,D])]
- DOUBLE
- FLOAT (from MariaDB 10.4.5)
- INTEGER
  - Short for `SIGNED INTEGER`
- SIGNED [INTEGER]
- UNSIGNED [INTEGER]
- TIME
- VARCHAR (in Oracle mode, from MariaDB 10.3)

The main difference between `CAST` and CONVERT() is that `CONVERT(expr,type)` is ODBC syntax while `CAST(expr as type)` and `CONVERT(... USING ...)` are SQL92 syntax.

In MariaDB 10.4 and later, you can use the `CAST()` function with the `INTERVAL` keyword.

Until MariaDB 5.5.31, `X'HHHH'`, the standard SQL syntax for binary string literals, erroneously worked in the same way as `0xHHHH`. In 5.5.31 it was intentionally changed to behave as a string in all contexts (and never as a number).

This introduced an incompatibility with previous versions of MariaDB, and all versions of MySQL (see the example below).

Examples

Simple casts:

```
SELECT CAST("abc" AS BINARY);
SELECT CAST("1" AS UNSIGNED INTEGER);
SELECT CAST(123 AS CHAR CHARACTER SET utf8)
```

Note that when one casts to CHAR without specifying the character set, the collation_connection character set collation will be used. When used with `CHAR CHARACTER SET`, the default collation for that character set will be used.

```
SELECT COLLATION(CAST(123 AS CHAR));
+-----------------------------+
| COLLATION(CAST(123 AS CHAR)) |
+-----------------------------+
| latin1_swedish_ci           |
+-----------------------------+

SELECT COLLATION(CAST(123 AS CHAR CHARACTER SET utf8));
+-----------------------------------------------+
| COLLATION(CAST(123 AS CHAR CHARACTER SET utf8)) |
+-----------------------------------------------+
| utf8_general_ci                               |
+-----------------------------------------------+
```

If you also want to change the collation, you have to use the `COLLATE` operator:

```
SELECT COLLATION(CAST(123 AS CHAR CHARACTER SET utf8)
  COLLATE utf8_unicode_ci);
+-----------------------------------------------------------------------------+
| COLLATION(CAST(123 AS CHAR CHARACTER SET utf8) COLLATE utf8_unicode_ci) |
+-----------------------------------------------------------------------------+
| utf8_unicode_ci                                                             |
+-----------------------------------------------------------------------------+
```

Using `CAST()` to order an `ENUM` field as a `CHAR` rather than the internal numerical value:

```
CREATE TABLE enum_list (enum_field enum('c','a','b'));

INSERT INTO enum_list (enum_field)
VALUES('c'),('a'),('c'),('b');

SELECT * FROM enum_list
ORDER BY enum_field;
+------------+
| enum_field |
+------------+
| c          |
| c          |
| a          |
| b          |
+------------+

SELECT * FROM enum_list
ORDER BY CAST(enum_field AS CHAR);
+------------+
| enum_field |
+------------+
| a          |
| b          |
| c          |
| c          |
+------------+
```

From MariaDB 5.5.31, the following will trigger warnings, since `x'aa'` and `'X'aa'` no longer behave as a number. Previously, and in all versions of MySQL, no warnings are triggered since they did erroneously behave as a number:

```
SELECT CAST(0xAA AS UNSIGNED), CAST(x'aa' AS UNSIGNED), CAST(X'aa' AS UNSIGNED);
+------------------------+-------------------------+-------------------------+
| CAST(0xAA AS UNSIGNED) | CAST(x'aa' AS UNSIGNED) | CAST(X'aa' AS UNSIGNED) |
+------------------------+-------------------------+-------------------------+
|                    170 |                       0 |                       0 |
+------------------------+-------------------------+-------------------------+
1 row in set, 2 warnings (0.00 sec)

Warning (Code 1292): Truncated incorrect INTEGER value: '\xAA'
Warning (Code 1292): Truncated incorrect INTEGER value: '\xAA'
```

Casting to intervals:

```
SELECT CAST(2019-01-04 INTERVAL AS DAY_SECOND(2)) AS "Cast";

+-------------+
| Cast        |
+-------------+
| 00:20:17.00 |
+-------------+
```

1.2.2.8 CHAR Function

Syntax

```
CHAR(N,... [USING charset_name])
```

Description

`CHAR()` interprets each argument as an `INT` and returns a string consisting of the characters given by the code values of those integers. `NULL` values are skipped. By default, `CHAR()` returns a binary string. To produce a string in a given character set, use the optional `USING` clause:

```
SELECT CHARSET(CHAR(0x65)), CHARSET(CHAR(0x65 USING utf8));
+---------------------+-------------------------------+
| CHARSET(CHAR(0x65)) | CHARSET(CHAR(0x65 USING utf8)) |
+---------------------+-------------------------------+
| binary              | utf8                          |
+---------------------+-------------------------------+
```

If `USING` is given and the result string is illegal for the given character set, a warning is issued. Also, if strict SQL mode is enabled, the result from `CHAR()` becomes `NULL`.

Examples

```
SELECT CHAR(77,97,114,'105',97,'68',66);
+----------------------------------+
| CHAR(77,97,114,'105',97,'68',66) |
+----------------------------------+
| MariaDB                          |
+----------------------------------+

SELECT CHAR(77,77.3,'77.3');
+----------------------+
| CHAR(77,77.3,'77.3') |
+----------------------+
| MMM                  |
+----------------------+
1 row in set, 1 warning (0.00 sec)

Warning (Code 1292): Truncated incorrect INTEGER value: '77.3'
```

1.2.2.9 CHAR_LENGTH

Syntax

```
CHAR_LENGTH(str)
CHARACTER_LENGTH(str)
```

Contents

Description

Returns the length of the given string argument, measured in characters. A multi-byte character counts as a single character. This means that for a string containing five two-byte characters, LENGTH() (or OCTET_LENGTH() in Oracle mode) returns 10, whereas `CHAR_LENGTH()` returns 5. If the argument is `NULL`, it returns `NULL`.

If the argument is not a string value, it is converted into a string.

It is synonymous with the `CHARACTER_LENGTH()` function.

Examples

```
SELECT CHAR_LENGTH('MariaDB');
+------------------------+
| CHAR_LENGTH('MariaDB') |
+------------------------+
|                      7 |
+------------------------+
```

When Oracle mode from MariaDB 10.3 is not set:

```
SELECT CHAR_LENGTH('π'), LENGTH('π'), LENGTHB('π'), OCTET_LENGTH('π');
+--------------------+--------------+---------------+--------------------+
| CHAR_LENGTH('π')   | LENGTH('π')  | LENGTHB('π')  | OCTET_LENGTH('π')  |
+--------------------+--------------+---------------+--------------------+
|                  1 |            2 |             2 |                  2 |
+--------------------+--------------+---------------+--------------------+
```

In Oracle mode from MariaDB 10.3:

```
SELECT CHAR_LENGTH('π'), LENGTH('π'), LENGTHB('π'), OCTET_LENGTH('π');
+--------------------+--------------+---------------+--------------------+
| CHAR_LENGTH('π')   | LENGTH('π')  | LENGTHB('π')  | OCTET_LENGTH('π')  |
+--------------------+--------------+---------------+--------------------+
|                  1 |            1 |             2 |                  2 |
+--------------------+--------------+---------------+--------------------+
```

1.2.2.10 CHARACTER_LENGTH

Syntax

```
CHARACTER_LENGTH(str)
```

Description

`CHARACTER_LENGTH()` is a synonym for `CHAR_LENGTH()`.

1.2.2.11 CHR

Syntax

```
CHR(N)
```

Description

`CHR()` interprets each argument N as an integer and returns a `VARCHAR(1)` string consisting of the character given by the code values of the integer. The character set and collation of the string are set according to the values of the `character_set_database` and `collation_database` system variables.

`CHR()` is similar to the `CHAR()` function, but only accepts a single argument.

`CHR()` is available in all sql_modes.

Examples

```
SELECT CHR(67);
+---------+
| CHR(67) |
+---------+
| C       |
+---------+

SELECT CHR('67');
+-----------+
| CHR('67') |
+-----------+
| C         |
+-----------+

SELECT CHR('C');
+----------+
| CHR('C') |
+----------+
|          |
+----------+
1 row in set, 1 warning (0.000 sec)

SHOW WARNINGS;
+---------+------+----------------------------------------+
| Level   | Code | Message                                |
+---------+------+----------------------------------------+
| Warning | 1292 | Truncated incorrect INTEGER value: 'C' |
+---------+------+----------------------------------------+
```

1.2.2.12 CONCAT

Syntax

```
CONCAT(str1,str2,...)
```

Contents

Description

Returns the string that results from concatenating the arguments. May have one or more arguments. If all arguments are non-binary strings, the result is a non-binary string. If the arguments include any binary strings, the result is a binary string. A numeric argument is converted to its equivalent binary string form; if you want to avoid that, you can use an explicit type cast, as in this example:

```
SELECT CONCAT(CAST(int_col AS CHAR), char_col);
```

`CONCAT()` returns `NULL` if any argument is `NULL`.

A `NULL` parameter hides all information contained in other parameters from the result. Sometimes this is not desirable; to avoid this, you can:

- Use the `CONCAT_WS()` function with an empty separator, because that function is `NULL`-safe.
- Use `IFNULL()` to turn NULLs into empty strings.

Oracle Mode

In Oracle mode, `CONCAT` ignores NULL.

Examples

```
SELECT CONCAT('Ma', 'ria', 'DB');
+--------------------------+
| CONCAT('Ma', 'ria', 'DB') |
+--------------------------+
| MariaDB                  |
+--------------------------+

SELECT CONCAT('Ma', 'ria', NULL, 'DB');
+---------------------------------+
| CONCAT('Ma', 'ria', NULL, 'DB') |
+---------------------------------+
| NULL                            |
+---------------------------------+

SELECT CONCAT(42.0);
+-------------+
| CONCAT(42.0) |
+-------------+
| 42.0         |
+-------------+
```

Using `IFNULL()` to handle NULLs:

```
SELECT CONCAT('The value of @v is: ', IFNULL(@v, ''));
+-----------------------------------------------+
| CONCAT('The value of @v is: ', IFNULL(@v, '')) |
+-----------------------------------------------+
| The value of @v is:                           |
+-----------------------------------------------+
```

In Oracle mode, from MariaDB 10.3:

```
SELECT CONCAT('Ma', 'ria', NULL, 'DB');
+---------------------------------+
| CONCAT('Ma', 'ria', NULL, 'DB') |
+---------------------------------+
| MariaDB                         |
+---------------------------------+
```

1.2.2.13 CONCAT_WS

Syntax

```
CONCAT_WS(separator,str1,str2,...)
```

Description

`CONCAT_WS()` stands for Concatenate With Separator and is a special form of `CONCAT()`. The first argument is the separator for the rest of the arguments. The separator is added between the strings to be concatenated. The separator can be a string, as can the rest of the arguments.

If the separator is `NULL`, the result is `NULL`; all other `NULL` values are skipped. This makes `CONCAT_WS()` suitable when you want to concatenate some values and avoid losing all information if one of them is `NULL`.

Examples

```
SELECT CONCAT_WS(',','First name','Second name','Last Name');
+----------------------------------------------------+
| CONCAT_WS(',','First name','Second name','Last Name') |
+----------------------------------------------------+
| First name,Second name,Last Name                   |
+----------------------------------------------------+

SELECT CONCAT_WS('-','Floor',NULL,'Room');
+-----------------------------------+
| CONCAT_WS('-','Floor',NULL,'Room') |
+-----------------------------------+
| Floor-Room                        |
+-----------------------------------+
```

In some cases, remember to include a space in the separator string:

```
SET @a = 'gnu', @b = 'penguin', @c = 'sea lion';
Query OK, 0 rows affected (0.00 sec)

SELECT CONCAT_WS(', ', @a, @b, @c);
+-----------------------------+
| CONCAT_WS(', ', @a, @b, @c) |
+-----------------------------+
| gnu, penguin, sea lion      |
+-----------------------------+
```

Using `CONCAT_WS()` to handle `NULL`s:

```
SET @a = 'a', @b = NULL, @c = 'c';

SELECT CONCAT_WS('', @a, @b, @c);
+--------------------------+
| CONCAT_WS('', @a, @b, @c) |
+--------------------------+
| ac                       |
+--------------------------+
```

1.2.2.14 CONVERT

Syntax

```
CONVERT(expr,type), CONVERT(expr USING transcoding_name)
```

Contents

Description

The `CONVERT()` and CAST() functions take a value of one type and produce a value of another type.

The type can be one of the following values:

- BINARY
- CHAR
- DATE
- DATETIME
- DECIMAL[(M[,D])]
- DOUBLE
- FLOAT (from MariaDB 10.4.5)
- INTEGER
  - Short for `SIGNED INTEGER`
- SIGNED [INTEGER]
- UNSIGNED [INTEGER]
- TIME
- VARCHAR (in Oracle mode, from MariaDB 10.3)

Note that in MariaDB, `INT` and `INTEGER` are the same thing.

`BINARY` produces a string with the BINARY data type. If the optional length is given, `BINARY(N)` causes the cast to use no more than `N` bytes of the argument. Values shorter than the given number in bytes are padded with 0x00 bytes to make them equal the length value.

`CHAR(N)` causes the cast to use no more than the number of characters given in the argument.

The main difference between the CAST() and `CONVERT()` is that `CONVERT(expr,type)` is ODBC syntax while CAST(expr as type) and `CONVERT(... USING ...)` are SQL92 syntax.

`CONVERT()` with `USING` is used to convert data between different character sets. In MariaDB, transcoding names are the same as the corresponding character set names. For example, this statement converts the string 'abc' in the default character set to the corresponding string in the utf8 character set:

```
SELECT CONVERT('abc' USING utf8);
```

Examples

```
SELECT enum_col FROM tbl_name
ORDER BY CAST(enum_col AS CHAR);
```

Converting a BINARY to string to permit the LOWER function to work:

```
SET @x = 'AardVark';

SET @x = BINARY 'AardVark';

SELECT LOWER(@x), LOWER(CONVERT (@x USING latin1));
+-----------+-----------------------------------+
| LOWER(@x) | LOWER(CONVERT (@x USING latin1)) |
+-----------+-----------------------------------+
| AardVark  | aardvark                          |
+-----------+-----------------------------------+
```

1.2.2.15 ELT

Syntax

```
ELT(N, str1[, str2, str3,...])
```

Description

Takes a numeric argument and a series of string arguments. Returns the string that corresponds to the given numeric position. For instance, it returns `str1` if `N` is 1, `str2` if `N` is 2, and so on. If the numeric argument is a `FLOAT`, MariaDB rounds it to the nearest `INTEGER`. If the numeric argument is less than 1, greater than the total number of arguments, or not a number, `ELT()` returns `NULL`. It must have at least two arguments.

It is complementary to the `FIELD()` function.

Examples

```
SELECT ELT(1, 'ej', 'Heja', 'hej', 'foo');
+----------------------------------+
| ELT(1, 'ej', 'Heja', 'hej', 'foo') |
+----------------------------------+
| ej                               |
+----------------------------------+

SELECT ELT(4, 'ej', 'Heja', 'hej', 'foo');
+----------------------------------+
| ELT(4, 'ej', 'Heja', 'hej', 'foo') |
+----------------------------------+
| foo                              |
+----------------------------------+
```

1.2.2.16 EXPORT_SET

Syntax

```
EXPORT_SET(bits, on, off[, separator[, number_of_bits]])
```

Description

Takes a minimum of three arguments. Returns a string where each bit in the given `bits` argument is returned, with the string values given for `on` and `off`.

Bits are examined from right to left, (from low-order to high-order bits). Strings are added to the result from left to right, separated by a separator string (defaults as ' , '). You can optionally limit the number of bits the `EXPORT_SET()` function examines using the `number_of_bits` option.

If any of the arguments are set as `NULL`, the function returns `NULL`.

Examples

```
SELECT EXPORT_SET(5,'Y','N',',',4);
+-----------------------------+
| EXPORT_SET(5,'Y','N',',',4) |
+-----------------------------+
| Y,N,Y,N                     |
+-----------------------------+

SELECT EXPORT_SET(6,'1','0',',',10);
+------------------------------+
| EXPORT_SET(6,'1','0',',',10) |
+------------------------------+
| 0,1,1,0,0,0,0,0,0,0          |
+------------------------------+
```

1.2.2.17 EXTRACTVALUE

Syntax

```
EXTRACTVALUE(xml_frag, xpath_expr)
```

Contents

Description

The `EXTRACTVALUE()` function takes two string arguments: a fragment of XML markup and an XPath expression, (also known as a locator). It returns the text (That is, CDDATA), of the first text node which is a child of the element or elements matching the XPath expression.

In cases where a valid XPath expression does not match any text nodes in a valid XML fragment, (including the implicit `/text()` expression), the `EXTRACTVALUE()` function returns an empty string.

Invalid Arguments

When either the XML fragment or the XPath expression is `NULL`, the `EXTRACTVALUE()` function returns `NULL`. When the XML fragment is invalid, it raises a warning Code 1525:

```
Warning (Code 1525): Incorrect XML value: 'parse error at line 1 pos 11: unexpected END-OF-
INPUT'
```

When the XPath value is invalid, it generates an Error 1105:

```
ERROR 1105 (HY000): XPATH syntax error: ')'
```

Explicit text() Expressions

This function is the equivalent of performing a match using the XPath expression after appending `/text()`. In other words:

```
SELECT
   EXTRACTVALUE('<cases><case>example</case></cases>', '/cases/case')
    AS 'Base Example',
   EXTRACTVALUE('<cases><case>example</case></cases>', '/cases/case/text()')
    AS 'text() Example';
+--------------+----------------+
| Base Example | text() Example |
+--------------+----------------+
| example      | example        |
+--------------+----------------+
```

Count Matches

When `EXTRACTVALUE()` returns multiple matches, it returns the content of the first child text node of each matching element, in the matched order, as a single, space-delimited string.

By design, the `EXTRACTVALUE()` function makes no distinction between a match on an empty element and no match at all. If you need to determine whether no matching element was found in the XML fragment or if an element was found that contained no child text nodes, use the XPath `count()` function.

For instance, when looking for a value that exists, but contains no child text nodes, you would get a count of the number of matching instances:

```
SELECT
   EXTRACTVALUE('<cases><case/></cases>', '/cases/case')
    AS 'Empty Example',
   EXTRACTVALUE('<cases><case/></cases>', 'count(/cases/case)')
    AS 'count() Example';
+---------------+-----------------+
| Empty Example | count() Example |
+---------------+-----------------+
|               |               1 |
+---------------+-----------------+
```

Alternatively, when looking for a value that doesn't exist, `count()` returns 0.

```
SELECT
   EXTRACTVALUE('<cases><case/></cases>', '/cases/person')
    AS 'No Match Example',
   EXTRACTVALUE('<cases><case/></cases>', 'count(/cases/person)')
    AS 'count() Example';
+------------------+-----------------+
| No Match Example | count() Example |
+------------------+-----------------+
|                  |               0|
+------------------+-----------------+
```

Matches

Important: The `EXTRACTVALUE()` function only returns CDDATA. It does not return tags that the element might contain or the text that these child elements contain.

```
SELECT
  EXTRACTVALUE('<cases><case>Person<email>x@example.com</email></case></cases>', '/cases')
   AS Case;
+--------+
| Case   |
+--------+
| Person |
+--------+
```

Note, in the above example, while the XPath expression matches to the parent `<case>` instance, it does not return the contained `<email>` tag or its content.

Examples

```
SELECT
    ExtractValue('<a>ccc<b>ddd</b></a>', '/a')            AS val1,
    ExtractValue('<a>ccc<b>ddd</b></a>', '/a/b')          AS val2,
    ExtractValue('<a>ccc<b>ddd</b></a>', '//b')           AS val3,
    ExtractValue('<a>ccc<b>ddd</b></a>', '/b')            AS val4,
    ExtractValue('<a>ccc<b>ddd</b><b>eee</b></a>', '//b') AS val5;
+------+------+------+------+---------+
| val1 | val2 | val3 | val4 | val5    |
+------+------+------+------+---------+
| ccc  | ddd  | ddd  |      | ddd eee |
+------+------+------+------+---------+
```

1.2.2.18 FIELD

Syntax

```
FIELD(pattern, str1[,str2,...])
```

Description

Returns the index position of the string or number matching the given pattern. Returns `0` in the event that none of the arguments match the pattern. Raises an Error 1582 if not given at least two arguments.

When all arguments given to the `FIELD()` function are strings, they are treated as case-insensitive. When all the arguments are numbers, they are treated as numbers. Otherwise, they are treated as doubles.

If the given pattern occurs more than once, the `FIELD()` function only returns the index of the first instance. If the given pattern is `NULL`, the function returns `0`, as a `NULL` pattern always fails to match.

This function is complementary to the `ELT()` function.

Examples

```
SELECT FIELD('ej', 'Hej', 'ej', 'Heja', 'hej', 'foo')
   AS 'Field Results';
+---------------+
| Field Results |
+---------------+
|             2 |
+---------------+

SELECT FIELD('fo', 'Hej', 'ej', 'Heja', 'hej', 'foo')
   AS 'Field Results';
+---------------+
| Field Results |
+---------------+
|             0 |
+---------------+

SELECT FIELD(1, 2, 3, 4, 5, 1) AS 'Field Results';
+---------------+
| Field Results |
+---------------+
|             5 |
+---------------+

SELECT FIELD(NULL, 2, 3) AS 'Field Results';
+---------------+
| Field Results |
+---------------+
|             0 |
+---------------+

SELECT FIELD('fail') AS 'Field Results';
Error 1582 (42000): Incorrect parameter count in call
to native function 'field'
```

1.2.2.19 FIND_IN_SET

Syntax

```
FIND_IN_SET(pattern, strlist)
```

Description

Returns the index position where the given pattern occurs in a string list. The first argument is the pattern you want to search for. The second argument is a string containing comma-separated variables. If the second argument is of the `SET` data-type, the function is optimized to use bit arithmetic.

If the pattern does not occur in the string list or if the string list is an empty string, the function returns `0`. If either argument is `NULL`, the function returns `NULL`. The function does not return the correct result if the pattern contains a comma (" , ") character.

Examples

```
SELECT FIND_IN_SET('b','a,b,c,d') AS "Found Results";
+---------------+
| Found Results |
+---------------+
|             2 |
+---------------+
```

1.2.2.20 FORMAT

Syntax

```
FORMAT(num, decimal_position[, locale])
```

Description

Formats the given number for display as a string, adding separators to appropriate position and rounding the results to the given decimal position. For instance, it would format `15233.345` to `15,233.35`.

If the given decimal position is `0`, it rounds to return no decimal point or fractional part. You can optionally specify a locale value to format numbers to the pattern appropriate for the given region.

Examples

```
SELECT FORMAT(1234567890.09876543210, 4) AS 'Format';
+--------------------+
| Format             |
+--------------------+
| 1,234,567,890.0988 |
+--------------------+

SELECT FORMAT(1234567.89, 4) AS 'Format';
+----------------+
| Format         |
+----------------+
| 1,234,567.8900 |
+----------------+

SELECT FORMAT(1234567.89, 0) AS 'Format';
+-----------+
| Format    |
+-----------+
| 1,234,568 |
+-----------+

SELECT FORMAT(123456789,2,'rm_CH') AS 'Format';
+----------------+
| Format         |
+----------------+
| 123'456'789,00 |
+----------------+
```

1.2.2.21 FROM_BASE64

Syntax

```
FROM_BASE64(str)
```

Description

Decodes the given base-64 encode string, returning the result as a binary string. Returns `NULL` if the given string is `NULL` or if it's invalid.

It is the reverse of the `TO_BASE64` function.

There are numerous methods to base-64 encode a string. MariaDB uses the following:

- It encodes alphabet value 64 as '`+`'.
- It encodes alphabet value 63 as '`/`'.
- It codes output in groups of four printable characters. Each three byte of data encoded uses four characters. If the final group is incomplete, it pads the difference with the '`=`' character.
- It divides long output, adding a new line very 76 characters.
- In decoding, it recognizes and ignores newlines, carriage returns, tabs and space whitespace characters.

```
SELECT TO_BASE64('Maria') AS 'Input';
+-----------+
| Input     |
+-----------+
| TWFyaWE=  |
+-----------+

SELECT FROM_BASE64('TWFyaWE=') AS 'Output';
+--------+
| Output |
+--------+
| Maria  |
+--------+
```

1.2.2.22 HEX

Syntax

```
HEX(N_or_S)
```

Description

If `N_or_S` is a number, returns a string representation of the hexadecimal value of `N`, where `N` is a `longlong` (`BIGINT`) number. This is equivalent to `CONV(N,10,16)`.

If `N_or_S` is a string, returns a hexadecimal string representation of `N_or_S` where each byte of each character in `N_or_S` is converted to two hexadecimal digits. If `N_or_S` is NULL, returns NULL. The inverse of this operation is performed by the UNHEX() function.

MariaDB starting with 10.5.0

HEX() with an INET6 argument returns a hexadecimal representation of the underlying 16-byte binary string.

Examples

```
SELECT HEX(255);
+----------+
| HEX(255) |
+----------+
| FF       |
+----------+

SELECT 0x4D617269614442;
+------------------+
| 0x4D617269614442 |
+------------------+
| MariaDB          |
+------------------+

SELECT HEX('MariaDB');
+----------------+
| HEX('MariaDB') |
+----------------+
| 4D617269614442 |
+----------------+
```

From MariaDB 10.5.0:

```
SELECT HEX(CAST('2001:db8::ff00:42:8329' AS INET6));
+---------------------------------------------+
| HEX(CAST('2001:db8::ff00:42:8329' AS INET6)) |
+---------------------------------------------+
| 20010DB8000000000000FF0000428329             |
+---------------------------------------------+
```

1.2.2.23 INSTR

Syntax

```
INSTR(str,substr)
```

Contents

Description

Returns the position of the first occurrence of substring *substr* in string *str*. This is the same as the two-argument form of LOCATE(), except that the order of the arguments is reversed.

`INSTR()` performs a case-insensitive search.

If any argument is `NULL`, returns `NULL`.

Examples

```
SELECT INSTR('foobarbar', 'bar');
+--------------------------+
| INSTR('foobarbar', 'bar') |
+--------------------------+
|                        4 |
+--------------------------+

SELECT INSTR('My', 'Maria');
+----------------------+
| INSTR('My', 'Maria') |
+----------------------+
|                    0 |
+----------------------+
```

1.2.2.24 LCASE

Syntax

```
LCASE(str)
```

Description

LCASE() is a synonym for LOWER().

1.2.2.25 LEFT

Syntax

```
LEFT(str,len)
```

Description

Returns the leftmost `len` characters from the string `str`, or NULL if any argument is NULL.

Examples

```
SELECT LEFT('MariaDB', 5);
+--------------------+
| LEFT('MariaDB', 5) |
+--------------------+
| Maria              |
+--------------------+
```

1.2.2.26 INSERT Function

Syntax

```
INSERT(str,pos,len,newstr)
```

Description

Returns the string `str`, with the substring beginning at position `pos` and `len` characters long replaced by the string `newstr`. Returns the original string if `pos` is not within the length of the string. Replaces the rest of the string from position

`pos` if `len` is not within the length of the rest of the string. Returns NULL if any argument is NULL.

Examples

```
SELECT INSERT('Quadratic', 3, 4, 'What');
+--------------------------------+
| INSERT('Quadratic', 3, 4, 'What') |
+--------------------------------+
| QuWhattic                      |
+--------------------------------+

SELECT INSERT('Quadratic', -1, 4, 'What');
+---------------------------------+
| INSERT('Quadratic', -1, 4, 'What') |
+---------------------------------+
| Quadratic                       |
+---------------------------------+

SELECT INSERT('Quadratic', 3, 100, 'What');
+-----------------------------------+
| INSERT('Quadratic', 3, 100, 'What') |
+-----------------------------------+
| QuWhat                            |
+-----------------------------------+
```

1.2.2.27 LENGTH

Syntax

```
LENGTH(str)
```

Contents

Description

Returns the length of the string `str`.

In the default mode, when Oracle mode from MariaDB 10.3 is not set, the length is measured in bytes. In this case, a multi-byte character counts as multiple bytes. This means that for a string containing five two-byte characters, `LENGTH()` returns 10, whereas CHAR_LENGTH() returns 5.

When running Oracle mode from MariaDB 10.3, the length is measured in characters, and `LENGTH` is a synonym for CHAR_LENGTH().

If `str` is not a string value, it is converted into a string. If `str` is `NULL`, the function returns `NULL`.

Examples

```
SELECT LENGTH('MariaDB');
+-------------------+
| LENGTH('MariaDB') |
+-------------------+
|                 7 |
+-------------------+
```

When Oracle mode from MariaDB 10.3 is not set:

```
SELECT CHAR_LENGTH('π'), LENGTH('π'), LENGTHB('π'), OCTET_LENGTH('π');
+------------------+-------------+--------------+--------------------+
| CHAR_LENGTH('π') | LENGTH('π') | LENGTHB('π') | OCTET_LENGTH('π') |
+------------------+-------------+--------------+--------------------+
|                1 |           2 |            2 |                  2 |
+------------------+-------------+--------------+--------------------+
```

In Oracle mode from MariaDB 10.3:

```
SELECT CHAR_LENGTH('π'), LENGTH('π'), LENGTHB('π'), OCTET_LENGTH('π');
+------------------+-------------+--------------+--------------------+
| CHAR_LENGTH('π') | LENGTH('π') | LENGTHB('π') | OCTET_LENGTH('π') |
+------------------+-------------+--------------+--------------------+
|                1 |           1 |            2 |                  2 |
+------------------+-------------+--------------+--------------------+
```

1.2.2.28 LENGTHB

MariaDB starting with 10.3.1

Introduced in MariaDB 10.3.1 as part of the Oracle compatibility enhancements.

Syntax

```
LENGTHB(str)
```

Description

`LENGTHB()` returns the length of the given string, in bytes. When Oracle mode is not set, this is a synonym for LENGTH.

A multi-byte character counts as multiple bytes. This means that for a string containing five two-byte characters, `LENGTHB()` returns 10, whereas CHAR_LENGTH() returns 5.

If `str` is not a string value, it is converted into a string. If `str` is `NULL`, the function returns `NULL`.

Examples

When Oracle mode from MariaDB 10.3 is not set:

```
SELECT CHAR_LENGTH('π'), LENGTH('π'), LENGTHB('π'), OCTET_LENGTH('π');
+------------------+-------------+--------------+--------------------+
| CHAR_LENGTH('π') | LENGTH('π') | LENGTHB('π') | OCTET_LENGTH('π') |
+------------------+-------------+--------------+--------------------+
|                1 |           2 |            2 |                  2 |
+------------------+-------------+--------------+--------------------+
```

In Oracle mode from MariaDB 10.3:

```
SELECT CHAR_LENGTH('π'), LENGTH('π'), LENGTHB('π'), OCTET_LENGTH('π');
+------------------+-------------+--------------+--------------------+
| CHAR_LENGTH('π') | LENGTH('π') | LENGTHB('π') | OCTET_LENGTH('π') |
+------------------+-------------+--------------+--------------------+
|                1 |           1 |            2 |                  2 |
+------------------+-------------+--------------+--------------------+
```

1.2.2.29 LIKE

Syntax

```
expr LIKE pat [ESCAPE 'escape_char']
expr NOT LIKE pat [ESCAPE 'escape_char']
```

Contents

Description

Tests whether *expr* matches the pattern *pat*. Returns either 1 (`TRUE`) or 0 (`FALSE`). Both *expr* and *pat* may be any valid expression and are evaluated to strings. Patterns may use the following wildcard characters:

- `%` matches any number of characters, including zero.
- `_` matches any single character.

Use `NOT LIKE` to test if a string does not match a pattern. This is equivalent to using the `NOT` operator on the entire `LIKE` expression.

If either the expression or the pattern is `NULL` , the result is `NULL` .

`LIKE` performs case-insensitive substring matches if the collation for the expression and pattern is case-insensitive. For case-sensitive matches, declare either argument to use a binary collation using `COLLATE` , or coerce either of them to a `BINARY` string using `CAST` . Use `SHOW COLLATION` to get a list of available collations. Collations ending in `_bin` are case-sensitive.

Numeric arguments are coerced to binary strings.

The `_` wildcard matches a single character, not byte. It will only match a multi-byte character if it is valid in the expression's character set. For example, `_` will match `_utf8"€"` , but it will not match `_latin1"€"` because the Euro sign is not a valid latin1 character. If necessary, use `CONVERT` to use the expression in a different character set.

If you need to match the characters `_` or `%` , you must escape them. By default, you can prefix the wildcard characters the backslash character `\` to escape them. The backslash is used both to encode special characters like newlines when a string is parsed as well as to escape wildcards in a pattern after parsing. Thus, to match an actual backslash, you sometimes need to double-escape it as `"\ \ \ \"` .

To avoid difficulties with the backslash character, you can change the wildcard escape character using `ESCAPE` in a `LIKE` expression. The argument to `ESCAPE` must be a single-character string.

Examples

Select the days that begin with "T":

```
CREATE TABLE t1 (d VARCHAR(16));
INSERT INTO t1 VALUES
  ("Monday"), ("Tuesday"), ("Wednesday"),
  ("Thursday"), ("Friday"), ("Saturday"), ("Sunday");
SELECT * FROM t1 WHERE d LIKE "T%";
```

```
SELECT * FROM t1 WHERE d LIKE "T%";
+-----------+
| d         |
+-----------+
| Tuesday   |
| Thursday  |
+-----------+
```

Select the days that contain the substring "es":

```
SELECT * FROM t1 WHERE d LIKE "%es%";
```

```
SELECT * FROM t1 WHERE d LIKE "%es%";
+-----------+
| d         |
+-----------+
| Tuesday   |
| Wednesday |
+-----------+
```

Select the six-character day names:

```
SELECT * FROM t1 WHERE d like "___day";
```

```
SELECT * FROM t1 WHERE d like "___day";
+---------+
| d       |
+---------+
| Monday  |
| Friday  |
| Sunday  |
+---------+
```

With the default collations, `LIKE` is case-insensitive:

```
SELECT * FROM t1 where d like "t%";
```

```
SELECT * FROM t1 where d like "t%";
+----------+
| d        |
+----------+
| Tuesday  |
| Thursday |
+----------+
```

Use `COLLATE` to specify a binary collation, forcing case-sensitive matches:

```
SELECT * FROM t1 WHERE d like "t%" COLLATE latin1_bin;
```

```
SELECT * FROM t1 WHERE d like "t%" COLLATE latin1_bin;
Empty set (0.00 sec)
```

You can include functions and operators in the expression to match. Select dates based on their day name:

```
CREATE TABLE t2 (d DATETIME);
INSERT INTO t2 VALUES
    ("2007-01-30 21:31:07"),
    ("1983-10-15 06:42:51"),
    ("2011-04-21 12:34:56"),
    ("2011-10-30 06:31:41"),
    ("2011-01-30 14:03:25"),
    ("2004-10-07 11:19:34");
SELECT * FROM t2 WHERE DAYNAME(d) LIKE "T%";
```

```
SELECT * FROM t2 WHERE DAYNAME(d) LIKE "T%";
+------------------+
| d                |
+------------------+
| 2007-01-30 21:31 |
| 2011-04-21 12:34 |
| 2004-10-07 11:19 |
+------------------+
3 rows in set, 7 warnings (0.00 sec)
```

Optimizing LIKE

- MariaDB can use indexes for LIKE on string columns in the case where the LIKE doesn't start with `%` or `_`.
- Starting from MariaDB 10.0, one can set the optimizer_use_condition_selectivity variable to 5. If this is done, then the optimizer will read optimizer_selectivity_sampling_limit rows to calculate the selectivity of the LIKE expression before starting to calculate the query plan. This can help speed up some LIKE queries by providing the optimizer with more information about your data.

1.1.1.4.2.4.4 LOAD_FILE

1.2.2.31 LOCATE

Syntax

```
LOCATE(substr,str), LOCATE(substr,str,pos)
```

Description

The first syntax returns the position of the first occurrence of substring `substr` in string `str`. The second syntax returns the position of the first occurrence of substring `substr` in string `str`, starting at position `pos`. Returns 0 if `substr` is not in `str`.

`LOCATE()` performs a case-insensitive search.

If any argument is `NULL`, returns `NULL`.

INSTR() is the same as the two-argument form of `LOCATE()`, except that the order of the arguments is reversed.

Examples

```
SELECT LOCATE('bar', 'foobarbar');
+----------------------------+
| LOCATE('bar', 'foobarbar') |
+----------------------------+
|                          4 |
+----------------------------+

SELECT LOCATE('My', 'Maria');
+-----------------------+
| LOCATE('My', 'Maria') |
+-----------------------+
|                     0 |
+-----------------------+

SELECT LOCATE('bar', 'foobarbar', 5);
+-------------------------------+
| LOCATE('bar', 'foobarbar', 5) |
+-------------------------------+
|                             7 |
+-------------------------------+
```

1.2.2.32 LOWER

Syntax

```
LOWER(str)
LCASE(str)
```

Contents

Description

Returns the string `str` with all characters changed to lowercase according to the current character set mapping. The default is latin1 (cp1252 West European).

`LCASE` is a synonym for `LOWER`

Examples

```
 SELECT LOWER('QUADRATICALLY');
+------------------------+
| LOWER('QUADRATICALLY') |
+------------------------+
| quadratically          |
+------------------------+
```

`LOWER()` (and `UPPER()`) are ineffective when applied to binary strings (`BINARY`, `VARBINARY`, `BLOB`). To perform lettercase conversion, CONVERT the string to a non-binary string:

```
SET @str = BINARY 'North Carolina';

SELECT LOWER(@str), LOWER(CONVERT(@str USING latin1));
+----------------+-----------------------------------+
| LOWER(@str)    | LOWER(CONVERT(@str USING latin1)) |
+----------------+-----------------------------------+
| North Carolina | north carolina                    |
+----------------+-----------------------------------+
```

1.2.2.33 LPAD

Syntax

```
LPAD(str, len [,padstr])
```

Contents

Description

Returns the string `str`, left-padded with the string `padstr` to a length of `len` characters. If `str` is longer than `len`, the return value is shortened to `len` characters. If `padstr` is omitted, the LPAD function pads spaces.

Prior to MariaDB 10.3.1, the `padstr` parameter was mandatory.

Returns NULL if given a NULL argument. If the result is empty (zero length), returns either an empty string or, from MariaDB 10.3.6 with SQL_MODE=Oracle, NULL.

The Oracle mode version of the function can be accessed outside of Oracle mode by using `LPAD_ORACLE` as the function name.

Examples

```
SELECT LPAD('hello',10,'.');
+----------------------+
| LPAD('hello',10,'.') |
+----------------------+
| .....hello           |
+----------------------+

SELECT LPAD('hello',2,'.');
+--------------------+
| LPAD('hello',2,'.') |
+--------------------+
| he                 |
+--------------------+
```

From MariaDB 10.3.1, with the pad string defaulting to space.

```
SELECT LPAD('hello',10);
+------------------+
| LPAD('hello',10) |
+------------------+
|      hello       |
+------------------+
```

Oracle mode version from MariaDB 10.3.6:

```
SELECT LPAD('',0),LPAD_ORACLE('',0);
+-----------+-------------------+
| LPAD('',0) | LPAD_ORACLE('',0) |
+-----------+-------------------+
|           | NULL              |
+-----------+-------------------+
```

1.2.2.34 LTRIM

Syntax

```
LTRIM(str)
```

Contents

Description

Returns the string `str` with leading space characters removed.

Returns NULL if given a NULL argument. If the result is empty, returns either an empty string, or, from MariaDB 10.3.6 with SQL_MODE=Oracle, NULL.

The Oracle mode version of the function can be accessed outside of Oracle mode by using `LTRIM_ORACLE` as the function name.

Examples

```
SELECT QUOTE(LTRIM('   MariaDB   '));
+-------------------------------+
| QUOTE(LTRIM('   MariaDB   ')) |
+-------------------------------+
| 'MariaDB   '                  |
+-------------------------------+
```

Oracle mode version from MariaDB 10.3.6:

```
SELECT LTRIM(''),LTRIM_ORACLE('');
+-----------+-----------------+
| LTRIM('') | LTRIM_ORACLE('') |
+-----------+-----------------+
|           | NULL            |
+-----------+-----------------+
```

1.2.2.35 MAKE_SET

Syntax

```
MAKE_SET(bits,str1,str2,...)
```

Description

Returns a set value (a string containing substrings separated by "," characters) consisting of the strings that have the corresponding bit in bits set. *str1* corresponds to bit 0, *str2* to bit 1, and so on. NULL values in *str1*, *str2*, ... are not appended to the result.

Examples

```
SELECT MAKE_SET(1,'a','b','c');
+------------------------+
| MAKE_SET(1,'a','b','c') |
+------------------------+
| a                      |
+------------------------+

SELECT MAKE_SET(1 | 4,'hello','nice','world');
+-------------------------------------+
| MAKE_SET(1 | 4,'hello','nice','world') |
+-------------------------------------+
| hello,world                         |
+-------------------------------------+

SELECT MAKE_SET(1 | 4,'hello','nice',NULL,'world');
+-----------------------------------------+
| MAKE_SET(1 | 4,'hello','nice',NULL,'world') |
+-----------------------------------------+
| hello                                   |
+-----------------------------------------+

SELECT QUOTE(MAKE_SET(0,'a','b','c'));
+--------------------------------+
| QUOTE(MAKE_SET(0,'a','b','c')) |
+--------------------------------+
| ''                             |
+--------------------------------+
```

1.2.2.36 MATCH AGAINST

Syntax

```
MATCH (col1,col2,...) AGAINST (expr [search_modifier])
```

Description

A special construct used to perform a fulltext search on a fulltext index.

See Fulltext Index Overview for a full description, and Full-text Indexes for more articles on the topic.

Examples

```
CREATE TABLE ft_myisam(copy TEXT,FULLTEXT(copy)) ENGINE=MyISAM;

INSERT INTO ft_myisam(copy) VALUES ('Once upon a time'), ('There was a wicked witch'),
 ('Who ate everybody up');

SELECT * FROM ft_myisam WHERE MATCH(copy) AGAINST('wicked');
+--------------------------+
| copy                     |
+--------------------------+
| There was a wicked witch |
+--------------------------+
```

```
SELECT id, body, MATCH (title,body) AGAINST
    ('Security implications of running MySQL as root'
    IN NATURAL LANGUAGE MODE) AS score
    FROM articles WHERE MATCH (title,body) AGAINST
    ('Security implications of running MySQL as root'
    IN NATURAL LANGUAGE MODE);
+----+-------------------------------------+-----------------+
| id | body                                | score           |
+----+-------------------------------------+-----------------+
|  4 | 1. Never run mysqld as root. 2. ... | 1.5219271183014 |
|  6 | When configured properly, MySQL ... | 1.3114095926285 |
+----+-------------------------------------+-----------------+
```

3.3.3.3.2 Full-Text Index Stopwords

1.2.2.38 MID

Syntax

```
MID(str,pos,len)
```

Description

MID(str,pos,len) is a synonym for SUBSTRING(str,pos,len).

Examples

```
SELECT MID('abcd',4,1);
+-----------------+
| MID('abcd',4,1) |
+-----------------+
| d               |
+-----------------+

SELECT MID('abcd',2,2);
+-----------------+
| MID('abcd',2,2) |
+-----------------+
| bc              |
+-----------------+
```

A negative starting position:

```
SELECT MID('abcd',-2,4);
+------------------+
| MID('abcd',-2,4) |
+------------------+
| cd               |
+------------------+
```

1.2.2.39 NATURAL_SORT_KEY

MariaDB starting with 10.7.0

NATURAL_SORT_KEY was added in MariaDB 10.7.0.

Syntax

```
NATURAL_SORT_KEY(str)
```

Contents

Description

The `NATURAL_SORT_KEY` function is used for sorting that is closer to natural sorting. Strings are sorted in alphabetical order, while numbers are treated in a way such that, for example, `10` is greater than `2`, whereas in other forms of sorting, `2` would be greater than `10`, just like `z` is greater than `ya`.

There are multiple natural sort implementations, differing in the way they handle leading zeroes, fractions, i18n, negatives, decimals and so on.

MariaDB's implementation ignores leading zeroes when performing the sort.

You can use also use `NATURAL_SORT_KEY` with generated columns. The value is not stored permanently in the table. When using a generated column, the virtual column must be longer than the base column to cater for embedded numbers in the string and MDEV-24582.

Examples

Strings and Numbers

```
CREATE TABLE t1 (c TEXT);

INSERT INTO t1 VALUES ('b1'),('a2'),('a11'),('a1');

SELECT c FROM t1;
+------+
| c    |
+------+
| b1   |
| a2   |
| a11  |
| a1   |
+------+

SELECT c FROM t1 ORDER BY c;
+------+
| c    |
+------+
| a1   |
| a11  |
| a2   |
| b1   |
+------+
```

Unsorted, regular sort and natural sort:

```
TRUNCATE t1;

INSERT INTO t1 VALUES
  ('5.5.31'),('10.7.0'),('10.2.1'),
  ('10.1.22'),('10.3.32'),('10.2.12');

SELECT c FROM t1;
+---------+
| c       |
+---------+
| 5.5.31  |
| 10.7.0  |
| 10.2.1  |
| 10.1.22 |
| 10.3.32 |
| 10.2.12 |
+---------+

SELECT c FROM t1 ORDER BY c;
+---------+
| c       |
+---------+
| 10.1.22 |
| 10.2.1  |
| 10.2.12 |
| 10.3.32 |
| 10.7.0  |
| 5.5.31  |
+---------+

SELECT c FROM t1 ORDER BY NATURAL_SORT_KEY(c);
+---------+
| c       |
+---------+
| 5.5.31  |
| 10.1.22 |
| 10.2.1  |
| 10.2.12 |
| 10.3.32 |
| 10.7.0  |
+---------+
```

IPs

Sorting IPs, unsorted, regular sort and natural sort::

```
TRUNCATE t1;

INSERT INTO t1 VALUES
  ('192.167.3.1'),('192.167.1.12'),('100.200.300.400'),
  ('100.50.60.70'),('100.8.9.9'),('127.0.0.1'),('0.0.0.0');

SELECT c FROM t1;
+-----------------+
| c               |
+-----------------+
| 192.167.3.1     |
| 192.167.1.12    |
| 100.200.300.400 |
| 100.50.60.70    |
| 100.8.9.9       |
| 127.0.0.1       |
| 0.0.0.0         |
+-----------------+

SELECT c FROM t1 ORDER BY c;
+-----------------+
| c               |
+-----------------+
| 0.0.0.0         |
| 100.200.300.400 |
| 100.50.60.70    |
| 100.8.9.9       |
| 127.0.0.1       |
| 192.167.1.12    |
| 192.167.3.1     |
+-----------------+

SELECT c FROM t1 ORDER BY NATURAL_SORT_KEY(c);
+-----------------+
| c               |
+-----------------+
| 0.0.0.0         |
| 100.8.9.9       |
| 100.50.60.70    |
| 100.200.300.400 |
| 127.0.0.1       |
| 192.167.1.12    |
| 192.167.3.1     |
+-----------------+
```

Generated Columns

Using with a generated column:

```
CREATE TABLE t(c VARCHAR(3), k VARCHAR(4) AS (NATURAL_SORT_KEY(c)) INVISIBLE);

INSERT INTO t(c) VALUES ('b1'),('a2'),('a11'),('a10');

SELECT * FROM t ORDER by k;
+------+
| c    |
+------+
| a2   |
| a10  |
| a11  |
| b1   |
+------+
```

Note that if the virtual column is not longer, results may not be as expected:

```
CREATE TABLE t2(c VARCHAR(3), k VARCHAR(3) AS (NATURAL_SORT_KEY(c)) INVISIBLE);

INSERT INTO t2(c) VALUES ('b1'),('a2'),('a11'),('a10');

SELECT * FROM t2 ORDER by k;
+------+
| c    |
+------+
| a2   |
| a11  |
| a10  |
| b1   |
+------+
```

Leading Zeroes

Ignoring leading zeroes can lead to undesirable results in certain contexts. For example:

```
CREATE TABLE t3 (a VARCHAR(4));

INSERT INTO t3 VALUES
  ('a1'), ('a001'), ('a10'), ('a001'), ('a10'),
  ('a01'), ('a01'), ('a01b'), ('a01b'), ('a1');

SELECT a FROM t3 ORDER BY a;
+------+
| a    |
+------+
| a001 |
| a001 |
| a01  |
| a01  |
| a01b |
| a01b |
| a1   |
| a1   |
| a10  |
| a10  |
+------+
10 rows in set (0.000 sec)

SELECT a FROM t3 ORDER BY NATURAL_SORT_KEY(a);
+------+
| a    |
+------+
| a1   |
| a01  |
| a01  |
| a001 |
| a001 |
| a1   |
| a01b |
| a01b |
| a10  |
| a10  |
+------+
```

This may not be what we were hoping for in a 'natural' sort. A workaround is to sort by both NATURAL_SORT_KEY and regular sort.

```
SELECT a FROM t3 ORDER BY NATURAL_SORT_KEY(a), a;
+------+
| a    |
+------+
| a001 |
| a001 |
| a01  |
| a01  |
| a1   |
| a1   |
| a01b |
| a01b |
| a10  |
| a10  |
+------+
```

1.2.2.40 NOT LIKE

Syntax

```
expr NOT LIKE pat [ESCAPE 'escape_char']
```

Description

This is the same as NOT (expr LIKE pat [ESCAPE 'escape_char']).

1.2.2.1.3 NOT REGEXP

1.2.2.42 OCTET_LENGTH

Syntax

```
OCTET_LENGTH(str)
```

Description

`OCTET_LENGTH()` returns the length of the given string, in octets (bytes). This is a synonym for LENGTHB(), and, when Oracle mode from MariaDB 10.3 is not set, a synonym for LENGTH().

A multi-byte character counts as multiple bytes. This means that for a string containing five two-byte characters, `OCTET_LENGTH()` returns 10, whereas CHAR_LENGTH() returns 5.

If `str` is not a string value, it is converted into a string. If `str` is `NULL`, the function returns `NULL`.

Examples

When Oracle mode from MariaDB 10.3 is not set:

```
SELECT CHAR_LENGTH('π'), LENGTH('π'), LENGTHB('π'), OCTET_LENGTH('π');
+-------------------+--------------+---------------+--------------------+
| CHAR_LENGTH('π')  | LENGTH('π')  | LENGTHB('π')  | OCTET_LENGTH('π')  |
+-------------------+--------------+---------------+--------------------+
|                 1 |            2 |             2 |                  2 |
+-------------------+--------------+---------------+--------------------+
```

In Oracle mode from MariaDB 10.3:

```
SELECT CHAR_LENGTH('π'), LENGTH('π'), LENGTHB('π'), OCTET_LENGTH('π');
+------------------+--------------+---------------+--------------------+
| CHAR_LENGTH('π') | LENGTH('π')  | LENGTHB('π')  | OCTET_LENGTH('π')  |
+------------------+--------------+---------------+--------------------+
|                1 |            1 |             2 |                  2 |
+------------------+--------------+---------------+--------------------+
```

1.2.2.43 ORD

Syntax

```
ORD(str)
```

Description

If the leftmost character of the string `str` is a multi-byte character, returns the code for that character, calculated from the numeric values of its constituent bytes using this formula:

```
  (1st byte code)
+ (2nd byte code x 256)
+ (3rd byte code x 256 x 256) ...
```

If the leftmost character is not a multi-byte character, ORD() returns the same value as the ASCII() function.

Examples

```
SELECT ORD('2');
+----------+
| ORD('2') |
+----------+
|       50 |
+----------+
```

1.2.2.44 POSITION

Syntax

```
POSITION(substr IN str)
```

Description

POSITION(substr IN str) is a synonym for LOCATE(substr,str).

It's part of ODBC 3.0.

1.2.2.45 QUOTE

Syntax

```
QUOTE(str)
```

Description

Quotes a string to produce a result that can be used as a properly escaped data value in an SQL statement. The string is returned enclosed by single quotes and with each instance of single quote (" ' "), backslash (" \ "), `ASCII NUL`, and Control-Z preceded by a backslash. If the argument is `NULL`, the return value is the word " `NULL` " without enclosing single quotes.

Examples

```
SELECT QUOTE("Don't!");
+-----------------+
| QUOTE("Don't!") |
+-----------------+
| 'Don\'t!'       |
+-----------------+

SELECT QUOTE(NULL);
+-------------+
| QUOTE(NULL) |
+-------------+
| NULL        |
+-------------+
```

1.2.2.46 REPEAT Function

Syntax

```
REPEAT(str,count)
```

Description

Returns a string consisting of the string `str` repeated `count` times. If `count` is less than 1, returns an empty string. Returns NULL if `str` or `count` are NULL.

Examples

```
SELECT QUOTE(REPEAT('MariaDB ',4));
+-----------------------------------+
| QUOTE(REPEAT('MariaDB ',4))       |
+-----------------------------------+
| 'MariaDB MariaDB MariaDB MariaDB ' |
+-----------------------------------+
```

1.2.2.47 REPLACE Function

Syntax

```
REPLACE(str,from_str,to_str)
```

Description

Returns the string `str` with all occurrences of the string `from_str` replaced by the string `to_str`. REPLACE() performs a case-sensitive match when searching for `from_str`.

Examples

```
SELECT REPLACE('www.mariadb.org', 'w', 'Ww');
+--------------------------------------+
| REPLACE('www.mariadb.org', 'w', 'Ww') |
+--------------------------------------+
| WwWwWw.mariadb.org                   |
+--------------------------------------+
```

1.2.2.48 REVERSE

Syntax

```
REVERSE(str)
```

Description

Returns the string `str` with the order of the characters reversed.

Examples

```
SELECT REVERSE('desserts');
+---------------------+
| REVERSE('desserts') |
+---------------------+
| stressed            |
+---------------------+
```

1.2.2.49 RIGHT

Syntax

```
RIGHT(str,len)
```

Description

Returns the rightmost *`len`* characters from the string *`str`*, or NULL if any argument is NULL.

Examples

```
SELECT RIGHT('MariaDB', 2);
+---------------------+
| RIGHT('MariaDB', 2) |
+---------------------+
| DB                  |
+---------------------+
```

1.2.2.50 RPAD

Syntax

```
RPAD(str, len [, padstr])
```

Contents

Description

Returns the string `str`, right-padded with the string `padstr` to a length of `len` characters. If `str` is longer than `len`, the return value is shortened to `len` characters. If `padstr` is omitted, the RPAD function pads spaces.

Prior to MariaDB 10.3.1, the `padstr` parameter was mandatory.

Returns NULL if given a NULL argument. If the result is empty (a length of zero), returns either an empty string, or, from MariaDB 10.3.6 with SQL_MODE=Oracle, NULL.

The Oracle mode version of the function can be accessed outside of Oracle mode by using `RPAD_ORACLE` as the function name.

Examples

```
SELECT RPAD('hello',10,'.');
+----------------------+
| RPAD('hello',10,'.') |
+----------------------+
| hello.....           |
+----------------------+

SELECT RPAD('hello',2,'.');
+---------------------+
| RPAD('hello',2,'.') |
+---------------------+
| he                  |
+---------------------+
```

From MariaDB 10.3.1, with the pad string defaulting to space.

```
SELECT RPAD('hello',30);
+--------------------------------+
| RPAD('hello',30)               |
+--------------------------------+
| hello                          |
+--------------------------------+
```

Oracle mode version from MariaDB 10.3.6:

```
SELECT RPAD('',0),RPAD_ORACLE('',0);
+------------+-------------------+
| RPAD('',0) | RPAD_ORACLE('',0) |
+------------+-------------------+
|            | NULL              |
+------------+-------------------+
```

1.2.2.51 RTRIM

Syntax

```
RTRIM(str)
```

Contents

Description

Returns the string `str` with trailing space characters removed.

Returns NULL if given a NULL argument. If the result is empty, returns either an empty string, or, from MariaDB 10.3.6 with SQL_MODE=Oracle, NULL.

The Oracle mode version of the function can be accessed outside of Oracle mode by using `RTRIM_ORACLE` as the function name.

Examples

```
SELECT QUOTE(RTRIM('MariaDB    '));
+-----------------------------+
| QUOTE(RTRIM('MariaDB    ')) |
+-----------------------------+
| 'MariaDB'                   |
+-----------------------------+
```

Oracle mode version from MariaDB 10.3.6:

```
SELECT RTRIM(''),RTRIM_ORACLE('');
+-----------+------------------+
| RTRIM('') | RTRIM_ORACLE('') |
+-----------+------------------+
|           | NULL             |
+-----------+------------------+
```

1.2.2.52 SFORMAT

MariaDB starting with 10.7.0
SFORMAT was added in MariaDB 10.7.0.

Description

The `SFORMAT` function takes an input string and a formatting specification and returns the string formatted using the rules the user passed in the specification.

It uses the fmtlib library for Python-like (as well as Rust, C++20, etc) string formatting.

Only fmtlib 7.0.0+ is supported.

There is no native support for temporal and decimal values:

- TIME_RESULT is handled as STRING_RESULT
- DECIMAL_RESULT as REAL_RESULT

Examples

```
SELECT SFORMAT("The answer is {}.", 42);
+---------------------------------+
| SFORMAT("The answer is {}.", 42) |
+---------------------------------+
| The answer is 42.               |
+---------------------------------+

CREATE TABLE test_sformat(mdb_release char(6), mdev int, feature char(20));

INSERT INTO test_sformat VALUES('10.7.0', 25015, 'Python style sformat'),
  ('10.7.0', 4958, 'UUID');

SELECT * FROM test_sformat;
+-------------+-------+----------------------+
| mdb_release | mdev  | feature              |
+-------------+-------+----------------------+
| 10.7.0      | 25015 | Python style sformat |
| 10.7.0      |  4958 | UUID                 |
+-------------+-------+----------------------+

SELECT SFORMAT('MariaDB Server {} has a preview for MDEV-{} which is about {}',
  mdb_release, mdev, feature) AS 'Preview Release Examples'
  FROM test_sformat;
+-----------------------------------------------------------------------------------------+
| Preview Release Examples                                                                |
+-----------------------------------------------------------------------------------------+
| MariaDB Server 10.7.0 has a preview for MDEV-25015 which is about Python style sformat |
| MariaDB Server 10.7.0 has a preview for MDEV-4958 which is about UUID                   |
+-----------------------------------------------------------------------------------------+
```

1.2.2.53 SOUNDEX

Syntax

```
SOUNDEX(str)
```

Description

Returns a soundex string from *str*. Two strings that sound almost the same should have identical soundex strings. A standard soundex string is four characters long, but the `SOUNDEX()` function returns an arbitrarily long string. You can use `SUBSTRING()` on the result to get a standard soundex string. All non-alphabetic characters in *str* are ignored. All international alphabetic characters outside the A-Z range are treated as vowels.

Important: When using SOUNDEX(), you should be aware of the following details:

- This function, as currently implemented, is intended to work well with strings that are in the English language only. Strings in other languages may not produce reasonable results.
- This function implements the original Soundex algorithm, not the more popular enhanced version (also described by D. Knuth). The difference is that original version discards vowels first and duplicates second, whereas the enhanced version discards duplicates first and vowels second.

Examples

```
SOUNDEX('Hello');
+------------------+
| SOUNDEX('Hello') |
+------------------+
| H400             |
+------------------+
```

```
SELECT SOUNDEX('MariaDB');
+--------------------+
| SOUNDEX('MariaDB') |
+--------------------+
| M631               |
+--------------------+
```

```
SELECT SOUNDEX('Knowledgebase');
+--------------------------+
| SOUNDEX('Knowledgebase') |
+--------------------------+
| K543212                  |
+--------------------------+
```

```
SELECT givenname, surname FROM users WHERE SOUNDEX(givenname) = SOUNDEX("robert");
+-----------+---------+
| givenname | surname |
+-----------+---------+
| Roberto   | Castro  |
+-----------+---------+
```

1.2.2.54 SOUNDS LIKE

Syntax

```
expr1 SOUNDS LIKE expr2
```

Description

This is the same as `SOUNDEX(expr1) = SOUNDEX(expr2)`.

Example

```
SELECT givenname, surname FROM users WHERE givenname SOUNDS LIKE "robert";
+-----------+---------+
| givenname | surname |
+-----------+---------+
| Roberto   | Castro  |
+-----------+---------+
```

1.2.2.55 SPACE

Syntax

```
SPACE(N)
```

Description

Returns a string consisting of *N* space characters. If `N` is NULL, returns NULL.

Examples

```
SELECT QUOTE(SPACE(6));
+-----------------+
| QUOTE(SPACE(6)) |
+-----------------+
| '      '        |
+-----------------+
```

1.2.2.56 STRCMP

Syntax

```
STRCMP(expr1,expr2)
```

Description

`STRCMP()` returns `0` if the strings are the same, `-1` if the first argument is smaller than the second according to the current sort order, and `1` if the strings are otherwise not the same. Returns `NULL` is either argument is `NULL`.

Examples

```
SELECT STRCMP('text', 'text2');
+-------------------------+
| STRCMP('text', 'text2') |
+-------------------------+
|                      -1 |
+-------------------------+

SELECT STRCMP('text2', 'text');
+-------------------------+
| STRCMP('text2', 'text') |
+-------------------------+
|                       1 |
+-------------------------+

SELECT STRCMP('text', 'text');
+-----------------------+
| STRCMP('text', 'text') |
+-----------------------+
|                     0 |
+-----------------------+
```

1.2.2.57 SUBSTR

Description

`SUBSTR()` is a synonym for `SUBSTRING()`.

1.2.2.58 SUBSTRING

Syntax

```
SUBSTRING(str,pos),
SUBSTRING(str FROM pos),
SUBSTRING(str,pos,len),
SUBSTRING(str FROM pos FOR len)

SUBSTR(str,pos),
SUBSTR(str FROM pos),
SUBSTR(str,pos,len),
SUBSTR(str FROM pos FOR len)
```

Contents

Description

The forms without a *len* argument return a substring from string *str* starting at position *pos*.

The forms with a *len* argument return a substring *len* characters long from string *str*, starting at position *pos*.

The forms that use *FROM* are standard SQL syntax.

It is also possible to use a negative value for *pos*. In this case, the beginning of the substring is *pos* characters from the end of the string, rather than the beginning. A negative value may be used for *pos* in any of the forms of this function.

By default, the position of the first character in the string from which the substring is to be extracted is reckoned as 1. For Oracle-compatibility, from MariaDB 10.3.3, when sql_mode is set to 'oracle', position zero is treated as position 1 (although the first character is still reckoned as 1).

If any argument is `NULL`, returns `NULL`.

Examples

```
SELECT SUBSTRING('Knowledgebase',5);
+-----------------------------+
| SUBSTRING('Knowledgebase',5) |
+-----------------------------+
| ledgebase                   |
+-----------------------------+

SELECT SUBSTRING('MariaDB' FROM 6);
+----------------------------+
| SUBSTRING('MariaDB' FROM 6) |
+----------------------------+
| DB                         |
+----------------------------+

SELECT SUBSTRING('Knowledgebase',3,7);
+-------------------------------+
| SUBSTRING('Knowledgebase',3,7) |
+-------------------------------+
| owledge                       |
+-------------------------------+

SELECT SUBSTRING('Knowledgebase', -4);
+--------------------------------+
| SUBSTRING('Knowledgebase', -4) |
+--------------------------------+
| base                           |
+--------------------------------+

SELECT SUBSTRING('Knowledgebase', -8, 4);
+-----------------------------------+
| SUBSTRING('Knowledgebase', -8, 4) |
+-----------------------------------+
| edge                              |
+-----------------------------------+

SELECT SUBSTRING('Knowledgebase' FROM -8 FOR 4);
+-----------------------------------------+
| SUBSTRING('Knowledgebase' FROM -8 FOR 4) |
+-----------------------------------------+
| edge                                    |
+-----------------------------------------+
```

Oracle mode from MariaDB 10.3.3:

```
SELECT SUBSTR('abc',0,3);
+------------------+
| SUBSTR('abc',0,3) |
+------------------+
|                  |
+------------------+

SELECT SUBSTR('abc',1,2);
+------------------+
| SUBSTR('abc',1,2) |
+------------------+
| ab               |
+------------------+

SET sql_mode='oracle';

SELECT SUBSTR('abc',0,3);
+------------------+
| SUBSTR('abc',0,3) |
+------------------+
| abc              |
+------------------+

SELECT SUBSTR('abc',1,2);
+------------------+
| SUBSTR('abc',1,2) |
+------------------+
| ab               |
+------------------+
```

1.2.2.59 SUBSTRING_INDEX

Syntax

```
SUBSTRING_INDEX(str,delim,count)
```

Contents

Description

Returns the substring from string *str* before count occurrences of the delimiter *delim*. If *count* is positive, everything to the left of the final delimiter (counting from the left) is returned. If *count* is negative, everything to the right of the final delimiter (counting from the right) is returned. SUBSTRING_INDEX() performs a case-sensitive match when searching for *delim*.

If any argument is NULL, returns NULL.

For example

```
SUBSTRING_INDEX('www.mariadb.org', '.', 2)
```

means "Return all of the characters up to the 2nd occurrence of ."

Examples

```
SELECT SUBSTRING_INDEX('www.mariadb.org', '.', 2);
+--------------------------------------------+
| SUBSTRING_INDEX('www.mariadb.org', '.', 2) |
+--------------------------------------------+
| www.mariadb                                |
+--------------------------------------------+

SELECT SUBSTRING_INDEX('www.mariadb.org', '.', -2);
+---------------------------------------------+
| SUBSTRING_INDEX('www.mariadb.org', '.', -2) |
+---------------------------------------------+
| mariadb.org                                 |
+---------------------------------------------+
```

1.2.2.60 TO_BASE64

Syntax

```
TO_BASE64(str)
```

Description

Converts the string argument `str` to its base-64 encoded form, returning the result as a character string in the connection character set and collation.

The argument `str` will be converted to string first if it is not a string. A NULL argument will return a NULL result.

The reverse function, FROM_BASE64(), decodes an encoded base-64 string.

There are a numerous different methods to base-64 encode a string. The following are used by MariaDB and MySQL:

- Alphabet value 64 is encoded as '+'.
- Alphabet value 63 is encoded as '/'.
- Encoding output is made up of groups of four printable characters, with each three bytes of data encoded using four characters. If the final group is not complete, it is padded with '=' characters to make up a length of four.
- To divide long output, a newline is added after every 76 characters.
- Decoding will recognize and ignore newlines, carriage returns, tabs, and spaces.

Examples

```
SELECT TO_BASE64('Maria');
+--------------------+
| TO_BASE64('Maria') |
+--------------------+
| TWFyaWE=           |
+--------------------+
```

1.2.2.61 TO_CHAR

MariaDB starting with 10.6.1

The TO_CHAR function was introduced in MariaDB 10.6.1 to enhance Oracle compatibility.

Syntax

```
TO_CHAR(expr[, fmt])
```

Contents

Description

The `TO_CHAR` function converts an *expr* of type date, datetime, time or timestamp to a string. The optional *fmt* argument supports YYYY/YYY/YY/RRRR/RR/MM/MON/MONTH/MI/DD/DY/HH/HH12/HH24/SS and special characters. The default value is "YYYY-MM-DD HH24:MI:SS".

In Oracle, TO_CHAR can also be used to convert numbers to strings, but this is not supported in MariaDB and will give an error.

Examples

```
SELECT TO_CHAR('1980-01-11 04:50:39', 'YYYY-MM-DD');
+-------------------------------------------+
| TO_CHAR('1980-01-11 04:50:39', 'YYYY-MM-DD') |
+-------------------------------------------+
| 1980-01-11                                |
+-------------------------------------------+

SELECT TO_CHAR('1980-01-11 04:50:39', 'HH24-MI-SS');
+-------------------------------------------+
| TO_CHAR('1980-01-11 04:50:39', 'HH24-MI-SS') |
+-------------------------------------------+
| 04-50-39                                  |
+-------------------------------------------+

SELECT TO_CHAR('00-01-01 00:00:00', 'YY-MM-DD HH24:MI:SS');
+-----------------------------------------------------+
| TO_CHAR('00-01-01 00:00:00', 'YY-MM-DD HH24:MI:SS') |
+-----------------------------------------------------+
| 00-01-01 00:00:00                                   |
+-----------------------------------------------------+

SELECT TO_CHAR('99-12-31 23:59:59', 'YY-MM-DD HH24:MI:SS');
+-----------------------------------------------------+
| TO_CHAR('99-12-31 23:59:59', 'YY-MM-DD HH24:MI:SS') |
+-----------------------------------------------------+
| 99-12-31 23:59:59                                   |
+-----------------------------------------------------+

SELECT TO_CHAR('9999-12-31 23:59:59', 'YY-MM-DD HH24:MI:SS');
+-----------------------------------------------------+
| TO_CHAR('9999-12-31 23:59:59', 'YY-MM-DD HH24:MI:SS') |
+-----------------------------------------------------+
| 99-12-31 23:59:59                                    |
+-----------------------------------------------------+

SELECT TO_CHAR('21-01-03 08:30:00', 'Y-MONTH-DY HH:MI:SS');
+-----------------------------------------------------+
| TO_CHAR('21-01-03 08:30:00', 'Y-MONTH-DY HH:MI:SS') |
+-----------------------------------------------------+
| 1-January  -Sun 08:30:00                            |
+-----------------------------------------------------+
```

1.2.2.62 TRIM

Syntax

```
TRIM([{BOTH | LEADING | TRAILING} [remstr] FROM] str), TRIM([remstr FROM] str)
```

From MariaDB 10.3.6

```
TRIM_ORACLE([{BOTH | LEADING | TRAILING} [remstr] FROM] str), TRIM([remstr FROM] str)
```

Contents

Description

Returns the string `str` with all `remstr` prefixes or suffixes removed. If none of the specifiers `BOTH`, `LEADING`, or `TRAILING` is given, `BOTH` is assumed. `remstr` is optional and, if not specified, spaces are removed.

Returns NULL if given a NULL argument. If the result is empty, returns either an empty string, or, from MariaDB 10.3.6 with SQL_MODE=Oracle, NULL. `SQL_MODE=Oracle` is not set by default.

The Oracle mode version of the function can be accessed in any mode by using `TRIM_ORACLE` as the function name.

Examples

```
SELECT TRIM('  bar   ')\G
*************************** 1. row ***************************
TRIM('  bar   '): bar

SELECT TRIM(LEADING 'x' FROM 'xxxbarxxx')\G
*************************** 1. row ***************************
TRIM(LEADING 'x' FROM 'xxxbarxxx'): barxxx

SELECT TRIM(BOTH 'x' FROM 'xxxbarxxx')\G
*************************** 1. row ***************************
TRIM(BOTH 'x' FROM 'xxxbarxxx'): bar

SELECT TRIM(TRAILING 'xyz' FROM 'barxxyz')\G
*************************** 1. row ***************************
TRIM(TRAILING 'xyz' FROM 'barxxyz'): barx
```

From MariaDB 10.3.6, with SQL_MODE=Oracle not set:

```
SELECT TRIM(''),TRIM_ORACLE('');
+----------+-----------------+
| TRIM('') | TRIM_ORACLE('') |
+----------+-----------------+
|          | NULL            |
+----------+-----------------+
```

From MariaDB 10.3.6, with SQL_MODE=Oracle set:

```
SELECT TRIM(''),TRIM_ORACLE('');
+----------+-----------------+
| TRIM('') | TRIM_ORACLE('') |
+----------+-----------------+
| NULL     | NULL            |
+----------+-----------------+
```

1.2.2.63 TRIM_ORACLE

MariaDB starting with 10.3.6

`TRIM_ORACLE` is a synonym for the Oracle mode version of the TRIM function, and is available in all modes.

1.2.2.64 UCASE

Syntax

```
UCASE(str)
```

Description

`UCASE()` is a synonym for `UPPER()`.

1.2.2.65 UNCOMPRESS

Syntax

```
UNCOMPRESS(string_to_uncompress)
```

Description

Uncompresses a string compressed by the `COMPRESS()` function. If the argument is not a compressed value, the result is `NULL`. This function requires MariaDB to have been compiled with a compression library such as zlib. Otherwise, the return value is always `NULL`. The have_compress server system variable indicates whether a compression library is present.

Examples

```
SELECT UNCOMPRESS(COMPRESS('a string'));
+----------------------------------+
| UNCOMPRESS(COMPRESS('a string')) |
+----------------------------------+
| a string                         |
+----------------------------------+

SELECT UNCOMPRESS('a string');
+------------------------+
| UNCOMPRESS('a string') |
+------------------------+
| NULL                   |
+------------------------+
```

1.2.2.66 UNCOMPRESSED_LENGTH

Syntax

```
UNCOMPRESSED_LENGTH(compressed_string)
```

Description

Returns the length that the compressed string had before being compressed with `COMPRESS()`.

`UNCOMPRESSED_LENGTH()` returns `NULL` or an incorrect result if the string is not compressed.

Until MariaDB 10.3.1, returns `MYSQL_TYPE_LONGLONG`, or bigint(10), in all cases. From MariaDB 10.3.1, returns `MYSQL_TYPE_LONG`, or int(10), when the result would fit within 32-bits.

Examples

```
SELECT UNCOMPRESSED_LENGTH(COMPRESS(REPEAT('a',30)));
+-----------------------------------------------+
| UNCOMPRESSED_LENGTH(COMPRESS(REPEAT('a',30))) |
+-----------------------------------------------+
|                                            30 |
+-----------------------------------------------+
```

1.2.2.67 UNHEX

Syntax

```
UNHEX(str)
```

Description

Performs the inverse operation of HEX(str). That is, it interprets each pair of hexadecimal digits in the argument as a number and converts it to the character represented by the number. The resulting characters are returned as a binary string.

If `str` is `NULL`, `UNHEX()` returns `NULL`.

Examples

```
SELECT HEX('MariaDB');
+----------------+
| HEX('MariaDB') |
+----------------+
| 4D617269614442 |
+----------------+

SELECT UNHEX('4D617269614442');
+-------------------------+
| UNHEX('4D617269614442') |
+-------------------------+
| MariaDB                 |
+-------------------------+

SELECT 0x4D617269614442;
+------------------+
| 0x4D617269614442 |
+------------------+
| MariaDB          |
+------------------+

SELECT UNHEX(HEX('string'));
+----------------------+
| UNHEX(HEX('string')) |
+----------------------+
| string               |
+----------------------+

SELECT HEX(UNHEX('1267'));
+--------------------+
| HEX(UNHEX('1267')) |
+--------------------+
| 1267               |
+--------------------+
```

1.2.2.68 UPDATEXML

Syntax

```
UpdateXML(xml_target, xpath_expr, new_xml)
```

Description

This function replaces a single portion of a given fragment of XML markup `xml_target` with a new XML fragment `new_xml`, and then returns the changed XML. The portion of `xml_target` that is replaced matches an XPath expression `xpath_expr` supplied by the user. If no expression matching `xpath_expr` is found, or if multiple matches are found, the function returns the original `xml_target` XML fragment. All three arguments should be strings.

Examples

```
SELECT
    UpdateXML('<a><b>ccc</b><d></d></a>', '/a', '<e>fff</e>') AS val1,
    UpdateXML('<a><b>ccc</b><d></d></a>', '/b', '<e>fff</e>') AS val2,
    UpdateXML('<a><b>ccc</b><d></d></a>', '//b', '<e>fff</e>') AS val3,
    UpdateXML('<a><b>ccc</b><d></d></a>', '/a/d', '<e>fff</e>') AS val4,
    UpdateXML('<a><d></d><b>ccc</b><d></d></a>', '/a/d', '<e>fff</e>') AS val5
    \G
*************************** 1. row ***************************
val1: <e>fff</e>
val2: <a><b>ccc</b><d></d></a>
val3: <a><e>fff</e><d></d></a>
val4: <a><b>ccc</b><e>fff</e></a>
val5: <a><d></d><b>ccc</b><d></d></a>
1 row in set (0.00 sec)
```

1.2.2.69 UPPER

Syntax

```
UPPER(str)
UCASE(str)
```

Description

Returns the string `str` with all characters changed to uppercase according to the current character set mapping. The default is latin1 (cp1252 West European).

`UCASE` is a synonym.

```
SELECT UPPER(surname), givenname FROM users ORDER BY surname;
+----------------+------------+
| UPPER(surname) | givenname  |
+----------------+------------+
| ABEL           | Jacinto    |
| CASTRO         | Robert     |
| COSTA          | Phestos    |
| MOSCHELLA      | Hippolytos |
+----------------+------------+
```

`UPPER()` is ineffective when applied to binary strings (BINARY, VARBINARY, BLOB). The description of LOWER() shows how to perform lettercase conversion of binary strings.

Prior to MariaDB 11.3, the query optimizer did not handle queries of the format `UCASE(varchar_col)=...`. An optimizer_switch option, `sargable_casefold=ON`, was added in MariaDB 11.3.0 to handle this case. (MDEV-31496)

1.2.2.70 WEIGHT_STRING

Syntax

```
WEIGHT_STRING(str [AS {CHAR|BINARY}(N)] [LEVEL levels] [flags])
  levels: N [ASC|DESC|REVERSE] [, N [ASC|DESC|REVERSE]] ...
```

Description

Returns a binary string representing the string's sorting and comparison value. A string with a lower result means that for sorting purposes the string appears before a string with a higher result.

WEIGHT_STRING() is particularly useful when adding new collations, for testing purposes.

If `str` is a non-binary string (CHAR, VARCHAR or TEXT), WEIGHT_STRING returns the string's collation weight. If `str` is a binary string (BINARY, VARBINARY or BLOB), the return value is simply the input value, since the weight for each byte in a binary string is the byte value.

WEIGHT_STRING() returns NULL if given a NULL input.

The optional AS clause permits casting the input string to a binary or non-binary string, as well as to a particular length.

AS BINARY(N) measures the length in bytes rather than characters, and right pads with 0x00 bytes to the desired length.

AS CHAR(N) measures the length in characters, and right pads with spaces to the desired length.

N has a minimum value of 1, and if it is less than the length of the input string, the string is truncated without warning.

The optional LEVEL clause specifies that the return value should contain weights for specific collation levels. The `levels` specifier can either be a single integer, a comma-separated list of integers, or a range of integers separated by a dash (whitespace is ignored). Integers can range from 1 to a maximum of 6, dependent on the collation, and need to be listed in ascending order.

If the LEVEL clause is no provided, a default of 1 to the maximum for the collation is assumed.

If the LEVEL is specified without using a range, an optional modifier is permitted.

`ASC`, the default, returns the weights without any modification.

`DESC` returns bitwise-inverted weights.

`REVERSE` returns the weights in reverse order.

Examples

The examples below use the HEX() function to represent non-printable results in hexadecimal format.

```
SELECT HEX(WEIGHT_STRING('x'));
+-------------------------+
| HEX(WEIGHT_STRING('x')) |
+-------------------------+
| 0058                    |
+-------------------------+

SELECT HEX(WEIGHT_STRING('x' AS BINARY(4)));
+--------------------------------------+
| HEX(WEIGHT_STRING('x' AS BINARY(4))) |
+--------------------------------------+
| 78000000                             |
+--------------------------------------+

SELECT HEX(WEIGHT_STRING('x' AS CHAR(4)));
+-----------------------------------+
| HEX(WEIGHT_STRING('x' AS CHAR(4))) |
+-----------------------------------+
| 0058002000200020                  |
+-----------------------------------+

SELECT HEX(WEIGHT_STRING(0xaa22ee LEVEL 1));
+--------------------------------------+
| HEX(WEIGHT_STRING(0xaa22ee LEVEL 1)) |
+--------------------------------------+
| AA22EE                               |
+--------------------------------------+

SELECT HEX(WEIGHT_STRING(0xaa22ee LEVEL 1 DESC));
+-----------------------------------------+
| HEX(WEIGHT_STRING(0xaa22ee LEVEL 1 DESC)) |
+-----------------------------------------+
| 55DD11                                  |
+-----------------------------------------+

SELECT HEX(WEIGHT_STRING(0xaa22ee LEVEL 1 REVERSE));
+-------------------------------------------+
| HEX(WEIGHT_STRING(0xaa22ee LEVEL 1 REVERSE)) |
+-------------------------------------------+
| EE22AA                                    |
+-------------------------------------------+
```

1.2.2.71 Type Conversion

Contents

Implicit type conversion takes place when MariaDB is using operands or different types, in order to make the operands compatible.

It is best practice not to rely upon implicit conversion; rather use CAST to explicitly convert types.

Rules for Conversion on Comparison

- If either argument is NULL, the result of the comparison is NULL unless the NULL-safe <=> equality comparison operator is used.
- If both arguments are integers, they are compared as integers.
- If both arguments are strings, they are compared as strings.
- If one argument is decimal and the other argument is decimal or integer, they are compared as decimals.
- If one argument is decimal and the other argument is a floating point, they are compared as floating point values.
- If one argument is string and the other argument is integer, they are compared as decimals. This conversion was added in MariaDB 10.3.36. Prior to 10.3.36, this combination was compared as floating point values, which did not always work well for huge 64-bit integers because of a possible precision loss on conversion to double.
- If a hexadecimal argument is not compared to a number, it is treated as a binary string.
- If a constant is compared to a TIMESTAMP or DATETIME, the constant is converted to a timestamp, unless used as an argument to the IN function.

- In other cases, arguments are compared as floating point, or real, numbers.

Note that if a string column is being compared with a numeric value, MariaDB will not use the index on the column, as there are numerous alternatives that may evaluate as equal (see examples below).

Comparison Examples

Converting a string to a number:

```
SELECT 15+'15';
+---------+
| 15+'15' |
+---------+
|      30 |
+---------+
```

Converting a number to a string:

```
SELECT CONCAT(15,'15');
+-----------------+
| CONCAT(15,'15') |
+-----------------+
| 1515            |
+-----------------+
```

Floating point number errors:

```
SELECT '9746718491924563214' = 9746718491924563213;
+-------------------------------------------+
| '9746718491924563214' = 9746718491924563213 |
+-------------------------------------------+
|                                         1 |
+-------------------------------------------+
```

Numeric equivalence with strings:

```
SELECT '5' = 5;
+---------+
| '5' = 5 |
+---------+
|       1 |
+---------+

SELECT '   5' = 5;
+------------+
| '   5' = 5 |
+------------+
|          1 |
+------------+

SELECT '   5  ' = 5;
+--------------+
| '   5  ' = 5 |
+--------------+
|            1 |
+--------------+
1 row in set, 1 warning (0.000 sec)

SHOW WARNINGS;
+-------+------+-------------------------------------------+
| Level | Code | Message                                   |
+-------+------+-------------------------------------------+
| Note  | 1292 | Truncated incorrect DOUBLE value: '   5  ' |
+-------+------+-------------------------------------------+
```

As a result of the above, MariaDB cannot use the index when comparing a string with a numeric value in the example below:

```
CREATE TABLE t (a VARCHAR(10), b VARCHAR(10), INDEX idx_a (a));

INSERT INTO t VALUES
  ('1', '1'), ('2', '2'), ('3', '3'),
  ('4', '4'), ('5', '5'), ('1', '5');

EXPLAIN SELECT * FROM t WHERE a = '3' \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t
         type: ref
possible_keys: idx_a
          key: idx_a
      key_len: 13
          ref: const
         rows: 1
        Extra: Using index condition

EXPLAIN SELECT * FROM t WHERE a = 3 \G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t
         type: ALL
possible_keys: idx_a
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 6
        Extra: Using where
```

Rules for Conversion on Dyadic Arithmetic Operations

Implicit type conversion also takes place on dyadic arithmetic operations (+,-,*,/). MariaDB chooses the minimum data type that is guaranteed to fit the result and converts both arguments to the result data type.

For addition (+), subtraction (-) and multiplication (*), the result data type is chosen as follows:

- If either of the arguments is an approximate number (float, double), the result is double.
- If either of the arguments is a string (char, varchar, text), the result is double.
- If either of the arguments is a decimal number, the result is decimal.
- If either of the arguments is of a temporal type with a non-zero fractional second precision (time(N), datetime(N), timestamp(N)), the result is decimal.
- If either of the arguments is of a temporal type with a zero fractional second precision (time(0), date, datetime(0), timestamp(0)), the result may vary between int, int unsigned, bigint or bigint unsigned, depending on the exact data type combination.
- If both arguments are integer numbers (tinyint, smallint, mediumint, bigint), the result may vary between int, int unsigned, bigint or bigint unsigned, depending of the exact data types and their signs.

For division (/), the result data type is chosen as follows:

- If either of the arguments is an approximate number (float, double), the result is double.
- If either of the arguments is a string (char, varchar, text), the result is double.
- Otherwise, the result is decimal.

Arithmetic Examples

Note, the above rules mean that when an argument of a temporal data type appears in addition or subtraction, it's treated as a number by default.

```
SELECT TIME'10:20:30' + 1;
+--------------------+
| TIME'10:20:30' + 1 |
+--------------------+
|             102031 |
+--------------------+
```

In order to do temporal addition or subtraction instead, use the DATE_ADD() or DATE_SUB() functions, or an INTERVAL expression as the second argument:

```
SELECT TIME'10:20:30' + INTERVAL 1 SECOND;
+-----------------------------------+
| TIME'10:20:30' + INTERVAL 1 SECOND |
+-----------------------------------+
| 10:20:31                          |
+-----------------------------------+
```

```
SELECT "2.2" + 3;
+-----------+
| "2.2" + 3 |
+-----------+
|       5.2 |
+-----------+

SELECT 2.2 + 3;
+---------+
| 2.2 + 3 |
+---------+
| 5.2     |
+---------+

SELECT 2.2 / 3;
+---------+
| 2.2 / 3 |
+---------+
| 0.73333 |
+---------+

SELECT "2.2" / 3;
+--------------------+
| "2.2" / 3          |
+--------------------+
| 0.7333333333333334 |
+--------------------+
```

1.2.3 Date & Time Functions

Functions for handling date and time, e.g. TIME, DATE, DAYNAME etc.

Microseconds in MariaDB

Microseconds have been supported since MariaDB 5.3.

Date and Time Units

Date or time units

ADD_MONTHS

Adds a number of months to a date.

ADDDATE

Add days or another interval to a date.

ADDTIME

Adds a time to a time or datetime.

CONVERT_TZ

Converts a datetime from one time zone to another.

CURDATE

Returns the current date.

CURRENT_DATE

Synonym for CURDATE().

CURRENT_TIME

Synonym for CURTIME().

CURRENT_TIMESTAMP
Synonym for NOW().

CURTIME
Returns the current time.

DATE FUNCTION
Extracts the date portion of a datetime.

DATEDIFF
Difference in days between two date/time values.

DATE_ADD
Date arithmetic - addition.

DATE_FORMAT
Formats the date value according to the format string.

DATE_SUB
Date arithmetic - subtraction.

DAY
Synonym for DAYOFMONTH().

DAYNAME
Return the name of the weekday.

DAYOFMONTH
Returns the day of the month.

DAYOFWEEK
Returns the day of the week index.

DAYOFYEAR
Returns the day of the year.

EXTRACT
Extracts a portion of the date.

FORMAT_PICO_TIME
Given a time in picoseconds, returns a human-readable time value and unit indicator.

FROM_DAYS
Returns a date given a day.

FROM_UNIXTIME
Returns a datetime from a Unix timestamp.

GET_FORMAT
Returns a format string.

HOUR
Returns the hour.

LAST_DAY
Returns the last day of the month.

LOCALTIME
Synonym for NOW().

LOCALTIMESTAMP
Synonym for NOW().

MAKEDATE

Returns a date given a year and day.

MAKETIME

Returns a time.

MICROSECOND

Returns microseconds from a date or datetime.

MINUTE

Returns a minute from 0 to 59.

MONTH

Returns a month from 1 to 12.

MONTHNAME

Returns the full name of the month.

NOW

Returns the current date and time.

PERIOD_ADD

Add months to a period.

PERIOD_DIFF

Number of months between two periods.

QUARTER

Returns year quarter from 1 to 4.

SECOND

Returns the second of a time.

SEC_TO_TIME

Converts a second to a time.

STR_TO_DATE

Converts a string to date.

SUBDATE

Subtract a date unit or number of days.

SUBTIME

Subtracts a time from a date/time.

SYSDATE

Returns the current date and time.

TIME Function

Extracts the time.

TIMEDIFF

Returns the difference between two date/times.

TIMESTAMP FUNCTION

Return the datetime, or add a time to a date/time.

TIMESTAMPADD

Add interval to a date or datetime.

TIMESTAMPDIFF

Difference between two datetimes.

TIME_FORMAT
Formats the time value according to the format string.

TIME_TO_SEC
Returns the time argument, converted to seconds.

TO_DAYS
Number of days since year 0.

TO_SECONDS
Number of seconds since year 0.

UNIX_TIMESTAMP
Returns a Unix timestamp.

UTC_DATE
Returns the current UTC date.

UTC_TIME
Returns the current UTC time.

UTC_TIMESTAMP
Returns the current UTC date and time.

WEEK
Returns the week number.

WEEKDAY
Returns the weekday index.

WEEKOFYEAR
Returns the calendar week of the date as a number in the range from 1 to 53.

YEAR
Returns the year for the given date.

YEARWEEK
Returns year and week for a date.

There are 6 related questions.

1.2.3.1 Microseconds in MariaDB

Contents

The TIME, DATETIME, and TIMESTAMP types, along with the temporal functions, CAST and dynamic columns, support microseconds. The datetime precision of a column can be specified when creating the table with CREATE TABLE, for example:

```
CREATE TABLE example(
  col_microsec DATETIME(6),
  col_millisec TIME(3)
);
```

Generally, the precision can be specified for any `TIME`, `DATETIME`, or `TIMESTAMP` column, in parentheses, after the type name. The datetime precision specifies number of digits after the decimal dot and can be any integer number from 0 to 6. If no precision is specified it is assumed to be 0, for backward compatibility reasons.

A datetime precision can be specified wherever a type name is used. For example:

- when declaring arguments of stored routines.

- when specifying a return type of a stored function.
- when declaring variables.
- in a `CAST` function:

```
create function example(x datetime(5)) returns time(4)
begin
  declare y timestamp(6);
  return cast(x as time(2));
end;
```

`%f` is used as the formatting option for microseconds in the STR_TO_DATE, DATE_FORMAT and FROM_UNIXTIME functions, for example:

```
SELECT STR_TO_DATE('20200809 020917076','%Y%m%d %H%i%s%f');
+-------------------------------------------------------+
| STR_TO_DATE('20200809 020917076','%Y%m%d %H%i%s%f') |
+-------------------------------------------------------+
| 2020-08-09 02:09:17.076000                            |
+-------------------------------------------------------+
```

Additional Information

- when comparing anything to a temporal value (`DATETIME` , `TIME` , `DATE` , or `TIMESTAMP`), both values are compared as temporal values, not as strings.
- The INFORMATION_SCHEMA.COLUMNS table has a new column `DATETIME_PRECISION`
- NOW(), CURTIME(), UTC_TIMESTAMP(), UTC_TIME(), CURRENT_TIME(), CURRENT_TIMESTAMP(), LOCALTIME() and LOCALTIMESTAMP() now accept datetime precision as an optional argument. For example:

```
SELECT CURTIME(4);
--> 10:11:12.3456
```

- TIME_TO_SEC() and UNIX_TIMESTAMP() preserve microseconds of the argument. These functions will return a decimal number if the result non-zero datetime precision and an integer otherwise (for backward compatibility).

```
SELECT TIME_TO_SEC('10:10:10.12345');
--> 36610.12345
```

- Current versions of this patch fix a bug in the following optimization: in certain queries with `DISTINCT` MariaDB can ignore this clause if it can prove that all result rows are unique anyway, for example, when a primary key is compared with a constant. Sometimes this optimization was applied incorrectly, though — for example, when comparing a string with a date constant. This is now fixed.
- `DATE_ADD()` and `DATE_SUB()` functions can now take a `TIME` expression as an argument (not just `DATETIME` as before).

```
SELECT TIME('10:10:10') + INTERVAL 100 MICROSECOND;
--> 10:10:10.000100
```

- The `event_time` field in the mysql.general_log table and the `start_time` , `query_time` , and `lock_time` fields in the mysql.slow_log table now store values with microsecond precision.
- This patch fixed a bug when comparing a temporal value using the `BETWEEN` operator and one of the operands is `NULL` .
- The old syntax `TIMESTAMP(N)` , where `N` is the display width, is no longer supported. It was deprecated in MySQL 4.1.0 (released on 2003-04-03).
- when a `DATETIME` value is compared to a `TIME` value, the latter is treated as a full datetime with a zero date part, similar to comparing `DATE` to a `DATETIME` , or to comparing `DECIMAL` numbers. Earlier versions of MariaDB used to compare only the time part of both operands in such a case.
- In MariaDB, an extra column `TIME_MS` has been added to the `INFORMATION_SCHEMA.PROCESSLIST` table, as well as to the output of `SHOW FULL PROCESSLIST` .

Note: When you convert a temporal value to a value with a smaller precision, it will be truncated, not rounded. This is done to guarantee that the date part is not changed. For example:

```
SELECT CAST('2009-12-31 23:59:59.998877' as DATETIME(3));
-> 2009-12-31 23:59:59.998
```

MySQL 5.6 Microseconds

MySQL 5.6 introduced microseconds using a slightly different implementation to MariaDB 5.3. Since MariaDB 10.1, MariaDB has defaulted to the MySQL format, by means of the --mysql56-temporal-format variable. The MySQL version requires slightly more storage but has some advantages in permitting the eventual support of negative dates, and in replication.

1.2.3.2 Date and Time Units

The `INTERVAL` keyword can be used to add or subtract a time interval of time to a `DATETIME`, `DATE` or `TIME` value.

The syntax is:

```
INTERVAL time_quantity time_unit
```

For example, the `SECOND` unit is used below by the `DATE_ADD()` function:

```
SELECT '2008-12-31 23:59:59' + INTERVAL 1 SECOND;
+---------------------------------------+
| '2008-12-31 23:59:59' + INTERVAL 1 SECOND |
+---------------------------------------+
| 2009-01-01 00:00:00                   |
+---------------------------------------+
```

The following units are valid:

Unit	Description
`MICROSECOND`	Microseconds
`SECOND`	Seconds
`MINUTE`	Minutes
`HOUR`	Hours
`DAY`	Days
`WEEK`	Weeks
`MONTH`	Months
`QUARTER`	Quarters
`YEAR`	Years
`SECOND_MICROSECOND`	Seconds.Microseconds
`MINUTE_MICROSECOND`	Minutes.Seconds.Microseconds
`MINUTE_SECOND`	Minutes.Seconds
`HOUR_MICROSECOND`	Hours.Minutes.Seconds.Microseconds
`HOUR_SECOND`	Hours.Minutes.Seconds
`HOUR_MINUTE`	Hours.Minutes
`DAY_MICROSECOND`	Days Hours.Minutes.Seconds.Microseconds
`DAY_SECOND`	Days Hours.Minutes.Seconds
`DAY_MINUTE`	Days Hours.Minutes
`DAY_HOUR`	Days Hours
`YEAR_MONTH`	Years-Months

The time units containing an underscore are composite; that is, they consist of multiple base time units. For base time units, `time_quantity` is an integer number. For composite units, the quantity must be expressed as a string with multiple integer numbers separated by any punctuation character.

Example of composite units:

```
INTERVAL '2:2' YEAR_MONTH
INTERVAL '1:30:30' HOUR_SECOND
INTERVAL '1!30!30' HOUR_SECOND -- same as above
```

Time units can be used in the following contexts:

- after a `+` or a `-` operator;
- with the following `DATE` or `TIME` functions: `ADDDATE()`, `SUBDATE()`, `DATE_ADD()`, `DATE_SUB()`, `TIMESTAMPADD()`, `TIMESTAMPDIFF()`, `EXTRACT()`;
- in the `ON SCHEDULE` clause of `CREATE EVENT` and `ALTER EVENT`.
- when defining a partitioning `BY SYSTEM_TIME`

1.2.3.3 ADD_MONTHS

MariaDB starting with 10.6.1

The ADD_MONTHS function was introduced in MariaDB 10.6.1 to enhance Oracle compatibility. Similar functionality can be achieved with the DATE_ADD function.

Syntax

```
ADD_MONTHS(date, months)
```

Contents

Description

`ADD_MONTHS` adds an integer *months* to a given *date* (DATE, DATETIME or TIMESTAMP), returning the resulting date.

months can be positive or negative.

The resulting day component will remain the same as that specified in *date*, unless the resulting month has fewer days than the day component of the given date, in which case the day will be the last day of the resulting month.

Returns NULL if given an invalid date, or a NULL argument.

Examples

```
SELECT ADD_MONTHS('2012-01-31', 2);
+-----------------------------+
| ADD_MONTHS('2012-01-31', 2) |
+-----------------------------+
| 2012-03-31                  |
+-----------------------------+

SELECT ADD_MONTHS('2012-01-31', -5);
+-----------------------------+
| ADD_MONTHS('2012-01-31', -5) |
+-----------------------------+
| 2011-08-31                   |
+-----------------------------+

SELECT ADD_MONTHS('2011-01-31', 1);
+-----------------------------+
| ADD_MONTHS('2011-01-31', 1) |
+-----------------------------+
| 2011-02-28                  |
+-----------------------------+

SELECT ADD_MONTHS('2012-01-31', 1);
+-----------------------------+
| ADD_MONTHS('2012-01-31', 1) |
+-----------------------------+
| 2012-02-29                  |
+-----------------------------+

SELECT ADD_MONTHS('2012-01-31', 2);
+-----------------------------+
| ADD_MONTHS('2012-01-31', 2) |
+-----------------------------+
| 2012-03-31                  |
+-----------------------------+

SELECT ADD_MONTHS('2012-01-31', 3);
+-----------------------------+
| ADD_MONTHS('2012-01-31', 3) |
+-----------------------------+
| 2012-04-30                  |
+-----------------------------+
```

1.2.3.4 ADDDATE

Syntax

```
ADDDATE(date,INTERVAL expr unit), ADDDATE(expr,days)
```

Description

When invoked with the `INTERVAL` form of the second argument, `ADDDATE()` is a synonym for `DATE_ADD()`. The related function `SUBDATE()` is a synonym for `DATE_SUB()`. For information on the `INTERVAL` unit argument, see the discussion for `DATE_ADD()`.

When invoked with the days form of the second argument, MariaDB treats it as an integer number of days to be added to *expr*.

Examples

```
SELECT DATE_ADD('2008-01-02', INTERVAL 31 DAY);
+----------------------------------------+
| DATE_ADD('2008-01-02', INTERVAL 31 DAY) |
+----------------------------------------+
| 2008-02-02                             |
+----------------------------------------+

SELECT ADDDATE('2008-01-02', INTERVAL 31 DAY);
+----------------------------------------+
| ADDDATE('2008-01-02', INTERVAL 31 DAY) |
+----------------------------------------+
| 2008-02-02                             |
+----------------------------------------+
```

```
SELECT ADDDATE('2008-01-02', 31);
+---------------------------+
| ADDDATE('2008-01-02', 31) |
+---------------------------+
| 2008-02-02                |
+---------------------------+
```

```
CREATE TABLE t1 (d DATETIME);
INSERT INTO t1 VALUES
    ("2007-01-30 21:31:07"),
    ("1983-10-15 06:42:51"),
    ("2011-04-21 12:34:56"),
    ("2011-10-30 06:31:41"),
    ("2011-01-30 14:03:25"),
    ("2004-10-07 11:19:34");
```

```
SELECT d, ADDDATE(d, 10) from t1;
+---------------------+---------------------+
| d                   | ADDDATE(d, 10)      |
+---------------------+---------------------+
| 2007-01-30 21:31:07 | 2007-02-09 21:31:07 |
| 1983-10-15 06:42:51 | 1983-10-25 06:42:51 |
| 2011-04-21 12:34:56 | 2011-05-01 12:34:56 |
| 2011-10-30 06:31:41 | 2011-11-09 06:31:41 |
| 2011-01-30 14:03:25 | 2011-02-09 14:03:25 |
| 2004-10-07 11:19:34 | 2004-10-17 11:19:34 |
+---------------------+---------------------+

SELECT d, ADDDATE(d, INTERVAL 10 HOUR) from t1;
+---------------------+-----------------------------+
| d                   | ADDDATE(d, INTERVAL 10 HOUR) |
+---------------------+-----------------------------+
| 2007-01-30 21:31:07 | 2007-01-31 07:31:07         |
| 1983-10-15 06:42:51 | 1983-10-15 16:42:51         |
| 2011-04-21 12:34:56 | 2011-04-21 22:34:56         |
| 2011-10-30 06:31:41 | 2011-10-30 16:31:41         |
| 2011-01-30 14:03:25 | 2011-01-31 00:03:25         |
| 2004-10-07 11:19:34 | 2004-10-07 21:19:34         |
+---------------------+-----------------------------+
```

1.2.3.5 ADDTIME

Syntax

```
ADDTIME(expr1,expr2)
```

Description

`ADDTIME()` adds *expr2* to *expr1* and returns the result. *expr1* is a time or datetime expression, and *expr2* is a time expression.

Examples

```
SELECT ADDTIME('2007-12-31 23:59:59.999999', '1 1:1:1.000002');
+----------------------------------------------------------+
| ADDTIME('2007-12-31 23:59:59.999999', '1 1:1:1.000002') |
+----------------------------------------------------------+
| 2008-01-02 01:01:01.000001                               |
+----------------------------------------------------------+

SELECT ADDTIME('01:00:00.999999', '02:00:00.999998');
+---------------------------------------------+
| ADDTIME('01:00:00.999999', '02:00:00.999998') |
+---------------------------------------------+
| 03:00:01.999997                             |
+---------------------------------------------+
```

1.2.3.6 CONVERT_TZ

Syntax

```
CONVERT_TZ(dt,from_tz,to_tz)
```

Description

CONVERT_TZ() converts a datetime value *dt* from the time zone given by *from_tz* to the time zone given by *to_tz* and returns the resulting value.

In order to use named time zones, such as GMT, MET or Africa/Johannesburg, the time_zone tables must be loaded (see mysql_tzinfo_to_sql).

No conversion will take place if the value falls outside of the supported TIMESTAMP range ('1970-01-01 00:00:01' to '2038-01-19 05:14:07' UTC) when converted from *from_tz* to UTC.

This function returns NULL if the arguments are invalid (or named time zones have not been loaded).

See time zones for more information.

Examples

```
SELECT CONVERT_TZ('2016-01-01 12:00:00','+00:00','+10:00');
+-----------------------------------------------------+
| CONVERT_TZ('2016-01-01 12:00:00','+00:00','+10:00') |
+-----------------------------------------------------+
| 2016-01-01 22:00:00                                 |
+-----------------------------------------------------+
```

Using named time zones (with the time zone tables loaded):

```
SELECT CONVERT_TZ('2016-01-01 12:00:00','GMT','Africa/Johannesburg');
+------------------------------------------------------------+
| CONVERT_TZ('2016-01-01 12:00:00','GMT','Africa/Johannesburg') |
+------------------------------------------------------------+
| 2016-01-01 14:00:00                                        |
+------------------------------------------------------------+
```

The value is out of the TIMESTAMP range, so no conversion takes place:

```
SELECT CONVERT_TZ('1969-12-31 22:00:00','+00:00','+10:00');
+-----------------------------------------------------+
| CONVERT_TZ('1969-12-31 22:00:00','+00:00','+10:00') |
+-----------------------------------------------------+
| 1969-12-31 22:00:00                                 |
+-----------------------------------------------------+
```

1.2.3.7 CURDATE

Syntax

```
CURDATE()
CURRENT_DATE
CURRENT_DATE()
```

Description

`CURDATE` returns the current date as a value in 'YYYY-MM-DD' or YYYYMMDD format, depending on whether the function is used in a string or numeric context.

`CURRENT_DATE` and `CURRENT_DATE()` are synonyms.

Examples

```
SELECT CURDATE();
+------------+
| CURDATE()  |
+------------+
| 2019-03-05 |
+------------+
```

In a numeric context (note this is not performing date calculations):

```
SELECT CURDATE() +0;
+--------------+
| CURDATE() +0 |
+--------------+
|     20190305 |
+--------------+
```

Data calculation:

```
SELECT CURDATE() - INTERVAL 5 DAY;
+----------------------------+
| CURDATE() - INTERVAL 5 DAY |
+----------------------------+
| 2019-02-28                 |
+----------------------------+
```

1.2.3.8 CURRENT_DATE

Syntax

```
CURRENT_DATE, CURRENT_DATE()
```

Description

`CURRENT_DATE` and `CURRENT_DATE()` are synonyms for CURDATE().

1.2.3.9 CURRENT_TIME

Syntax

```
CURRENT_TIME
CURRENT_TIME([precision])
```

Description

`CURRENT_TIME` and `CURRENT_TIME()` are synonyms for `CURTIME()` .

1.2.3.10 CURRENT_TIMESTAMP

Syntax

```
CURRENT_TIMESTAMP
CURRENT_TIMESTAMP([precision])
```

Description

`CURRENT_TIMESTAMP` and `CURRENT_TIMESTAMP()` are synonyms for `NOW()` .

1.2.3.11 CURTIME

Syntax

```
CURTIME([precision])
```

Description

Returns the current time as a value in 'HH:MM:SS' or HHMMSS.uuuuuu format, depending on whether the function is used in a string or numeric context. The value is expressed in the current time zone.

The optional *precision* determines the microsecond precision. See Microseconds in MariaDB.

Examples

```
SELECT CURTIME();
+-----------+
| CURTIME() |
+-----------+
| 12:45:39  |
+-----------+

SELECT CURTIME() + 0;
+---------------+
| CURTIME() + 0 |
+---------------+
| 124545.000000 |
+---------------+
```

With precision:

```
SELECT CURTIME(2);
+-------------+
| CURTIME(2)  |
+-------------+
| 09:49:08.09 |
+-------------+
```

1.2.3.12 DATE FUNCTION

Syntax

```
DATE(expr)
```

Description

Extracts the date part of the date or datetime expression *expr*. Returns NULL and throws a warning when passed an invalid date.

Examples

```
SELECT DATE('2013-07-18 12:21:32');
+----------------------------+
| DATE('2013-07-18 12:21:32') |
+----------------------------+
| 2013-07-18                 |
+----------------------------+
```

1.2.3.13 DATEDIFF

Syntax

```
DATEDIFF(expr1,expr2)
```

Description

`DATEDIFF()` returns (*expr1* – *expr2*) expressed as a value in days from one date to the other. *expr1* and *expr2* are date or date-and-time expressions. Only the date parts of the values are used in the calculation.

Examples

```
SELECT DATEDIFF('2007-12-31 23:59:59','2007-12-30');
+-----------------------------------------------+
| DATEDIFF('2007-12-31 23:59:59','2007-12-30') |
+-----------------------------------------------+
|                                             1 |
+-----------------------------------------------+

SELECT DATEDIFF('2010-11-30 23:59:59','2010-12-31');
+-----------------------------------------------+
| DATEDIFF('2010-11-30 23:59:59','2010-12-31') |
+-----------------------------------------------+
|                                           -31 |
+-----------------------------------------------+
```

```
CREATE TABLE t1 (d DATETIME);
INSERT INTO t1 VALUES
    ("2007-01-30 21:31:07"),
    ("1983-10-15 06:42:51"),
    ("2011-04-21 12:34:56"),
    ("2011-10-30 06:31:41"),
    ("2011-01-30 14:03:25"),
    ("2004-10-07 11:19:34");
```

```
SELECT NOW();
+---------------------+
| NOW()               |
+---------------------+
| 2011-05-23 10:56:05 |
+---------------------+

SELECT d, DATEDIFF(NOW(),d) FROM t1;
+---------------------+-------------------+
| d                   | DATEDIFF(NOW(),d) |
+---------------------+-------------------+
| 2007-01-30 21:31:07 |              1574 |
| 1983-10-15 06:42:51 |             10082 |
| 2011-04-21 12:34:56 |                32 |
| 2011-10-30 06:31:41 |              -160 |
| 2011-01-30 14:03:25 |               113 |
| 2004-10-07 11:19:34 |              2419 |
+---------------------+-------------------+
```

1.2.3.14 DATE_ADD

Syntax

```
DATE_ADD(date,INTERVAL expr unit)
```

Contents

Description

Performs date arithmetic. The *date* argument specifies the starting date or datetime value. *expr* is an expression specifying the interval value to be added or subtracted from the starting date. *expr* is a string; it may start with a " - " for negative intervals. *unit* is a keyword indicating the units in which the expression should be interpreted. See Date and Time Units for a complete list of permitted units.

Examples

```
SELECT '2008-12-31 23:59:59' + INTERVAL 1 SECOND;
+-----------------------------------------+
| '2008-12-31 23:59:59' + INTERVAL 1 SECOND |
+-----------------------------------------+
| 2009-01-01 00:00:00                     |
+-----------------------------------------+
```

```
SELECT INTERVAL 1 DAY + '2008-12-31';
+-------------------------------+
| INTERVAL 1 DAY + '2008-12-31' |
+-------------------------------+
| 2009-01-01                    |
+-------------------------------+
```

```
SELECT '2005-01-01' - INTERVAL 1 SECOND;
+---------------------------------+
| '2005-01-01' - INTERVAL 1 SECOND |
+---------------------------------+
| 2004-12-31 23:59:59              |
+---------------------------------+
```

```
SELECT DATE_ADD('2000-12-31 23:59:59', INTERVAL 1 SECOND);
+-----------------------------------------------------+
| DATE_ADD('2000-12-31 23:59:59', INTERVAL 1 SECOND) |
+-----------------------------------------------------+
| 2001-01-01 00:00:00                                 |
+-----------------------------------------------------+
```

```
SELECT DATE_ADD('2010-12-31 23:59:59', INTERVAL 1 DAY);
+------------------------------------------------+
| DATE_ADD('2010-12-31 23:59:59', INTERVAL 1 DAY) |
+------------------------------------------------+
| 2011-01-01 23:59:59                             |
+------------------------------------------------+
```

```
SELECT DATE_ADD('2100-12-31 23:59:59', INTERVAL '1:1' MINUTE_SECOND);
+-------------------------------------------------------------+
| DATE_ADD('2100-12-31 23:59:59', INTERVAL '1:1' MINUTE_SECOND) |
+-------------------------------------------------------------+
| 2101-01-01 00:01:00                                          |
+-------------------------------------------------------------+
```

```
SELECT DATE_ADD('1900-01-01 00:00:00', INTERVAL '-1 10' DAY_HOUR);
+-----------------------------------------------------------+
| DATE_ADD('1900-01-01 00:00:00', INTERVAL '-1 10' DAY_HOUR) |
+-----------------------------------------------------------+
| 1899-12-30 14:00:00                                        |
+-----------------------------------------------------------+
```

```
SELECT DATE_ADD('1992-12-31 23:59:59.000002', INTERVAL '1.999999' SECOND_MICROSECOND);
+-----------------------------------------------------------------------------+
| DATE_ADD('1992-12-31 23:59:59.000002', INTERVAL '1.999999' SECOND_MICROSECOND) |
+-----------------------------------------------------------------------------+
| 1993-01-01 00:00:01.000001                                                   |
+-----------------------------------------------------------------------------+
```

1.2.3.15 DATE_FORMAT

Syntax

```
DATE_FORMAT(date, format[, locale])
```

Contents

Description

Formats the date value according to the format string.

The language used for the names is controlled by the value of the lc_time_names system variable. See server locale for more on the supported locales.

The options that can be used by DATE_FORMAT(), as well as its inverse STR_TO_DATE() and the FROM_UNIXTIME()

function, are:

Option	Description
`%a`	Short weekday name in current locale (Variable lc_time_names).
`%b`	Short form month name in current locale. For locale en_US this is one of: Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov or Dec.
`%c`	Month with 1 or 2 digits.
`%D`	Day with English suffix 'th', 'nd', 'st' or 'rd''. (1st, 2nd, 3rd...).
`%d`	Day with 2 digits.
`%e`	Day with 1 or 2 digits.
`%f`	Microseconds 6 digits.
`%H`	Hour with 2 digits between 00-23.
`%h`	Hour with 2 digits between 01-12.
`%I`	Hour with 2 digits between 01-12.
`%i`	Minute with 2 digits.
`%j`	Day of the year (001-366)
`%k`	Hour with 1 digits between 0-23.
`%l`	Hour with 1 digits between 1-12.
`%M`	Full month name in current locale (Variable lc_time_names).
`%m`	Month with 2 digits.
`%p`	AM/PM according to current locale (Variable lc_time_names).
`%r`	Time in 12 hour format, followed by AM/PM. Short for '%I:%i:%S %p'.
`%S`	Seconds with 2 digits.
`%s`	Seconds with 2 digits.
`%T`	Time in 24 hour format. Short for '%H:%i:%S'.
`%U`	Week number (00-53), when first day of the week is Sunday.
`%u`	Week number (00-53), when first day of the week is Monday.
`%V`	Week number (01-53), when first day of the week is Sunday. Used with %X.
`%v`	Week number (01-53), when first day of the week is Monday. Used with %x.
`%W`	Full weekday name in current locale (Variable lc_time_names).
`%w`	Day of the week. 0 = Sunday, 6 = Saturday.
`%X`	Year with 4 digits when first day of the week is Sunday. Used with %V.
`%x`	Year with 4 digits when first day of the week is Monday. Used with %v.
`%Y`	Year with 4 digits.
`%y`	Year with 2 digits.
`%Z`	Timezone abbreviation. From MariaDB 11.3.0.
`%z`	Numeric timezone +hhmm or -hhmm presenting the hour and minute offset from UTC. From MariaDB 11.3.0.
`%#`	For str_to_date(), skip all numbers.
`%.`	For str_to_date(), skip all punctation characters.
`%@`	For str_to_date(), skip all alpha characters.
`%%`	A literal `%` character.

To get a date in one of the standard formats, `GET_FORMAT()` can be used.

Examples

```
SELECT DATE_FORMAT('2009-10-04 22:23:00', '%W %M %Y');
+-----------------------------------------------+
| DATE_FORMAT('2009-10-04 22:23:00', '%W %M %Y') |
+-----------------------------------------------+
| Sunday October 2009                           |
+-----------------------------------------------+

SELECT DATE_FORMAT('2007-10-04 22:23:00', '%H:%i:%s');
+-----------------------------------------------+
| DATE_FORMAT('2007-10-04 22:23:00', '%H:%i:%s') |
+-----------------------------------------------+
| 22:23:00                                      |
+-----------------------------------------------+

SELECT DATE_FORMAT('1900-10-04 22:23:00', '%D %y %a %d %m %b %j');
+------------------------------------------------------------+
| DATE_FORMAT('1900-10-04 22:23:00', '%D %y %a %d %m %b %j') |
+------------------------------------------------------------+
| 4th 00 Thu 04 10 Oct 277                                   |
+------------------------------------------------------------+

SELECT DATE_FORMAT('1997-10-04 22:23:00', '%H %k %I %r %T %S %w');
+------------------------------------------------------------+
| DATE_FORMAT('1997-10-04 22:23:00', '%H %k %I %r %T %S %w') |
+------------------------------------------------------------+
| 22 22 10 10:23:00 PM 22:23:00 00 6                         |
+------------------------------------------------------------+

SELECT DATE_FORMAT('1999-01-01', '%X %V');
+-----------------------------------+
| DATE_FORMAT('1999-01-01', '%X %V') |
+-----------------------------------+
| 1998 52                           |
+-----------------------------------+

SELECT DATE_FORMAT('2006-06-00', '%d');
+---------------------------------+
| DATE_FORMAT('2006-06-00', '%d') |
+---------------------------------+
| 00                              |
+---------------------------------+
```

Optionally, the locale can be explicitly specified as the third DATE_FORMAT() argument. Doing so makes the function independent from the session settings, and the three argument version of DATE_FORMAT() can be used in virtual indexed and persistent generated-columns:

```
SELECT DATE_FORMAT('2006-01-01', '%W', 'el_GR');
+-----------------------------------------+
| DATE_FORMAT('2006-01-01', '%W', 'el_GR') |
+-----------------------------------------+
| Κυριακή                                 |
+-----------------------------------------+
```

From MariaDB 11.3, the timezone information:

```
SELECT DATE_FORMAT(NOW(), '%W %d %M %Y %H:%i:%s %Z %z');
+-------------------------------------------------+
| DATE_FORMAT(NOW(), '%W %d %M %Y %H:%i:%s %Z %z') |
+-------------------------------------------------+
| Wednesday 20 September 2023 15:00:23 SAST +0200  |
+-------------------------------------------------+
```

1.2.3.16 DATE_SUB

Syntax

```
DATE_SUB(date,INTERVAL expr unit)
```

Description

Performs date arithmetic. The *date* argument specifies the starting date or datetime value. *expr* is an expression specifying the interval value to be added or subtracted from the starting date. *expr* is a string; it may start with a " - " for negative intervals. *unit* is a keyword indicating the units in which the expression should be interpreted. See Date and Time Units for a complete list of permitted units.

See also `DATE_ADD()` .

Examples

```
SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
+-----------------------------------------+
| DATE_SUB('1998-01-02', INTERVAL 31 DAY) |
+-----------------------------------------+
| 1997-12-02                              |
+-----------------------------------------+
```

```
SELECT DATE_SUB('2005-01-01 00:00:00', INTERVAL '1 1:1:1' DAY_SECOND);
+-----------------------------------------------------------------+
| DATE_SUB('2005-01-01 00:00:00', INTERVAL '1 1:1:1' DAY_SECOND) |
+-----------------------------------------------------------------+
| 2004-12-30 22:58:59                                             |
+-----------------------------------------------------------------+
```

1.2.3.17 DAY

Syntax

```
DAY(date)
```

Description

`DAY()` is a synonym for `DAYOFMONTH()` .

1.2.3.18 DAYNAME

Syntax

```
DAYNAME(date)
```

Description

Returns the name of the weekday for date. The language used for the name is controlled by the value of the lc_time_names system variable. See server locale for more on the supported locales.

Examples

```
SELECT DAYNAME('2007-02-03');
+-----------------------+
| DAYNAME('2007-02-03') |
+-----------------------+
| Saturday              |
+-----------------------+
```

```
CREATE TABLE t1 (d DATETIME);
INSERT INTO t1 VALUES
    ("2007-01-30 21:31:07"),
    ("1983-10-15 06:42:51"),
    ("2011-04-21 12:34:56"),
    ("2011-10-30 06:31:41"),
    ("2011-01-30 14:03:25"),
    ("2004-10-07 11:19:34");
```

```
SELECT d, DAYNAME(d) FROM t1;
+---------------------+------------+
| d                   | DAYNAME(d) |
+---------------------+------------+
| 2007-01-30 21:31:07 | Tuesday    |
| 1983-10-15 06:42:51 | Saturday   |
| 2011-04-21 12:34:56 | Thursday   |
| 2011-10-30 06:31:41 | Sunday     |
| 2011-01-30 14:03:25 | Sunday     |
| 2004-10-07 11:19:34 | Thursday   |
+---------------------+------------+
```

Changing the locale:

```
SET lc_time_names = 'fr_CA';

SELECT DAYNAME('2013-04-01');
+-----------------------+
| DAYNAME('2013-04-01') |
+-----------------------+
| lundi                 |
+-----------------------+
```

1.2.3.19 DAYOFMONTH

Syntax

```
DAYOFMONTH(date)
```

Description

Returns the day of the month for date, in the range `1` to `31`, or `0` for dates such as `'0000-00-00'` or `'2008-00-00'` which have a zero day part.

DAY() is a synonym.

Examples

```
SELECT DAYOFMONTH('2007-02-03');
+--------------------------+
| DAYOFMONTH('2007-02-03') |
+--------------------------+
|                        3 |
+--------------------------+
```

```
CREATE TABLE t1 (d DATETIME);
INSERT INTO t1 VALUES
    ("2007-01-30 21:31:07"),
    ("1983-10-15 06:42:51"),
    ("2011-04-21 12:34:56"),
    ("2011-10-30 06:31:41"),
    ("2011-01-30 14:03:25"),
    ("2004-10-07 11:19:34");
```

```
SELECT d FROM t1 where DAYOFMONTH(d) = 30;
+---------------------+
| d                   |
+---------------------+
| 2007-01-30 21:31:07 |
| 2011-10-30 06:31:41 |
| 2011-01-30 14:03:25 |
+---------------------+
```

1.2.3.20 DAYOFWEEK

Syntax

```
DAYOFWEEK(date)
```

Description

Returns the day of the week index for the date (1 = Sunday, 2 = Monday, ..., 7 = Saturday). These index values correspond to the ODBC standard.

This contrasts with `WEEKDAY()` which follows a different index numbering (`0` = Monday, `1` = Tuesday, ... `6` = Sunday).

Examples

```
SELECT DAYOFWEEK('2007-02-03');
+--------------------------+
| DAYOFWEEK('2007-02-03') |
+--------------------------+
|                        7 |
+--------------------------+
```

```
CREATE TABLE t1 (d DATETIME);
INSERT INTO t1 VALUES
    ("2007-01-30 21:31:07"),
    ("1983-10-15 06:42:51"),
    ("2011-04-21 12:34:56"),
    ("2011-10-30 06:31:41"),
    ("2011-01-30 14:03:25"),
    ("2004-10-07 11:19:34");
```

```
SELECT d, DAYNAME(d), DAYOFWEEK(d), WEEKDAY(d) from t1;
+---------------------+------------+--------------+------------+
| d                   | DAYNAME(d) | DAYOFWEEK(d) | WEEKDAY(d) |
+---------------------+------------+--------------+------------+
| 2007-01-30 21:31:07 | Tuesday    |            3 |          1 |
| 1983-10-15 06:42:51 | Saturday   |            7 |          5 |
| 2011-04-21 12:34:56 | Thursday   |            5 |          3 |
| 2011-10-30 06:31:41 | Sunday     |            1 |          6 |
| 2011-01-30 14:03:25 | Sunday     |            1 |          6 |
| 2004-10-07 11:19:34 | Thursday   |            5 |          3 |
+---------------------+------------+--------------+------------+
```

1.2.3.21 DAYOFYEAR

Syntax

```
DAYOFYEAR(date)
```

Description

Returns the day of the year for date, in the range 1 to 366.

Examples

```
SELECT DAYOFYEAR('2018-02-16');
+-------------------------+
| DAYOFYEAR('2018-02-16') |
+-------------------------+
|                      47 |
+-------------------------+
```

1.2.3.22 EXTRACT

Syntax

```
EXTRACT(unit FROM date)
```

Contents

Description

The EXTRACT() function extracts the required unit from the date. See Date and Time Units for a complete list of permitted units.

In MariaDB 10.0.7 and MariaDB 5.5.35, `EXTRACT (HOUR FROM ...)` was changed to return a value from 0 to 23, adhering to the SQL standard. Until MariaDB 10.0.6 and MariaDB 5.5.34, and in all versions of MySQL at least as of MySQL 5.7, it could return a value > 23. HOUR() is not a standard function, so continues to adhere to the old behaviour inherited from MySQL.

Examples

```
SELECT EXTRACT(YEAR FROM '2009-07-02');
+---------------------------------+
| EXTRACT(YEAR FROM '2009-07-02') |
+---------------------------------+
|                            2009 |
+---------------------------------+

SELECT EXTRACT(YEAR_MONTH FROM '2009-07-02 01:02:03');
+-----------------------------------------------+
| EXTRACT(YEAR_MONTH FROM '2009-07-02 01:02:03') |
+-----------------------------------------------+
|                                        200907 |
+-----------------------------------------------+

SELECT EXTRACT(DAY_MINUTE FROM '2009-07-02 01:02:03');
+-----------------------------------------------+
| EXTRACT(DAY_MINUTE FROM '2009-07-02 01:02:03') |
+-----------------------------------------------+
|                                         20102 |
+-----------------------------------------------+

SELECT EXTRACT(MICROSECOND FROM '2003-01-02 10:30:00.000123');
+-------------------------------------------------------+
| EXTRACT(MICROSECOND FROM '2003-01-02 10:30:00.000123') |
+-------------------------------------------------------+
|                                                   123 |
+-------------------------------------------------------+
```

From MariaDB 10.0.7 and MariaDB 5.5.35, `EXTRACT (HOUR FROM...)` returns a value from 0 to 23, as per the SQL standard. `HOUR` is not a standard function, so continues to adhere to the old behaviour inherited from MySQL.

```
SELECT EXTRACT(HOUR FROM '26:30:00'), HOUR('26:30:00');
+--------------------------------+------------------+
| EXTRACT(HOUR FROM '26:30:00') | HOUR('26:30:00') |
+--------------------------------+------------------+
|                              2 |               26 |
+--------------------------------+------------------+
```

1.2.3.23 FORMAT_PICO_TIME

MariaDB starting with 11.0.2
Introduced in MariaDB 11.0.2

Syntax

```
FORMAT_PICO_TIME(time_val)
```

Contents

Description

Given a time in picoseconds, returns a human-readable time value and unit indicator. Resulting unit is dependent on the length of the argument, and can be:

- ps - picoseconds
- ns - nanoseconds
- us - microseconds
- ms - milliseconds
- s - seconds
- min - minutes
- h - hours

- d - days

With the exception of results under one nanosecond, which are not rounded and are represented as whole numbers, the result is rounded to 2 decimal places, with a minimum of 3 significant digits.

Returns NULL if the argument is NULL.

This function is very similar to the Sys Schema FORMAT_TIME function, but with the following differences:

- Represents minutes as `min` rather than `m`.
- Does not represent weeks.

Examples

```
SELECT
    FORMAT_PICO_TIME(43) AS ps,
    FORMAT_PICO_TIME(4321) AS ns,
    FORMAT_PICO_TIME(43211234) AS us,
    FORMAT_PICO_TIME(432112344321) AS ms,
    FORMAT_PICO_TIME(43211234432123) AS s,
    FORMAT_PICO_TIME(432112344321234) AS m,
    FORMAT_PICO_TIME(4321123443212345) AS h,
    FORMAT_PICO_TIME(432112344321234545) AS d;
+--------+---------+----------+-----------+---------+----------+--------+--------+
| ps     | ns      | us       | ms        | s       | m        | h      | d      |
+--------+---------+----------+-----------+---------+----------+--------+--------+
|  43 ps | 4.32 ns | 43.21 us | 432.11 ms | 43.21 s | 7.20 min | 1.20 h | 5.00 d |
+--------+---------+----------+-----------+---------+----------+--------+--------+
```

1.2.3.24 FROM_DAYS

Syntax

```
FROM_DAYS(N)
```

Description

Given a day number N, returns a DATE value. The day count is based on the number of days from the start of the standard calendar (0000-00-00).

The function is not designed for use with dates before the advent of the Gregorian calendar in October 1582. Results will not be reliable since it doesn't account for the lost days when the calendar changed from the Julian calendar.

This is the converse of the TO_DAYS() function.

Examples

```
SELECT FROM_DAYS(730669);
+-------------------+
| FROM_DAYS(730669) |
+-------------------+
| 2000-07-03        |
+-------------------+
```

1.2.3.25 FROM_UNIXTIME

Syntax

```
FROM_UNIXTIME(unix_timestamp), FROM_UNIXTIME(unix_timestamp,format)
```

Contents

Description

Returns a representation of the unix_timestamp argument as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS.uuuuuu format, depending on whether the function is used in a string or numeric context. The value is expressed in the current time zone. unix_timestamp is an internal timestamp value such as is produced by the UNIX_TIMESTAMP() function.

If format is given, the result is formatted according to the format string, which is used the same way as listed in the entry for the DATE_FORMAT() function.

Timestamps in MariaDB have a maximum value of 2147483647, equivalent to 2038-01-19 05:14:07. This is due to the underlying 32-bit limitation. Using the function on a timestamp beyond this will result in NULL being returned. Use DATETIME as a storage type if you require dates beyond this.

The options that can be used by FROM_UNIXTIME(), as well as DATE_FORMAT() and STR_TO_DATE(), are:

Option	Description
`%a`	Short weekday name in current locale (Variable lc_time_names).
`%b`	Short form month name in current locale. For locale en_US this is one of: Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov or Dec.
`%c`	Month with 1 or 2 digits.
`%D`	Day with English suffix 'th', 'nd', 'st' or 'rd''. (1st, 2nd, 3rd...).
`%d`	Day with 2 digits.
`%e`	Day with 1 or 2 digits.
`%f`	Microseconds 6 digits.
`%H`	Hour with 2 digits between 00-23.
`%h`	Hour with 2 digits between 01-12.
`%I`	Hour with 2 digits between 01-12.
`%i`	Minute with 2 digits.
`%j`	Day of the year (001-366)
`%k`	Hour with 1 digits between 0-23.
`%l`	Hour with 1 digits between 1-12.
`%M`	Full month name in current locale (Variable lc_time_names).
`%m`	Month with 2 digits.
`%p`	AM/PM according to current locale (Variable lc_time_names).
`%r`	Time in 12 hour format, followed by AM/PM. Short for '%I:%i:%S %p'.
`%S`	Seconds with 2 digits.
`%s`	Seconds with 2 digits.
`%T`	Time in 24 hour format. Short for '%H:%i:%S'.
`%U`	Week number (00-53), when first day of the week is Sunday.
`%u`	Week number (00-53), when first day of the week is Monday.
`%V`	Week number (01-53), when first day of the week is Sunday. Used with %X.
`%v`	Week number (01-53), when first day of the week is Monday. Used with %x.
`%W`	Full weekday name in current locale (Variable lc_time_names).

%w	Day of the week. 0 = Sunday, 6 = Saturday.
%X	Year with 4 digits when first day of the week is Sunday. Used with %V.
%x	Year with 4 digits when first day of the week is Sunday. Used with %v.
%Y	Year with 4 digits.
%y	Year with 2 digits.
%#	For str_to_date(), skip all numbers.
%.	For str_to_date(), skip all punctation characters.
%@	For str_to_date(), skip all alpha characters.
%%	A literal % character.

Performance Considerations

If your session time zone is set to `SYSTEM` (the default), `FROM_UNIXTIME()` will call the OS function to convert the data using the system time zone. At least on Linux, the corresponding function (`localtime_r`) uses a global mutex inside glibc that can cause contention under high concurrent load.

Set your time zone to a named time zone to avoid this issue. See mysql time zone tables for details on how to do this.

Examples

```
SELECT FROM_UNIXTIME(1196440219);
+---------------------------+
| FROM_UNIXTIME(1196440219) |
+---------------------------+
| 2007-11-30 11:30:19       |
+---------------------------+

SELECT FROM_UNIXTIME(1196440219) + 0;
+------------------------------+
| FROM_UNIXTIME(1196440219) + 0 |
+------------------------------+
|         20071130113019.000000 |
+------------------------------+

SELECT FROM_UNIXTIME(UNIX_TIMESTAMP(), '%Y %D %M %h:%i:%s %x');
+-----------------------------------------------------------+
| FROM_UNIXTIME(UNIX_TIMESTAMP(), '%Y %D %M %h:%i:%s %x') |
+-----------------------------------------------------------+
| 2010 27th March 01:03:47 2010                             |
+-----------------------------------------------------------+
```

1.2.3.26 GET_FORMAT

Syntax

```
GET_FORMAT({DATE|DATETIME|TIME}, {'EUR'|'USA'|'JIS'|'ISO'|'INTERNAL'})
```

Description

Returns a format string. This function is useful in combination with the DATE_FORMAT() and the STR_TO_DATE() functions.

Possible result formats are:

Function Call	Result Format
GET_FORMAT(DATE,'EUR')	'%d.%m.%Y'

GET_FORMAT(DATE,'USA')	'%m.%d.%Y'
GET_FORMAT(DATE,'JIS')	'%Y-%m-%d'
GET_FORMAT(DATE,'ISO')	'%Y-%m-%d'
GET_FORMAT(DATE,'INTERNAL')	'%Y%m%d'
GET_FORMAT(DATETIME,'EUR')	'%Y-%m-%d %H.%i.%s'
GET_FORMAT(DATETIME,'USA')	'%Y-%m-%d %H.%i.%s'
GET_FORMAT(DATETIME,'JIS')	'%Y-%m-%d %H:%i:%s'
GET_FORMAT(DATETIME,'ISO')	'%Y-%m-%d %H:%i:%s'
GET_FORMAT(DATETIME,'INTERNAL')	'%Y%m%d%H%i%s'
GET_FORMAT(TIME,'EUR')	'%H.%i.%s'
GET_FORMAT(TIME,'USA')	'%h:%i:%s %p'
GET_FORMAT(TIME,'JIS')	'%H:%i:%s'
GET_FORMAT(TIME,'ISO')	'%H:%i:%s'
GET_FORMAT(TIME,'INTERNAL')	'%H%i%s'

Examples

Obtaining the string matching to the standard European date format:

```
SELECT GET_FORMAT(DATE, 'EUR');
+--------------------------+
| GET_FORMAT(DATE, 'EUR') |
+--------------------------+
| %d.%m.%Y                 |
+--------------------------+
```

Using the same string to format a date:

```
SELECT DATE_FORMAT('2003-10-03',GET_FORMAT(DATE,'EUR'));
+-----------------------------------------------+
| DATE_FORMAT('2003-10-03',GET_FORMAT(DATE,'EUR')) |
+-----------------------------------------------+
| 03.10.2003                                    |
+-----------------------------------------------+

SELECT STR_TO_DATE('10.31.2003',GET_FORMAT(DATE,'USA'));
+-----------------------------------------------+
| STR_TO_DATE('10.31.2003',GET_FORMAT(DATE,'USA')) |
+-----------------------------------------------+
| 2003-10-31                                    |
+-----------------------------------------------+
```

1.2.3.27 HOUR

Syntax

```
HOUR(time)
```

Description

Returns the hour for time. The range of the return value is 0 to 23 for time-of-day values. However, the range of `TIME` values actually is much larger, so HOUR can return values greater than 23.

The return value is always positive, even if a negative `TIME` value is provided.

Examples

```
SELECT HOUR('10:05:03');
+------------------+
| HOUR('10:05:03') |
+------------------+
|               10 |
+------------------+

SELECT HOUR('272:59:59');
+--------------------+
| HOUR('272:59:59') |
+--------------------+
|                272 |
+--------------------+
```

Difference between `EXTRACT (HOUR FROM ...)` (>= MariaDB 10.0.7 and MariaDB 5.5.35) and `HOUR`:

```
SELECT EXTRACT(HOUR FROM '26:30:00'), HOUR('26:30:00');
+--------------------------------+------------------+
| EXTRACT(HOUR FROM '26:30:00') | HOUR('26:30:00') |
+--------------------------------+------------------+
|                              2 |               26 |
+--------------------------------+------------------+
```

1.2.3.28 LAST_DAY

Syntax

```
LAST_DAY(date)
```

Description

Takes a date or datetime value and returns the corresponding value for the last day of the month. Returns NULL if the argument is invalid.

Examples

```
SELECT LAST_DAY('2003-02-05');
+-----------------------+
| LAST_DAY('2003-02-05') |
+-----------------------+
| 2003-02-28            |
+-----------------------+

SELECT LAST_DAY('2004-02-05');
+-----------------------+
| LAST_DAY('2004-02-05') |
+-----------------------+
| 2004-02-29            |
+-----------------------+

SELECT LAST_DAY('2004-01-01 01:01:01');
+--------------------------------+
| LAST_DAY('2004-01-01 01:01:01') |
+--------------------------------+
| 2004-01-31                     |
+--------------------------------+

SELECT LAST_DAY('2003-03-32');
+-----------------------+
| LAST_DAY('2003-03-32') |
+-----------------------+
| NULL                  |
+-----------------------+
1 row in set, 1 warning (0.00 sec)

Warning (Code 1292): Incorrect datetime value: '2003-03-32'
```

1.2.3.29 LOCALTIME

Syntax

```
LOCALTIME
LOCALTIME([precision])
```

Description

`LOCALTIME` and `LOCALTIME()` are synonyms for `NOW()`.

1.2.3.30 LOCALTIMESTAMP

Syntax

```
LOCALTIMESTAMP
LOCALTIMESTAMP([precision])
```

Description

`LOCALTIMESTAMP` and `LOCALTIMESTAMP()` are synonyms for `NOW()`.

1.2.3.31 MAKEDATE

Syntax

```
MAKEDATE(year,dayofyear)
```

Description

Returns a date, given `year` and `day-of-year values`. `dayofyear` must be greater than 0 or the result is NULL.

Examples

```
SELECT MAKEDATE(2011,31), MAKEDATE(2011,32);
+-------------------+-------------------+
| MAKEDATE(2011,31) | MAKEDATE(2011,32) |
+-------------------+-------------------+
| 2011-01-31        | 2011-02-01        |
+-------------------+-------------------+

SELECT MAKEDATE(2011,365), MAKEDATE(2014,365);
+--------------------+--------------------+
| MAKEDATE(2011,365) | MAKEDATE(2014,365) |
+--------------------+--------------------+
| 2011-12-31         | 2014-12-31         |
+--------------------+--------------------+

SELECT MAKEDATE(2011,0);
+------------------+
| MAKEDATE(2011,0) |
+------------------+
| NULL             |
+------------------+
```

1.2.3.32 MAKETIME

Syntax

```
MAKETIME(hour,minute,second)
```

Description

Returns a time value calculated from the `hour`, `minute`, and `second` arguments.

If `minute` or `second` are out of the range 0 to 60, NULL is returned. The `hour` can be in the range -838 to 838, outside of which the value is truncated with a warning.

Examples

```
SELECT MAKETIME(13,57,33);
+--------------------+
| MAKETIME(13,57,33) |
+--------------------+
| 13:57:33           |
+--------------------+

SELECT MAKETIME(-13,57,33);
+---------------------+
| MAKETIME(-13,57,33) |
+---------------------+
| -13:57:33           |
+---------------------+

SELECT MAKETIME(13,67,33);
+--------------------+
| MAKETIME(13,67,33) |
+--------------------+
| NULL               |
+--------------------+

SELECT MAKETIME(-1000,57,33);
+-----------------------+
| MAKETIME(-1000,57,33) |
+-----------------------+
| -838:59:59            |
+-----------------------+
1 row in set, 1 warning (0.00 sec)

SHOW WARNINGS;
+---------+------+-----------------------------------------------+
| Level   | Code | Message                                       |
+---------+------+-----------------------------------------------+
| Warning | 1292 | Truncated incorrect time value: '-1000:57:33' |
+---------+------+-----------------------------------------------+
```

1.2.3.33 MICROSECOND

Syntax

```
MICROSECOND(expr)
```

Contents

Description

Returns the microseconds from the time or datetime expression *expr* as a number in the range from 0 to 999999.

If *expr* is a time with no microseconds, zero is returned, while if *expr* is a date with no time, zero with a warning is returned.

Examples

```
SELECT MICROSECOND('12:00:00.123456');
+-------------------------------+
| MICROSECOND('12:00:00.123456') |
+-------------------------------+
|                         123456 |
+-------------------------------+

SELECT MICROSECOND('2009-12-31 23:59:59.000010');
+----------------------------------------+
| MICROSECOND('2009-12-31 23:59:59.000010') |
+----------------------------------------+
|                                     10 |
+----------------------------------------+

SELECT MICROSECOND('2013-08-07 12:13:14');
+-----------------------------------+
| MICROSECOND('2013-08-07 12:13:14') |
+-----------------------------------+
|                                 0 |
+-----------------------------------+

SELECT MICROSECOND('2013-08-07');
+---------------------------+
| MICROSECOND('2013-08-07') |
+---------------------------+
|                         0 |
+---------------------------+
1 row in set, 1 warning (0.00 sec)

SHOW WARNINGS;
+---------+------+-----------------------------------------------+
| Level   | Code | Message                                       |
+---------+------+-----------------------------------------------+
| Warning | 1292 | Truncated incorrect time value: '2013-08-07' |
+---------+------+-----------------------------------------------+
```

1.2.3.34 MINUTE

Syntax

```
MINUTE(time)
```

Description

Returns the minute for *time*, in the range 0 to 59.

Examples

```
SELECT MINUTE('2013-08-03 11:04:03');
+-------------------------------+
| MINUTE('2013-08-03 11:04:03') |
+-------------------------------+
|                             4 |
+-------------------------------+

 SELECT MINUTE ('23:12:50');
+---------------------+
| MINUTE ('23:12:50') |
+---------------------+
|                  12 |
+---------------------+
```

1.2.3.35 MONTH

Syntax

```
MONTH(date)
```

Description

Returns the month for `date` in the range 1 to 12 for January to December, or 0 for dates such as '0000-00-00' or '2008-00-00' that have a zero month part.

Examples

```
SELECT MONTH('2019-01-03');
+---------------------+
| MONTH('2019-01-03') |
+---------------------+
|                   1 |
+---------------------+

SELECT MONTH('2019-00-03');
+---------------------+
| MONTH('2019-00-03') |
+---------------------+
|                   0 |
+---------------------+
```

1.2.3.36 MONTHNAME

Syntax

```
MONTHNAME(date)
```

Description

Returns the full name of the month for date. The language used for the name is controlled by the value of the lc_time_names system variable. See server locale for more on the supported locales.

Examples

```
SELECT MONTHNAME('2019-02-03');
+--------------------------+
| MONTHNAME('2019-02-03') |
+--------------------------+
| February                 |
+--------------------------+
```

Changing the locale:

```
SET lc_time_names = 'fr_CA';

SELECT MONTHNAME('2019-05-21');
+--------------------------+
| MONTHNAME('2019-05-21') |
+--------------------------+
| mai                      |
+--------------------------+
```

1.2.3.37 NOW

Syntax

```
NOW([precision])
CURRENT_TIMESTAMP
CURRENT_TIMESTAMP([precision])
LOCALTIME, LOCALTIME([precision])
LOCALTIMESTAMP
LOCALTIMESTAMP([precision])
```

Contents

Description

Returns the current date and time as a value in `'YYYY-MM-DD HH:MM:SS'` or `YYYYMMDDHHMMSS.uuuuuu` format, depending on whether the function is used in a string or numeric context. The value is expressed in the current time zone.

The optional *precision* determines the microsecond precision. See Microseconds in MariaDB.

`NOW()` (or its synonyms) can be used as the default value for TIMESTAMP columns as well as, since MariaDB 10.0.1, DATETIME columns. Before MariaDB 10.0.1, it was only possible for a single TIMESTAMP column per table to contain the CURRENT_TIMESTAMP as its default.

When displayed in the INFORMATION_SCHEMA.COLUMNS table, a default CURRENT TIMESTAMP is displayed as `CURRENT_TIMESTAMP` up until MariaDB 10.2.2, and as `current_timestamp()` from MariaDB 10.2.3, due to to MariaDB 10.2 accepting expressions in the DEFAULT clause.

Changing the timestamp system variable with a SET `timestamp` statement affects the value returned by NOW(), but not by SYSDATE().

Examples

```
SELECT NOW();
+---------------------+
| NOW()               |
+---------------------+
| 2010-03-27 13:13:25 |
+---------------------+

SELECT NOW() + 0;
+-----------------------+
| NOW() + 0             |
+-----------------------+
| 20100327131329.000000 |
+-----------------------+
```

With precision:

```
SELECT CURRENT_TIMESTAMP(2);
+------------------------+
| CURRENT_TIMESTAMP(2)   |
+------------------------+
| 2018-07-10 09:47:26.24 |
+------------------------+
```

Used as a default TIMESTAMP:

```
CREATE TABLE t (createdTS TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP);
```

From MariaDB 10.2.2:

```
SELECT * FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_SCHEMA='test'
  AND COLUMN_NAME LIKE '%ts%'\G
*************************** 1. row ***************************
           TABLE_CATALOG: def
            TABLE_SCHEMA: test
              TABLE_NAME: t
             COLUMN_NAME: ts
        ORDINAL_POSITION: 1
          COLUMN_DEFAULT: current_timestamp()
...
```

<= MariaDB 10.2.1

```
SELECT * FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_SCHEMA='test'
  AND COLUMN_NAME LIKE '%ts%'\G
*************************** 1. row ***************************
           TABLE_CATALOG: def
            TABLE_SCHEMA: test
              TABLE_NAME: t
             COLUMN_NAME: createdTS
        ORDINAL_POSITION: 1
          COLUMN_DEFAULT: CURRENT_TIMESTAMP
...
```

1.2.3.38 PERIOD_ADD

Syntax

```
PERIOD_ADD(P,N)
```

Description

Adds `N` months to period `P`. `P` is in the format YYMM or YYYYMM, and is not a date value. If `P` contains a two-digit year, values from 00 to 69 are converted to from 2000 to 2069, while values from 70 are converted to 1970 upwards.

Returns a value in the format YYYYMM.

Examples

```
SELECT PERIOD_ADD(200801,2);
+----------------------+
| PERIOD_ADD(200801,2) |
+----------------------+
|               200803 |
+----------------------+

SELECT PERIOD_ADD(6910,2);
+--------------------+
| PERIOD_ADD(6910,2) |
+--------------------+
|             206912 |
+--------------------+

SELECT PERIOD_ADD(7010,2);
+--------------------+
| PERIOD_ADD(7010,2) |
+--------------------+
|             197012 |
+--------------------+
```

1.2.3.39 PERIOD_DIFF

Syntax

```
PERIOD_DIFF(P1,P2)
```

Description

Returns the number of months between periods P1 and P2. P1 and P2 can be in the format `YYMM` or `YYYYMM`, and are not date values.

If P1 or P2 contains a two-digit year, values from 00 to 69 are converted to from 2000 to 2069, while values from 70 are converted to 1970 upwards.

Examples

```
SELECT PERIOD_DIFF(200802,200703);
+----------------------------+
| PERIOD_DIFF(200802,200703) |
+----------------------------+
|                         11 |
+----------------------------+

SELECT PERIOD_DIFF(6902,6803);
+------------------------+
| PERIOD_DIFF(6902,6803) |
+------------------------+
|                     11 |
+------------------------+

SELECT PERIOD_DIFF(7002,6803);
+------------------------+
| PERIOD_DIFF(7002,6803) |
+------------------------+
|                  -1177 |
+------------------------+
```

1.2.3.40 QUARTER

Syntax

```
QUARTER(date)
```

Description

Returns the quarter of the year for `date`, in the range 1 to 4. Returns 0 if month contains a zero value, or `NULL` if the given value is not otherwise a valid date (zero values are accepted).

Examples

```
SELECT QUARTER('2008-04-01');
+-----------------------+
| QUARTER('2008-04-01') |
+-----------------------+
|                     2 |
+-----------------------+

SELECT QUARTER('2019-00-01');
+-----------------------+
| QUARTER('2019-00-01') |
+-----------------------+
|                     0 |
+-----------------------+
```

1.2.3.41 SECOND

Syntax

```
SECOND(time)
```

Description

Returns the second for a given `time` (which can include microseconds), in the range 0 to 59, or `NULL` if not given a valid time value.

Examples

```
SELECT SECOND('10:05:03');
+--------------------+
| SECOND('10:05:03') |
+--------------------+
|                  3 |
+--------------------+

SELECT SECOND('10:05:01.999999');
+---------------------------+
| SECOND('10:05:01.999999') |
+---------------------------+
|                         1 |
+---------------------------+
```

1.2.3.42 SEC_TO_TIME

Syntax

```
SEC_TO_TIME(seconds)
```

Description

Returns the seconds argument, converted to hours, minutes, and seconds, as a TIME value. The range of the result is constrained to that of the TIME data type. A warning occurs if the argument corresponds to a value outside that range.

The time will be returned in the format `hh:mm:ss`, or `hhmmss` if used in a numeric calculation.

Examples

```
SELECT SEC_TO_TIME(12414);
+--------------------+
| SEC_TO_TIME(12414) |
+--------------------+
| 03:26:54           |
+--------------------+

SELECT SEC_TO_TIME(12414)+0;
+----------------------+
| SEC_TO_TIME(12414)+0 |
+----------------------+
|                32654 |
+----------------------+

SELECT SEC_TO_TIME(9999999);
+----------------------+
| SEC_TO_TIME(9999999) |
+----------------------+
| 838:59:59            |
+----------------------+
1 row in set, 1 warning (0.00 sec)

SHOW WARNINGS;
+---------+------+-----------------------------------------+
| Level   | Code | Message                                 |
+---------+------+-----------------------------------------+
| Warning | 1292 | Truncated incorrect time value: '9999999' |
+---------+------+-----------------------------------------+
```

1.2.3.43 STR_TO_DATE

Syntax

```
STR_TO_DATE(str,format)
```

Contents

Description

This is the inverse of the DATE_FORMAT() function. It takes a string `str` and a format string `format`. `STR_TO_DATE()` returns a `DATETIME` value if the format string contains both date and time parts, or a `DATE` or `TIME` value if the string contains only date or time parts.

The date, time, or datetime values contained in `str` should be given in the format indicated by format. If str contains an illegal date, time, or datetime value, `STR_TO_DATE()` returns `NULL`. An illegal value also produces a warning.

Under specific SQL_MODE settings an error may also be generated if the `str` isn't a valid date:

- ALLOW_INVALID_DATES
- NO_ZERO_DATE
- NO_ZERO_IN_DATE

The options that can be used by STR_TO_DATE(), as well as its inverse DATE_FORMAT() and the FROM_UNIXTIME() function, are:

Option	Description
%a	Short weekday name in current locale (Variable lc_time_names).
%b	Short form month name in current locale. For locale en_US this is one of: Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov or Dec.
%c	Month with 1 or 2 digits.

%D	Day with English suffix 'th', 'nd', 'st' or 'rd''. (1st, 2nd, 3rd...).
%d	Day with 2 digits.
%e	Day with 1 or 2 digits.
%f	Microseconds 6 digits.
%H	Hour with 2 digits between 00-23.
%h	Hour with 2 digits between 01-12.
%I	Hour with 2 digits between 01-12.
%i	Minute with 2 digits.
%j	Day of the year (001-366)
%k	Hour with 1 digits between 0-23.
%l	Hour with 1 digits between 1-12.
%M	Full month name in current locale (Variable lc_time_names).
%m	Month with 2 digits.
%p	AM/PM according to current locale (Variable lc_time_names).
%r	Time in 12 hour format, followed by AM/PM. Short for '%I:%i:%S %p'.
%S	Seconds with 2 digits.
%s	Seconds with 2 digits.
%T	Time in 24 hour format. Short for '%H:%i:%S'.
%U	Week number (00-53), when first day of the week is Sunday.
%u	Week number (00-53), when first day of the week is Monday.
%V	Week number (01-53), when first day of the week is Sunday. Used with %X.
%v	Week number (01-53), when first day of the week is Monday. Used with %x.
%W	Full weekday name in current locale (Variable lc_time_names).
%w	Day of the week. 0 = Sunday, 6 = Saturday.
%X	Year with 4 digits when first day of the week is Sunday. Used with %V.
%x	Year with 4 digits when first day of the week is Monday. Used with %v.
%Y	Year with 4 digits.
%y	Year with 2 digits.
%#	For str_to_date(), skip all numbers.
%.	For str_to_date(), skip all punctation characters.
%@	For str_to_date(), skip all alpha characters.
%%	A literal % character.

Examples

```
SELECT STR_TO_DATE('Wednesday, June 2, 2014', '%W, %M %e, %Y');
+-----------------------------------------------------+
| STR_TO_DATE('Wednesday, June 2, 2014', '%W, %M %e, %Y') |
+-----------------------------------------------------+
| 2014-06-02                                          |
+-----------------------------------------------------+


SELECT STR_TO_DATE('Wednesday23423, June 2, 2014', '%W, %M %e, %Y');
+-------------------------------------------------------------+
| STR_TO_DATE('Wednesday23423, June 2, 2014', '%W, %M %e, %Y') |
+-------------------------------------------------------------+
| NULL                                                        |
+-------------------------------------------------------------+
1 row in set, 1 warning (0.00 sec)

SHOW WARNINGS;
+---------+------+-----------------------------------------------------------------------------------------
| Level   | Code | Message
+---------+------+-----------------------------------------------------------------------------------------
| Warning | 1411 | Incorrect datetime value: 'Wednesday23423, June 2, 2014' for function str_to_d
+---------+------+-----------------------------------------------------------------------------------------

SELECT STR_TO_DATE('Wednesday23423, June 2, 2014', '%W%#, %M %e, %Y');
+-------------------------------------------------------------+
| STR_TO_DATE('Wednesday23423, June 2, 2014', '%W%#, %M %e, %Y') |
+-------------------------------------------------------------+
| 2014-06-02                                                  |
+-------------------------------------------------------------+
```

1.2.3.44 SUBDATE

Syntax

```
SUBDATE(date,INTERVAL expr unit), SUBDATE(expr,days)
```

Description

When invoked with the `INTERVAL` form of the second argument, `SUBDATE()` is a synonym for `DATE_SUB()`. See Date and Time Units for a complete list of permitted units.

The second form allows the use of an integer value for days. In such cases, it is interpreted as the number of days to be subtracted from the date or datetime expression expr.

Examples

```
SELECT DATE_SUB('2008-01-02', INTERVAL 31 DAY);
+----------------------------------------+
| DATE_SUB('2008-01-02', INTERVAL 31 DAY) |
+----------------------------------------+
| 2007-12-02                             |
+----------------------------------------+

SELECT SUBDATE('2008-01-02', INTERVAL 31 DAY);
+----------------------------------------+
| SUBDATE('2008-01-02', INTERVAL 31 DAY) |
+----------------------------------------+
| 2007-12-02                             |
+----------------------------------------+
```

```
SELECT SUBDATE('2008-01-02 12:00:00', 31);
+-----------------------------------+
| SUBDATE('2008-01-02 12:00:00', 31) |
+-----------------------------------+
| 2007-12-02 12:00:00                |
+-----------------------------------+
```

```
CREATE TABLE t1 (d DATETIME);
INSERT INTO t1 VALUES
    ("2007-01-30 21:31:07"),
    ("1983-10-15 06:42:51"),
    ("2011-04-21 12:34:56"),
    ("2011-10-30 06:31:41"),
    ("2011-01-30 14:03:25"),
    ("2004-10-07 11:19:34");
```

```
SELECT d, SUBDATE(d, 10) from t1;
+---------------------+---------------------+
| d                   | SUBDATE(d, 10)      |
+---------------------+---------------------+
| 2007-01-30 21:31:07 | 2007-01-20 21:31:07 |
| 1983-10-15 06:42:51 | 1983-10-05 06:42:51 |
| 2011-04-21 12:34:56 | 2011-04-11 12:34:56 |
| 2011-10-30 06:31:41 | 2011-10-20 06:31:41 |
| 2011-01-30 14:03:25 | 2011-01-20 14:03:25 |
| 2004-10-07 11:19:34 | 2004-09-27 11:19:34 |
+---------------------+---------------------+

SELECT d, SUBDATE(d, INTERVAL 10 MINUTE) from t1;
+---------------------+-------------------------------+
| d                   | SUBDATE(d, INTERVAL 10 MINUTE) |
+---------------------+-------------------------------+
| 2007-01-30 21:31:07 | 2007-01-30 21:21:07            |
| 1983-10-15 06:42:51 | 1983-10-15 06:32:51            |
| 2011-04-21 12:34:56 | 2011-04-21 12:24:56            |
| 2011-10-30 06:31:41 | 2011-10-30 06:21:41            |
| 2011-01-30 14:03:25 | 2011-01-30 13:53:25            |
| 2004-10-07 11:19:34 | 2004-10-07 11:09:34            |
+---------------------+-------------------------------+
```

1.2.3.45 SUBTIME

Syntax

```
SUBTIME(expr1,expr2)
```

Description

SUBTIME() returns `expr1 - expr2` expressed as a value in the same format as `expr1`. `expr1` is a time or datetime expression, and expr2 is a time expression.

Examples

```
SELECT SUBTIME('2007-12-31 23:59:59.999999','1 1:1:1.000002');
+-----------------------------------------------------------+
| SUBTIME('2007-12-31 23:59:59.999999','1 1:1:1.000002') |
+-----------------------------------------------------------+
| 2007-12-30 22:58:58.999997                                |
+-----------------------------------------------------------+

SELECT SUBTIME('01:00:00.999999', '02:00:00.999998');
+---------------------------------------------+
| SUBTIME('01:00:00.999999', '02:00:00.999998') |
+---------------------------------------------+
| -00:59:59.999999                            |
+---------------------------------------------+
```

1.2.3.46 SYSDATE

Syntax

```
SYSDATE([precision])
```

Contents

Description

Returns the current date and time as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS.uuuuuu format, depending on whether the function is used in a string or numeric context.

The optional *precision* determines the microsecond precision. See Microseconds in MariaDB.

SYSDATE() returns the time at which it executes. This differs from the behavior for NOW(), which returns a constant time that indicates the time at which the statement began to execute. (Within a stored routine or trigger, NOW() returns the time at which the routine or triggering statement began to execute.)

In addition, changing the timestamp system variable with a SET `timestamp` statement affects the value returned by NOW() but not by SYSDATE(). This means that timestamp settings in the binary log have no effect on invocations of SYSDATE().

Because SYSDATE() can return different values even within the same statement, and is not affected by SET TIMESTAMP, it is non-deterministic and therefore unsafe for replication if statement-based binary logging is used. If that is a problem, you can use row-based logging, or start the server with the mysqld option --sysdate-is-now to cause SYSDATE() to be an alias for NOW(). The non-deterministic nature of SYSDATE() also means that indexes cannot be used for evaluating expressions that refer to it, and that statements using the SYSDATE() function are unsafe for statement-based replication.

Examples

Difference between NOW() and SYSDATE():

```
SELECT NOW(), SLEEP(2), NOW();
+---------------------+----------+---------------------+
| NOW()               | SLEEP(2) | NOW()               |
+---------------------+----------+---------------------+
| 2010-03-27 13:23:40 |        0 | 2010-03-27 13:23:40 |
+---------------------+----------+---------------------+

SELECT SYSDATE(), SLEEP(2), SYSDATE();
+---------------------+----------+---------------------+
| SYSDATE()           | SLEEP(2) | SYSDATE()           |
+---------------------+----------+---------------------+
| 2010-03-27 13:23:52 |        0 | 2010-03-27 13:23:54 |
+---------------------+----------+---------------------+
```

With precision:

```
SELECT SYSDATE(4);
+--------------------------+
| SYSDATE(4)               |
+--------------------------+
| 2018-07-10 10:17:13.1689 |
+--------------------------+
```

1.2.3.47 TIME Function

Syntax

```
TIME(expr)
```

Description

Extracts the time part of the time or datetime expression `expr` and returns it as a string.

Examples

```
SELECT TIME('2003-12-31 01:02:03');
+-----------------------------+
| TIME('2003-12-31 01:02:03') |
+-----------------------------+
| 01:02:03                    |
+-----------------------------+

SELECT TIME('2003-12-31 01:02:03.000123');
+--------------------------------------+
| TIME('2003-12-31 01:02:03.000123') |
+--------------------------------------+
| 01:02:03.000123                      |
+--------------------------------------+
```

1.2.3.48 TIMEDIFF

Syntax

```
TIMEDIFF(expr1,expr2)
```

Description

TIMEDIFF() returns `expr1 - expr2` expressed as a time value. `expr1` and `expr2` are time or date-and-time expressions, but both must be of the same type.

Examples

```
SELECT TIMEDIFF('2000:01:01 00:00:00', '2000:01:01 00:00:00.000001');
+----------------------------------------------------------------+
| TIMEDIFF('2000:01:01 00:00:00', '2000:01:01 00:00:00.000001') |
+----------------------------------------------------------------+
| -00:00:00.000001                                               |
+----------------------------------------------------------------+

SELECT TIMEDIFF('2008-12-31 23:59:59.000001', '2008-12-30 01:01:01.000002');
+----------------------------------------------------------------------+
| TIMEDIFF('2008-12-31 23:59:59.000001', '2008-12-30 01:01:01.000002') |
+----------------------------------------------------------------------+
| 46:58:57.999999                                                      |
+----------------------------------------------------------------------+
```

1.2.3.49 TIMESTAMP FUNCTION

Syntax

```
TIMESTAMP(expr), TIMESTAMP(expr1,expr2)
```

Description

With a single argument, this function returns the date or datetime expression `expr` as a datetime value. With two arguments, it adds the time expression `expr2` to the date or datetime expression `expr1` and returns the result as a datetime value.

Examples

```
SELECT TIMESTAMP('2003-12-31');
+-------------------------+
| TIMESTAMP('2003-12-31') |
+-------------------------+
| 2003-12-31 00:00:00     |
+-------------------------+

SELECT TIMESTAMP('2003-12-31 12:00:00','6:30:00');
+-----------------------------------------+
| TIMESTAMP('2003-12-31 12:00:00','6:30:00') |
+-----------------------------------------+
| 2003-12-31 18:30:00                     |
+-----------------------------------------+
```

1.2.3.50 TIMESTAMPADD

Syntax

```
TIMESTAMPADD(unit,interval,datetime_expr)
```

Description

Adds the integer expression interval to the date or datetime expression datetime_expr. The unit for interval is given by the unit argument, which should be one of the following values: MICROSECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, or YEAR.

The unit value may be specified using one of keywords as shown, or with a prefix of SQL_TSI_. For example, DAY and SQL_TSI_DAY both are legal.

Before MariaDB 5.5, FRAC_SECOND was permitted as a synonym for MICROSECOND.

Examples

```
SELECT TIMESTAMPADD(MINUTE,1,'2003-01-02');
+-------------------------------------+
| TIMESTAMPADD(MINUTE,1,'2003-01-02') |
+-------------------------------------+
| 2003-01-02 00:01:00                 |
+-------------------------------------+

SELECT TIMESTAMPADD(WEEK,1,'2003-01-02');
+----------------------------------+
| TIMESTAMPADD(WEEK,1,'2003-01-02') |
+----------------------------------+
| 2003-01-09                       |
+----------------------------------+
```

1.2.3.51 TIMESTAMPDIFF

Syntax

```
TIMESTAMPDIFF(unit,datetime_expr1,datetime_expr2)
```

Description

Returns `datetime_expr2 - datetime_expr1`, where `datetime_expr1` and `datetime_expr2` are date or datetime expressions. One expression may be a date and the other a datetime; a date value is treated as a datetime having the time part '00:00:00' where necessary. The unit for the result (an integer) is given by the unit argument. The legal values for unit are the same as those listed in the description of the TIMESTAMPADD() function, i.e MICROSECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, or YEAR.

`TIMESTAMPDIFF` can also be used to calculate age.

Examples

```
SELECT TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01');
+----------------------------------------------+
| TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01') |
+----------------------------------------------+
|                                            3 |
+----------------------------------------------+

SELECT TIMESTAMPDIFF(YEAR,'2002-05-01','2001-01-01');
+---------------------------------------------+
| TIMESTAMPDIFF(YEAR,'2002-05-01','2001-01-01') |
+---------------------------------------------+
|                                          -1 |
+---------------------------------------------+

SELECT TIMESTAMPDIFF(MINUTE,'2003-02-01','2003-05-01 12:05:55');
+-----------------------------------------------------------+
| TIMESTAMPDIFF(MINUTE,'2003-02-01','2003-05-01 12:05:55') |
+-----------------------------------------------------------+
|                                                    128885 |
+-----------------------------------------------------------+
```

Calculating age:

```
SELECT CURDATE();
+------------+
| CURDATE()  |
+------------+
| 2019-05-27 |
+------------+

SELECT TIMESTAMPDIFF(YEAR, '1971-06-06', CURDATE()) AS age;
+------+
| age  |
+------+
|   47 |
+------+

SELECT TIMESTAMPDIFF(YEAR, '1971-05-06', CURDATE()) AS age;
+------+
| age  |
+------+
|   48 |
+------+
```

Age as of 2014-08-02:

```
SELECT name, date_of_birth, TIMESTAMPDIFF(YEAR,date_of_birth,'2014-08-02') AS age
  FROM student_details;
+---------+---------------+------+
| name    | date_of_birth | age  |
+---------+---------------+------+
| Chun    | 1993-12-31    |   20 |
| Esben   | 1946-01-01    |   68 |
| Kaolin  | 1996-07-16    |   18 |
| Tatiana | 1988-04-13    |   26 |
+---------+---------------+------+
```

1.2.3.52 TIME_FORMAT

Syntax

```
TIME_FORMAT(time,format)
```

Description

This is used like the DATE_FORMAT() function, but the format string may contain format specifiers only for hours, minutes, and seconds. Other specifiers produce a NULL value or 0.

Examples

```
SELECT TIME_FORMAT('100:00:00', '%H %k %h %I %l');
+----------------------------------------+
| TIME_FORMAT('100:00:00', '%H %k %h %I %l') |
+----------------------------------------+
| 100 100 04 04 4                        |
+----------------------------------------+
```

1.2.3.53 TIME_TO_SEC

Syntax

```
TIME_TO_SEC(time)
```

Description

Returns the time argument, converted to seconds.

The value returned by `TIME_TO_SEC` is of type `DOUBLE`. Before MariaDB 5.3 (and MySQL 5.6), the type was `INT`. The returned value preserves microseconds of the argument. See also Microseconds in MariaDB.

Examples

```
SELECT TIME_TO_SEC('22:23:00');
+-------------------------+
| TIME_TO_SEC('22:23:00') |
+-------------------------+
|                   80580 |
+-------------------------+
```

```
SELECT TIME_TO_SEC('00:39:38');
+-------------------------+
| TIME_TO_SEC('00:39:38') |
+-------------------------+
|                    2378 |
+-------------------------+
```

```
SELECT TIME_TO_SEC('09:12:55.2355');
+-------------------------------+
| TIME_TO_SEC('09:12:55.2355') |
+-------------------------------+
|                   33175.2355 |
+-------------------------------+
1 row in set (0.000 sec)
```

1.2.3.54 TO_DAYS

Syntax

```
TO_DAYS(date)
```

Description

Given a date `date`, returns the number of days since the start of the current calendar (0000-00-00).

The function is not designed for use with dates before the advent of the Gregorian calendar in October 1582. Results will not be reliable since it doesn't account for the lost days when the calendar changed from the Julian calendar.

This is the converse of the FROM_DAYS() function.

Examples

```
SELECT TO_DAYS('2007-10-07');
+-----------------------+
| TO_DAYS('2007-10-07') |
+-----------------------+
|                733321 |
+-----------------------+

SELECT TO_DAYS('0000-01-01');
+-----------------------+
| TO_DAYS('0000-01-01') |
+-----------------------+
|                     1 |
+-----------------------+

SELECT TO_DAYS(950501);
+-----------------+
| TO_DAYS(950501) |
+-----------------+
|          728779 |
+-----------------+
```

1.2.3.55 TO_SECONDS

Syntax

```
TO_SECONDS(expr)
```

Description

Returns the number of seconds from year 0 till `expr`, or NULL if `expr` is not a valid date or datetime.

Examples

```
SELECT TO_SECONDS('2013-06-13');
+--------------------------+
| TO_SECONDS('2013-06-13') |
+--------------------------+
|              63538300800 |
+--------------------------+

SELECT TO_SECONDS('2013-06-13 21:45:13');
+-----------------------------------+
| TO_SECONDS('2013-06-13 21:45:13') |
+-----------------------------------+
|                       63538379113 |
+-----------------------------------+

SELECT TO_SECONDS(NOW());
+-------------------+
| TO_SECONDS(NOW()) |
+-------------------+
|       63543530875 |
+-------------------+

SELECT TO_SECONDS(20130513);
+----------------------+
| TO_SECONDS(20130513) |
+----------------------+
|          63535622400 |
+----------------------+
1 row in set (0.00 sec)

SELECT TO_SECONDS(130513);
+--------------------+
| TO_SECONDS(130513) |
+--------------------+
|        63535622400 |
+--------------------+
```

1.2.3.56 UNIX_TIMESTAMP

Contents

Syntax

```
UNIX_TIMESTAMP()
UNIX_TIMESTAMP(date)
```

Description

If called with no argument, returns a Unix timestamp (seconds since '1970-01-01 00:00:00' UTC) as an unsigned integer. If `UNIX_TIMESTAMP()` is called with a date argument, it returns the value of the argument as seconds since '1970-01-01 00:00:00' UTC. date may be a `DATE` string, a `DATETIME` string, a `TIMESTAMP`, or a number in the format YYMMDD or YYYYMMDD. The server interprets date as a value in the current time zone and converts it to an internal value in UTC. Clients can set their time zone as described in time zones.

The inverse function of `UNIX_TIMESTAMP()` is `FROM_UNIXTIME()`

`UNIX_TIMESTAMP()` supports microseconds.

Timestamps in MariaDB have a maximum value of 2147483647, equivalent to 2038-01-19 05:14:07. This is due to the underlying 32-bit limitation. Using the function on a date beyond this will result in NULL being returned. Use DATETIME as a storage type if you require dates beyond this.

Error Handling

Returns NULL for wrong arguments to `UNIX_TIMESTAMP()`. In MySQL and MariaDB before 5.3 wrong arguments to `UNIX_TIMESTAMP()` returned 0.

Compatibility

As you can see in the examples above, UNIX_TIMESTAMP(constant-date-string) returns a timestamp with 6 decimals while MariaDB 5.2 and before returns it without decimals. This can cause a problem if you are using UNIX_TIMESTAMP() as a partitioning function. You can fix this by using FLOOR(UNIX_TIMESTAMP(..)) or changing the date string to a date number, like 20080101000000.

Examples

```
SELECT UNIX_TIMESTAMP();
+------------------+
| UNIX_TIMESTAMP() |
+------------------+
|       1269711082 |
+------------------+

SELECT UNIX_TIMESTAMP('2007-11-30 10:30:19');
+---------------------------------------+
| UNIX_TIMESTAMP('2007-11-30 10:30:19') |
+---------------------------------------+
|                   1196436619.000000 |
+---------------------------------------+

SELECT UNIX_TIMESTAMP("2007-11-30 10:30:19.123456");
+-----------------------------------------------+
| unix_timestamp("2007-11-30 10:30:19.123456") |
+-----------------------------------------------+
|                          1196411419.123456 |
+-----------------------------------------------+

SELECT FROM_UNIXTIME(UNIX_TIMESTAMP('2007-11-30 10:30:19'));
+-------------------------------------------------------+
| FROM_UNIXTIME(UNIX_TIMESTAMP('2007-11-30 10:30:19')) |
+-------------------------------------------------------+
| 2007-11-30 10:30:19.000000                            |
+-------------------------------------------------------+

SELECT FROM_UNIXTIME(FLOOR(UNIX_TIMESTAMP('2007-11-30 10:30:19')));
+-------------------------------------------------------------+
| FROM_UNIXTIME(FLOOR(UNIX_TIMESTAMP('2007-11-30 10:30:19'))) |
+-------------------------------------------------------------+
| 2007-11-30 10:30:19                                         |
+-------------------------------------------------------------+
```

1.2.3.57 UTC_DATE

Syntax

```
UTC_DATE, UTC_DATE()
```

Description

Returns the current UTC date as a value in 'YYYY-MM-DD' or YYYYMMDD format, depending on whether the function is used in a string or numeric context.

Examples

```
SELECT UTC_DATE(), UTC_DATE() + 0;
+------------+----------------+
| UTC_DATE() | UTC_DATE() + 0 |
+------------+----------------+
| 2010-03-27 |       20100327 |
+------------+----------------+
```

1.2.3.58 UTC_TIME

Syntax

```
UTC_TIME
UTC_TIME([precision])
```

Description

Returns the current UTC time as a value in 'HH:MM:SS' or HHMMSS.uuuuuu format, depending on whether the function is used in a string or numeric context.

The optional *precision* determines the microsecond precision. See Microseconds in MariaDB.

Examples

```
SELECT UTC_TIME(), UTC_TIME() + 0;
+------------+----------------+
| UTC_TIME() | UTC_TIME() + 0 |
+------------+----------------+
| 17:32:34   |  173234.000000 |
+------------+----------------+
```

With precision:

```
SELECT UTC_TIME(5);
+----------------+
| UTC_TIME(5)    |
+----------------+
| 07:52:50.78369 |
+----------------+
```

1.2.3.59 UTC_TIMESTAMP

Syntax

```
UTC_TIMESTAMP
UTC_TIMESTAMP([precision])
```

Description

Returns the current UTC date and time as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS.uuuuuu format, depending on whether the function is used in a string or numeric context.

The optional *precision* determines the microsecond precision. See Microseconds in MariaDB.

Examples

```
SELECT UTC_TIMESTAMP(), UTC_TIMESTAMP() + 0;
+---------------------+----------------------+
| UTC_TIMESTAMP()     | UTC_TIMESTAMP() + 0  |
+---------------------+----------------------+
| 2010-03-27 17:33:16 | 20100327173316.000000 |
+---------------------+----------------------+
```

With precision:

```
SELECT UTC_TIMESTAMP(4);
+--------------------------+
| UTC_TIMESTAMP(4)         |
+--------------------------+
| 2018-07-10 07:51:09.1019 |
+--------------------------+
```

1.2.3.60 WEEK

Syntax

```
WEEK(date[,mode])
```

Description

This function returns the week number for `date`. The two-argument form of `WEEK()` allows you to specify whether the week starts on Sunday or Monday and whether the return value should be in the range from 0 to 53 or from 1 to 53. If the `mode` argument is omitted, the value of the default_week_format system variable is used.

Modes

Mode	1st day of week	Range	Week 1 is the 1st week with
0	Sunday	0-53	a Sunday in this year
1	Monday	0-53	more than 3 days this year
2	Sunday	1-53	a Sunday in this year
3	Monday	1-53	more than 3 days this year
4	Sunday	0-53	more than 3 days this year
5	Monday	0-53	a Monday in this year
6	Sunday	1-53	more than 3 days this year
7	Monday	1-53	a Monday in this year

With the mode value of 3, which means 'more than 3 days this year', weeks are numbered according to ISO 8601:1988.

Examples

```
SELECT WEEK('2008-02-20');
+--------------------+
| WEEK('2008-02-20') |
+--------------------+
|                  7 |
+--------------------+

SELECT WEEK('2008-02-20',0);
+----------------------+
| WEEK('2008-02-20',0) |
+----------------------+
|                    7 |
+----------------------+

SELECT WEEK('2008-02-20',1);
+----------------------+
| WEEK('2008-02-20',1) |
+----------------------+
|                    8 |
+----------------------+

SELECT WEEK('2008-12-31',0);
+----------------------+
| WEEK('2008-12-31',0) |
+----------------------+
|                   52 |
+----------------------+

SELECT WEEK('2008-12-31',1);
+----------------------+
| WEEK('2008-12-31',1) |
+----------------------+
|                   53 |
+----------------------+

 SELECT WEEK('2019-12-30',3);
+----------------------+
| WEEK('2019-12-30',3) |
+----------------------+
|                    1 |
+----------------------+
```

```
CREATE TABLE t1 (d DATETIME);
INSERT INTO t1 VALUES
    ("2007-01-30 21:31:07"),
    ("1983-10-15 06:42:51"),
    ("2011-04-21 12:34:56"),
    ("2011-10-30 06:31:41"),
    ("2011-01-30 14:03:25"),
    ("2004-10-07 11:19:34");
```

```
SELECT d, WEEK(d,0), WEEK(d,1) from t1;
+---------------------+-----------+-----------+
| d                   | WEEK(d,0) | WEEK(d,1) |
+---------------------+-----------+-----------+
| 2007-01-30 21:31:07 |         4 |         5 |
| 1983-10-15 06:42:51 |        41 |        41 |
| 2011-04-21 12:34:56 |        16 |        16 |
| 2011-10-30 06:31:41 |        44 |        43 |
| 2011-01-30 14:03:25 |         5 |         4 |
| 2004-10-07 11:19:34 |        40 |        41 |
+---------------------+-----------+-----------+
```

1.2.3.61 WEEKDAY

Syntax

```
WEEKDAY(date)
```

Description

Returns the weekday index for `date` (`0` = Monday, `1` = Tuesday, ... `6` = Sunday).

This contrasts with `DAYOFWEEK()` which follows the ODBC standard (`1` = Sunday, `2` = Monday, ..., `7` = Saturday).

Examples

```
SELECT WEEKDAY('2008-02-03 22:23:00');
+--------------------------------+
| WEEKDAY('2008-02-03 22:23:00') |
+--------------------------------+
|                              6 |
+--------------------------------+

SELECT WEEKDAY('2007-11-06');
+-----------------------+
| WEEKDAY('2007-11-06') |
+-----------------------+
|                     1 |
+-----------------------+
```

```
CREATE TABLE t1 (d DATETIME);
INSERT INTO t1 VALUES
    ("2007-01-30 21:31:07"),
    ("1983-10-15 06:42:51"),
    ("2011-04-21 12:34:56"),
    ("2011-10-30 06:31:41"),
    ("2011-01-30 14:03:25"),
    ("2004-10-07 11:19:34");
```

```
SELECT d FROM t1 where WEEKDAY(d) = 6;
+---------------------+
| d                   |
+---------------------+
| 2011-10-30 06:31:41 |
| 2011-01-30 14:03:25 |
+---------------------+
```

1.2.3.62 WEEKOFYEAR

Syntax

```
WEEKOFYEAR(date)
```

Description

Returns the calendar week of the date as a number in the range from 1 to 53. `WEEKOFYEAR()` is a compatibility function that is equivalent to `WEEK(date,3)` .

Examples

```
SELECT WEEKOFYEAR('2008-02-20');
+--------------------------+
| WEEKOFYEAR('2008-02-20') |
+--------------------------+
|                        8 |
+--------------------------+
```

```
CREATE TABLE t1 (d DATETIME);
INSERT INTO t1 VALUES
    ("2007-01-30 21:31:07"),
    ("1983-10-15 06:42:51"),
    ("2011-04-21 12:34:56"),
    ("2011-10-30 06:31:41"),
    ("2011-01-30 14:03:25"),
    ("2004-10-07 11:19:34");
```

```
 select * from t1;
+---------------------+
| d                   |
+---------------------+
| 2007-01-30 21:31:07 |
| 1983-10-15 06:42:51 |
| 2011-04-21 12:34:56 |
| 2011-10-30 06:31:41 |
| 2011-01-30 14:03:25 |
| 2004-10-07 11:19:34 |
+---------------------+
```

```
SELECT d, WEEKOFYEAR(d), WEEK(d,3) from t1;
+---------------------+---------------+-----------+
| d                   | WEEKOFYEAR(d) | WEEK(d,3) |
+---------------------+---------------+-----------+
| 2007-01-30 21:31:07 |             5 |         5 |
| 1983-10-15 06:42:51 |            41 |        41 |
| 2011-04-21 12:34:56 |            16 |        16 |
| 2011-10-30 06:31:41 |            43 |        43 |
| 2011-01-30 14:03:25 |             4 |         4 |
| 2004-10-07 11:19:34 |            41 |        41 |
+---------------------+---------------+-----------+
```

1.2.3.63 YEAR

Syntax

```
YEAR(date)
```

Description

Returns the year for the given date, in the range 1000 to 9999, or 0 for the "zero" date.

Examples

```
CREATE TABLE t1 (d DATETIME);
INSERT INTO t1 VALUES
    ("2007-01-30 21:31:07"),
    ("1983-10-15 06:42:51"),
    ("2011-04-21 12:34:56"),
    ("2011-10-30 06:31:41"),
    ("2011-01-30 14:03:25"),
    ("2004-10-07 11:19:34");
```

```
SELECT * FROM t1;
+---------------------+
| d                   |
+---------------------+
| 2007-01-30 21:31:07 |
| 1983-10-15 06:42:51 |
| 2011-04-21 12:34:56 |
| 2011-10-30 06:31:41 |
| 2011-01-30 14:03:25 |
| 2004-10-07 11:19:34 |
+---------------------+

SELECT * FROM t1 WHERE YEAR(d) = 2011;
+---------------------+
| d                   |
+---------------------+
| 2011-04-21 12:34:56 |
| 2011-10-30 06:31:41 |
| 2011-01-30 14:03:25 |
+---------------------+
```

```
SELECT YEAR('1987-01-01');
+--------------------+
| YEAR('1987-01-01') |
+--------------------+
|               1987 |
+--------------------+
```

1.2.3.64 YEARWEEK

Syntax

```
YEARWEEK(date), YEARWEEK(date,mode)
```

Description

Returns year and week for a date. The mode argument works exactly like the mode argument to WEEK(). The year in the result may be different from the year in the date argument for the first and the last week of the year.

Examples

```
SELECT YEARWEEK('1987-01-01');
+------------------------+
| YEARWEEK('1987-01-01') |
+------------------------+
|                 198652 |
+------------------------+
```

```
CREATE TABLE t1 (d DATETIME);
INSERT INTO t1 VALUES
    ("2007-01-30 21:31:07"),
    ("1983-10-15 06:42:51"),
    ("2011-04-21 12:34:56"),
    ("2011-10-30 06:31:41"),
    ("2011-01-30 14:03:25"),
    ("2004-10-07 11:19:34");
```

```
SELECT * FROM t1;
+---------------------+
| d                   |
+---------------------+
| 2007-01-30 21:31:07 |
| 1983-10-15 06:42:51 |
| 2011-04-21 12:34:56 |
| 2011-10-30 06:31:41 |
| 2011-01-30 14:03:25 |
| 2004-10-07 11:19:34 |
+---------------------+
6 rows in set (0.02 sec)
```

```
SELECT YEARWEEK(d) FROM t1 WHERE YEAR(d) = 2011;
+-------------+
| YEARWEEK(d) |
+-------------+
|      201116 |
|      201144 |
|      201105 |
+-------------+
3 rows in set (0.03 sec)
```

1.2.4 Aggregate Functions

The following functions (also called aggregate functions) can be used with the GROUP BY clause:

Stored Aggregate Functions

Custom aggregate functions.

AVG

Returns the average value.

BIT_AND

Bitwise AND.

BIT_OR

Bitwise OR.

BIT_XOR

Bitwise XOR.

COUNT

Returns count of non-null values.

COUNT DISTINCT

Returns count of number of different non-NULL values.

GROUP_CONCAT

Returns string with concatenated values from a group.

JSON_ARRAYAGG

Returns a JSON array containing an element for each value in a given set of JSON or SQL values.

JSON_OBJECTAGG

Returns a JSON object containing key-value pairs.

MAX

Returns the maximum value.

MIN

Returns the minimum value.

STD
Population standard deviation.

STDDEV
Population standard deviation.

STDDEV_POP
Returns the population standard deviation.

STDDEV_SAMP
Standard deviation.

SUM
Sum total.

VARIANCE
Population standard variance.

VAR_POP
Population standard variance.

VAR_SAMP
Returns the sample variance.

1.2.4.1 Stored Aggregate Functions

MariaDB starting with 10.3.3
The ability to create stored aggregate functions was added in MariaDB 10.3.3.

Contents

Aggregate functions are functions that are computed over a sequence of rows and return one result for the sequence of rows.

Creating a custom aggregate function is done using the CREATE FUNCTION statement with two main differences:

- The addition of the AGGREGATE keyword, so `CREATE AGGREGATE FUNCTION`
- The `FETCH GROUP NEXT ROW` instruction inside the loop
- Oracle PL/SQL compatibility using SQL/PL is provided

Standard Syntax

```
CREATE AGGREGATE FUNCTION function_name (parameters) RETURNS return_type
BEGIN
      All types of declarations
      DECLARE CONTINUE HANDLER FOR NOT FOUND RETURN return_val;
      LOOP
           FETCH GROUP NEXT ROW; // fetches next row from table
           other instructions
      END LOOP;
END
```

Stored aggregate functions were a 2016 Google Summer of Code project by Varun Gupta.

Using SQL/PL

```
SET sql_mode=Oracle;
DELIMITER //

CREATE AGGREGATE FUNCTION function_name (parameters) RETURN return_type
   declarations
BEGIN
   LOOP
      FETCH GROUP NEXT ROW; -- fetches next row from table
      -- other instructions

   END LOOP;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      RETURN return_val;
END //

DELIMITER ;
```

Examples

First a simplified example:

```
CREATE TABLE marks(stud_id INT, grade_count INT);

INSERT INTO marks VALUES (1,6), (2,4), (3,7), (4,5), (5,8);

SELECT * FROM marks;
+---------+-------------+
| stud_id | grade_count |
+---------+-------------+
|       1 |           6 |
|       2 |           4 |
|       3 |           7 |
|       4 |           5 |
|       5 |           8 |
+---------+-------------+

DELIMITER //
CREATE AGGREGATE FUNCTION IF NOT EXISTS aggregate_count(x INT) RETURNS INT
BEGIN
 DECLARE count_students INT DEFAULT 0;
 DECLARE CONTINUE HANDLER FOR NOT FOUND
 RETURN count_students;
      LOOP
          FETCH GROUP NEXT ROW;
          IF x  THEN
            SET count_students = count_students+1;
          END IF;
      END LOOP;
END //
DELIMITER ;
```

A non-trivial example that cannot easily be rewritten using existing functions:

```
DELIMITER //
CREATE AGGREGATE FUNCTION medi_int(x INT) RETURNS DOUBLE
BEGIN
  DECLARE CONTINUE HANDLER FOR NOT FOUND
    BEGIN
      DECLARE res DOUBLE;
      DECLARE cnt INT DEFAULT (SELECT COUNT(*) FROM tt);
      DECLARE lim INT DEFAULT (cnt-1) DIV 2;
      IF cnt % 2 = 0 THEN
        SET res = (SELECT AVG(a) FROM (SELECT a FROM tt ORDER BY a LIMIT lim,2) ttt);
      ELSE
        SET res = (SELECT a FROM tt ORDER BY a LIMIT lim,1);
      END IF;
      DROP TEMPORARY TABLE tt;
      RETURN res;
    END;
  CREATE TEMPORARY TABLE tt (a INT);
  LOOP
    FETCH GROUP NEXT ROW;
    INSERT INTO tt VALUES (x);
  END LOOP;
END //
DELIMITER ;
```

SQL/PL Example

This uses the same marks table as created above.

```
SET sql_mode=Oracle;
DELIMITER //

CREATE AGGREGATE FUNCTION aggregate_count(x INT) RETURN INT AS count_students INT DEFAULT 0;
BEGIN
   LOOP
      FETCH GROUP NEXT ROW;
      IF x  THEN
        SET count_students := count_students+1;
      END IF;
   END LOOP;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      RETURN count_students;
END aggregate_count //
DELIMITER ;

SELECT aggregate_count(stud_id) FROM marks;
```

1.2.4.2 AVG

Syntax

```
AVG([DISTINCT] expr)
```

Contents

Description

Returns the average value of expr. The DISTINCT option can be used to return the average of the distinct values of expr. NULL values are ignored. It is an aggregate function, and so can be used with the GROUP BY clause.

AVG() returns NULL if there were no matching rows.

AVG() can be used as a window function.

Examples

```
CREATE TABLE sales (sales_value INT);

INSERT INTO sales VALUES(10),(20),(20),(40);

SELECT AVG(sales_value) FROM sales;
+------------------+
| AVG(sales_value) |
+------------------+
|          22.5000 |
+------------------+

SELECT AVG(DISTINCT(sales_value)) FROM sales;
+----------------------------+
| AVG(DISTINCT(sales_value)) |
+----------------------------+
|                    23.3333 |
+----------------------------+
```

Commonly, AVG() is used with a GROUP BY clause:

```
CREATE TABLE student (name CHAR(10), test CHAR(10), score TINYINT);

INSERT INTO student VALUES
  ('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),
  ('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),
  ('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),
  ('Tatiana', 'SQL', 87), ('Tatiana', 'Tuning', 83);

SELECT name, AVG(score) FROM student GROUP BY name;
+---------+------------+
| name    | AVG(score) |
+---------+------------+
| Chun    |    74.0000 |
| Esben   |    37.0000 |
| Kaolin  |    72.0000 |
| Tatiana |    85.0000 |
+---------+------------+
```

Be careful to avoid this common mistake, not grouping correctly and returning mismatched data:

```
SELECT name,test,AVG(score) FROM student;
+------+------+------------+
| name | test | MIN(score) |
+------+------+------------+
| Chun | SQL  |         31 |
+------+------+------------+
```

As a window function:

```
CREATE TABLE student_test (name CHAR(10), test CHAR(10), score TINYINT);

INSERT INTO student_test VALUES
    ('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),
    ('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),
    ('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),
    ('Tatiana', 'SQL', 87), ('Tatiana', 'Tuning', 83);

SELECT name, test, score, AVG(score) OVER (PARTITION BY test)
    AS average_by_test FROM student_test;
+---------+--------+-------+-----------------+
| name    | test   | score | average_by_test |
+---------+--------+-------+-----------------+
| Chun    | SQL    |    75 |         65.2500 |
| Chun    | Tuning |    73 |         68.7500 |
| Esben   | SQL    |    43 |         65.2500 |
| Esben   | Tuning |    31 |         68.7500 |
| Kaolin  | SQL    |    56 |         65.2500 |
| Kaolin  | Tuning |    88 |         68.7500 |
| Tatiana | SQL    |    87 |         65.2500 |
| Tatiana | Tuning |    83 |         68.7500 |
+---------+--------+-------+-----------------+
```

1.2.4.3 BIT_AND

Syntax

```
BIT_AND(expr) [over_clause]
```

Description

Returns the bitwise AND of all bits in *expr*. The calculation is performed with 64-bit (BIGINT) precision. It is an aggregate function, and so can be used with the GROUP BY clause.

If no rows match, `BIT_AND` will return a value with all bits set to 1. NULL values have no effect on the result unless all results are NULL, which is treated as no match.

`BIT_AND` can be used as a window function with the addition of the *over_clause*.

Examples

```
CREATE TABLE vals (x INT);

INSERT INTO vals VALUES(111),(110),(100);

SELECT BIT_AND(x), BIT_OR(x), BIT_XOR(x) FROM vals;
+------------+-----------+------------+
| BIT_AND(x) | BIT_OR(x) | BIT_XOR(x) |
+------------+-----------+------------+
|        100 |       111 |        101 |
+------------+-----------+------------+
```

As an aggregate function:

```
CREATE TABLE vals2 (category VARCHAR(1), x INT);

INSERT INTO vals2 VALUES
  ('a',111),('a',110),('a',100),
  ('b','000'),('b',001),('b',011);

SELECT category, BIT_AND(x), BIT_OR(x), BIT_XOR(x)
  FROM vals GROUP BY category;
+----------+------------+-----------+------------+
| category | BIT_AND(x) | BIT_OR(x) | BIT_XOR(x) |
+----------+------------+-----------+------------+
| a        |        100 |       111 |        101 |
| b        |          0 |        11 |         10 |
+----------+------------+-----------+------------+
```

No match:

```
SELECT BIT_AND(NULL);
+----------------------+
| BIT_AND(NULL)        |
+----------------------+
| 18446744073709551615 |
+----------------------+
```

1.2.4.4 BIT_OR

Syntax

```
BIT_OR(expr) [over_clause]
```

Contents

Description

Returns the bitwise OR of all bits in `expr`. The calculation is performed with 64-bit (BIGINT) precision. It is an aggregate function, and so can be used with the GROUP BY clause.

If no rows match, `BIT_OR` will return a value with all bits set to `0`. NULL values have no effect on the result unless all results are NULL, which is treated as no match.

`BIT_OR` can be used as a window function with the addition of the *over_clause*.

Examples

```
CREATE TABLE vals (x INT);

INSERT INTO vals VALUES(111),(110),(100);

SELECT BIT_AND(x), BIT_OR(x), BIT_XOR(x) FROM vals;
+------------+-----------+------------+
| BIT_AND(x) | BIT_OR(x) | BIT_XOR(x) |
+------------+-----------+------------+
|        100 |       111 |        101 |
+------------+-----------+------------+
```

As an aggregate function:

```
CREATE TABLE vals2 (category VARCHAR(1), x INT);

INSERT INTO vals2 VALUES
  ('a',111),('a',110),('a',100),
  ('b','000'),('b',001),('b',011);

SELECT category, BIT_AND(x), BIT_OR(x), BIT_XOR(x)
  FROM vals GROUP BY category;
+----------+------------+-----------+------------+
| category | BIT_AND(x) | BIT_OR(x) | BIT_XOR(x) |
+----------+------------+-----------+------------+
| a        |        100 |       111 |        101 |
| b        |          0 |        11 |         10 |
+----------+------------+-----------+------------+
```

No match:

```
SELECT BIT_OR(NULL);
+--------------+
| BIT_OR(NULL) |
+--------------+
|            0 |
+--------------+
```

1.2.4.5 BIT_XOR

Syntax

```
BIT_XOR(expr) [over_clause]
```

Contents

Description

Returns the bitwise XOR of all bits in `expr`. The calculation is performed with 64-bit (BIGINT) precision. It is an aggregate function, and so can be used with the GROUP BY clause.

If no rows match, `BIT_XOR` will return a value with all bits set to `0`. NULL values have no effect on the result unless all results are NULL, which is treated as no match.

`BIT_XOR` can be used as a window function with the addition of the *over_clause*.

Examples

```
CREATE TABLE vals (x INT);

INSERT INTO vals VALUES(111),(110),(100);

SELECT BIT_AND(x), BIT_OR(x), BIT_XOR(x) FROM vals;
+------------+-----------+------------+
| BIT_AND(x) | BIT_OR(x) | BIT_XOR(x) |
+------------+-----------+------------+
|        100 |       111 |        101 |
+------------+-----------+------------+
```

As an aggregate function:

```
CREATE TABLE vals2 (category VARCHAR(1), x INT);

INSERT INTO vals2 VALUES
  ('a',111),('a',110),('a',100),
  ('b','000'),('b',001),('b',011);

SELECT category, BIT_AND(x), BIT_OR(x), BIT_XOR(x)
  FROM vals GROUP BY category;
+----------+------------+-----------+------------+
| category | BIT_AND(x) | BIT_OR(x) | BIT_XOR(x) |
+----------+------------+-----------+------------+
| a        |        100 |       111 |        101 |
| b        |          0 |        11 |         10 |
+----------+------------+-----------+------------+
```

No match:

```
SELECT BIT_XOR(NULL);
+---------------+
| BIT_XOR(NULL) |
+---------------+
|             0 |
+---------------+
```

1.2.4.6 COUNT

Syntax

```
COUNT(expr)
```

Contents

Description

Returns a count of the number of non-NULL values of expr in the rows retrieved by a SELECT statement. The result is a BIGINT value. It is an aggregate function, and so can be used with the GROUP BY clause.

COUNT(*) counts the total number of rows in a table.

COUNT() returns 0 if there were no matching rows.

COUNT() can be used as a window function.

Examples

```
CREATE TABLE student (name CHAR(10), test CHAR(10), score TINYINT);

INSERT INTO student VALUES
  ('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),
  ('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),
  ('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),
  ('Tatiana', 'SQL', 87), ('Tatiana', 'Tuning', 83);

SELECT COUNT(*) FROM student;
+----------+
| COUNT(*) |
+----------+
|        8 |
+----------+
```

COUNT(DISTINCT) example:

```
SELECT COUNT(DISTINCT (name)) FROM student;
+------------------------+
| COUNT(DISTINCT (name)) |
+------------------------+
|                      4 |
+------------------------+
```

As a window function

```
CREATE OR REPLACE TABLE student_test (name CHAR(10), test CHAR(10), score TINYINT);

INSERT INTO student_test VALUES
    ('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),
    ('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),
    ('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),
    ('Tatiana', 'SQL', 87);

SELECT name, test, score, COUNT(score) OVER (PARTITION BY name)
    AS tests_written FROM student_test;
+---------+--------+-------+---------------+
| name    | test   | score | tests_written |
+---------+--------+-------+---------------+
| Chun    | SQL    |    75 |             2 |
| Chun    | Tuning |    73 |             2 |
| Esben   | SQL    |    43 |             2 |
| Esben   | Tuning |    31 |             2 |
| Kaolin  | SQL    |    56 |             2 |
| Kaolin  | Tuning |    88 |             2 |
| Tatiana | SQL    |    87 |             1 |
+---------+--------+-------+---------------+
```

1.2.4.7 COUNT DISTINCT

Syntax

```
COUNT(DISTINCT expr,[expr...])
```

Contents

Description

Returns a count of the number of different non-NULL values.

COUNT(DISTINCT) returns 0 if there were no matching rows.

Although, from MariaDB 10.2.0, COUNT can be used as a window function, COUNT DISTINCT cannot be.

Examples

```
CREATE TABLE student (name CHAR(10), test CHAR(10), score TINYINT);

INSERT INTO student VALUES
  ('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),
  ('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),
  ('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),
  ('Tatiana', 'SQL', 87), ('Tatiana', 'Tuning', 83);

SELECT COUNT(*) FROM student;
+----------+
| COUNT(*) |
+----------+
|        8 |
+----------+

SELECT COUNT(DISTINCT (name)) FROM student;
+------------------------+
| COUNT(DISTINCT (name)) |
+------------------------+
|                      4 |
+------------------------+
```

1.2.4.8 GROUP_CONCAT

Syntax

```
GROUP_CONCAT(expr)
```

Contents

Description

This function returns a string result with the concatenated non-NULL values from a group. It returns NULL if there are no non-NULL values.

The maximum returned length in bytes is determined by the group_concat_max_len server system variable, which defaults to 1M (>= MariaDB 10.2.4) or 1K (<= MariaDB 10.2.3).

If group_concat_max_len <= 512, the return type is VARBINARY or VARCHAR; otherwise, the return type is BLOB or TEXT. The choice between binary or non-binary types depends from the input.

The full syntax is as follows:

```
GROUP_CONCAT([DISTINCT] expr [,expr ...]
             [ORDER BY {unsigned_integer | col_name | expr}
                 [ASC | DESC] [,col_name ...]]
             [SEPARATOR str_val]
             [LIMIT {[offset,] row_count | row_count OFFSET offset}])
```

`DISTINCT` eliminates duplicate values from the output string.

ORDER BY determines the order of returned values.

`SEPARATOR` specifies a separator between the values. The default separator is a comma (`,`). It is possible to avoid using a separator by specifying an empty string.

LIMIT

MariaDB starting with 10.3.3

Until MariaDB 10.3.2, it was not possible to use the LIMIT clause with `GROUP_CONCAT` . This restriction was lifted in MariaDB 10.3.3.

Examples

```
SELECT student_name,
       GROUP_CONCAT(test_score)
       FROM student
       GROUP BY student_name;
```

Get a readable list of MariaDB users from the mysql.user table:

```
SELECT GROUP_CONCAT(DISTINCT User ORDER BY User SEPARATOR '\n')
   FROM mysql.user;
```

In the former example, `DISTINCT` is used because the same user may occur more than once. The new line (`\n`) used as a `SEPARATOR` makes the results easier to read.

Get a readable list of hosts from which each user can connect:

```
SELECT User, GROUP_CONCAT(Host ORDER BY Host SEPARATOR ', ')
   FROM mysql.user GROUP BY User ORDER BY User;
```

The former example shows the difference between the `GROUP_CONCAT` 's ORDER BY (which sorts the concatenated hosts), and the `SELECT` 's ORDER BY (which sorts the rows).

From MariaDB 10.3.3, LIMIT can be used with `GROUP_CONCAT` , so, for example, given the following table:

```
CREATE TABLE d (dd DATE, cc INT);

INSERT INTO d VALUES ('2017-01-01',1);
INSERT INTO d VALUES ('2017-01-02',2);
INSERT INTO d VALUES ('2017-01-04',3);
```

the following query:

```
SELECT SUBSTRING_INDEX(GROUP_CONCAT(CONCAT_WS(":",dd,cc) ORDER BY cc DESC),",",1) FROM d;
+-----------------------------------------------------------------------------+
| SUBSTRING_INDEX(GROUP_CONCAT(CONCAT_WS(":",dd,cc) ORDER BY cc DESC),",",1) |
+-----------------------------------------------------------------------------+
| 2017-01-04:3                                                                |
+-----------------------------------------------------------------------------+
```

can be more simply rewritten as:

```
SELECT GROUP_CONCAT(CONCAT_WS(":",dd,cc) ORDER BY cc DESC LIMIT 1) FROM d;
+-----------------------------------------------------------+
| GROUP_CONCAT(CONCAT_WS(":",dd,cc) ORDER BY cc DESC LIMIT 1) |
+-----------------------------------------------------------+
| 2017-01-04:3                                              |
+-----------------------------------------------------------+
```

1.2.4.9 JSON_ARRAYAGG

MariaDB starting with 10.5.0

JSON_ARRAYAGG was added in MariaDB 10.5.0.

Syntax

```
JSON_ARRAYAGG(column_or_expression)
```

Description

`JSON_ARRAYAGG` returns a JSON array containing an element for each value in a given set of JSON or SQL values. It acts

on a column or an expression that evaluates to a single value.

The maximum returned length in bytes is determined by the group_concat_max_len server system variable.

Returns NULL in the case of an error, or if the result contains no rows.

`JSON_ARRAYAGG` cannot currently be used as a window function.

The full syntax is as follows:

```
JSON_ARRAYAGG([DISTINCT] expr [,expr ...]
             [ORDER BY {unsigned_integer | col_name | expr}
                 [ASC | DESC] [,col_name ...]]
             [LIMIT {[offset,] row_count | row_count OFFSET offset}])
```

Examples

```
CREATE TABLE t1 (a INT, b INT);

INSERT INTO t1 VALUES (1, 1),(2, 1), (1, 1),(2, 1), (3, 2),(2, 2),(2, 2),(2, 2);

SELECT JSON_ARRAYAGG(a), JSON_ARRAYAGG(b) FROM t1;
+-------------------+-------------------+
| JSON_ARRAYAGG(a)  | JSON_ARRAYAGG(b)  |
+-------------------+-------------------+
| [1,2,1,2,3,2,2,2] | [1,1,1,1,2,2,2,2] |
+-------------------+-------------------+

SELECT JSON_ARRAYAGG(a), JSON_ARRAYAGG(b) FROM t1 GROUP BY b;
+-----------------+-----------------+
| JSON_ARRAYAGG(a) | JSON_ARRAYAGG(b) |
+-----------------+-----------------+
| [1,2,1,2]       | [1,1,1,1]       |
| [3,2,2,2]       | [2,2,2,2]       |
+-----------------+-----------------+
```

1.2.4.10 JSON_OBJECTAGG

MariaDB starting with 10.5.0

JSON_OBJECTAGG was added in MariaDB 10.5.0.

Syntax

```
JSON_OBJECTAGG(key, value)
```

Description

`JSON_OBJECTAGG` returns a JSON object containing key-value pairs. It takes two expressions that evaluate to a single value, or two column names, as arguments, the first used as a key, and the second as a value.

The maximum returned length in bytes is determined by the group_concat_max_len server system variable.

Returns NULL in the case of an error, or if the result contains no rows.

`JSON_OBJECTAGG` cannot currently be used as a window function.

Examples

```
select * from t1;
+------+-------+
| a    | b     |
+------+-------+
|    1 | Hello |
|    1 | World |
|    2 | This  |
+------+-------+

SELECT JSON_OBJECTAGG(a, b) FROM t1;
+--------------------------------------+
| JSON_OBJECTAGG(a, b)                 |
+--------------------------------------+
| {"1":"Hello", "1":"World", "2":"This"} |
+--------------------------------------+
```

1.2.4.11 MAX

Syntax

```
MAX([DISTINCT] expr)
```

Contents

Description

Returns the largest, or maximum, value of *expr*. `MAX()` can also take a string argument in which case it returns the maximum string value. The `DISTINCT` keyword can be used to find the maximum of the distinct values of *expr*, however, this produces the same result as omitting `DISTINCT`.

Note that SET and ENUM fields are currently compared by their string value rather than their relative position in the set, so MAX() may produce a different highest result than ORDER BY DESC.

It is an aggregate function, and so can be used with the GROUP BY clause.

MAX() can be used as a window function.

`MAX()` returns `NULL` if there were no matching rows.

Examples

```
CREATE TABLE student (name CHAR(10), test CHAR(10), score TINYINT);

INSERT INTO student VALUES
  ('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),
  ('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),
  ('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),
  ('Tatiana', 'SQL', 87), ('Tatiana', 'Tuning', 83);

SELECT name, MAX(score) FROM student GROUP BY name;
+---------+------------+
| name    | MAX(score) |
+---------+------------+
| Chun    |         75 |
| Esben   |         43 |
| Kaolin  |         88 |
| Tatiana |         87 |
+---------+------------+
```

MAX string:

```
SELECT MAX(name) FROM student;
+-----------+
| MAX(name) |
+-----------+
| Tatiana   |
+-----------+
```

Be careful to avoid this common mistake, not grouping correctly and returning mismatched data:

```
SELECT name,test,MAX(SCORE) FROM student;
+------+------+------------+
| name | test | MAX(SCORE) |
+------+------+------------+
| Chun | SQL  |         88 |
+------+------+------------+
```

Difference between ORDER BY DESC and MAX():

```
CREATE TABLE student2(name CHAR(10),grade ENUM('b','c','a'));

INSERT INTO student2 VALUES('Chun','b'),('Esben','c'),('Kaolin','a');

SELECT MAX(grade) FROM student2;
+------------+
| MAX(grade) |
+------------+
| c          |
+------------+

SELECT grade FROM student2 ORDER BY grade DESC LIMIT 1;
+-------+
| grade |
+-------+
| a     |
+-------+
```

As a window function:

```
CREATE OR REPLACE TABLE student_test (name CHAR(10), test CHAR(10), score TINYINT);
INSERT INTO student_test VALUES
    ('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),
    ('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),
    ('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),
    ('Tatiana', 'SQL', 87);

SELECT name, test, score, MAX(score)
  OVER (PARTITION BY name) AS highest_score FROM student_test;
+---------+--------+-------+---------------+
| name    | test   | score | highest_score |
+---------+--------+-------+---------------+
| Chun    | SQL    |    75 |            75 |
| Chun    | Tuning |    73 |            75 |
| Esben   | SQL    |    43 |            43 |
| Esben   | Tuning |    31 |            43 |
| Kaolin  | SQL    |    56 |            88 |
| Kaolin  | Tuning |    88 |            88 |
| Tatiana | SQL    |    87 |            87 |
+---------+--------+-------+---------------+
```

1.2.4.12 MIN

Syntax

```
MIN([DISTINCT] expr)
```

Contents

Description

Returns the minimum value of *expr*. MIN() may take a string argument, in which case it returns the minimum string value. The DISTINCT keyword can be used to find the minimum of the distinct values of *expr*, however, this produces the same result as omitting DISTINCT.

Note that SET and ENUM fields are currently compared by their string value rather than their relative position in the set, so MIN() may produce a different lowest result than ORDER BY ASC.

It is an aggregate function, and so can be used with the GROUP BY clause.

MIN() can be used as a window function.

MIN() returns NULL if there were no matching rows.

Examples

```
CREATE TABLE student (name CHAR(10), test CHAR(10), score TINYINT);

INSERT INTO student VALUES
  ('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),
  ('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),
  ('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),
  ('Tatiana', 'SQL', 87), ('Tatiana', 'Tuning', 83);

SELECT name, MIN(score) FROM student GROUP BY name;
+---------+------------+
| name    | MIN(score) |
+---------+------------+
| Chun    |         73 |
| Esben   |         31 |
| Kaolin  |         56 |
| Tatiana |         83 |
+---------+------------+
```

MIN() with a string:

```
SELECT MIN(name) FROM student;
+-----------+
| MIN(name) |
+-----------+
| Chun      |
+-----------+
```

Be careful to avoid this common mistake, not grouping correctly and returning mismatched data:

```
SELECT name,test,MIN(score) FROM student;
+------+------+------------+
| name | test | MIN(score) |
+------+------+------------+
| Chun | SQL  |         31 |
+------+------+------------+
```

Difference between ORDER BY ASC and MIN():

```
CREATE TABLE student2(name CHAR(10),grade ENUM('b','c','a'));

INSERT INTO student2 VALUES('Chun','b'),('Esben','c'),('Kaolin','a');

SELECT MIN(grade) FROM student2;
+------------+
| MIN(grade) |
+------------+
| a          |
+------------+

SELECT grade FROM student2 ORDER BY grade ASC LIMIT 1;
+-------+
| grade |
+-------+
| b     |
+-------+
```

As a window function:

```
CREATE OR REPLACE TABLE student_test (name CHAR(10), test CHAR(10), score TINYINT);
INSERT INTO student_test VALUES
    ('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),
    ('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),
    ('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),
    ('Tatiana', 'SQL', 87);


SELECT name, test, score, MIN(score)
  OVER (PARTITION BY name) AS lowest_score FROM student_test;
+---------+--------+-------+--------------+
| name    | test   | score | lowest_score |
+---------+--------+-------+--------------+
| Chun    | SQL    |    75 |           73 |
| Chun    | Tuning |    73 |           73 |
| Esben   | SQL    |    43 |           31 |
| Esben   | Tuning |    31 |           31 |
| Kaolin  | SQL    |    56 |           56 |
| Kaolin  | Tuning |    88 |           56 |
| Tatiana | SQL    |    87 |           87 |
+---------+--------+-------+--------------+
```

1.2.4.13 STD

Syntax

```
STD(expr)
```

Description

Returns the population standard deviation of `expr`. This is an extension to standard SQL. The standard SQL function `STDDEV_POP()` can be used instead.

It is an aggregate function, and so can be used with the GROUP BY clause.

STD() can be used as a window function.

This function returns `NULL` if there were no matching rows.

Examples

As an aggregate function:

```
CREATE OR REPLACE TABLE stats (category VARCHAR(2), x INT);

INSERT INTO stats VALUES
  ('a',1),('a',2),('a',3),
  ('b',11),('b',12),('b',20),('b',30),('b',60);

SELECT category, STDDEV_POP(x), STDDEV_SAMP(x), VAR_POP(x)
  FROM stats GROUP BY category;
+----------+--------------+---------------+------------+
| category | STDDEV_POP(x) | STDDEV_SAMP(x) | VAR_POP(x) |
+----------+--------------+---------------+------------+
| a        |       0.8165 |        1.0000 |     0.6667 |
| b        |      18.0400 |       20.1693 |   325.4400 |
+----------+--------------+---------------+------------+
```

As a window function:

```
CREATE OR REPLACE TABLE student_test (name CHAR(10), test CHAR(10), score TINYINT);

INSERT INTO student_test VALUES
    ('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),
    ('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),
    ('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),
    ('Tatiana', 'SQL', 87);

SELECT name, test, score, STDDEV_POP(score)
  OVER (PARTITION BY test) AS stddev_results FROM student_test;
+---------+--------+-------+----------------+
| name    | test   | score | stddev_results |
+---------+--------+-------+----------------+
| Chun    | SQL    |    75 |        16.9466 |
| Chun    | Tuning |    73 |        24.1247 |
| Esben   | SQL    |    43 |        16.9466 |
| Esben   | Tuning |    31 |        24.1247 |
| Kaolin  | SQL    |    56 |        16.9466 |
| Kaolin  | Tuning |    88 |        24.1247 |
| Tatiana | SQL    |    87 |        16.9466 |
+---------+--------+-------+----------------+
```

1.2.4.14 STDDEV

Syntax

```
STDDEV(expr)
```

Contents

Description

Returns the population standard deviation of `expr`. This function is provided for compatibility with Oracle. The standard SQL function `STDDEV_POP()` can be used instead.

It is an aggregate function, and so can be used with the GROUP BY clause.

STDDEV() can be used as a window function.

This function returns `NULL` if there were no matching rows.

Examples

As an aggregate function:

```
CREATE OR REPLACE TABLE stats (category VARCHAR(2), x INT);

INSERT INTO stats VALUES
  ('a',1),('a',2),('a',3),
  ('b',11),('b',12),('b',20),('b',30),('b',60);

SELECT category, STDDEV_POP(x), STDDEV_SAMP(x), VAR_POP(x)
  FROM stats GROUP BY category;
+----------+---------------+----------------+------------+
| category | STDDEV_POP(x) | STDDEV_SAMP(x) | VAR_POP(x) |
+----------+---------------+----------------+------------+
| a        |        0.8165 |         1.0000 |     0.6667 |
| b        |       18.0400 |        20.1693 |   325.4400 |
+----------+---------------+----------------+------------+
```

As a window function:

```
CREATE OR REPLACE TABLE student_test (name CHAR(10), test CHAR(10), score TINYINT);

INSERT INTO student_test VALUES
    ('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),
    ('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),
    ('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),
    ('Tatiana', 'SQL', 87);

SELECT name, test, score, STDDEV_POP(score)
  OVER (PARTITION BY test) AS stddev_results FROM student_test;
+---------+--------+-------+----------------+
| name    | test   | score | stddev_results |
+---------+--------+-------+----------------+
| Chun    | SQL    |    75 |        16.9466 |
| Chun    | Tuning |    73 |        24.1247 |
| Esben   | SQL    |    43 |        16.9466 |
| Esben   | Tuning |    31 |        24.1247 |
| Kaolin  | SQL    |    56 |        16.9466 |
| Kaolin  | Tuning |    88 |        24.1247 |
| Tatiana | SQL    |    87 |        16.9466 |
+---------+--------+-------+----------------+
```

1.2.4.15 STDDEV_POP

Syntax

```
STDDEV_POP(expr)
```

Contents

Description

Returns the population standard deviation of *expr* (the square root of `VAR_POP()`). You can also use `STD()` or `STDDEV()`, which are equivalent but not standard SQL.

It is an aggregate function, and so can be used with the GROUP BY clause.

STDDEV_POP() can be used as a window function.

STDDEV_POP() returns `NULL` if there were no matching rows.

Examples

As an aggregate function:

```
CREATE OR REPLACE TABLE stats (category VARCHAR(2), x INT);

INSERT INTO stats VALUES
  ('a',1),('a',2),('a',3),
  ('b',11),('b',12),('b',20),('b',30),('b',60);

SELECT category, STDDEV_POP(x), STDDEV_SAMP(x), VAR_POP(x)
  FROM stats GROUP BY category;
+----------+--------------+---------------+------------+
| category | STDDEV_POP(x) | STDDEV_SAMP(x) | VAR_POP(x) |
+----------+--------------+---------------+------------+
| a        |       0.8165 |        1.0000 |     0.6667 |
| b        |      18.0400 |       20.1693 |   325.4400 |
+----------+--------------+---------------+------------+
```

As a window function:

```
CREATE OR REPLACE TABLE student_test (name CHAR(10), test CHAR(10), score TINYINT);

INSERT INTO student_test VALUES
    ('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),
    ('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),
    ('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),
    ('Tatiana', 'SQL', 87);

SELECT name, test, score, STDDEV_POP(score)
  OVER (PARTITION BY test) AS stddev_results FROM student_test;
+---------+--------+-------+----------------+
| name    | test   | score | stddev_results |
+---------+--------+-------+----------------+
| Chun    | SQL    |    75 |        16.9466 |
| Chun    | Tuning |    73 |        24.1247 |
| Esben   | SQL    |    43 |        16.9466 |
| Esben   | Tuning |    31 |        24.1247 |
| Kaolin  | SQL    |    56 |        16.9466 |
| Kaolin  | Tuning |    88 |        24.1247 |
| Tatiana | SQL    |    87 |        16.9466 |
+---------+--------+-------+----------------+
```

1.2.4.16 STDDEV_SAMP

Syntax

```
STDDEV_SAMP(expr)
```

Description

Returns the sample standard deviation of `expr` (the square root of VAR_SAMP()).

It is an aggregate function, and so can be used with the GROUP BY clause.

STDDEV_SAMP() can be used as a window function.

STDDEV_SAMP() returns `NULL` if there were no matching rows.

1.2.4.17 SUM

Syntax

```
SUM([DISTINCT] expr)
```

Contents

Description

Returns the sum of `expr`. If the return set has no rows, `SUM()` returns `NULL`. The `DISTINCT` keyword can be used to sum only the distinct values of `expr`.

SUM() can be used as a window function, although not with the DISTINCT specifier.

Examples

```
CREATE TABLE sales (sales_value INT);
INSERT INTO sales VALUES(10),(20),(20),(40);

SELECT SUM(sales_value) FROM sales;
+------------------+
| SUM(sales_value) |
+------------------+
|               90 |
+------------------+

SELECT SUM(DISTINCT(sales_value)) FROM sales;
+----------------------------+
| SUM(DISTINCT(sales_value)) |
+----------------------------+
|                         70 |
+----------------------------+
```

Commonly, SUM is used with a GROUP BY clause:

```
CREATE TABLE sales (name CHAR(10), month CHAR(10), units INT);

INSERT INTO sales VALUES
  ('Chun', 'Jan', 75), ('Chun', 'Feb', 73),
  ('Esben', 'Jan', 43), ('Esben', 'Feb', 31),
  ('Kaolin', 'Jan', 56), ('Kaolin', 'Feb', 88),
  ('Tatiana', 'Jan', 87), ('Tatiana', 'Feb', 83);

SELECT name, SUM(units) FROM sales GROUP BY name;
+---------+------------+
| name    | SUM(units) |
+---------+------------+
| Chun    |        148 |
| Esben   |         74 |
| Kaolin  |        144 |
| Tatiana |        170 |
+---------+------------+
```

The GROUP BY clause is required when using an aggregate function along with regular column data, otherwise the result will be a mismatch, as in the following common type of mistake:

```
SELECT name,SUM(units) FROM sales
;+------+------------+
| name | SUM(units) |
+------+------------+
| Chun |        536 |
+------+------------+
```

As a window function:

```
CREATE OR REPLACE TABLE student_test (name CHAR(10), test CHAR(10), score TINYINT);
INSERT INTO student_test VALUES
    ('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),
    ('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),
    ('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),
    ('Tatiana', 'SQL', 87);

SELECT name, test, score, SUM(score) OVER (PARTITION BY name) AS total_score FROM student_test;
+---------+--------+-------+-------------+
| name    | test   | score | total_score |
+---------+--------+-------+-------------+
| Chun    | SQL    |    75 |         148 |
| Chun    | Tuning |    73 |         148 |
| Esben   | SQL    |    43 |          74 |
| Esben   | Tuning |    31 |          74 |
| Kaolin  | SQL    |    56 |         144 |
| Kaolin  | Tuning |    88 |         144 |
| Tatiana | SQL    |    87 |          87 |
+---------+--------+-------+-------------+
```

1.2.4.18 VARIANCE

Syntax

```
VARIANCE(expr)
```

Contents

Description

Returns the population standard variance of `expr`. This is an extension to standard SQL. The standard SQL function VAR_POP() can be used instead.

Variance is calculated by

- working out the mean for the set
- for each number, subtracting the mean and squaring the result
- calculate the average of the resulting differences

It is an aggregate function, and so can be used with the GROUP BY clause.

VARIANCE() can be used as a window function.

VARIANCE() returns `NULL` if there were no matching rows.

Examples

```
CREATE TABLE v(i tinyint);

INSERT INTO v VALUES(101),(99);

SELECT VARIANCE(i) FROM v;
+-------------+
| VARIANCE(i) |
+-------------+
|      1.0000 |
+-------------+

INSERT INTO v VALUES(120),(80);

SELECT VARIANCE(i) FROM v;
+-------------+
| VARIANCE(i) |
+-------------+
|    200.5000 |
+-------------+
```

As an aggregate function:

```
CREATE OR REPLACE TABLE stats (category VARCHAR(2), x INT);

INSERT INTO stats VALUES
  ('a',1),('a',2),('a',3),
  ('b',11),('b',12),('b',20),('b',30),('b',60);

SELECT category, STDDEV_POP(x), STDDEV_SAMP(x), VAR_POP(x)
  FROM stats GROUP BY category;
+----------+---------------+----------------+-------------+
| category | STDDEV_POP(x) | STDDEV_SAMP(x) | VAR_POP(x) |
+----------+---------------+----------------+-------------+
| a        |        0.8165 |         1.0000 |      0.6667 |
| b        |       18.0400 |        20.1693 |    325.4400 |
+----------+---------------+----------------+-------------+
```

As a window function:

```
CREATE OR REPLACE TABLE student_test (name CHAR(10), test CHAR(10), score TINYINT);

INSERT INTO student_test VALUES
    ('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),
    ('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),
    ('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),
    ('Tatiana', 'SQL', 87);

SELECT name, test, score, VAR_POP(score)
  OVER (PARTITION BY test) AS variance_results FROM student_test;
+---------+--------+-------+------------------+
| name    | test   | score | variance_results |
+---------+--------+-------+------------------+
| Chun    | SQL    |    75 |         287.1875 |
| Chun    | Tuning |    73 |         582.0000 |
| Esben   | SQL    |    43 |         287.1875 |
| Esben   | Tuning |    31 |         582.0000 |
| Kaolin  | SQL    |    56 |         287.1875 |
| Kaolin  | Tuning |    88 |         582.0000 |
| Tatiana | SQL    |    87 |         287.1875 |
+---------+--------+-------+------------------+
```

1.2.4.19 VAR_POP

Syntax

```
VAR_POP(expr)
```

Contents

Description

Returns the population standard variance of `expr`. It considers rows as the whole population, not as a sample, so it has the number of rows as the denominator. You can also use VARIANCE(), which is equivalent but is not standard SQL.

Variance is calculated by

- working out the mean for the set
- for each number, subtracting the mean and squaring the result
- calculate the average of the resulting differences

It is an aggregate function, and so can be used with the GROUP BY clause.

VAR_POP() can be used as a window function.

VAR_POP() returns `NULL` if there were no matching rows.

Examples

```
CREATE TABLE v(i tinyint);

INSERT INTO v VALUES(101),(99);

SELECT VAR_POP(i) FROM v;
+------------+
| VAR_POP(i) |
+------------+
|     1.0000 |
+------------+

INSERT INTO v VALUES(120),(80);

SELECT VAR_POP(i) FROM v;
+------------+
| VAR_POP(i) |
+------------+
|   200.5000 |
+------------+
```

As an aggregate function:

```
CREATE OR REPLACE TABLE stats (category VARCHAR(2), x INT);

INSERT INTO stats VALUES
  ('a',1),('a',2),('a',3),
  ('b',11),('b',12),('b',20),('b',30),('b',60);

SELECT category, STDDEV_POP(x), STDDEV_SAMP(x), VAR_POP(x)
  FROM stats GROUP BY category;
+----------+---------------+----------------+------------+
| category | STDDEV_POP(x) | STDDEV_SAMP(x) | VAR_POP(x) |
+----------+---------------+----------------+------------+
| a        |        0.8165 |         1.0000 |     0.6667 |
| b        |       18.0400 |        20.1693 |   325.4400 |
+----------+---------------+----------------+------------+
```

As a window function:

```
CREATE OR REPLACE TABLE student_test (name CHAR(10), test CHAR(10), score TINYINT);

INSERT INTO student_test VALUES
    ('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),
    ('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),
    ('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),
    ('Tatiana', 'SQL', 87);

SELECT name, test, score, VAR_POP(score)
  OVER (PARTITION BY test) AS variance_results FROM student_test;
+---------+--------+-------+------------------+
| name    | test   | score | variance_results |
+---------+--------+-------+------------------+
| Chun    | SQL    |    75 |         287.1875 |
| Esben   | SQL    |    43 |         287.1875 |
| Kaolin  | SQL    |    56 |         287.1875 |
| Tatiana | SQL    |    87 |         287.1875 |
| Chun    | Tuning |    73 |         582.0000 |
| Esben   | Tuning |    31 |         582.0000 |
| Kaolin  | Tuning |    88 |         582.0000 |
+---------+--------+-------+------------------+
```

1.2.4.20 VAR_SAMP

Syntax

```
VAR_SAMP(expr)
```

Contents

Description

Returns the sample variance of *expr*. That is, the denominator is the number of rows minus one.

It is an aggregate function, and so can be used with the GROUP BY clause.

VAR_SAMP() can be used as a window function.

VAR_SAMP() returns `NULL` if there were no matching rows.

Examples

As an aggregate function:

```
CREATE OR REPLACE TABLE stats (category VARCHAR(2), x INT);

INSERT INTO stats VALUES
  ('a',1),('a',2),('a',3),
  ('b',11),('b',12),('b',20),('b',30),('b',60);

SELECT category, STDDEV_POP(x), STDDEV_SAMP(x), VAR_POP(x)
  FROM stats GROUP BY category;
+----------+---------------+---------------+------------+
| category | STDDEV_POP(x) | STDDEV_SAMP(x) | VAR_POP(x) |
+----------+---------------+---------------+------------+
| a        |        0.8165 |         1.0000 |     0.6667 |
| b        |       18.0400 |        20.1693 |   325.4400 |
+----------+---------------+---------------+------------+
```

As a window function:

```
CREATE OR REPLACE TABLE student_test (name CHAR(10), test CHAR(10), score TINYINT);

INSERT INTO student_test VALUES
    ('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),
    ('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),
    ('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),
    ('Tatiana', 'SQL', 87);

SELECT name, test, score, VAR_SAMP(score)
  OVER (PARTITION BY test) AS variance_results FROM student_test;
+---------+--------+-------+------------------+
| name    | test   | score | variance_results |
+---------+--------+-------+------------------+
| Chun    | SQL    |    75 |         382.9167 |
| Chun    | Tuning |    73 |         873.0000 |
| Esben   | SQL    |    43 |         382.9167 |
| Esben   | Tuning |    31 |         873.0000 |
| Kaolin  | SQL    |    56 |         382.9167 |
| Kaolin  | Tuning |    88 |         873.0000 |
| Tatiana | SQL    |    87 |         382.9167 |
+---------+--------+-------+------------------+
```

1.2.5 Numeric Functions

Functions dealing with numerals, including ABS, CEIL, DIV, EXP, PI, SIN, etc.

Addition Operator (+)

Addition.

Subtraction Operator (-)

Subtraction and unary minus.

Division Operator (/)

Division.

Multiplication Operator (*)

Multiplication.

Modulo Operator (%)

Modulo operator. Returns the remainder of N divided by M.

DIV

Integer division.

ABS

Returns an absolute value.

ACOS

Returns an arc cosine.

ASIN

Returns the arc sine.

ATAN

Returns the arc tangent.

ATAN2

Returns the arc tangent of two variables.

CEIL

Synonym for CEILING().

CEILING

Returns the smallest integer not less than X.

CONV
Converts numbers between different number bases.

COS
Returns the cosine.

COT
Returns the cotangent.

CRC32
Computes a cyclic redundancy check (CRC) value.

CRC32C
Computes a cyclic redundancy check (CRC) value using the Castagnoli polynomial.

DEGREES
Converts from radians to degrees.

EXP
e raised to the power of the argument.

FLOOR
Largest integer value not greater than the argument.

GREATEST
Returns the largest argument.

LEAST
Returns the smallest argument.

LN
Returns natural logarithm.

LOG
Returns the natural logarithm.

LOG10
Returns the base-10 logarithm.

LOG2
Returns the base-2 logarithm.

MOD
Modulo operation. Remainder of N divided by M.

OCT
Returns octal value.

PI
Returns the value of π (pi).

POW
Returns X raised to the power of Y.

POWER
Synonym for POW().

RADIANS
Converts from degrees to radians.

RAND
Random floating-point value.

ROUND

Rounds a number.

SIGN

Returns 1, 0 or -1.

SIN

Returns the sine.

SQRT

Square root.

TAN

Returns the tangent.

TRUNCATE

The TRUNCATE function truncates a number to a specified number of decimal places.

1.1.5.1.1 Addition Operator (+)

1.1.5.1.7 Subtraction Operator (-)

1.1.5.1.3 Division Operator (/)

1.1.5.1.6 Multiplication Operator (*)

1.1.5.1.5 Modulo Operator (%)

1.2.5.6 DIV

Syntax

```
DIV
```

Description

Integer division. Similar to FLOOR(), but is safe with BIGINT values. Incorrect results may occur for non-integer operands that exceed BIGINT range.

If the `ERROR_ON_DIVISION_BY_ZERO` SQL_MODE is used, a division by zero produces an error. Otherwise, it returns NULL.

The remainder of a division can be obtained using the MOD operator.

Examples

```
SELECT 300 DIV 7;
+-----------+
| 300 DIV 7 |
+-----------+
|        42 |
+-----------+

SELECT 300 DIV 0;
+-----------+
| 300 DIV 0 |
+-----------+
|      NULL |
+-----------+
```

1.2.5.7 ABS

Syntax

```
ABS(X)
```

Contents

Description

Returns the absolute (non-negative) value of `X`. If `X` is not a number, it is converted to a numeric type.

Examples

```
SELECT ABS(42);
+---------+
| ABS(42) |
+---------+
|      42 |
+---------+

SELECT ABS(-42);
+----------+
| ABS(-42) |
+----------+
|       42 |
+----------+

SELECT ABS(DATE '1994-01-01');
+------------------------+
| ABS(DATE '1994-01-01') |
+------------------------+
|               19940101 |
+------------------------+
```

1.2.5.8 ACOS

Syntax

```
ACOS(X)
```

Contents

Description

Returns the arc cosine of `X`, that is, the value whose cosine is `X`. Returns `NULL` if `X` is not in the range `-1` to `1`.

Examples

```
SELECT ACOS(1);
+---------+
| ACOS(1) |
+---------+
|       0 |
+---------+

SELECT ACOS(1.0001);
+--------------+
| ACOS(1.0001) |
+--------------+
|         NULL |
+--------------+

SELECT ACOS(0);
+-----------------+
| ACOS(0)         |
+-----------------+
| 1.5707963267949 |
+-----------------+

SELECT ACOS(0.234);
+------------------+
| ACOS(0.234)      |
+------------------+
| 1.33460644244679 |
+------------------+
```

1.2.5.9 ASIN

Syntax

```
ASIN(X)
```

Contents

Description

Returns the arc sine of X, that is, the value whose sine is X. Returns NULL if X is not in the range -1 to 1.

Examples

```
SELECT ASIN(0.2);
+--------------------+
| ASIN(0.2)          |
+--------------------+
| 0.2013579207903308 |
+--------------------+

SELECT ASIN('foo');
+-------------+
| ASIN('foo') |
+-------------+
|           0 |
+-------------+

SHOW WARNINGS;
+---------+------+----------------------------------------+
| Level   | Code | Message                                |
+---------+------+----------------------------------------+
| Warning | 1292 | Truncated incorrect DOUBLE value: 'foo' |
+---------+------+----------------------------------------+
```

1.2.5.10 ATAN

Syntax

```
ATAN(X)
```

Contents

Description

Returns the arc tangent of X, that is, the value whose tangent is X.

Examples

```
SELECT ATAN(2);
+--------------------+
| ATAN(2)            |
+--------------------+
| 1.1071487177940904 |
+--------------------+

SELECT ATAN(-2);
+---------------------+
| ATAN(-2)            |
+---------------------+
| -1.1071487177940904 |
+---------------------+
```

1.2.5.11 ATAN2

Syntax

```
ATAN(Y,X), ATAN2(Y,X)
```

Description

Returns the arc tangent of the two variables X and Y. It is similar to calculating the arc tangent of Y / X, except that the signs of both arguments are used to determine the quadrant of the result.

Examples

```
SELECT ATAN(-2,2);
+---------------------+
| ATAN(-2,2)          |
+---------------------+
| -0.7853981633974483 |
+---------------------+

SELECT ATAN2(PI(),0);
+--------------------+
| ATAN2(PI(),0)      |
+--------------------+
| 1.5707963267948966 |
+--------------------+
```

1.2.5.12 CEIL

Syntax

```
CEIL(X)
```

Description

CEIL() is a synonym for CEILING().

1.2.5.13 CEILING

Syntax

```
CEILING(X)
```

Description

Returns the smallest integer value not less than X.

Examples

```
SELECT CEILING(1.23);
+---------------+
| CEILING(1.23) |
+---------------+
|             2 |
+---------------+

SELECT CEILING(-1.23);
+----------------+
| CEILING(-1.23) |
+----------------+
|             -1 |
+----------------+
```

1.2.5.14 CONV

Syntax

```
CONV(N,from_base,to_base)
```

Description

Converts numbers between different number bases. Returns a string representation of the number `N`, converted from base `from_base` to base `to_base`.

Returns `NULL` if any argument is `NULL`, or if the second or third argument are not in the allowed range.

The argument `N` is interpreted as an integer, but may be specified as an integer or a string. The minimum base is 2 and the maximum base is 36. If `to_base` is a negative number, `N` is regarded as a signed number. Otherwise, `N` is treated as unsigned. `CONV()` works with 64-bit precision.

Some shortcuts for this function are also available: `BIN()`, `OCT()`, `HEX()`, `UNHEX()`. Also, MariaDB allows binary literal values and hexadecimal literal values.

Examples

```
SELECT CONV('a',16,2);
+----------------+
| CONV('a',16,2) |
+----------------+
| 1010           |
+----------------+

SELECT CONV('6E',18,8);
+-----------------+
| CONV('6E',18,8) |
+-----------------+
| 172             |
+-----------------+

SELECT CONV(-17,10,-18);
+------------------+
| CONV(-17,10,-18) |
+------------------+
| -H               |
+------------------+

SELECT CONV(12+'10'+'10'+0xa,10,10);
+-------------------------------+
| CONV(12+'10'+'10'+0xa,10,10) |
+-------------------------------+
| 42                            |
+-------------------------------+
```

1.2.5.15 COS

Syntax

```
COS(X)
```

Description

Returns the cosine of X, where X is given in radians.

Examples

```
SELECT COS(PI());
+-----------+
| COS(PI()) |
+-----------+
|        -1 |
+-----------+
```

1.2.5.16 COT

Syntax

```
COT(X)
```

Description

Returns the cotangent of X.

Examples

```
SELECT COT(42);
+--------------------+
| COT(42)            |
+--------------------+
| 0.4364167060752729 |
+--------------------+

SELECT COT(12);
+---------------------+
| COT(12)             |
+---------------------+
| -1.5726734063976893 |
+---------------------+

SELECT COT(0);
ERROR 1690 (22003): DOUBLE value is out of range in 'cot(0)'
```

1.2.5.17 CRC32

Syntax

<= MariaDB 10.7

```
CRC32(expr)
```

From MariaDB 10.8

```
CRC32([par,]expr)
```

Description

Computes a cyclic redundancy check (CRC) value and returns a 32-bit unsigned value. The result is NULL if the argument is NULL. The argument is expected to be a string and (if possible) is treated as one if it is not.

Uses the ISO 3309 polynomial that used by zlib and many others. MariaDB 10.8 introduced the CRC32C() function, which uses the alternate Castagnoli polynomia.

MariaDB starting with 10.8

Often, CRC is computed in pieces. To facilitate this, MariaDB 10.8.0 introduced an optional parameter:

CRC32('MariaDB')=CRC32(CRC32('Maria'),'DB').

Examples

```
SELECT CRC32('MariaDB');
+------------------+
| CRC32('MariaDB') |
+------------------+
|       4227209140 |
+------------------+

SELECT CRC32('mariadb');
+------------------+
| CRC32('mariadb') |
+------------------+
|       2594253378 |
+------------------+
```

From MariaDB 10.8.0

```
SELECT CRC32(CRC32('Maria'),'DB');
+----------------------------+
| CRC32(CRC32('Maria'),'DB') |
+----------------------------+
|                 4227209140 |
+----------------------------+
```

1.2.5.18 CRC32C

MariaDB starting with 10.8

Introduced in MariaDB 10.8.0 to compute a cyclic redundancy check (CRC) value using the Castagnoli polynomial.

Syntax

```
CRC32C([par,]expr)
```

Description

MariaDB has always included a native unary function CRC32() that computes the CRC-32 of a string using the ISO 3309 polynomial that used by zlib and many others.

InnoDB and MyRocks use a different polynomial, which was implemented in SSE4.2 instructions that were introduced in the Intel Nehalem microarchitecture. This is commonly called CRC-32C (Castagnoli).

The CRC32C function uses the Castagnoli polynomial.

This allows `SELECT…INTO DUMPFILE` to be used for the creation of files with valid checksums, such as a logically empty InnoDB redo log file `ib_logfile0` corresponding to a particular log sequence number.

The optional parameter allows the checksum to be computed in pieces:
CRC32C('MariaDB')=CRC32C(CRC32C('Maria'),'DB').

Examples

```
SELECT CRC32C('MariaDB');
+-------------------+
| CRC32C('MariaDB') |
+-------------------+
|         809606978 |
+-------------------+

SELECT CRC32C(CRC32C('Maria'),'DB');
+------------------------------+
| CRC32C(CRC32C('Maria'),'DB') |
+------------------------------+
|                    809606978 |
+------------------------------+
```

1.2.5.19 DEGREES

Syntax

```
DEGREES(X)
```

Description

Returns the argument *X*, converted from radians to degrees.

This is the converse of the RADIANS() function.

Examples

```
SELECT DEGREES(PI());
+---------------+
| DEGREES(PI()) |
+---------------+
|           180 |
+---------------+

SELECT DEGREES(PI() / 2);
+-------------------+
| DEGREES(PI() / 2) |
+-------------------+
|                90 |
+-------------------+

SELECT DEGREES(45);
+-----------------+
| DEGREES(45)     |
+-----------------+
| 2578.3100780887 |
+-----------------+
```

1.2.5.20 EXP

Syntax

```
EXP(X)
```

Description

Returns the value of e (the base of natural logarithms) raised to the power of X. The inverse of this function is LOG() (using a single argument only) or LN().

If `X` is `NULL`, this function returns `NULL`.

Examples

```
SELECT EXP(2);
+------------------+
| EXP(2)           |
+------------------+
| 7.38905609893065 |
+------------------+

SELECT EXP(-2);
+--------------------+
| EXP(-2)            |
+--------------------+
| 0.1353352832366127 |
+--------------------+

SELECT EXP(0);
+--------+
| EXP(0) |
+--------+
|      1 |
+--------+

SELECT EXP(NULL);
+-----------+
| EXP(NULL) |
+-----------+
|      NULL |
+-----------+
```

1.2.5.21 FLOOR

Syntax

```
FLOOR(X)
```

Description

Returns the largest integer value not greater than X.

Examples

```
SELECT FLOOR(1.23);
+-------------+
| FLOOR(1.23) |
+-------------+
|           1 |
+-------------+

SELECT FLOOR(-1.23);
+--------------+
| FLOOR(-1.23) |
+--------------+
|           -2 |
+--------------+
```

1.1.5.4.10 GREATEST

1.1.5.4.18 LEAST

1.2.5.24 LN

Syntax

```
LN(X)
```

Description

Returns the natural logarithm of X; that is, the base-e logarithm of X. If X is less than or equal to 0, or `NULL`, then NULL is returned.

The inverse of this function is `EXP()`.

Examples

```
SELECT LN(2);
+-------------------+
| LN(2)             |
+-------------------+
| 0.693147180559945 |
+-------------------+

SELECT LN(-2);
+--------+
| LN(-2) |
+--------+
|   NULL |
+--------+
```

1.2.5.25 LOG

Syntax

```
LOG(X), LOG(B,X)
```

Description

If called with one parameter, this function returns the natural logarithm of X. If X is less than or equal to 0, then NULL is returned.

If called with two parameters, it returns the logarithm of X to the base B. If B is <= 1 or X <= 0, the function returns NULL.

If any argument is `NULL`, the function returns `NULL`.

The inverse of this function (when called with a single argument) is the EXP() function.

Examples

LOG(X):

```
SELECT LOG(2);
+-------------------+
| LOG(2)            |
+-------------------+
| 0.693147180559945 |
+-------------------+

SELECT LOG(-2);
+---------+
| LOG(-2) |
+---------+
|    NULL |
+---------+
```

LOG(B,X)

```
SELECT LOG(2,16);
+-----------+
| LOG(2,16) |
+-----------+
|         4 |
+-----------+

SELECT LOG(3,27);
+-----------+
| LOG(3,27) |
+-----------+
|         3 |
+-----------+

SELECT LOG(3,1);
+----------+
| LOG(3,1) |
+----------+
|        0 |
+----------+

SELECT LOG(3,0);
+----------+
| LOG(3,0) |
+----------+
|     NULL |
+----------+
```

1.2.5.26 LOG10

Syntax

```
LOG10(X)
```

Description

Returns the base-10 logarithm of X.

Examples

```
SELECT LOG10(2);
+-------------------+
| LOG10(2)          |
+-------------------+
| 0.301029995663981 |
+-------------------+

SELECT LOG10(100);
+------------+
| LOG10(100) |
+------------+
|          2 |
+------------+

SELECT LOG10(-100);
+-------------+
| LOG10(-100) |
+-------------+
|        NULL |
+-------------+
```

1.2.5.27 LOG2

Syntax

```
LOG2(X)
```

Description

Returns the base-2 logarithm of X.

Examples

```
SELECT LOG2(4398046511104);
+----------------------+
| LOG2(4398046511104) |
+----------------------+
|                   42 |
+----------------------+

SELECT LOG2(65536);
+-------------+
| LOG2(65536) |
+-------------+
|          16 |
+-------------+

SELECT LOG2(-100);
+------------+
| LOG2(-100) |
+------------+
|       NULL |
+------------+
```

1.2.5.28 MOD

Syntax

```
MOD(N,M), N % M, N MOD M
```

Description

Modulo operation. Returns the remainder of N divided by M. See also Modulo Operator.

If the `ERROR_ON_DIVISION_BY_ZERO` SQL_MODE is used, any number modulus zero produces an error. Otherwise, it returns NULL.

The integer part of a division can be obtained using DIV.

Examples

```
SELECT 1042 % 50;
+-----------+
| 1042 % 50 |
+-----------+
|        42 |
+-----------+

SELECT MOD(234, 10);
+--------------+
| MOD(234, 10) |
+--------------+
|            4 |
+--------------+

SELECT 253 % 7;
+---------+
| 253 % 7 |
+---------+
|       1 |
+---------+

SELECT MOD(29,9);
+-----------+
| MOD(29,9) |
+-----------+
|         2 |
+-----------+

SELECT 29 MOD 9;
+----------+
| 29 MOD 9 |
+----------+
|        2 |
+----------+
```

1.2.5.29 OCT

Syntax

```
OCT(N)
```

Contents

Description

Returns a string representation of the octal value of N, where N is a longlong (BIGINT) number. This is equivalent to CONV(N,10,8). Returns NULL if N is NULL.

Examples

```
SELECT OCT(34);
+---------+
| OCT(34) |
+---------+
| 42      |
+---------+

SELECT OCT(12);
+---------+
| OCT(12) |
+---------+
| 14      |
+---------+
```

1.2.5.30 PI

Syntax

```
PI()
```

Description

Returns the value of π (pi). The default number of decimal places displayed is six, but MariaDB uses the full double-precision value internally.

Examples

```
SELECT PI();
+----------+
| PI()     |
+----------+
| 3.141593 |
+----------+

SELECT PI()+0.00000000000000000000;
+---------------------------------+
| PI()+0.00000000000000000000 |
+---------------------------------+
|      3.141592653589793115980 |
+---------------------------------+
```

1.2.5.31 POW

Syntax

```
POW(X,Y)
```

Description

Returns the value of X raised to the power of Y.

POWER() is a synonym.

Examples

```
SELECT POW(2,3);
+----------+
| POW(2,3) |
+----------+
|        8 |
+----------+

SELECT POW(2,-2);
+-----------+
| POW(2,-2) |
+-----------+
|      0.25 |
+-----------+
```

1.2.5.32 POWER

Syntax

```
POWER(X,Y)
```

Description

This is a synonym for POW(), which returns the value of X raised to the power of Y.

1.2.5.33 RADIANS

Syntax

```
RADIANS(X)
```

Description

Returns the argument X, converted from degrees to radians. Note that π radians equals 180 degrees.

This is the converse of the DEGREES() function.

Examples

```
SELECT RADIANS(45);
+-------------------+
| RADIANS(45)       |
+-------------------+
| 0.785398163397448 |
+-------------------+

SELECT RADIANS(90);
+------------------+
| RADIANS(90)      |
+------------------+
| 1.5707963267949 |
+------------------+

SELECT RADIANS(PI());
+--------------------+
| RADIANS(PI())      |
+--------------------+
| 0.0548311355616075 |
+--------------------+

SELECT RADIANS(180);
+-------------------+
| RADIANS(180)      |
+-------------------+
| 3.14159265358979 |
+-------------------+
```

1.2.5.34 RAND

Contents

Syntax

```
RAND(), RAND(N)
```

Description

Returns a random `DOUBLE` precision floating point value v in the range 0 <= v < 1.0. If a constant integer argument N is specified, it is used as the seed value, which produces a repeatable sequence of column values. In the example below, note that the sequences of values produced by RAND(3) is the same both places where it occurs.

In a WHERE clause, RAND() is evaluated each time the WHERE is executed.

Statements using the RAND() function are not safe for statement-based replication.

Practical uses

The expression to get a random integer from a given range is the following:

```
FLOOR(min_value + RAND() * (max_value - min_value +1))
```

RAND() is often used to read random rows from a table, as follows:

```
SELECT * FROM my_table ORDER BY RAND() LIMIT 10;
```

Note, however, that this technique should never be used on a large table as it will be extremely slow. MariaDB will read all rows in the table, generate a random value for each of them, order them, and finally will apply the LIMIT clause.

Examples

```
CREATE TABLE t (i INT);

INSERT INTO t VALUES(1),(2),(3);

SELECT i, RAND() FROM t;
+------+------------------+
| i    | RAND()           |
+------+------------------+
|    1 | 0.255651095188829 |
|    2 | 0.833920199269355 |
|    3 |  0.40264774151393 |
+------+------------------+

SELECT i, RAND(3) FROM t;
+------+------------------+
| i    | RAND(3)          |
+------+------------------+
|    1 |  0.90576975597606 |
|    2 | 0.373079058130345 |
|    3 | 0.148086053457191 |
+------+------------------+

SELECT i, RAND() FROM t;
+------+------------------+
| i    | RAND()           |
+------+------------------+
|    1 | 0.511478140495232 |
|    2 | 0.349447508668012 |
|    3 | 0.212803152588013 |
+------+------------------+
```

Using the same seed, the same sequence will be returned:

```
SELECT i, RAND(3) FROM t;
+------+------------------+
| i    | RAND(3)          |
+------+------------------+
|    1 |  0.90576975597606 |
|    2 | 0.373079058130345 |
|    3 | 0.148086053457191 |
+------+------------------+
```

Generating a random number from 5 to 15:

```
SELECT FLOOR(5 + (RAND() * 11));
```

1.2.5.35 ROUND

Syntax

```
ROUND(X), ROUND(X,D)
```

Description

Rounds the argument `X` to `D` decimal places. `D` defaults to `0` if not specified. `D` can be negative to cause `D` digits left of the decimal point of the value `X` to become zero.

The rounding algorithm depends on the data type of `X`:

- for floating point types (FLOAT, DOUBLE) the C libraries rounding function is used, so the behavior *may* differ between operating systems
- for fixed point types (DECIMAL, DEC/NUMBER/FIXED) the "round half up" rule is used, meaning that e.g. a value ending in exactly .5 is always rounded up.

Examples

```
SELECT ROUND(-1.23);
+--------------+
| ROUND(-1.23) |
+--------------+
|           -1 |
+--------------+

SELECT ROUND(-1.58);
+--------------+
| ROUND(-1.58) |
+--------------+
|           -2 |
+--------------+

SELECT ROUND(1.58);
+-------------+
| ROUND(1.58) |
+-------------+
|           2 |
+-------------+

SELECT ROUND(1.298, 1);
+-----------------+
| ROUND(1.298, 1) |
+-----------------+
|             1.3 |
+-----------------+

SELECT ROUND(1.298, 0);
+-----------------+
| ROUND(1.298, 0) |
+-----------------+
|               1 |
+-----------------+

SELECT ROUND(23.298, -1);
+-------------------+
| ROUND(23.298, -1) |
+-------------------+
|                20 |
+-------------------+
```

1.2.5.36 SIGN

Syntax

```
SIGN(X)
```

Description

Returns the sign of the argument as -1, 0, or 1, depending on whether X is negative, zero, or positive.

Examples

```
SELECT SIGN(-32);
+-----------+
| SIGN(-32) |
+-----------+
|        -1 |
+-----------+

SELECT SIGN(0);
+---------+
| SIGN(0) |
+---------+
|       0 |
+---------+

SELECT SIGN(234);
+-----------+
| SIGN(234) |
+-----------+
|         1 |
+-----------+
```

1.2.5.37 SIN

Syntax

```
SIN(X)
```

Description

Returns the sine of X, where X is given in radians.

Examples

```
SELECT SIN(1.5707963267948966);
+--------------------------+
| SIN(1.5707963267948966) |
+--------------------------+
|                        1 |
+--------------------------+

SELECT SIN(PI());
+----------------------+
| SIN(PI())            |
+----------------------+
| 1.22460635382238e-16 |
+----------------------+

SELECT ROUND(SIN(PI()));
+------------------+
| ROUND(SIN(PI())) |
+------------------+
|                0 |
+------------------+
```

1.2.5.38 SQRT

Syntax

```
SQRT(X)
```

Description

Returns the square root of X. If X is negative, NULL is returned.

Examples

```
SELECT SQRT(4);
+---------+
| SQRT(4) |
+---------+
|       2 |
+---------+

SELECT SQRT(20);
+------------------+
| SQRT(20)         |
+------------------+
| 4.47213595499958 |
+------------------+

SELECT SQRT(-16);
+-----------+
| SQRT(-16) |
+-----------+
|      NULL |
+-----------+

SELECT SQRT(1764);
+------------+
| SQRT(1764) |
+------------+
|         42 |
+------------+
```

1.2.5.39 TAN

Syntax

```
TAN(X)
```

Description

Returns the tangent of X, where X is given in radians.

Examples

```
SELECT TAN(0.7853981633974483);
+--------------------------+
| TAN(0.7853981633974483) |
+--------------------------+
|       0.9999999999999999 |
+--------------------------+

SELECT TAN(PI());
+-----------------------+
| TAN(PI())             |
+-----------------------+
| -1.22460635382238e-16 |
+-----------------------+

SELECT TAN(PI()+1);
+-----------------+
| TAN(PI()+1)     |
+-----------------+
| 1.5574077246549 |
+-----------------+

SELECT TAN(RADIANS(PI()));
+--------------------+
| TAN(RADIANS(PI())) |
+--------------------+
| 0.0548861508080033 |
+--------------------+
```

1.2.5.40 TRUNCATE

This page documents the TRUNCATE function. See TRUNCATE TABLE for the DDL statement.

Syntax

```
TRUNCATE(X,D)
```

Contents

Description

Returns the number X, truncated to D decimal places. If D is 0, the result has no decimal point or fractional part. D can be negative to cause D digits left of the decimal point of the value X to become zero.

Examples

```
SELECT TRUNCATE(1.223,1);
+-------------------+
| TRUNCATE(1.223,1) |
+-------------------+
|               1.2 |
+-------------------+

SELECT TRUNCATE(1.999,1);
+-------------------+
| TRUNCATE(1.999,1) |
+-------------------+
|               1.9 |
+-------------------+

SELECT TRUNCATE(1.999,0);
+-------------------+
| TRUNCATE(1.999,0) |
+-------------------+
|                 1 |
+-------------------+

SELECT TRUNCATE(-1.999,1);
+--------------------+
| TRUNCATE(-1.999,1) |
+--------------------+
|               -1.9 |
+--------------------+

SELECT TRUNCATE(122,-2);
+------------------+
| TRUNCATE(122,-2) |
+------------------+
|              100 |
+------------------+

SELECT TRUNCATE(10.28*100,0);
+------------------------+
| TRUNCATE(10.28*100,0) |
+------------------------+
|                   1028 |
+------------------------+
```

1.2.6 Control Flow Functions

Built-In functions for assessing data to determine what results to return.

CASE OPERATOR

Returns the result where value=compare_value or for the first condition that is true.

DECODE

Decrypts a string encoded with ENCODE(), or, in Oracle mode, matches expressions.

DECODE_ORACLE

Synonym for the Oracle mode version of DECODE().

IF Function

If expr1 is TRUE, returns expr2; otherwise it returns expr3.

IFNULL

Check whether an expression is NULL.

NULLIF

Returns NULL if expr1 = expr2.

NVL

Synonym for IFNULL.

NVL2

Returns a value based on whether a specified expression is NULL or not.

There are 1 related questions.

1.2.6.1 CASE OPERATOR

Syntax

```
CASE value WHEN [compare_value] THEN result [WHEN [compare_value] THEN
result ...] [ELSE result] END

CASE WHEN [condition] THEN result [WHEN [condition] THEN result ...]
[ELSE result] END
```

Contents

Description

The first version returns the result where value=compare_value. The second version returns the result for the first condition that is true. If there was no matching result value, the result after ELSE is returned, or NULL if there is no ELSE part.

There is also a CASE statement, which differs from the CASE operator described here.

Examples

```
SELECT CASE 1 WHEN 1 THEN 'one' WHEN 2 THEN 'two' ELSE 'more' END;
+-------------------------------------------------------------+
| CASE 1 WHEN 1 THEN 'one' WHEN 2 THEN 'two' ELSE 'more' END |
+-------------------------------------------------------------+
| one                                                         |
+-------------------------------------------------------------+

SELECT CASE WHEN 1>0 THEN 'true' ELSE 'false' END;
+---------------------------------------+
| CASE WHEN 1>0 THEN 'true' ELSE 'false' END |
+---------------------------------------+
| true                                  |
+---------------------------------------+


SELECT CASE BINARY 'B' WHEN 'a' THEN 1 WHEN 'b' THEN 2 END;
+-----------------------------------------------------+
| CASE BINARY 'B' WHEN 'a' THEN 1 WHEN 'b' THEN 2 END |
+-----------------------------------------------------+
|                                                NULL |
+-----------------------------------------------------+
```

1.2.6.2 DECODE

Syntax

```
DECODE(crypt_str,pass_str)
```

In Oracle mode from MariaDB 10.3.2:

```
DECODE(expr, search_expr, result_expr [, search_expr2, result_expr2 ...] [default_expr])
```

In all modes from MariaDB 10.3.2:

```
DECODE_ORACLE(expr, search_expr, result_expr [, search_expr2, result_expr2 ...] [default_expr])
```

Description

In the default mode, `DECODE` decrypts the encrypted string *crypt_str* using *pass_str* as the password. *crypt_str* should be a string returned from ENCODE(). The resulting string will be the original string only if *pass_str* is the same.

In Oracle mode from MariaDB 10.3.2, `DECODE` compares *expr* to the search expressions, in order. If it finds a match, the corresponding result expression is returned. If no matches are found, the default expression is returned, or NULL if no default is provided.

NULLs are treated as equivalent.

`DECODE_ORACLE` is a synonym for the Oracle-mode version of the function, and is available in all modes.

Examples

From MariaDB 10.3.2:

```
SELECT DECODE_ORACLE(2+1,3*1,'found1',3*2,'found2','default');
+-----------------------------------------------------------+
| DECODE_ORACLE(2+1,3*1,'found1',3*2,'found2','default') |
+-----------------------------------------------------------+
| found1                                                    |
+-----------------------------------------------------------+

SELECT DECODE_ORACLE(2+4,3*1,'found1',3*2,'found2','default');
+-----------------------------------------------------------+
| DECODE_ORACLE(2+4,3*1,'found1',3*2,'found2','default') |
+-----------------------------------------------------------+
| found2                                                    |
+-----------------------------------------------------------+

SELECT DECODE_ORACLE(2+2,3*1,'found1',3*2,'found2','default');
+-----------------------------------------------------------+
| DECODE_ORACLE(2+2,3*1,'found1',3*2,'found2','default') |
+-----------------------------------------------------------+
| default                                                   |
+-----------------------------------------------------------+
```

Nulls are treated as equivalent:

```
SELECT DECODE_ORACLE(NULL,NULL,'Nulls are equivalent','Nulls are not equivalent');
+-----------------------------------------------------------------------------+
| DECODE_ORACLE(NULL,NULL,'Nulls are equivalent','Nulls are not equivalent') |
+-----------------------------------------------------------------------------+
| Nulls are equivalent                                                        |
+-----------------------------------------------------------------------------+
```

1.2.6.3 DECODE_ORACLE

MariaDB starting with 10.3.2

`DECODE_ORACLE` is a synonym for the Oracle mode version of the DECODE function, and is available in all modes.

1.2.6.4 IF Function

Syntax

```
IF(expr1,expr2,expr3)
```

Contents

Description

If `expr1` is `TRUE` (`expr1 <> 0` and `expr1 <> NULL`) then `IF()` returns `expr2` ; otherwise it returns `expr3` . `IF()` returns a numeric or string value, depending on the context in which it is used.

Note: There is also an IF statement which differs from the `IF()` function described here.

Examples

```
SELECT IF(1>2,2,3);
+-------------+
| IF(1>2,2,3) |
+-------------+
|           3 |
+-------------+
```

```
SELECT IF(1<2,'yes','no');
+--------------------+
| IF(1<2,'yes','no') |
+--------------------+
| yes                |
+--------------------+
```

```
SELECT IF(STRCMP('test','test1'),'no','yes');
+---------------------------------------+
| IF(STRCMP('test','test1'),'no','yes') |
+---------------------------------------+
| no                                    |
+---------------------------------------+
```

1.2.6.5 IFNULL

Syntax

```
IFNULL(expr1,expr2)
NVL(expr1,expr2)
```

Contents

Description

If *`expr1`* is not NULL, IFNULL() returns *`expr1`* ; otherwise it returns *`expr2`* . IFNULL() returns a numeric or string value, depending on the context in which it is used.

From MariaDB 10.3, NVL() is an alias for IFNULL().

Examples

```
SELECT IFNULL(1,0);
+-------------+
| IFNULL(1,0) |
+-------------+
|           1 |
+-------------+

SELECT IFNULL(NULL,10);
+-----------------+
| IFNULL(NULL,10) |
+-----------------+
|              10 |
+-----------------+

SELECT IFNULL(1/0,10);
+----------------+
| IFNULL(1/0,10) |
+----------------+
|        10.0000 |
+----------------+

SELECT IFNULL(1/0,'yes');
+-------------------+
| IFNULL(1/0,'yes') |
+-------------------+
| yes               |
+-------------------+
```

1.2.6.6 NULLIF

Syntax

```
NULLIF(expr1,expr2)
```

Contents

Description

Returns NULL if expr1 = expr2 is true, otherwise returns expr1. This is the same as CASE WHEN expr1 = expr2 THEN NULL ELSE expr1 END.

Examples

```
SELECT NULLIF(1,1);
+-------------+
| NULLIF(1,1) |
+-------------+
|        NULL |
+-------------+

SELECT NULLIF(1,2);
+-------------+
| NULLIF(1,2) |
+-------------+
|           1 |
+-------------+
```

1.2.6.7 NVL

MariaDB starting with 10.3

From MariaDB 10.3, NVL is a synonym for IFNULL.

1.2.6.8 NVL2

MariaDB starting with 10.3

The NLV2 function was introduced in MariaDB 10.3.0.

Syntax

```
NVL2(expr1,expr2,expr3)
```

Contents

Description

The `NVL2` function returns a value based on whether a specified expression is NULL or not. If *expr1* is not NULL, then NVL2 returns *expr2*. If *expr1* is NULL, then NVL2 returns *expr3*.

Examples

```
SELECT NVL2(NULL,1,2);
+----------------+
| NVL2(NULL,1,2) |
+----------------+
|              2 |
+----------------+

SELECT NVL2('x',1,2);
+---------------+
| NVL2('x',1,2) |
+---------------+
|             1 |
+---------------+
```

1.2.7 Pseudo Columns

MariaDB has pseudo columns that can be used for different purposes.

_rowid

_rowid is an alias for the primary key column

1.2.7.1 _rowid

Syntax

_rowid

Description

The `_rowid` pseudo column is mapped to the primary key in the related table. This can be used as a replacement of the `rowid` pseudo column in other databases. Another usage is to simplify sql queries as one doesn't have to know the name of the primary key.

Examples

```
create table t1 (a int primary key, b varchar(80));
insert into t1 values (1,"one"),(2,"two");
select * from t1 where _rowid=1;
```

```
+---+------+
| a | b    |
+---+------+
| 1 | one  |
+---+------+
```

```
update t1 set b="three" where _rowid=2;
select * from t1 where _rowid>=1 and _rowid<=10;
```

```
+---+-------+
| a | b     |
+---+-------+
| 1 | one   |
| 2 | three |
+---+-------+
```

1.2.8 Secondary Functions

These are commonly used functions, but they are not primary functions.

Bit Functions and Operators

Operators for comparison and setting of values, and related functions.

Encryption, Hashing and Compression Functions

Functions used for encryption, hashing and compression.

Information Functions

Functions which return information on the server, the user, or a given query.

Miscellaneous Functions

Functions for very singular and specific needs.

1.2.8.1 Bit Functions and Operators

Operators for comparison and setting of values, and related functions. They all return a result of the `BIGINT UNSIGNED` type

Operator Precedence

Precedence of SQL operators

&

Bitwise AND

<<

Left shift

>>

Shift right

BIT_COUNT

Returns the number of set bits

^

Bitwise XOR

|

Bitwise OR

~

Bitwise NOT

Parentheses

Parentheses modify the precedence of other operators in an expression

TRUE FALSE

TRUE and FALSE evaluate to 1 and 0

1.1.5.6 Operator Precedence

1.2.8.1.2 &

Syntax

```
&
```

Description

Bitwise AND. Converts the values to binary and compares bits. Only if both the corresponding bits are 1 is the resulting bit also 1.

See also bitwise OR.

Examples

```
SELECT 2&1;
+-----+
| 2&1 |
+-----+
|   0 |
+-----+

SELECT 3&1;
+-----+
| 3&1 |
+-----+
|   1 |
+-----+

SELECT 29 & 15;
+---------+
| 29 & 15 |
+---------+
|      13 |
+---------+
```

1.2.8.1.3 <<

Syntax

```
value1 << value2
```

Description

Converts a longlong (BIGINT) number (*value1*) to binary and shifts *value2* units to the left.

Examples

```
SELECT 1 << 2;
+--------+
| 1 << 2 |
+--------+
|      4 |
+--------+
```

1.2.8.1.4 >>

Syntax

```
value1 >> value2
```

Description

Converts a longlong (BIGINT) number (*value1*) to binary and shifts *value2* units to the right.

Examples

```
SELECT 4 >> 2;
+--------+
| 4 >> 2 |
+--------+
|      1 |
+--------+
```

1.2.8.1.5 BIT_COUNT

Syntax

```
BIT_COUNT(N)
```

Description

Returns the number of bits that are set in the argument N.

Examples

```
SELECT BIT_COUNT(29), BIT_COUNT(b'101010');
+---------------+----------------------+
| BIT_COUNT(29) | BIT_COUNT(b'101010') |
+---------------+----------------------+
|             4 |                    3 |
+---------------+----------------------+
```

1.2.8.1.6 ^

Syntax

```
^
```

Description

Bitwise XOR. Converts the values to binary and compares bits. If one (and only one) of the corresponding bits is 1 is the resulting bit also 1.

Examples

```
SELECT 1 ^ 1;
+-------+
| 1 ^ 1 |
+-------+
|     0 |
+-------+

SELECT 1 ^ 0;
+-------+
| 1 ^ 0 |
+-------+
|     1 |
+-------+

SELECT 11 ^ 3;
+--------+
| 11 ^ 3 |
+--------+
|      8 |
+--------+
```

1.2.8.1.7 |

Syntax

```
|
```

Description

Bitwise OR. Converts the values to binary and compares bits. If either of the corresponding bits has a value of 1, the resulting bit is also 1.

See also bitwise AND.

Examples

```
SELECT 2|1;
+-----+
| 2|1 |
+-----+
|   3 |
+-----+

SELECT 29 | 15;
+---------+
| 29 | 15 |
+---------+
|      31 |
+---------+
```

1.2.8.1.8 ~

Syntax

```
~
```

Description

Bitwise NOT. Converts the value to 4 bytes binary and inverts all bits.

Examples

```
SELECT 3 & ~1;
+--------+
| 3 & ~1 |
+--------+
|      2 |
+--------+

SELECT 5 & ~1;
+--------+
| 5 & ~1 |
+--------+
|      4 |
+--------+
```

1.2.8.1.9 Parentheses

Parentheses are sometimes called precedence operators - this means that they can be used to change the other operator's precedence in an expression. The expressions that are written between parentheses are computed before the expressions that are written outside. Parentheses must always contain an expression (that is, they cannot be empty), and can be nested.

For example, the following expressions could return different results:

- `NOT a OR b`
- `NOT (a OR b)`

In the first case, `NOT` applies to `a`, so if `a` is `FALSE` or `b` is `TRUE`, the expression returns `TRUE`. In the second case, `NOT` applies to the result of `a OR b`, so if at least one of `a` or `b` is `TRUE`, the expression is `TRUE`.

When the precedence of operators is not intuitive, you can use parentheses to make it immediately clear for whoever reads the statement.

The precedence of the `NOT` operator can also be affected by the `HIGH_NOT_PRECEDENCE` SQL_MODE flag.

Other uses

Parentheses must always be used to enclose subqueries.

Parentheses can also be used in a `JOIN` statement between multiple tables to determine which tables must be joined first.

Also, parentheses are used to enclose the list of parameters to be passed to built-in functions, user-defined functions and stored routines. However, when no parameter is passed to a stored procedure, parentheses are optional. For builtin functions and user-defined functions, spaces are not allowed between the function name and the open parenthesis, unless the `IGNORE_SPACE` SQL_MODE is set. For stored routines (and for functions if `IGNORE_SPACE` is set) spaces are allowed before the open parenthesis, including tab characters and new line characters.

Syntax errors

If there are more open parentheses than closed parentheses, the error usually looks like this:

```
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that
corresponds to your MariaDB server version for the right syntax to use near '' a
t line 1
```

Note the empty string.

If there are more closed parentheses than open parentheses, the error usually looks like this:

```
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that
corresponds to your MariaDB server version for the right syntax to use near ')'
at line 1
```

Note the quoted closed parenthesis.

1.2.8.1.10 TRUE FALSE

Description

The constants TRUE and FALSE evaluate to 1 and 0, respectively. The constant names can be written in any lettercase.

Examples

```
SELECT TRUE, true, FALSE, false;
+------+------+-------+-------+
| TRUE | TRUE | FALSE | FALSE |
+------+------+-------+-------+
|    1 |    1 |     0 |     0 |
+------+------+-------+-------+
```

1.2.8.2 Encryption, Hashing and Compression Functions

Encryption, hashing and compression functions, such as ENCRYPT, DECRYPT, COMPRESS, PASSWORD etc.

AES_DECRYPT

Decryption data encrypted with AES_ENCRYPT.

AES_ENCRYPT

Encrypts a string with the AES algorithm.

COMPRESS

Returns a binary, compressed string.

DECODE

Decrypts a string encoded with ENCODE(), or, in Oracle mode, matches expressions.

DES_DECRYPT

Decrypts a string encrypted with DES_ENCRYPT().

DES_ENCRYPT

Encrypts a string using the Triple-DES algorithm.

ENCODE

Encrypts a string.

ENCRYPT

Encrypts a string with Unix crypt().

KDF

Key derivation function

MD5
MD5 checksum.

OLD_PASSWORD
Pre MySQL 4.1 password implementation.

PASSWORD
Calculates a password string.

RANDOM_BYTES
Generates a binary string of random bytes.

SHA1
Calculates an SHA-1 checksum.

SHA2
Calculates an SHA-2 checksum.

UNCOMPRESS
Uncompresses string compressed with COMPRESS().

UNCOMPRESSED_LENGTH
Returns length of a string before being compressed with COMPRESS().

There are 1 related questions.

1.2.8.2.1 AES_DECRYPT

Syntax

```
AES_DECRYPT(crypt_str,key_str)
```

From MariaDB 11.2.0

```
AES_ENCRYPT(crypt_str, key_str, [, iv [, mode]])
```

Description

This function allows decryption of data using the official AES (Advanced Encryption Standard) algorithm. For more information, see the description of AES_ENCRYPT().

MariaDB starting with 11.2

From MariaDB 11.2, the function supports an initialization vector, and control of the block encryption mode. The default mode is specified by the block_encryption_mode system variable, which can be changed when calling the function with a mode. *mode* is aes-{128,192,256}-{ecb,cbc,ctr} for example: "AES-128-cbc".

For modes that require it, the initialization_vector *iv* should be 16 bytes (it can be longer, but the extra bytes are ignored). A shorter *iv*, where one is required, results in the function returning NULL. Calling RANDOM_BYTES(16) will generate a random series of bytes that can be used for the *iv*.

Examples

From MariaDB 11.2.0:

```
SELECT HEX(AES_ENCRYPT('foo', 'bar', '0123456789abcdef', 'aes-128-ctr')) AS x;
+--------+
| x      |
+--------+
| C57C4B |
+--------+


SELECT AES_DECRYPT(x'C57C4B', 'bar', '0123456789abcdef', 'aes-128-ctr');
+-----------------------------------------------------------------+
| AES_DECRYPT(x'C57C4B', 'bar', '0123456789abcdef', 'aes-128-ctr') |
+-----------------------------------------------------------------+
| foo                                                             |
+-----------------------------------------------------------------+
```

1.2.8.2.2 AES_ENCRYPT

Syntax

```
AES_ENCRYPT(str,key_str)
```

From MariaDB 11.2.0

```
AES_ENCRYPT(str, key, [, iv [, mode]])
```

Description

`AES_ENCRYPT()` and AES_DECRYPT() allow encryption and decryption of data using the official AES (Advanced Encryption Standard) algorithm, previously known as "Rijndael." Encoding with a 128-bit key length is used (from MariaDB 11.2.0, this is the default, and can be changed). 128 bits is much faster and is secure enough for most purposes.

`AES_ENCRYPT()` encrypts a string *`str`* using the key *`key_str`*, and returns a binary string.

`AES_DECRYPT()` decrypts the encrypted string and returns the original string.

The input arguments may be any length. If either argument is NULL, the result of this function is also `NULL`.

Because AES is a block-level algorithm, padding is used to encode uneven length strings and so the result string length may be calculated using this formula:

```
16 x (trunc(string_length / 16) + 1)
```

If `AES_DECRYPT()` detects invalid data or incorrect padding, it returns `NULL`. However, it is possible for `AES_DECRYPT()` to return a non-`NULL` value (possibly garbage) if the input data or the key is invalid.

MariaDB starting with 11.2

From MariaDB 11.2, the function supports an initialization vector, and control of the block encryption mode. The default mode is specified by the block_encryption_mode system variable, which can be changed when calling the function with a mode. *mode* is aes-{128,192,256}-{ecb,cbc,ctr} for example: "AES-128-cbc".

AES_ENCRYPT(str, key) can no longer be used in persistent virtual columns (and the like).

Examples

```
INSERT INTO t VALUES (AES_ENCRYPT('text',SHA2('password',512)));
```

From MariaDB 11.2.0:

```
SELECT HEX(AES_ENCRYPT('foo', 'bar', '0123456789abcdef', 'aes-256-cbc')) AS x;
+----------------------------------+
| x                                |
+----------------------------------+
| 42A3EB91E6DFC40A900D278F99E0726E |
+----------------------------------+
```

1.2.8.2.3 COMPRESS

Syntax

```
COMPRESS(string_to_compress)
```

Description

Compresses a string and returns the result as a binary string. This function requires MariaDB to have been compiled with a compression library such as zlib. Otherwise, the return value is always `NULL`. The compressed string can be uncompressed with `UNCOMPRESS()`.

The have_compress server system variable indicates whether a compression library is present.

Examples

```
SELECT LENGTH(COMPRESS(REPEAT('a',1000)));
+-----------------------------------+
| LENGTH(COMPRESS(REPEAT('a',1000))) |
+-----------------------------------+
|                                21 |
+-----------------------------------+

SELECT LENGTH(COMPRESS(''));
+----------------------+
| LENGTH(COMPRESS('')) |
+----------------------+
|                    0 |
+----------------------+

SELECT LENGTH(COMPRESS('a'));
+-----------------------+
| LENGTH(COMPRESS('a')) |
+-----------------------+
|                    13 |
+-----------------------+

SELECT LENGTH(COMPRESS(REPEAT('a',16)));
+---------------------------------+
| LENGTH(COMPRESS(REPEAT('a',16))) |
+---------------------------------+
|                              15 |
+---------------------------------+
```

1.2.6.2 DECODE

1.2.8.2.5 DES_DECRYPT

DES_DECRYPT has been deprecated from MariaDB 10.10.0, and will be removed in a future release.

Syntax

```
DES_DECRYPT(crypt_str[,key_str])
```

Description

Decrypts a string encrypted with `DES_ENCRYPT()`. If an error occurs, this function returns `NULL`.

This function works only if MariaDB has been configured with TLS support.

If no `key_str` argument is given, `DES_DECRYPT()` examines the first byte of the encrypted string to determine the DES key number that was used to encrypt the original string, and then reads the key from the DES key file to decrypt the message. For this to work, the user must have the SUPER privilege. The key file can be specified with the `--des-key-file` server option.

If you pass this function a `key_str` argument, that string is used as the key for decrypting the message.

If the `crypt_str` argument does not appear to be an encrypted string, MariaDB returns the given crypt_str.

1.2.8.2.6 DES_ENCRYPT

DES_ENCRYPT has been deprecated from MariaDB 10.10.0, and will be removed in a future release.

Syntax

```
DES_ENCRYPT(str[,{key_num|key_str}])
```

Description

Encrypts the string with the given key using the Triple-DES algorithm.

This function works only if MariaDB has been configured with TLS support.

The encryption key to use is chosen based on the second argument to `DES_ENCRYPT()`, if one was given. With no argument, the first key from the DES key file is used. With a *`key_num`* argument, the given key number (0-9) from the DES key file is used. With a *`key_str`* argument, the given key string is used to encrypt *`str`*.

The key file can be specified with the `--des-key-file` server option.

The return string is a binary string where the first character is `CHAR(128 | key_num)`. If an error occurs, `DES_ENCRYPT()` returns `NULL`.

The 128 is added to make it easier to recognize an encrypted key. If you use a string key, *`key_num`* is 127.

The string length for the result is given by this formula:

```
new_len = orig_len + (8 - (orig_len % 8)) + 1
```

Each line in the DES key file has the following format:

```
key_num des_key_str
```

Each *`key_num`* value must be a number in the range from 0 to 9. Lines in the file may be in any order. *`des_key_str`* is the string that is used to encrypt the message. There should be at least one space between the number and the key. The first key is the default key that is used if you do not specify any key argument to `DES_ENCRYPT()`.

You can tell MariaDB to read new key values from the key file with the FLUSH DES_KEY_FILE statement. This requires the RELOAD privilege.

One benefit of having a set of default keys is that it gives applications a way to check for the existence of encrypted column values, without giving the end user the right to decrypt those values.

Examples

```
SELECT customer_address FROM customer_table
   WHERE crypted_credit_card = DES_ENCRYPT('credit_card_number');
```

1.2.8.2.7 ENCODE

Syntax

```
ENCODE(str,pass_str)
```

Description

ENCODE is not considered cryptographically secure, and should not be used for password encryption.

Encrypt `str` using `pass_str` as the password. To decrypt the result, use `DECODE()`.

The result is a binary string of the same length as `str`.

The strength of the encryption is based on how good the random generator is.

It is not recommended to rely on the encryption performed by the ENCODE function. Using a salt value (changed when a password is updated) will improve matters somewhat, but for storing passwords, consider a more cryptographically secure function, such as SHA2().

Examples

```
ENCODE('not so secret text', CONCAT('random_salt','password'))
```

1.2.8.2.8 ENCRYPT

Syntax

```
ENCRYPT(str[,salt])
```

Description

Encrypts a string using the Unix crypt() system call, returning an encrypted binary string. The `salt` argument should be a string with at least two characters or the returned result will be NULL. If no salt argument is given, a random value of sufficient length is used.

It is not recommended to use ENCRYPT() with utf16, utf32 or ucs2 multi-byte character sets because the crypt() system call expects a string terminated with a zero byte.

Note that the underlying crypt() system call may have some limitations, such as ignoring all but the first eight characters.

If the have_crypt system variable is set to `NO` (because the crypt() system call is not available), the ENCRYPT function will always return NULL.

Examples

```
SELECT ENCRYPT('encrypt me');
+----------------------+
| ENCRYPT('encrypt me') |
+----------------------+
| 4I5BsEx0lqTDk         |
+----------------------+
```

1.2.8.2.9 KDF

MariaDB starting with 11.3

KDF() is a key derivation function introduced in MariaDB 11.3.0.

Syntax

```
KDF(key_str, salt [, {info | iterations} [, kdf_name [, width ]]])
```

Description

`KDF` is a key derivation function, similar to OpenSSL's EVP_KDF_derive(), useful for generating good encryption keys for AES_ENCRYPT.

- *kdf_name* is "hkdf" or "pbkdf2_hmac" (default)
- *width* (in bits) can be any number divisible by 8, by default it's taken from @@block_encryption_mode
- *iterations* must be positive, and is 1000 by default

Note that OpenSSL 1.0 doesn't support HKDF, so in this case NULL is returned. This OpenSSL version is still used in SLES 12 and CentOS 7.

Examples

```
select hex(kdf('foo', 'bar', 'infa', 'hkdf'));
+----------------------------------------+
| hex(kdf('foo', 'bar', 'infa', 'hkdf')) |
+----------------------------------------+
| 612875F859CFB4EE0DFEFF9F2A18E836       |
+----------------------------------------+
```

1.2.8.2.10 MD5

Syntax

```
MD5(str)
```

Description

Calculates an MD5 128-bit checksum for the string.

The return value is a 32-hex digit string, and as of MariaDB 5.5, is a nonbinary string in the connection character set and collation, determined by the values of the character_set_connection and collation_connection system variables. Before 5.5, the return value was a binary string.

NULL is returned if the argument was NULL.

Examples

```
SELECT MD5('testing');
+----------------------------------+
| MD5('testing')                   |
+----------------------------------+
| ae2b1fca515949e5d54fb22b8ed95575 |
+----------------------------------+
```

1.2.8.2.11 OLD_PASSWORD

Syntax

```
OLD_PASSWORD(str)
```

Contents

Description

`OLD_PASSWORD()` was added to MySQL when the implementation of PASSWORD() was changed to improve security. `OLD_PASSWORD()` returns the value of the old (pre-MySQL 4.1) implementation of PASSWORD() as a string, and is intended to permit you to reset passwords for any pre-4.1 clients that need to connect to a more recent MySQL server version, or any version of MariaDB, without locking them out.

As of MariaDB 5.5, the return value is a nonbinary string in the connection character set and collation, determined by the values of the character_set_connection and collation_connection system variables. Before 5.5, the return value was a binary string.

The return value is 16 bytes in length, or NULL if the argument was NULL.

1.2.8.2.12 PASSWORD

Syntax

```
PASSWORD(str)
```

Contents

Description

> The PASSWORD() function is used for hashing passwords for use in authentication by the MariaDB server. It is not intended for use in other applications.

Calculates and returns a hashed password string from the plaintext password *str*. Returns an empty string (>= MariaDB 10.0.4) if the argument was NULL.

The return value is a nonbinary string in the connection character set and collation, determined by the values of the character_set_connection and collation_connection system variables.

This is the function that is used for hashing MariaDB passwords for storage in the Password column of the user table (see privileges), usually used with the SET PASSWORD statement. It is not intended for use in other applications.

Until MariaDB 10.3, the return value is 41-bytes in length, and the first character is always '*'. From MariaDB 10.4, the function takes into account the authentication plugin where applicable (A CREATE USER or SET PASSWORD statement). For example, when used in conjunction with a user authenticated by the ed25519 plugin, the statement will create a longer hash:

```
CREATE USER edtest@localhost IDENTIFIED VIA ed25519 USING PASSWORD('secret');

CREATE USER edtest2@localhost IDENTIFIED BY 'secret';

SELECT CONCAT(user, '@', host, ' => ', JSON_DETAILED(priv)) FROM mysql.global_priv
  WHERE user LIKE 'edtest%'\G
*************************** 1. row ***************************
CONCAT(user, '@', host, ' => ', JSON_DETAILED(priv)): edtest@localhost => {
...
    "plugin": "ed25519",
    "authentication_string": "ZIgUREUg5PVgQ6LskhXmO+eZLS0nC8be6HPjYWR4YJY",
...
}
*************************** 2. row ***************************
CONCAT(user, '@', host, ' => ', JSON_DETAILED(priv)): edtest2@localhost => {
...
    "plugin": "mysql_native_password",
    "authentication_string": "*14E65567ABDB5135D0CFD9A70B3032C179A49EE7",
...
}
```

The behavior of this function is affected by the value of the old_passwords system variable. If this is set to `1` (`0` is default), MariaDB reverts to using the mysql_old_password authentication plugin by default for newly created users and passwords.

Examples

```
SELECT PASSWORD('notagoodpwd');
+-------------------------------------------+
| PASSWORD('notagoodpwd')                   |
+-------------------------------------------+
| *3A70EE9FC6594F88CE9E959CD51C5A1C002DC937 |
+-------------------------------------------+
```

```
SET PASSWORD FOR 'bob'@'%.loc.gov' = PASSWORD('newpass');
```

1.2.8.2.13 RANDOM_BYTES

MariaDB starting with 10.10.0

The RANDOM_BYTES function generates a binary string of random bytes. It was added in MariaDB 10.10.0.

Syntax

```
RANDOM_BYTES(length)
```

Description

Given a *length* from 1 to 1024, generates a binary string of *length* consisting of random bytes generated by the SSL library's random number generator.

See the RAND_bytes() function documentation of your SSL library for information on the random number generator. In the case of OpenSSL, a cryptographically secure pseudo random generator (CSPRNG) is used.

Statements containing the RANDOM_BYTES function are unsafe for statement-based replication.

An error occurs if *length* is outside the range 1 to 1024.

1.2.8.2.14 SHA1

Syntax

Syntax

```
SHA1(str), SHA(str)
```

Description

Calculates an SHA-1 160-bit checksum for the string *str*, as described in RFC 3174 (Secure Hash Algorithm).

The value is returned as a string of 40 hex digits, or NULL if the argument was NULL. As of MariaDB 5.5, the return value is a nonbinary string in the connection character set and collation, determined by the values of the character_set_connection and collation_connection system variables. Before 5.5, the return value was a binary string.

Examples

```
SELECT SHA1('some boring text');
+------------------------------------------+
| SHA1('some boring text')                 |
+------------------------------------------+
| af969fc2085b1bb6d31e517d5c456def5cdd7093 |
+------------------------------------------+
```

1.2.8.2.15 SHA2

Syntax

```
SHA2(str,hash_len)
```

Description

Given a string *str*, calculates an SHA-2 checksum, which is considered more cryptographically secure than its SHA-1 equivalent. The SHA-2 family includes SHA-224, SHA-256, SHA-384, and SHA-512, and the *hash_len* must correspond to one of these, i.e. 224, 256, 384 or 512. 0 is equivalent to 256.

The return value is a nonbinary string in the connection character set and collation, determined by the values of the character_set_connection and collation_connection system variables.

NULL is returned if the hash length is not valid, or the string *str* is NULL.

SHA2 will only work if MariaDB was has been configured with TLS support.

Examples

```
SELECT SHA2('Maria',224);
+----------------------------------------------------------+
| SHA2('Maria',224)                                        |
+----------------------------------------------------------+
| 6cc67add32286412efcab9d0e1675a43a5c2ef3cec8879f81516ff83 |
+----------------------------------------------------------+

SELECT SHA2('Maria',256);
+------------------------------------------------------------------+
| SHA2('Maria',256)                                                |
+------------------------------------------------------------------+
| 9ff18ebe7449349f358e3af0b57cf7a032c1c6b2272cb2656ff85eb112232f16 |
+------------------------------------------------------------------+

SELECT SHA2('Maria',0);
+------------------------------------------------------------------+
| SHA2('Maria',0)                                                  |
+------------------------------------------------------------------+
| 9ff18ebe7449349f358e3af0b57cf7a032c1c6b2272cb2656ff85eb112232f16 |
+------------------------------------------------------------------+
```

1.2.2.65 UNCOMPRESS

1.2.2.66 UNCOMPRESSED_LENGTH

1.2.8.3 Information Functions

General information functions, including BENCHMARK, CHARSET, DATABASE, USER, VERSION, etc.

BENCHMARK

Executes an expression repeatedly.

BINLOG_GTID_POS

Returns a string representation of the corresponding GTID position.

CHARSET

Returns the character set.

COERCIBILITY

Returns the collation coercibility value.

COLLATION

Collation of the string argument.

CONNECTION_ID

Connection ID.

CURRENT_ROLE

Current role name.

CURRENT_USER

Username/host that authenticated the current client.

DATABASE

Current default database.

DECODE_HISTOGRAM

Returns comma separated numerics corresponding to a probability distribution.

DEFAULT

Returns column default.

FOUND_ROWS

Number of (potentially) returned rows.

LAST_INSERT_ID

Last inserted auto_increment value.

LAST_VALUE

Returns the last value in a list or set of values.

PROCEDURE ANALYSE

Suggests optimal data types for each column.

ROWNUM

Function that returns the number of accepted rows so far.

ROW_COUNT

Number of rows affected by previous statement.

SCHEMA

Synonym for DATABASE().

SESSION_USER
Synonym for USER().

SYSTEM_USER
Synonym for USER().

USER
Current user/host.

VERSION
MariaDB server version.

There are 1 related questions.

1.2.8.3.1 BENCHMARK

Syntax

```
BENCHMARK(count,expr)
```

Description

The BENCHMARK() function executes the expression `expr` repeatedly `count` times. It may be used to time how quickly MariaDB processes the expression. The result value is always 0. The intended use is from within the mariadb client, which reports query execution times.

Examples

```
SELECT BENCHMARK(1000000,ENCODE('hello','goodbye'));
+----------------------------------------------+
| BENCHMARK(1000000,ENCODE('hello','goodbye')) |
+----------------------------------------------+
|                                            0 |
+----------------------------------------------+
1 row in set (0.21 sec)
```

1.2.8.3.2 BINLOG_GTID_POS

Syntax

```
BINLOG_GTID_POS(binlog_filename,binlog_offset)
```

Description

The BINLOG_GTID_POS() function takes as input an old-style binary log position in the form of a file name and a file offset. It looks up the position in the current binlog, and returns a string representation of the corresponding GTID position. If the position is not found in the current binlog, NULL is returned.

Examples

```
SELECT BINLOG_GTID_POS("master-bin.000001", 600);
```

1.2.8.3.3 CHARSET

Syntax

```
CHARSET(str)
```

Description

Returns the character set of the string argument. If `str` is not a string, it is considered as a binary string (so the function returns 'binary'). This applies to `NULL`, too. The return value is a string in the utf8 character set.

Examples

```
SELECT CHARSET('abc');
+----------------+
| CHARSET('abc') |
+----------------+
| latin1         |
+----------------+

SELECT CHARSET(CONVERT('abc' USING utf8));
+-----------------------------------+
| CHARSET(CONVERT('abc' USING utf8)) |
+-----------------------------------+
| utf8                              |
+-----------------------------------+

SELECT CHARSET(USER());
+-----------------+
| CHARSET(USER()) |
+-----------------+
| utf8            |
+-----------------+
```

1.2.8.3.4 COERCIBILITY

Syntax

```
COERCIBILITY(str)
```

Description

Returns the collation coercibility value of the string argument. Coercibility defines what will be converted to what in case of collation conflict, with an expression with higher coercibility being converted to the collation of an expression with lower coercibility.

Coercibility	Description	Example
0	Explicit	Value using a COLLATE clause
1	No collation	Concatenated strings using different collations
2	Implicit	A string data type column value, CAST to a string data type
3	System constant	DATABASE(), USER() return value
4	Coercible	Literal string
5	Numeric	Numeric and temporal values
6	Ignorable	NULL or derived from NULL

Examples

```
SELECT COERCIBILITY('abc' COLLATE latin1_swedish_ci);
+-------------------------------------------------+
| COERCIBILITY('abc' COLLATE latin1_swedish_ci) |
+-------------------------------------------------+
|                                               0 |
+-------------------------------------------------+

SELECT COERCIBILITY(CAST(1 AS CHAR));
+--------------------------------+
| COERCIBILITY(CAST(1 AS CHAR)) |
+--------------------------------+
|                              2 |
+--------------------------------+

SELECT COERCIBILITY(USER());
+----------------------+
| COERCIBILITY(USER()) |
+----------------------+
|                    3 |
+----------------------+

SELECT COERCIBILITY('abc');
+---------------------+
| COERCIBILITY('abc') |
+---------------------+
|                   4 |
+---------------------+

SELECT COERCIBILITY(1);
+-----------------+
| COERCIBILITY(1) |
+-----------------+
|               5 |
+-----------------+

SELECT COERCIBILITY(NULL);
+--------------------+
| COERCIBILITY(NULL) |
+--------------------+
|                  6 |
+--------------------+
```

1.2.8.3.5 COLLATION

Syntax

```
COLLATION(str)
```

Contents

Description

Returns the collation of the string argument. If `str` is not a string, it is considered as a binary string (so the function returns 'binary'). This applies to `NULL`, too. The return value is a string in the utf8 character set.

See Character Sets and Collations.

Examples

```
SELECT COLLATION('abc');
+------------------+
| COLLATION('abc')  |
+------------------+
| latin1_swedish_ci |
+------------------+

SELECT COLLATION(_utf8'abc');
+----------------------+
| COLLATION(_utf8'abc') |
+----------------------+
| utf8_general_ci       |
+----------------------+
```

1.2.8.3.6 CONNECTION_ID

Syntax

```
CONNECTION_ID()
```

Description

Returns the connection ID for the connection. Every connection (including events) has an ID that is unique among the set of currently connected clients.

Until MariaDB 10.3.1, returns `MYSQL_TYPE_LONGLONG`, or bigint(10), in all cases. From MariaDB 10.3.1, returns `MYSQL_TYPE_LONG`, or int(10), when the result would fit within 32-bits.

Examples

```
SELECT CONNECTION_ID();
+-----------------+
| CONNECTION_ID() |
+-----------------+
|               3 |
+-----------------+
```

1.2.8.3.7 CURRENT_ROLE

Syntax

```
CURRENT_ROLE, CURRENT_ROLE()
```

Description

Returns the current role name. This determines your access privileges. The return value is a string in the utf8 character set.

If there is no current role, NULL is returned.

The output of `SELECT CURRENT_ROLE` is equivalent to the contents of the ENABLED_ROLES Information Schema table.

USER() returns the combination of user and host used to login. CURRENT_USER() returns the account used to determine current connection's privileges.

Statements using the `CURRENT_ROLE` function are not safe for statement-based replication.

Examples

```
SELECT CURRENT_ROLE;
+--------------+
| CURRENT_ROLE |
+--------------+
| NULL         |
+--------------+

SET ROLE staff;

SELECT CURRENT_ROLE;
+--------------+
| CURRENT_ROLE |
+--------------+
| staff        |
+--------------+
```

1.2.8.3.8 CURRENT_USER

Syntax

```
CURRENT_USER, CURRENT_USER()
```

Description

Returns the user name and host name combination for the MariaDB account that the server used to authenticate the current client. This account determines your access privileges. The return value is a string in the utf8 character set.

The value of `CURRENT_USER()` can differ from the value of USER(). CURRENT_ROLE() returns the current active role.

Statements using the `CURRENT_USER` function are not safe for statement-based replication.

Examples

```
shell> mysql --user="anonymous"

select user(),current_user();
+---------------------+----------------+
| user()              | current_user() |
+---------------------+----------------+
| anonymous@localhost | @localhost     |
+---------------------+----------------+
```

When calling `CURRENT_USER()` in a stored procedure, it returns the owner of the stored procedure, as defined with `DEFINER`.

1.2.8.3.9 DATABASE

Syntax

```
DATABASE()
SCHEMA()
```

Description

Returns the default (current) database name as a string in the utf8 character set. If there is no default database, DATABASE() returns NULL. Within a stored routine, the default database is the database that the routine is associated with, which is not necessarily the same as the database that is the default in the calling context.

SCHEMA() is a synonym for DATABASE().

To select a default database, the USE statement can be run. Another way to set the default database is specifying its name at mariadb command line client startup.

Examples

```
SELECT DATABASE();
+------------+
| DATABASE() |
+------------+
| NULL       |
+------------+

USE test;
Database changed

SELECT DATABASE();
+------------+
| DATABASE() |
+------------+
| test       |
+------------+
```

1.2.8.3.10 DECODE_HISTOGRAM

Syntax

```
DECODE_HISTOGRAM(hist_type,histogram)
```

Description

Returns a string of comma separated numeric values corresponding to a probability distribution represented by the histogram of type `hist_type` (`SINGLE_PREC_HB` or `DOUBLE_PREC_HB`). The `hist_type` and `histogram` would be commonly used from the mysql.column_stats table.

See Histogram Based Statistics for details.

Examples

```
CREATE TABLE origin (
  i INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  v INT UNSIGNED NOT NULL
);

INSERT INTO origin(v) VALUES
  (1),(2),(3),(4),(5),(10),(20),
  (30),(40),(50),(60),(70),(80),
  (90),(100),(200),(400),(800);

SET histogram_size=10,histogram_type=SINGLE_PREC_HB;

ANALYZE TABLE origin PERSISTENT FOR ALL;
+-------------+---------+----------+-----------------------------------------+
| Table       | Op      | Msg_type | Msg_text                                |
+-------------+---------+----------+-----------------------------------------+
| test.origin | analyze | status   | Engine-independent statistics collected |
| test.origin | analyze | status   | OK                                      |
+-------------+---------+----------+-----------------------------------------+

SELECT db_name,table_name,column_name,hist_type,
  hex(histogram),decode_histogram(hist_type,histogram)
  FROM mysql.column_stats WHERE db_name='test' and table_name='origin';
+---------+------------+-------------+---------------+----------------------+---------------
-------------------------------------------------+
| db_name | table_name | column_name | hist_type      | hex(histogram)       |
decode_histogram(hist_type,histogram)                              |
+---------+------------+-------------+---------------+----------------------+---------------
-------------------------------------------------+
| test    | origin     | i           | SINGLE_PREC_HB | 0F2D3C5A7887A5C3D2F0 |
0.059,0.118,0.059,0.118,0.118,0.059,0.118,0.118,0.059,0.118,0.059 |
| test    | origin     | v           | SINGLE_PREC_HB | 000001060C0F161C1F7F |
0.000,0.000,0.004,0.020,0.024,0.012,0.027,0.024,0.012,0.376,0.502 |
+---------+------------+-------------+---------------+----------------------+---------------
-------------------------------------------------+

SET histogram_size=20,histogram_type=DOUBLE_PREC_HB;

ANALYZE TABLE origin PERSISTENT FOR ALL;
+-------------+---------+----------+-----------------------------------------+
| Table       | Op      | Msg_type | Msg_text                                |
+-------------+---------+----------+-----------------------------------------+
| test.origin | analyze | status   | Engine-independent statistics collected |
| test.origin | analyze | status   | OK                                      |
+-------------+---------+----------+-----------------------------------------+

SELECT db_name,table_name,column_name,
  hist_type,hex(histogram),decode_histogram(hist_type,histogram)
  FROM mysql.column_stats WHERE db_name='test' and table_name='origin';
+---------+------------+-------------+---------------+-----------------------------------------
----+-----------------------------------------------------------------------------------------
-+
| db_name | table_name | column_name | hist_type      | hex(histogram)
| decode_histogram(hist_type,histogram)                                                    |
+---------+------------+-------------+---------------+-----------------------------------------
----+-----------------------------------------------------------------------------------------
-+
| test    | origin     | i           | DOUBLE_PREC_HB |
0F0F2D2D3C3C5A5A78788787A5A5C3C3D2D2F0F0 |
0.05882,0.11765,0.05882,0.11765,0.11765,0.05882,0.11765,0.11765,0.05882,0.11765,0.05882 |
| test    | origin     | v           | DOUBLE_PREC_HB |
5200F600480116067E0CB30F1B16831CB81FD67F |
0.00125,0.00250,0.00125,0.01877,0.02502,0.01253,0.02502,0.02502,0.01253,0.37546,0.50063 |
```

1.2.8.3.11 DEFAULT

Syntax

```
DEFAULT(col_name)
```

Description

Returns the default value for a table column. If the column has no default value (and is not NULLABLE - NULLABLE fields have a NULL default), an error is returned.

For integer columns using AUTO_INCREMENT, `0` is returned.

When using `DEFAULT` as a value to set in an INSERT or UPDATE statement, you can use the bare keyword `DEFAULT` without the parentheses and argument to refer to the column in context. You can only use `DEFAULT` as a bare keyword if you are using it alone without a surrounding expression or function.

Examples

Select only non-default values for a column:

```
SELECT i FROM t WHERE i != DEFAULT(i);
```

Update values to be one greater than the default value:

```
UPDATE t SET i = DEFAULT(i)+1 WHERE i < 100;
```

When referring to the default value exactly in `UPDATE` or `INSERT`, you can omit the argument:

```
INSERT INTO t (i) VALUES (DEFAULT);
UPDATE t SET i = DEFAULT WHERE i < 100;
```

```
CREATE OR REPLACE TABLE t (
  i INT NOT NULL AUTO_INCREMENT,
  j INT NOT NULL,
  k INT DEFAULT 3,
  l INT NOT NULL DEFAULT 4,
  m INT,
  PRIMARY KEY (i)
);

DESC t;
+-------+---------+------+-----+---------+----------------+
| Field | Type    | Null | Key | Default | Extra          |
+-------+---------+------+-----+---------+----------------+
| i     | int(11) | NO   | PRI | NULL    | auto_increment |
| j     | int(11) | NO   |     | NULL    |                |
| k     | int(11) | YES  |     | 3       |                |
| l     | int(11) | NO   |     | 4       |                |
| m     | int(11) | YES  |     | NULL    |                |
+-------+---------+------+-----+---------+----------------+

INSERT INTO t (j) VALUES (1);
INSERT INTO t (j,m) VALUES (2,2);
INSERT INTO t (j,l,m) VALUES (3,3,3);

SELECT * FROM t;
+---+---+------+---+------+
| i | j | k    | l | m    |
+---+---+------+---+------+
| 1 | 1 |    3 | 4 | NULL |
| 2 | 2 |    3 | 4 |    2 |
| 3 | 3 |    3 | 3 |    3 |
+---+---+------+---+------+

SELECT DEFAULT(i), DEFAULT(k), DEFAULT (l), DEFAULT(m) FROM t;
+------------+------------+-------------+------------+
| DEFAULT(i) | DEFAULT(k) | DEFAULT (l) | DEFAULT(m) |
+------------+------------+-------------+------------+
|          0 |          3 |           4 |       NULL |
|          0 |          3 |           4 |       NULL |
|          0 |          3 |           4 |       NULL |
+------------+------------+-------------+------------+

SELECT DEFAULT(i), DEFAULT(k), DEFAULT (l), DEFAULT(m), DEFAULT(j)  FROM t;
ERROR 1364 (HY000): Field 'j' doesn't have a default value
```

```
ERROR 1364 (HY000): Field 'j' doesn't have a default value

SELECT * FROM t WHERE i = DEFAULT(i);
Empty set (0.001 sec)

SELECT * FROM t WHERE j = DEFAULT(j);
ERROR 1364 (HY000): Field 'j' doesn't have a default value

SELECT * FROM t WHERE k = DEFAULT(k);
+---+---+------+---+------+
| i | j | k    | l | m    |
+---+---+------+---+------+
| 1 | 1 |    3 | 4 | NULL |
| 2 | 2 |    3 | 4 |    2 |
| 3 | 3 |    3 | 3 |    3 |
+---+---+------+---+------+

SELECT * FROM t WHERE l = DEFAULT(l);
+---+---+------+---+------+
| i | j | k    | l | m    |
+---+---+------+---+------+
| 1 | 1 |    3 | 4 | NULL |
| 2 | 2 |    3 | 4 |    2 |
+---+---+------+---+------+

SELECT * FROM t WHERE m = DEFAULT(m);
Empty set (0.001 sec)

SELECT * FROM t WHERE m <=> DEFAULT(m);
+---+---+------+---+------+
| i | j | k    | l | m    |
+---+---+------+---+------+
| 1 | 1 |    3 | 4 | NULL |
+---+---+------+---+------+
```

1.2.8.3.12 FOUND_ROWS

Syntax

```
FOUND_ROWS()
```

Description

A SELECT statement may include a LIMIT clause to restrict the number of rows the server returns to the client. In some cases, it is desirable to know how many rows the statement would have returned without the LIMIT, but without running the statement again. To obtain this row count, include a SQL_CALC_FOUND_ROWS option in the SELECT statement, and then invoke FOUND_ROWS() afterwards.

You can also use FOUND_ROWS() to obtain the number of rows returned by a SELECT which does not contain a LIMIT clause. In this case you don't need to use the SQL_CALC_FOUND_ROWS option. This can be useful for example in a stored procedure.

Also, this function works with some other statements which return a resultset, including SHOW, DESC and HELP. For DELETE ... RETURNING you should use ROW_COUNT(). It also works as a prepared statement, or after executing a prepared statement.

Statements which don't return any results don't affect FOUND_ROWS() - the previous value will still be returned.

Warning: When used after a CALL statement, this function returns the number of rows selected by the last query in the procedure, not by the whole procedure.

Statements using the FOUND_ROWS() function are not safe for statement-based replication.

Examples

```
SHOW ENGINES\G
*************************** 1. row ***************************
      Engine: CSV
     Support: YES
     Comment: Stores tables as CSV files
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 2. row ***************************
      Engine: MRG_MyISAM
     Support: YES
     Comment: Collection of identical MyISAM tables
Transactions: NO
          XA: NO
  Savepoints: NO

...

*************************** 8. row ***************************
      Engine: PERFORMANCE_SCHEMA
     Support: YES
     Comment: Performance Schema
Transactions: NO
          XA: NO
  Savepoints: NO
8 rows in set (0.000 sec)

SELECT FOUND_ROWS();
+--------------+
| FOUND_ROWS() |
+--------------+
|            8 |
+--------------+

SELECT SQL_CALC_FOUND_ROWS * FROM tbl_name WHERE id > 100 LIMIT 10;

SELECT FOUND_ROWS();
+--------------+
| FOUND_ROWS() |
+--------------+
|           23 |
+--------------+
```

1.2.8.3.13 LAST_INSERT_ID

Syntax

```
LAST_INSERT_ID(), LAST_INSERT_ID(expr)
```

Contents

Description

LAST_INSERT_ID() (no arguments) returns the first automatically generated value successfully inserted for an AUTO_INCREMENT column as a result of the most recently executed INSERT statement. The value of LAST_INSERT_ID() remains unchanged if no rows are successfully inserted.

If one gives an argument to LAST_INSERT_ID(), then it will return the value of the expression and the next call to LAST_INSERT_ID() will return the same value. The value will also be sent to the client and can be accessed by the mysql_insert_id function.

For example, after inserting a row that generates an AUTO_INCREMENT value, you can get the value like this:

```
SELECT LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                9 |
+------------------+
```

You can also use LAST_INSERT_ID() to delete the last inserted row:

```
DELETE FROM product WHERE id = LAST_INSERT_ID();
```

If no rows were successfully inserted, LAST_INSERT_ID() returns 0.

The value of LAST_INSERT_ID() will be consistent across all versions if all rows in the INSERT or UPDATE statement were successful.

The currently executing statement does not affect the value of LAST_INSERT_ID(). Suppose that you generate an AUTO_INCREMENT value with one statement, and then refer to LAST_INSERT_ID() in a multiple-row INSERT statement that inserts rows into a table with its own AUTO_INCREMENT column. The value of LAST_INSERT_ID() will remain stable in the second statement; its value for the second and later rows is not affected by the earlier row insertions. (However, if you mix references to LAST_INSERT_ID() and LAST_INSERT_ID(expr), the effect is undefined.)

If the previous statement returned an error, the value of LAST_INSERT_ID() is undefined. For transactional tables, if the statement is rolled back due to an error, the value of LAST_INSERT_ID() is left undefined. For manual ROLLBACK, the value of LAST_INSERT_ID() is not restored to that before the transaction; it remains as it was at the point of the ROLLBACK.

Within the body of a stored routine (procedure or function) or a trigger, the value of LAST_INSERT_ID() changes the same way as for statements executed outside the body of these kinds of objects. The effect of a stored routine or trigger upon the value of LAST_INSERT_ID() that is seen by following statements depends on the kind of routine:

- If a stored procedure executes statements that change the value of LAST_INSERT_ID(), the new value will be seen by statements that follow the procedure call.
- For stored functions and triggers that change the value, the value is restored when the function or trigger ends, so following statements will not see a changed value.

Examples

```
CREATE TABLE t (
  id INTEGER UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  f VARCHAR(1))
ENGINE = InnoDB;

INSERT INTO t(f) VALUES('a');

SELECT LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                1 |
+------------------+

INSERT INTO t(f) VALUES('b');

INSERT INTO t(f) VALUES('c');

SELECT LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                3 |
+------------------+

INSERT INTO t(f) VALUES('d'),('e');

SELECT LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                4 |
+------------------+
```

```
SELECT * FROM t;
+----+------+
| id | f    |
+----+------+
|  1 | a    |
|  2 | b    |
|  3 | c    |
|  4 | d    |
|  5 | e    |
+----+------+

SELECT LAST_INSERT_ID(12);
+--------------------+
| LAST_INSERT_ID(12) |
+--------------------+
|                 12 |
+--------------------+

SELECT LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
+------------------+
|               12 |
+------------------+

INSERT INTO t(f) VALUES('f');

SELECT LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                6 |
+------------------+

SELECT * FROM t;
+----+------+
| id | f    |
+----+------+
|  1 | a    |
|  2 | b    |
|  3 | c    |
|  4 | d    |
|  5 | e    |
|  6 | f    |
+----+------+

SELECT LAST_INSERT_ID(12);
+--------------------+
| LAST_INSERT_ID(12) |
+--------------------+
|                 12 |
+--------------------+

INSERT INTO t(f) VALUES('g');

SELECT * FROM t;
+----+------+
| id | f    |
+----+------+
|  1 | a    |
|  2 | b    |
|  3 | c    |
|  4 | d    |
|  5 | e    |
|  6 | f    |
|  7 | g    |
+----+------+
```

1.2.8.3.14 LAST_VALUE

Syntax

```
LAST_VALUE(expr,[expr,...])
```

```
LAST_VALUE(expr) OVER (
  [ PARTITION BY partition_expression ]
  [ ORDER BY order_list ]
)
```

Contents

Description

`LAST_VALUE()` evaluates all expressions and returns the last.

This is useful together with setting user variables to a value with @var:=expr, for example when you want to get data of rows updated/deleted without having to do two queries against the table.

LAST_VALUE can be used as a window function.

Returns NULL if no last value exists.

Examples

```
CREATE TABLE t1 (a int, b int);
INSERT INTO t1 VALUES(1,10),(2,20);
DELETE FROM t1 WHERE a=1 AND last_value(@a:=a,@b:=b,1);
SELECT @a,@b;
+------+------+
| @a   | @b   |
+------+------+
|    1 |   10 |
+------+------+
```

As a window function:

```
CREATE TABLE t1 (
  pk int primary key,
  a int,
  b int,
  c char(10),
  d decimal(10, 3),
  e real
);

INSERT INTO t1 VALUES
( 1, 0, 1,    'one',    0.1,  0.001),
( 2, 0, 2,    'two',    0.2,  0.002),
( 3, 0, 3,    'three',  0.3,  0.003),
( 4, 1, 2,    'three',  0.4,  0.004),
( 5, 1, 1,    'two',    0.5,  0.005),
( 6, 1, 1,    'one',    0.6,  0.006),
( 7, 2, NULL, 'n_one',  0.5,  0.007),
( 8, 2, 1,    'n_two',  NULL, 0.008),
( 9, 2, 2,    NULL,     0.7,  0.009),
(10, 2, 0,    'n_four', 0.8,  0.010),
(11, 2, 10,   NULL,     0.9,  NULL);

SELECT pk, FIRST_VALUE(pk) OVER (ORDER BY pk) AS first_asc,
           LAST_VALUE(pk) OVER (ORDER BY pk) AS last_asc,
           FIRST_VALUE(pk) OVER (ORDER BY pk DESC) AS first_desc,
           LAST_VALUE(pk) OVER (ORDER BY pk DESC) AS last_desc
FROM t1
ORDER BY pk DESC;

+----+-----------+----------+------------+-----------+
| pk | first_asc | last_asc | first_desc | last_desc |
+----+-----------+----------+------------+-----------+
| 11 |         1 |       11 |         11 |        11 |
| 10 |         1 |       10 |         11 |        10 |
|  9 |         1 |        9 |         11 |         9 |
|  8 |         1 |        8 |         11 |         8 |
|  7 |         1 |        7 |         11 |         7 |
|  6 |         1 |        6 |         11 |         6 |
|  5 |         1 |        5 |         11 |         5 |
|  4 |         1 |        4 |         11 |         4 |
|  3 |         1 |        3 |         11 |         3 |
|  2 |         1 |        2 |         11 |         2 |
|  1 |         1 |        1 |         11 |         1 |
+----+-----------+----------+------------+-----------+
```

```
CREATE OR REPLACE TABLE t1 (i int);
INSERT INTO t1 VALUES (1),(2),(3),(4),(5),(6),(7),(8),(9),(10);

SELECT i,
  FIRST_VALUE(i) OVER (ORDER BY i ROWS BETWEEN CURRENT ROW and 1 FOLLOWING) AS f_1f,
  LAST_VALUE(i) OVER (ORDER BY i ROWS BETWEEN CURRENT ROW and 1 FOLLOWING) AS l_1f,
  FIRST_VALUE(i) OVER (ORDER BY i ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS f_1p1f,
  LAST_VALUE(i) OVER (ORDER BY i ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS l_1p1f,
  FIRST_VALUE(i) OVER (ORDER BY i ROWS BETWEEN 2 PRECEDING AND 1 PRECEDING) AS f_2p1p,
  LAST_VALUE(i) OVER (ORDER BY i ROWS BETWEEN 2 PRECEDING AND 1 PRECEDING) AS l_2p1p,
  FIRST_VALUE(i) OVER (ORDER BY i ROWS BETWEEN 1 FOLLOWING AND 2 FOLLOWING) AS f_1f2f,
  LAST_VALUE(i) OVER (ORDER BY i ROWS BETWEEN 1 FOLLOWING AND 2 FOLLOWING) AS l_1f2f
FROM t1;

+------+------+------+--------+--------+--------+--------+--------+--------+
| i    | f_1f | l_1f | f_1p1f | l_1p1f | f_2p1p | l_2p1p | f_1f2f | l_1f2f |
+------+------+------+--------+--------+--------+--------+--------+--------+
|    1 |    1 |    2 |      1 |      2 |   NULL |   NULL |      2 |      3 |
|    2 |    2 |    3 |      1 |      3 |      1 |      1 |      3 |      4 |
|    3 |    3 |    4 |      2 |      4 |      1 |      2 |      4 |      5 |
|    4 |    4 |    5 |      3 |      5 |      2 |      3 |      5 |      6 |
|    5 |    5 |    6 |      4 |      6 |      3 |      4 |      6 |      7 |
|    6 |    6 |    7 |      5 |      7 |      4 |      5 |      7 |      8 |
|    7 |    7 |    8 |      6 |      8 |      5 |      6 |      8 |      9 |
|    8 |    8 |    9 |      7 |      9 |      6 |      7 |      9 |     10 |
|    9 |    9 |   10 |      8 |     10 |      7 |      8 |     10 |     10 |
|   10 |   10 |   10 |      9 |     10 |      8 |      9 |   NULL |   NULL |
+------+------+------+--------+--------+--------+--------+--------+--------+
```

1.2.8.3.15 PROCEDURE ANALYSE

Syntax

```
analyse([max_elements[,max_memory]])
```

Description

This procedure is defined in the sql/sql_analyse.cc file. It examines the result from a query and returns an analysis of the results that suggests optimal data types for each column. To obtain this analysis, append PROCEDURE ANALYSE to the end of a SELECT statement:

```
SELECT ... FROM ... WHERE ... PROCEDURE ANALYSE([max_elements,[max_memory]])
```

For example:

```
SELECT col1, col2 FROM table1 PROCEDURE ANALYSE(10, 2000);
```

The results show some statistics for the values returned by the query, and propose an optimal data type for the columns. This can be helpful for checking your existing tables, or after importing new data. You may need to try different settings for the arguments so that PROCEDURE ANALYSE() does not suggest the ENUM data type when it is not appropriate.

The arguments are optional and are used as follows:

- max_elements (default 256) is the maximum number of distinct values that analyse notices per column. This is used by analyse to check whether the optimal data type should be of type ENUM; if there are more than max_elements distinct values, then ENUM is not a suggested type.
- max_memory (default 8192) is the maximum amount of memory that analyse should allocate per column while trying to find all distinct values.

1.2.8.3.16 ROWNUM

MariaDB starting with 10.6.1

From MariaDB 10.6.1, the `ROWNUM()` function is supported.

Contents

Syntax

```
ROWNUM()
```

In Oracle mode one can just use `ROWNUM`, without the parentheses.

Description

`ROWNUM()` returns the current number of accepted rows in the current context. It main purpose is to emulate the `ROWNUM` pseudo column in Oracle. For MariaDB native applications, we recommend the usage of LIMIT, as it is easier to use and gives more predictable results than the usage of `ROWNUM()`.

The main difference between using `LIMIT` and `ROWNUM()` to limit the rows in the result is that `LIMIT` works on the result set while `ROWNUM` works on the number of accepted rows (before any `ORDER` or `GROUP BY` clauses).

The following queries will return the same results:

```
SELECT * from t1 LIMIT 10;
SELECT * from t1 WHERE ROWNUM() <= 10;
```

While the following may return different results based on in which orders the rows are found:

```
SELECT * from t1 ORDER BY a LIMIT 10;
SELECT * from t1 ORDER BY a WHERE ROWNUM() <= 10;
```

The recommended way to use `ROWNUM` to limit the number of returned rows and get predictable results is to have the query in a subquery and test for `ROWNUM()` in the outer query:

```
SELECT * FROM (select * from t1 ORDER BY a) WHERE ROWNUM() <= 10;
```

`ROWNUM()` can be used in the following contexts:

- SELECT
- INSERT
- UPDATE
- DELETE
- LOAD DATA INFILE

Used in other contexts, `ROWNUM()` will return 0.

Examples

```
INSERT INTO t1 VALUES (1,ROWNUM()),(2,ROWNUM()),(3,ROWNUM());

INSERT INTO t1 VALUES (1),(2) returning a, ROWNUM();

UPDATE t1 SET row_num_column=ROWNUM();

DELETE FROM t1 WHERE a < 10 AND ROWNUM() < 2;

LOAD DATA INFILE 'filename' into table t1 fields terminated by ','
  lines terminated by "\r\n" (a,b) set c=ROWNUM();
```

Optimizations

In many cases where `ROWNUM()` is used, MariaDB will use the same optimizations it uses with LIMIT.

`LIMIT` optimization is possible when using `ROWNUM` in the following manner:

- When one is in a top level `WHERE` clause comparing `ROWNUM()` with a numerical constant using any of the following expressions:
  - `ROWNUM()` < number
  - `ROWNUM()` <= number
  - `ROWNUM()` = 1 `ROWNUM()` can be also be the right argument to the comparison function.

In the above cases, `LIMIT` optimization can be done in the following cases:

- For the current sub query when the ROWNUM comparison is done on the top level:

```
SELECT * from t1 WHERE ROWNUM() <= 2 AND t1.a > 0
```

- For an inner sub query, when the upper level has only a `ROWNUM()` comparison in the `WHERE` clause:

```
SELECT * from (select * from t1) as t WHERE ROWNUM() <= 2
```

Other Changes Related to ROWNUM

When `ROWNUM()` is used anywhere in a query, the optimization to ignore `ORDER BY` in subqueries are disabled.

This was done to get the following common Oracle query to work as expected:

```
 select * from (select * from t1 order by a desc) as t where rownum() <= 2;
```

By default MariaDB ignores any `ORDER BY` in subqueries both because the SQL standard defines results sets in subqueries to be un-ordered and because of performance reasons (especially when using views in subqueries). See MDEV-3926 "Wrong result with GROUP BY ... WITH ROLLUP" for a discussion of this topic.

Other Considerations

While MariaDB tries to emulate Oracle's usage of `ROWNUM()` as closely as possible, there are cases where the result is different:

- When the optimizer finds rows in a different order (because of different storage methods or optimization). This may also happen in Oracle if one adds or deletes an index, in which case the rows may be found in a different order.

Note that usage of `ROWNUM()` in functions or stored procedures will use their own context, not the caller's context.

1.2.8.3.17 ROW_COUNT

Syntax

```
ROW_COUNT()
```

Description

ROW_COUNT() returns the number of rows updated, inserted or deleted by the preceding statement. This is the same as the row count that the mariadb client displays and the value from the mysql_affected_rows() C API function.

Generally:

- For statements which return a result set (such as SELECT, SHOW, DESC or HELP), returns -1, even when the result set is empty. This is also true for administrative statements, such as OPTIMIZE.
- For DML statements other than SELECT and for ALTER TABLE, returns the number of affected rows.
- For DDL statements (including TRUNCATE) and for other statements which don't return any result set (such as USE, DO, SIGNAL or DEALLOCATE PREPARE), returns 0.

For UPDATE, affected rows is by default the number of rows that were actually changed. If the CLIENT_FOUND_ROWS flag to mysql_real_connect() is specified when connecting to mysqld, affected rows is instead the number of rows matched by the WHERE clause.

For REPLACE, deleted rows are also counted. So, if REPLACE deletes a row and adds a new row, ROW_COUNT() returns 2.

For INSERT ... ON DUPLICATE KEY, updated rows are counted twice. So, if INSERT adds a new rows and modifies another row, ROW_COUNT() returns 3.

ROW_COUNT() does not take into account rows that are not directly deleted/updated by the last statement. This means that rows deleted by foreign keys or triggers are not counted.

Warning: You can use ROW_COUNT() with prepared statements, but you need to call it after EXECUTE, not after DEALLOCATE PREPARE, because the row count for allocate prepare is always 0.

Warning: When used after a CALL statement, this function returns the number of rows affected by the last statement in the procedure, not by the whole procedure.

Warning: After INSERT DELAYED, ROW_COUNT() returns the number of the rows you tried to insert, not the number of the successful writes.

This information can also be found in the diagnostics area.

Statements using the ROW_COUNT() function are not safe for statement-based replication.

Examples

```
CREATE TABLE t (A INT);

INSERT INTO t VALUES(1),(2),(3);

SELECT ROW_COUNT();
+-------------+
| ROW_COUNT() |
+-------------+
|           3 |
+-------------+

DELETE FROM t WHERE A IN(1,2);

SELECT ROW_COUNT();
+-------------+
| ROW_COUNT() |
+-------------+
|           2 |
+-------------+
```

Example with prepared statements:

```
SET @q = 'INSERT INTO t VALUES(1),(2),(3);';

PREPARE stmt FROM @q;

EXECUTE stmt;
Query OK, 3 rows affected (0.39 sec)
Records: 3  Duplicates: 0  Warnings: 0

SELECT ROW_COUNT();
+-------------+
| ROW_COUNT() |
+-------------+
|           3 |
+-------------+
```

1.2.8.3.18 SCHEMA

Syntax

```
SCHEMA()
```

Description

This function is a synonym for DATABASE().

1.2.8.3.19 SESSION_USER

Syntax

```
SESSION_USER()
```

Description

SESSION_USER() is a synonym for USER().

1.2.8.3.20 SYSTEM_USER

Syntax

```
SYSTEM_USER()
```

Description

SYSTEM_USER() is a synonym for USER().

1.2.8.3.21 USER

Syntax

```
USER()
```

Description

Returns the current MariaDB user name and host name, given when authenticating to MariaDB, as a string in the utf8 character set.

Note that the value of USER() may differ from the value of CURRENT_USER(), which is the user used to authenticate the current client. `CURRENT_ROLE()` returns the current active role.

`SYSTEM_USER()` and `SESSION_USER` are synonyms for `USER()` .

Statements using the `USER()` function or one of its synonyms are not safe for statement level replication.

Examples

```
shell> mysql --user="anonymous"

SELECT USER(),CURRENT_USER();
+---------------------+----------------+
| USER()              | CURRENT_USER() |
+---------------------+----------------+
| anonymous@localhost | @localhost     |
+---------------------+----------------+
```

To select only the IP address, use SUBSTRING_INDEX(),

```
SELECT SUBSTRING_INDEX(USER(), '@', -1);
+---------------------------------+
| SUBSTRING_INDEX(USER(), '@', -1) |
+---------------------------------+
| 192.168.0.101                   |
+---------------------------------+
```

1.2.8.3.22 VERSION

Syntax

```
VERSION()
```

Description

Returns a string that indicates the MariaDB server version. The string uses the utf8 character set.

Examples

```
SELECT VERSION();
+----------------+
| VERSION()      |
+----------------+
| 10.4.7-MariaDB |
+----------------+
```

The `VERSION()` string may have one or more of the following suffixes:

Suffix	Description
-embedded	The server is an embedded server (libmariadbd).
-log	General logging, slow logging or binary (replication) logging is enabled.
-debug	The server is compiled for debugging.
-valgrind	The server is compiled to be instrumented with valgrind.

Changing the Version String

Some old legacy code may break because they are parsing the `VERSION` string and expecting a MySQL string or a simple version string like Joomla til API17, see MDEV-7780.

One can fool these applications by setting the version string from the command line or the my.cnf files with --version=....

1.2.8.4 Miscellaneous Functions

Miscellaneous functions include DEFAULT, GET_LOCK, SLEEP, UUID, etc.

GET_LOCK
Obtain LOCK.

INET6_ATON
Given an IPv6 or IPv4 network address, returns a VARBINARY numeric value.

INET6_NTOA
Given an IPv6 or IPv4 network address, returns the address as a nonbinary string.

INET_ATON
Returns numeric value of IPv4 address.

INET_NTOA
Returns dotted-quad representation of IPv4 address.

IS_FREE_LOCK
Checks whether lock is free to use.

IS_IPV4
Whether or not an expression is a valid IPv4 address.

IS_IPV4_COMPAT
Whether or not an IPv6 address is IPv4-compatible.

IS_IPV4_MAPPED
Whether an IPv6 address is a valid IPv4-mapped address.

IS_IPV6
Whether or not an expression is a valid IPv6 address.

IS_USED_LOCK
Check if lock is in use.

MASTER_GTID_WAIT
Wait until slave reaches the GTID position.

MASTER_POS_WAIT
Blocks until the replica has applied all specified updates.

NAME_CONST
Returns the given value.

RELEASE_ALL_LOCKS
Releases all named locks held by the current session.

RELEASE_LOCK
Releases lock obtained with GET_LOCK().

SLEEP
Pauses for the given number of seconds.

SYS_GUID
Returns a globally unique identifier (GUID).

UUID
Returns a Universal Unique Identifier.

UUID_SHORT
Return short universal identifier.

VALUES / VALUE
Refer to columns in INSERT ... ON DUPLICATE KEY UPDATE.

There are 1 related questions.

1.2.8.4.1 GET_LOCK

Syntax

```
GET_LOCK(str,timeout)
```

Contents

Description

Tries to obtain a lock with a name given by the string `str`, using a timeout of `timeout` seconds. Returns `1` if the lock was obtained successfully, `0` if the attempt timed out (for example, because another client has previously locked the name), or `NULL` if an error occurred (such as running out of memory or the thread was killed with mariadb-admin kill).

A lock is released with RELEASE_LOCK(), when the connection terminates (either normally or abnormally). A connection can hold multiple locks at the same time, so a lock that is no longer needed needs to be explicitly released.

The IS_FREE_LOCK function returns whether a specified lock a free or not, and the IS_USED_LOCK whether the function is in use or not.

Locks obtained with `GET_LOCK()` do not interact with transactions. That is, committing a transaction does not release any such locks obtained during the transaction.

It is also possible to recursively set the same lock. If a lock with the same name is set `n` times, it needs to be released `n` times as well.

`str` is case insensitive for `GET_LOCK()` and related functions. If `str` is an empty string or `NULL`, `GET_LOCK()` returns `NULL` and does nothing. `timeout` supports microseconds.

If the metadata_lock_info plugin is installed, locks acquired with this function are visible in the Information Schema METADATA_LOCK_INFO table.

This function can be used to implement application locks or to simulate record locks. Names are locked on a server-wide basis. If a name has been locked by one client, `GET_LOCK()` blocks any request by another client for a lock with the same name. This allows clients that agree on a given lock name to use the name to perform cooperative advisory locking. But be aware that it also allows a client that is not among the set of cooperating clients to lock a name, either inadvertently or deliberately, and thus prevent any of the cooperating clients from locking that name. One way to reduce the likelihood of this is to use lock names that are database-specific or application-specific. For example, use lock names of the form `db_name.str` or `app_name.str`.

Statements using the `GET_LOCK` function are not safe for statement-based replication.

The patch to permit multiple locks was contributed by Konstantin "Kostja" Osipov (MDEV-3917).

Examples

```
SELECT GET_LOCK('lock1',10);
+----------------------+
| GET_LOCK('lock1',10) |
+----------------------+
|                    1 |
+----------------------+

SELECT IS_FREE_LOCK('lock1'), IS_USED_LOCK('lock1');
+----------------------+----------------------+
| IS_FREE_LOCK('lock1') | IS_USED_LOCK('lock1') |
+----------------------+----------------------+
|                    0 |                   46 |
+----------------------+----------------------+

SELECT IS_FREE_LOCK('lock2'), IS_USED_LOCK('lock2');
+----------------------+----------------------+
| IS_FREE_LOCK('lock2') | IS_USED_LOCK('lock2') |
+----------------------+----------------------+
|                    1 |                 NULL |
+----------------------+----------------------+
```

Multiple locks can be held:

```
SELECT GET_LOCK('lock2',10);
+----------------------+
| GET_LOCK('lock2',10) |
+----------------------+
|                    1 |
+----------------------+

SELECT IS_FREE_LOCK('lock1'), IS_FREE_LOCK('lock2');
+----------------------+----------------------+
| IS_FREE_LOCK('lock1') | IS_FREE_LOCK('lock2') |
+----------------------+----------------------+
|                    0 |                    0 |
+----------------------+----------------------+

SELECT RELEASE_LOCK('lock1'), RELEASE_LOCK('lock2');
+----------------------+----------------------+
| RELEASE_LOCK('lock1') | RELEASE_LOCK('lock2') |
+----------------------+----------------------+
|                    1 |                    1 |
+----------------------+----------------------+
```

It is possible to hold the same lock recursively. This example is viewed using the metadata_lock_info plugin:

```
SELECT GET_LOCK('lock3',10);
+----------------------+
| GET_LOCK('lock3',10) |
+----------------------+
|                    1 |
+----------------------+

SELECT GET_LOCK('lock3',10);
+----------------------+
| GET_LOCK('lock3',10) |
+----------------------+
|                    1 |
+----------------------+

SELECT * FROM INFORMATION_SCHEMA.METADATA_LOCK_INFO;
+-----------+---------------------+---------------+-----------+--------------+------------+
| THREAD_ID | LOCK_MODE           | LOCK_DURATION | LOCK_TYPE | TABLE_SCHEMA | TABLE_NAME |
+-----------+---------------------+---------------+-----------+--------------+------------+
|        46 | MDL_SHARED_NO_WRITE | NULL          | User lock | lock3        |            |
+-----------+---------------------+---------------+-----------+--------------+------------+

SELECT RELEASE_LOCK('lock3');
+----------------------+
| RELEASE_LOCK('lock3') |
+----------------------+
|                    1 |
+----------------------+

SELECT * FROM INFORMATION_SCHEMA.METADATA_LOCK_INFO;
+-----------+---------------------+---------------+-----------+--------------+------------+
| THREAD_ID | LOCK_MODE           | LOCK_DURATION | LOCK_TYPE | TABLE_SCHEMA | TABLE_NAME |
+-----------+---------------------+---------------+-----------+--------------+------------+
|        46 | MDL_SHARED_NO_WRITE | NULL          | User lock | lock3        |            |
+-----------+---------------------+---------------+-----------+--------------+------------+

SELECT RELEASE_LOCK('lock3');
+----------------------+
| RELEASE_LOCK('lock3') |
+----------------------+
|                    1 |
+----------------------+

SELECT * FROM INFORMATION_SCHEMA.METADATA_LOCK_INFO;
Empty set (0.000 sec)
```

Timeout example: Connection 1:

```
SELECT GET_LOCK('lock4',10);
+----------------------+
| GET_LOCK('lock4',10) |
+----------------------+
|                    1 |
+----------------------+
```

Connection 2:

```
SELECT GET_LOCK('lock4',10);
```

After 10 seconds...

```
+----------------------+
| GET_LOCK('lock4',10) |
+----------------------+
|                    0 |
+----------------------+
```

Deadlocks are automatically detected and resolved. Connection 1:

```
SELECT GET_LOCK('lock5',10);
+----------------------+
| GET_LOCK('lock5',10) |
+----------------------+
|                    1 |
+----------------------+
```

Connection 2:

```
SELECT GET_LOCK('lock6',10);
+----------------------+
| GET_LOCK('lock6',10) |
+----------------------+
|                    1 |
+----------------------+
```

Connection 1:

```
SELECT GET_LOCK('lock6',10);
+----------------------+
| GET_LOCK('lock6',10) |
+----------------------+
|                    0 |
+----------------------+
```

Connection 2:

```
SELECT GET_LOCK('lock5',10);
ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting transaction
```

1.2.8.4.2 INET6_ATON

Syntax

```
INET6_ATON(expr)
```

Description

Given an IPv6 or IPv4 network address as a string, returns a binary string that represents the numeric value of the address.

No trailing zone ID's or traling network masks are permitted. For IPv4 addresses, or IPv6 addresses with IPv4 address

parts, no classful addresses or trailing port numbers are permitted and octal numbers are not supported.

The returned binary string will be VARBINARY(16) or VARBINARY(4) for IPv6 and IPv4 addresses respectively.

Returns NULL if the argument is not understood.

MariaDB starting with 10.5.0

From MariaDB 10.5.0, `INET6_ATON` can take INET6 as an argument.

Examples

```
SELECT HEX(INET6_ATON('10.0.1.1'));
+----------------------------+
| HEX(INET6_ATON('10.0.1.1')) |
+----------------------------+
| 0A000101                   |
+----------------------------+

SELECT HEX(INET6_ATON('48f3::d432:1431:ba23:846f'));
+--------------------------------------------+
| HEX(INET6_ATON('48f3::d432:1431:ba23:846f')) |
+--------------------------------------------+
| 48F3000000000000D4321431BA23846F           |
+--------------------------------------------+
```

1.2.8.4.3 INET6_NTOA

Contents

1. Syntax
2. Description
3. Examples

Syntax

```
INET6_NTOA(expr)
```

Description

Given an IPv6 or IPv4 network address as a numeric binary string, returns the address as a nonbinary string in the connection character set.

The return string is lowercase, and is platform independent, since it does not use functions specific to the operating system. It has a maximum length of 39 characters.

Returns NULL if the argument is not understood.

Examples

```
SELECT INET6_NTOA(UNHEX('0A000101'));
+-------------------------------+
| INET6_NTOA(UNHEX('0A000101')) |
+-------------------------------+
| 10.0.1.1                      |
+-------------------------------+

SELECT INET6_NTOA(UNHEX('48F3000000000000D4321431BA23846F'));
+-----------------------------------------------------+
| INET6_NTOA(UNHEX('48F3000000000000D4321431BA23846F')) |
+-----------------------------------------------------+
| 48f3::d432:1431:ba23:846f                           |
+-----------------------------------------------------+
```

1.2.8.4.4 INET_ATON

Syntax

```
INET_ATON(expr)
```

Description

Given the dotted-quad representation of an IPv4 network address as a string, returns an integer that represents the numeric value of the address. Addresses may be 4- or 8-byte addresses.

Returns NULL if the argument is not understood.

Examples

```
SELECT INET_ATON('192.168.1.1');
+--------------------------+
| INET_ATON('192.168.1.1') |
+--------------------------+
|               3232235777 |
+--------------------------+
```

This is calculated as follows: 192 x 256^3 + 168 x 256^2 + 1 x 256 + 1

1.2.8.4.5 INET_NTOA

Syntax

```
INET_NTOA(expr)
```

Description

Given a numeric IPv4 network address in network byte order (4 or 8 byte), returns the dotted-quad representation of the address as a string.

Examples

```
SELECT INET_NTOA(3232235777);
+-----------------------+
| INET_NTOA(3232235777) |
+-----------------------+
| 192.168.1.1           |
+-----------------------+
```

192.168.1.1 corresponds to 3232235777 since 192 x 256^3 + 168 x 256^2 + 1 x 256 + 1 = 3232235777

1.2.8.4.6 IS_FREE_LOCK

Syntax

```
IS_FREE_LOCK(str)
```

Description

Checks whether the lock named `str` is free to use (that is, not locked). Returns `1` if the lock is free (no one is using the lock), `0` if the lock is in use, and `NULL` if an error occurs (such as an incorrect argument, like an empty string or `NULL`). `str` is case insensitive.

If the metadata_lock_info plugin is installed, the Information Schema metadata_lock_info table contains information about locks of this kind (as well as metadata locks).

Statements using the `IS_FREE_LOCK` function are not safe for statement-based replication.

1.2.8.4.7 IS_IPV4

Syntax

```
IS_IPV4(expr)
```

Description

If the expression is a valid IPv4 address, returns 1, otherwise returns 0.

IS_IPV4() is stricter than INET_ATON(), but as strict as INET6_ATON(), in determining the validity of an IPv4 address. This implies that if IS_IPV4 returns 1, the same expression will always return a non-NULL result when passed to INET_ATON(), but that the reverse may not apply.

Examples

```
SELECT IS_IPV4('1110.0.1.1');
+-----------------------+
| IS_IPV4('1110.0.1.1') |
+-----------------------+
|                     0 |
+-----------------------+

SELECT IS_IPV4('48f3::d432:1431:ba23:846f');
+--------------------------------------+
| IS_IPV4('48f3::d432:1431:ba23:846f') |
+--------------------------------------+
|                                    0 |
+--------------------------------------+
```

1.2.8.4.8 IS_IPV4_COMPAT

Syntax

```
IS_IPV4_COMPAT(expr)
```

Description

Returns 1 if a given numeric binary string IPv6 address, such as returned by INET6_ATON(), is IPv4-compatible, otherwise returns 0.

MariaDB starting with 10.5.0

From MariaDB 10.5.0, when the argument is not INET6, automatic implicit CAST to INET6 is applied. As a consequence, `IS_IPV4_COMPAT` now understands arguments in both text representation and binary(16)

representation. Before MariaDB 10.5.0, the function understood only binary(16) representation.

Examples

```
SELECT IS_IPV4_COMPAT(INET6_ATON('::10.0.1.1'));
+---------------------------------------+
| IS_IPV4_COMPAT(INET6_ATON('::10.0.1.1')) |
+---------------------------------------+
|                                     1 |
+---------------------------------------+

SELECT IS_IPV4_COMPAT(INET6_ATON('::48f3::d432:1431:ba23:846f'));
+-----------------------------------------------------------+
| IS_IPV4_COMPAT(INET6_ATON('::48f3::d432:1431:ba23:846f')) |
+-----------------------------------------------------------+
|                                                         0 |
+-----------------------------------------------------------+
```

1.2.8.4.9 IS_IPV4_MAPPED

Syntax

```
IS_IPV4_MAPPED(expr)
```

Description

Returns 1 if a given a numeric binary string IPv6 address, such as returned by INET6_ATON(), is a valid IPv4-mapped address, otherwise returns 0.

MariaDB starting with 10.5.0

From MariaDB 10.5.0, when the argument is not INET6, automatic implicit CAST to INET6 is applied. As a consequence, `IS_IPV4_MAPPED` now understands arguments in both text representation and binary(16) representation. Before MariaDB 10.5.0, the function understood only binary(16) representation.

Examples

```
SELECT IS_IPV4_MAPPED(INET6_ATON('::10.0.1.1'));
+---------------------------------------+
| IS_IPV4_MAPPED(INET6_ATON('::10.0.1.1')) |
+---------------------------------------+
|                                     0 |
+---------------------------------------+

SELECT IS_IPV4_MAPPED(INET6_ATON('::ffff:10.0.1.1'));
+-----------------------------------------------+
| IS_IPV4_MAPPED(INET6_ATON('::ffff:10.0.1.1')) |
+-----------------------------------------------+
|                                             1 |
+-----------------------------------------------+
```

1.2.8.4.10 IS_IPV6

Syntax

```
IS_IPV6(expr)
```

Description

Returns 1 if the expression is a valid IPv6 address specified as a string, otherwise returns 0. Does not consider IPv4 addresses to be valid IPv6 addresses.

Examples

```
 SELECT IS_IPV6('48f3::d432:1431:ba23:846f');
+--------------------------------------+
| IS_IPV6('48f3::d432:1431:ba23:846f') |
+--------------------------------------+
|                                    1 |
+--------------------------------------+
1 row in set (0.02 sec)

SELECT IS_IPV6('10.0.1.1');
+----------------------+
| IS_IPV6('10.0.1.1') |
+----------------------+
|                    0 |
+----------------------+
```

1.2.8.4.11 IS_USED_LOCK

Syntax

```
IS_USED_LOCK(str)
```

Contents

Description

Checks whether the lock named `str` is in use (that is, locked). If so, it returns the connection identifier of the client that holds the lock. Otherwise, it returns `NULL`. `str` is case insensitive.

If the metadata_lock_info plugin is installed, the Information Schema metadata_lock_info table contains information about locks of this kind (as well as metadata locks).

Statements using the `IS_USED_LOCK` function are not safe for statement-based replication.

1.2.8.4.12 MASTER_GTID_WAIT

Syntax

```
MASTER_GTID_WAIT(gtid-list[, timeout)
```

Description

This function takes a string containing a comma-separated list of global transaction id's (similar to the value of, for example, gtid_binlog_pos). It waits until the value of gtid_slave_pos has the same or higher seq_no within all replication domains specified in the gtid-list; in other words, it waits until the slave has reached the specified GTID position.

An optional second argument gives a timeout in seconds. If the timeout expires before the specified GTID position is reached, then the function returns -1. Passing NULL or a negative number for the timeout means no timeout, and the function will wait indefinitely.

If the wait completes without a timeout, 0 is returned. Passing NULL for the gtid-list makes the function return NULL immediately, without waiting.

The gtid-list may be the empty string, in which case MASTER_GTID_WAIT() returns immediately. If the gtid-list contains fewer domains than gtid_slave_pos, then only those domains are waited upon. If gtid-list contains a domain that is not present in @@gtid_slave_pos, then MASTER_GTID_WAIT() will wait until an event containing such domain_id arrives on the slave (or until timed out or killed).

MASTER_GTID_WAIT() can be useful to ensure that a slave has caught up to a master. Simply take the value of gtid_binlog_pos on the master, and use it in a MASTER_GTID_WAIT() call on the slave; when the call completes, the slave will have caught up with that master position.

MASTER_GTID_WAIT() can also be used in client applications together with the last_gtid session variable. This is useful in a read-scaleout replication setup, where the application writes to a single master but divides the reads out to a number of slaves to distribute the load. In such a setup, there is a risk that an application could first do an update on the master, and then a bit later do a read on a slave, and if the slave is not fast enough, the data read from the slave might not include the update just made, possibly confusing the application and/or the end-user. One way to avoid this is to request the value of last_gtid on the master just after the update. Then before doing the read on the slave, do a MASTER_GTID_WAIT() on the value obtained from the master; this will ensure that the read is not performed until the slave has replicated sufficiently far for the update to have become visible.

Note that MASTER_GTID_WAIT() can be used even if the slave is configured not to use GTID for connections (CHANGE MASTER TO master_use_gtid=no). This is because from MariaDB 10, GTIDs are always logged on the master server, and always recorded on the slave servers.

Differences to MASTER_POS_WAIT()

- MASTER_GTID_WAIT() is global; it waits for any master connection to reach the specified GTID position. MASTER_POS_WAIT() works only against a specific connection. This also means that while MASTER_POS_WAIT() aborts if its master connection is terminated with STOP SLAVE or due to an error, MASTER_GTID_WAIT() continues to wait while slaves are stopped.
- MASTER_GTID_WAIT() can take its timeout as a floating-point value, so a timeout in fractional seconds is supported, eg. MASTER_GTID_WAIT("0-1-100", 0.5). (The minimum wait is one microsecond, 0.000001 seconds).
- MASTER_GTID_WAIT() allows one to specify a timeout of zero in order to do a non-blocking check to see if the slaves have progressed to a specific GTID position (MASTER_POS_WAIT() takes a zero timeout as meaning an infinite wait). To do an infinite MASTER_GTID_WAIT(), specify a negative timeout, or omit the timeout argument.
- MASTER_GTID_WAIT() does not return the number of events executed since the wait started, nor does it return NULL if a slave thread is stopped. It always returns either 0 for successful wait completed, or -1 for timeout reached (or NULL if the specified gtid-pos is NULL).

Since MASTER_GTID_WAIT() looks only at the seq_no part of the GTIDs, not the server_id, care is needed if a slave becomes diverged from another server so that two different GTIDs with the same seq_no (in the same domain) arrive at the same server. This situation is in any case best avoided; setting gtid_strict_mode is recommended, as this will prevent any such out-of-order sequence numbers from ever being replicated on a slave.

1.2.8.4.13 MASTER_POS_WAIT

Syntax

```
MASTER_POS_WAIT(log_name,log_pos[,timeout,["connection_name"]])
```

Description

This function is useful in replication for controlling primary/replica synchronization. It blocks until the replica has read and applied all updates up to the specified position (`log_name,log_pos`) in the primary log. The return value is the number of log events the replica had to wait for to advance to the specified position. The function returns NULL if the replica SQL thread is not started, the replica's primary information is not initialized, the arguments are incorrect, or an error occurs. It returns -1 if the timeout has been exceeded. If the replica SQL thread stops while `MASTER_POS_WAIT()` is waiting, the function returns NULL. If the replica is past the specified position, the function returns immediately.

If a `timeout` value is specified, `MASTER_POS_WAIT()` stops waiting when `timeout` seconds have elapsed. `timeout` must be greater than 0; a zero or negative `timeout` means no `timeout` .

The `connection_name` is used when you are using multi-source-replication. If you don't specify it, it's set to the value of the default_master_connection system variable.

Statements using the MASTER_POS_WAIT() function are not safe for statement-based replication.

1.2.8.4.14 NAME_CONST

Syntax

```
NAME_CONST(name,value)
```

Description

Returns the given value. When used to produce a result set column, `NAME_CONST()` causes the column to have the given name. The arguments should be constants.

This function is used internally when replicating stored procedures. It makes little sense to use it explicitly in SQL statements, and it was not supposed to be used like that.

```
SELECT NAME_CONST('myname', 14);
+--------+
| myname |
+--------+
|     14 |
+--------+
```

1.2.8.4.15 RELEASE_ALL_LOCKS

MariaDB until 10.5.2

RELEASE_ALL_LOCKS was added in MariaDB 10.5.2.

Syntax

```
RELEASE_ALL_LOCKS()
```

Contents

Description

Releases all named locks held by the current session. Returns the number of locks released, or `0` if none were held.

Statements using the `RELEASE_ALL_LOCKS` function are not safe for statement-based replication.

Examples

```
SELECT RELEASE_ALL_LOCKS();
+---------------------+
| RELEASE_ALL_LOCKS() |
+---------------------+
|                   0 |
+---------------------+

SELECT GET_LOCK('lock1',10);
+----------------------+
| GET_LOCK('lock1',10) |
+----------------------+
|                    1 |
+----------------------+

SELECT RELEASE_ALL_LOCKS();
+---------------------+
| RELEASE_ALL_LOCKS() |
+---------------------+
|                   1 |
+---------------------+
```

1.2.8.4.16 RELEASE_LOCK

Syntax

```
RELEASE_LOCK(str)
```

Contents

Description

Releases the lock named by the string `str` that was obtained with GET_LOCK(). Returns 1 if the lock was released, 0 if the lock was not established by this thread (in which case the lock is not released), and `NULL` if the named lock did not exist. The lock does not exist if it was never obtained by a call to `GET_LOCK()` or if it has previously been released.

`str` is case insensitive. If `str` is an empty string or `NULL`, `RELEASE_LOCK()` returns `NULL` and does nothing.

Statements using the `RELEASE_LOCK` function are not safe for statement-based replication.

The DO statement is convenient to use with `RELEASE_LOCK()`.

Examples

Connection1:

```
SELECT GET_LOCK('lock1',10);
+----------------------+
| GET_LOCK('lock1',10) |
+----------------------+
|                    1 |
+----------------------+
```

Connection 2:

```
SELECT GET_LOCK('lock2',10);
+----------------------+
| GET_LOCK('lock2',10) |
+----------------------+
|                    1 |
+----------------------+
```

Connection 1:

```
SELECT RELEASE_LOCK('lock1'), RELEASE_LOCK('lock2'), RELEASE_LOCK('lock3');
+----------------------+----------------------+----------------------+
| RELEASE_LOCK('lock1') | RELEASE_LOCK('lock2') | RELEASE_LOCK('lock3') |
+----------------------+----------------------+----------------------+
|                    1 |                    0 |                 NULL |
+----------------------+----------------------+----------------------+
```

It is possible to hold the same lock recursively. This example is viewed using the metadata_lock_info plugin:

```
SELECT GET_LOCK('lock3',10);
+---------------------+
| GET_LOCK('lock3',10) |
+---------------------+
|                   1 |
+---------------------+

SELECT GET_LOCK('lock3',10);
+---------------------+
| GET_LOCK('lock3',10) |
+---------------------+
|                   1 |
+---------------------+

SELECT * FROM INFORMATION_SCHEMA.METADATA_LOCK_INFO;
+-----------+---------------------+---------------+-----------+--------------+------------+
| THREAD_ID | LOCK_MODE           | LOCK_DURATION | LOCK_TYPE | TABLE_SCHEMA | TABLE_NAME |
+-----------+---------------------+---------------+-----------+--------------+------------+
|        46 | MDL_SHARED_NO_WRITE | NULL          | User lock | lock3        |            |
+-----------+---------------------+---------------+-----------+--------------+------------+

SELECT RELEASE_LOCK('lock3');
+----------------------+
| RELEASE_LOCK('lock3') |
+----------------------+
|                    1 |
+----------------------+

SELECT * FROM INFORMATION_SCHEMA.METADATA_LOCK_INFO;
+-----------+---------------------+---------------+-----------+--------------+------------+
| THREAD_ID | LOCK_MODE           | LOCK_DURATION | LOCK_TYPE | TABLE_SCHEMA | TABLE_NAME |
+-----------+---------------------+---------------+-----------+--------------+------------+
|        46 | MDL_SHARED_NO_WRITE | NULL          | User lock | lock3        |            |
+-----------+---------------------+---------------+-----------+--------------+------------+

SELECT RELEASE_LOCK('lock3');
+----------------------+
| RELEASE_LOCK('lock3') |
+----------------------+
|                    1 |
+----------------------+

SELECT * FROM INFORMATION_SCHEMA.METADATA_LOCK_INFO;
Empty set (0.000 sec)
```

1.2.8.4.17 SLEEP

Syntax

```
SLEEP(duration)
```

Description

Sleeps (pauses) for the number of seconds given by the duration argument, then returns `0`. If `SLEEP()` is interrupted, it returns `1`. The duration may have a fractional part given in microseconds.

Statements using the SLEEP() function are not safe for statement-based replication.

Example

```
SELECT SLEEP(5.5);
+------------+
| SLEEP(5.5) |
+------------+
|          0 |
+------------+
1 row in set (5.50 sec)
```

1.2.8.4.18 SYS_GUID

MariaDB starting with 10.6.1

The SYS_GUID function was introduced in MariaDB 10.6.1 to enhance Oracle compatibility. Similar functionality can be achieved with the UUID function.

Syntax

```
SYS_GUID()
```

Description

Returns a 16-byte globally unique identifier (GUID), similar to the UUID function, but without the `-` character.

Example

```
SELECT SYS_GUID();
+----------------------------------+
| SYS_GUID()                       |
+----------------------------------+
| 2C574E45BA2811EBB265F859713E4BE4 |
+----------------------------------+
```

1.2.8.4.19 UUID

Syntax

```
UUID()
```

Contents

Description

Returns a Universally Unique Identifier (UUID).

A UUID is designed as a number that is globally unique in space and time. Two calls to `UUID()` are expected to generate two different values, even if these calls are performed on two separate computers that are not connected to each other.

UUID() results are intended to be unique, but cannot always be relied upon to be unpredictable and unguessable.

A UUID is a 128-bit number represented by a utf8 string of five hexadecimal numbers in `aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee` format:

- The first three numbers are generated from a timestamp.
- The fourth number preserves temporal uniqueness in case the timestamp value loses monotonicity (for example, due to daylight saving time).
- The fifth number is an IEEE 802 node number that provides spatial uniqueness. A random number is substituted if the latter is not available (for example, because the host computer has no Ethernet card, or we do not know how to find the hardware address of an interface on your operating system). In this case, spatial uniqueness cannot be guaranteed. Nevertheless, a collision should have very low probability.

Currently, the MAC address of an interface is taken into account only on FreeBSD and Linux. On other operating systems, MariaDB uses a randomly generated 48-bit number.

Statements using the UUID() function are not safe for statement-based replication.

The results are generated according to the "DCE 1.1:Remote Procedure Call" (Appendix A) CAE (Common Applications Environment) Specifications published by The Open Group in October 1997 (Document Number C706).

Examples

```
SELECT UUID();
+--------------------------------------+
| UUID()                               |
+--------------------------------------+
| cd41294a-afb0-11df-bc9b-00241dd75637 |
+--------------------------------------+
```

1.2.8.4.20 UUID_SHORT

Syntax

```
UUID_SHORT()
```

Contents

Description

Returns a "short" universally unique identifier as a 64-bit unsigned integer (rather than a string-form 128-bit identifier as returned by the UUID() function).

The value of `UUID_SHORT()` is guaranteed to be unique if the following conditions hold:

- The server_id of the current host is unique among your set of master and slave servers
- `server_id` is between 0 and 255
- You don't set back your system time for your server between mysqld restarts
- You do not invoke `UUID_SHORT()` on average more than 16 million times per second between mysqld restarts

The UUID_SHORT() return value is constructed this way:

```
  (server_id & 255) << 56
+ (server_startup_time_in_seconds << 24)
+ incremented_variable++;
```

Statements using the UUID_SHORT() function are not safe for statement-based replication.

Examples

```
SELECT UUID_SHORT();
+-------------------+
| UUID_SHORT()      |
+-------------------+
| 21517162376069120 |
+-------------------+
```

```
create table t1 (a bigint unsigned default(uuid_short()) primary key);
insert into t1 values(),();
select * from t1;
+-------------------+
| a                 |
+-------------------+
| 98113699159474176 |
| 98113699159474177 |
+-------------------+
```

1.2.8.4.21 VALUES / VALUE

Syntax

MariaDB starting with 10.3.3

```
VALUE(col_name)
```

MariaDB until 10.3.2

```
VALUES(col_name)
```

Description

In an INSERT ... ON DUPLICATE KEY UPDATE statement, you can use the `VALUES(col_name)` function in the UPDATE clause to refer to column values from the INSERT portion of the statement. In other words, `VALUES(col_name)` in the `UPDATE` clause refers to the value of col_name that would be inserted, had no duplicate-key conflict occurred. This function is especially useful in multiple-row inserts.

The `VALUES()` function is meaningful only in `INSERT ... ON DUPLICATE KEY UPDATE` statements and returns `NULL` otherwise.

In MariaDB 10.3.3 this function was renamed to `VALUE()`, because it's incompatible with the standard Table Value Constructors syntax, implemented in MariaDB 10.3.3.

The `VALUES()` function can still be used even from MariaDB 10.3.3, but only in `INSERT ... ON DUPLICATE KEY UPDATE` statements; it's a syntax error otherwise.

Examples

MariaDB starting with 10.3.3

```
INSERT INTO t (a,b,c) VALUES (1,2,3),(4,5,6)
    ON DUPLICATE KEY UPDATE c=VALUE(a)+VALUE(b);
```

MariaDB until 10.3.2

```
INSERT INTO t (a,b,c) VALUES (1,2,3),(4,5,6)
    ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);
```

1.2.9 Special Functions

There are many commonly used built-in functions. These are lesser used function for specific needs.

Dynamic Columns Functions
Functions for storing key/value pairs of data within a column.

Galera Functions
Built-in functions related to Galera.

Geographic Functions
Geographic, as well as geometric functions.

JSON Functions
Built-in functions related to JSON.

SEQUENCE Functions
Functions that can be used on SEQUENCEs.

Spider Functions
User-defined functions available with the Spider storage engine.

Window Functions
Window functions for performing calculations on a set of rows related to the current row.

1.2.2.2 Dynamic Columns Functions

1.2.9.2 Galera Functions

The following functions are for use with Galera.

WSREP_LAST_SEEN_GTID
Returns the Global Transaction ID of the most recent write transaction observed by the client.

WSREP_LAST_WRITTEN_GTID
Returns the Global Transaction ID of the most recent write transaction performed by the client.

WSREP_SYNC_WAIT_UPTO_GTID
Blocks the client until the transaction specified by the given GTID is applied and committed.

1.2.9.2.1 WSREP_LAST_SEEN_GTID

MariaDB starting with 10.4.2
WSREP_LAST_SEEN_GTID was added as part of Galera 4 in MariaDB 10.4.2.

Syntax

```
WSREP_LAST_SEEN_GTID()
```

Description

Returns the Global Transaction ID of the most recent write transaction observed by the client.

The result can be useful to determine the transaction to provide to WSREP_SYNC_WAIT_UPTO_GTID for waiting and unblocking purposes.

1.2.9.2.2 WSREP_LAST_WRITTEN_GTID

MariaDB starting with 10.4.2

MariaDB starting with 10.4.2
WSREP_LAST_WRITTEN_GTID was added as part of Galera 4 in MariaDB 10.4.2.

Syntax

```
WSREP_LAST_WRITTEN_GTID()
```

Description

Returns the Global Transaction ID of the most recent write transaction performed by the client.

1.2.9.2.3 WSREP_SYNC_WAIT_UPTO_GTID

MariaDB starting with 10.4.2
WSREP_SYNC_WAIT_UPTO_GTID was added as part of Galera 4 in MariaDB 10.4.2.

Syntax

```
WSREP_SYNC_WAIT_UPTO_GTID(gtid[,timeout])
```

Description

Blocks the client until the transaction specified by the given Global Transaction ID is applied and committed by the node.

The optional *timeout* argument can be used to specify a block timeout in seconds. If not provided, the timeout will be indefinite.

Returns the node that applied and committed the Global Transaction ID, `ER_LOCAL_WAIT_TIMEOUT` if the function is timed out before this, or `ER_WRONG_ARGUMENTS` if the function is given an invalid GTID.

The result from WSREP_LAST_SEEN_GTID can be useful to determine the transaction to provide to WSREP_SYNC_WAIT_UPTO_GTID for waiting and unblocking purposes.

1.2.9.3 Geographic Functions

Geographic and geometry functions. See Geographic Features for a full discussion of MariaDB's spatial extensions.

Geometry Constructors
Geometry constructors

Geometry Properties
Geometry properties

Geometry Relations
Geometry relations

LineString Properties
LineString properties

MBR (Minimum Bounding Rectangle)

Point Properties
Point properties

Polygon Properties
Polygon properties

WKB
Well-Known Binary format for geometric data

WKT
Well-Known Text geometry representation

There are 2 related questions.

1.2.9.3.1 Geometry Constructors

Geometry constructors

BUFFER
Synonym for ST_BUFFER.

CONVEXHULL
Synonym for ST_CONVEXHULL.

GEOMETRYCOLLECTION
Constructs a WKB GeometryCollection.

LINESTRING
Constructs a WKB LineString value from a number of WKB Point arguments.

MULTILINESTRING
Constructs a MultiLineString value.

MULTIPOINT
Constructs a WKB MultiPoint value.

MULTIPOLYGON
Constructs a WKB MultiPolygon.

POINT
Constructs a WKB Point.

PointOnSurface
Synonym for ST_PointOnSurface.

POLYGON
Constructs a WKB Polygon value from a number of WKB LineString arguments.

ST_BUFFER
A new geometry with a buffer added to the original geometry.

ST_CONVEXHULL
The minimum convex geometry enclosing all geometries within the set.

ST_INTERSECTION
The intersection, or shared portion, of two geometries.

ST_POINTONSURFACE
Returns a POINT guaranteed to intersect a surface.

ST_SYMDIFFERENCE
Portions of two geometries that don't intersect.

ST_UNION
Union of two geometries.

1.2.9.3.1.1 BUFFER

A synonym for ST_BUFFER.

1.2.9.3.1.2 CONVEXHULL

A synonym for ST_CONVEXHULL.

1.2.9.3.1.3 GEOMETRYCOLLECTION

Syntax

```
GeometryCollection(g1,g2,...)
```

Description

Constructs a WKB GeometryCollection. If any argument is not a well-formed WKB representation of a geometry, the return value is `NULL` .

Examples

```
CREATE TABLE gis_geometrycollection  (g GEOMETRYCOLLECTION);
SHOW FIELDS FROM gis_geometrycollection;
INSERT INTO gis_geometrycollection VALUES
    (GeomCollFromText('GEOMETRYCOLLECTION(POINT(0 0), LINESTRING(0 0,10 10))')),
    (GeometryFromWKB(AsWKB(GeometryCollection(Point(44, 6), LineString(Point(3, 6), Point(7, 9))))),
    (GeomFromText('GeometryCollection()')),
    (GeomFromText('GeometryCollection EMPTY'));
```

1.2.9.3.1.4 LINESTRING

Syntax

```
LineString(pt1,pt2,...)
```

Description

Constructs a WKB LineString value from a number of WKB Point arguments. If any argument is not a WKB Point, the return value is `NULL` . If the number of Point arguments is less than two, the return value is `NULL` .

Examples

```
SET @ls = 'LineString(1 1,2 2,3 3)';

SELECT AsText(EndPoint(GeomFromText(@ls)));
+-----------------------------------+
| AsText(EndPoint(GeomFromText(@ls))) |
+-----------------------------------+
| POINT(3 3)                        |
+-----------------------------------+

CREATE TABLE gis_line  (g LINESTRING);
INSERT INTO gis_line VALUES
    (LineFromText('LINESTRING(0 0,0 10,10 0)')),
    (LineStringFromText('LINESTRING(10 10,20 10,20 20,10 20,10 10)')),
    (LineStringFromWKB(AsWKB(LineString(Point(10, 10), Point(40, 10)))));
```

1.2.9.3.1.5 MULTILINESTRING

Syntax

```
MultiLineString(ls1,ls2,...)
```

Description

Constructs a WKB MultiLineString value using WKB LineString arguments. If any argument is not a WKB LineString, the return value is `NULL` .

Example

```
CREATE TABLE gis_multi_line (g MULTILINESTRING);
INSERT INTO gis_multi_line VALUES
 (MultiLineStringFromText('MULTILINESTRING((10 48,10 21,10 0),(16 0,16 23,16 48))')),
 (MLineFromText('MULTILINESTRING((10 48,10 21,10 0))')),
 (MLineFromWKB(AsWKB(MultiLineString(LineString(Point(1, 2),
    Point(3, 5)), LineString(Point(2, 5),Point(5, 8),Point(21, 7))))));
```

1.2.9.3.1.6 MULTIPOINT

Syntax

```
MultiPoint(pt1,pt2,...)
```

Description

Constructs a WKB MultiPoint value using WKB Point arguments. If any argument is not a WKB Point, the return value is `NULL` .

Examples

```
SET @g = ST_GEOMFROMTEXT('MultiPoint( 1 1, 2 2, 5 3, 7 2, 9 3, 8 4, 6 6, 6 9, 4 9, 1 5 )');

CREATE TABLE gis_multi_point (g MULTIPOINT);
INSERT INTO gis_multi_point VALUES
    (MultiPointFromText('MULTIPOINT(0 0,10 10,10 20,20 20)')),
    (MPointFromText('MULTIPOINT(1 1,11 11,11 21,21 21)')),
    (MPointFromWKB(AsWKB(MultiPoint(Point(3, 6), Point(4, 10)))));
```

1.2.9.3.1.7 MULTIPOLYGON

Syntax

```
MultiPolygon(poly1,poly2,...)
```

Description

Constructs a WKB MultiPolygon value from a set of WKB Polygon arguments. If any argument is not a WKB Polygon, the return value is `NULL`.

Example

```
CREATE TABLE gis_multi_polygon  (g MULTIPOLYGON);
INSERT INTO gis_multi_polygon VALUES
    (MultiPolygonFromText('MULTIPOLYGON(((28 26,28 0,84 0,84 42,28 26),(52 18,66 23,73 9,48
6,52 18)),
     ((59 18,67 18,67 13,59 13,59 18)))')),
    (MPolyFromText('MULTIPOLYGON(((28 26,28 0,84 0,84 42,28 26),(52 18,66 23,73 9,48 6,52 18)),
        ((59 18,67 18,67 13,59 13,59 18)))')),
    (MPolyFromWKB(AsWKB(MultiPolygon(Polygon(LineString(
       Point(0, 3), Point(3, 3), Point(3, 0), Point(0, 3)))))));
```

1.2.9.3.1.8 POINT

Syntax

```
Point(x,y)
```

Description

Constructs a WKB Point using the given coordinates.

Examples

```
SET @g = ST_GEOMFROMTEXT('Point(1 1)');

CREATE TABLE gis_point  (g POINT);
INSERT INTO gis_point VALUES
    (PointFromText('POINT(10 10)')),
    (PointFromText('POINT(20 10)')),
    (PointFromText('POINT(20 20)')),
    (PointFromWKB(AsWKB(PointFromText('POINT(10 20)'))));
```

1.2.9.3.1.9 PointOnSurface

A synonym for ST_PointOnSurface.

1.2.9.3.1.10 POLYGON

Syntax

```
Polygon(ls1,ls2,...)
```

Description

Constructs a WKB Polygon value from a number of WKB LineString arguments. If any argument does not represent the WKB of a LinearRing (that is, not a closed and simple LineString) the return value is `NULL`.

Note that according to the OpenGIS standard, a POLYGON should have exactly one ExteriorRing and all other rings should lie within that ExteriorRing and thus be the InteriorRings. Practically, however, some systems, including MariaDB's, permit polygons to have several 'ExteriorRings'. In the case of there being multiple, non-overlapping exterior rings ST_NUMINTERIORRINGS() will return 1.

Examples

```
SET @g = ST_GEOMFROMTEXT('POLYGON((1 1,1 5,4 9,6 9,9 3,7 2,1 1))');

CREATE TABLE gis_polygon    (g POLYGON);
INSERT INTO gis_polygon VALUES
    (PolygonFromText('POLYGON((10 10,20 10,20 20,10 20,10 10))')),
    (PolyFromText('POLYGON((0 0,50 0,50 50,0 50,0 0), (10 10,20 10,20 20,10 20,10 10))')),
    (PolyFromWKB(AsWKB(Polygon(LineString(Point(0, 0), Point(30, 0), Point(30, 30), Point(0, 0))
```

Non-overlapping 'polygon':

```
SELECT ST_NumInteriorRings(ST_PolyFromText('POLYGON((0 0,10 0,10 10,0 10,0 0),
  (-1 -1,-5 -1,-5 -5,-1 -5,-1 -1))')) AS NumInteriorRings;
+------------------+
| NumInteriorRings |
+------------------+
|                1 |
+------------------+
```

1.2.9.3.1.11 ST_BUFFER

Syntax

```
ST_BUFFER(g1,r)
BUFFER(g1,r)
```

Description

Returns a geometry that represents all points whose distance from geometry *g1* is less than or equal to distance, or radius, *r*.

Uses for this function could include creating for example a new geometry representing a buffer zone around an island.

BUFFER() is a synonym.

Examples

Determining whether a point is within a buffer zone:

```
SET @g1 = ST_GEOMFROMTEXT('POLYGON((10 10, 10 20, 20 20, 20 10, 10 10))');

SET @g2 = ST_GEOMFROMTEXT('POINT(8 8)');

SELECT ST_WITHIN(@g2,ST_BUFFER(@g1,5));
+--------------------------------+
| ST_WITHIN(@g2,ST_BUFFER(@g1,5)) |
+--------------------------------+
|                              1 |
+--------------------------------+

SELECT ST_WITHIN(@g2,ST_BUFFER(@g1,1));
+--------------------------------+
| ST_WITHIN(@g2,ST_BUFFER(@g1,1)) |
+--------------------------------+
|                              0 |
+--------------------------------+
```

1.2.9.3.1.12 ST_CONVEXHULL

MariaDB starting with 10.1.2
ST_ConvexHull() was introduced in MariaDB 10.1.2

Syntax

```
ST_ConvexHull(g)
ConvexHull(g)
```

Description

Given a geometry, returns a geometry that is the minimum convex geometry enclosing all geometries within the set. Returns NULL if the geometry value is NULL or an empty value.

ST_ConvexHull() and ConvexHull() are synonyms.

Examples

The ConvexHull of a single point is simply the single point:

```
SET @g = ST_GEOMFROMTEXT('Point(0 0)');

SELECT ST_ASTEXT(ST_CONVEXHULL(@g));
+------------------------------+
| ST_ASTEXT(ST_CONVEXHULL(@g)) |
+------------------------------+
| POINT(0 0)                   |
+------------------------------+
```

```
SET @g = ST_GEOMFROMTEXT('MultiPoint(0 0, 1 2, 2 3)');

SELECT ST_ASTEXT(ST_CONVEXHULL(@g));
+------------------------------+
| ST_ASTEXT(ST_CONVEXHULL(@g)) |
+------------------------------+
| POLYGON((0 0,1 2,2 3,0 0))   |
+------------------------------+
```

```
SET @g = ST_GEOMFROMTEXT('MultiPoint( 1 1, 2 2, 5 3, 7 2, 9 3, 8 4, 6 6, 6 9, 4 9, 1 5 )');

SELECT ST_ASTEXT(ST_CONVEXHULL(@g));
+--------------------------------------+
| ST_ASTEXT(ST_CONVEXHULL(@g))         |
+--------------------------------------+
| POLYGON((1 1,1 5,4 9,6 9,9 3,7 2,1 1)) |
+--------------------------------------+
```

1.2.9.3.1.13 ST_INTERSECTION

Syntax

```
ST_INTERSECTION(g1,g2)
```

Description

Returns a geometry that is the intersection, or shared portion, of geometry *g1* and geometry *g2*.

Examples

```
SET @g1 = ST_GEOMFROMTEXT('POINT(2 1)');

SET @g2 = ST_GEOMFROMTEXT('LINESTRING(2 1, 0 2)');

SELECT ASTEXT(ST_INTERSECTION(@g1,@g2));
+---------------------------------+
| ASTEXT(ST_INTERSECTION(@g1,@g2)) |
+---------------------------------+
| POINT(2 1)                      |
+---------------------------------+
```

1.2.9.3.1.14 ST_POINTONSURFACE

MariaDB starting with 10.1.2

ST_POINTONSURFACE() was introduced in MariaDB 10.1.2

Syntax

```
ST_PointOnSurface(g)
PointOnSurface(g)
```

Description

Given a geometry, returns a POINT guaranteed to intersect a surface. However, see MDEV-7514.

ST_PointOnSurface() and PointOnSurface() are synonyms.

1.2.9.3.1.15 ST_SYMDIFFERENCE

Syntax

```
ST_SYMDIFFERENCE(g1,g2)
```

Description

Returns a geometry that represents the portions of geometry `g1` and geometry `g2` that don't intersect.

Examples

```
SET @g1 = ST_GEOMFROMTEXT('LINESTRING(10 20, 10 40)');

SET @g2 = ST_GEOMFROMTEXT('LINESTRING(10 15, 10 25)');

SELECT ASTEXT(ST_SYMDIFFERENCE(@g1,@g2));
+-------------------------------------------+
| ASTEXT(ST_SYMDIFFERENCE(@g1,@g2))         |
+-------------------------------------------+
| MULTILINESTRING((10 15,10 20),(10 25,10 40)) |
+-------------------------------------------+

SET @g2 = ST_GeomFromText('LINESTRING(10 20, 10 41)');

SELECT ASTEXT(ST_SYMDIFFERENCE(@g1,@g2));
+-----------------------------------+
| ASTEXT(ST_SYMDIFFERENCE(@g1,@g2)) |
+-----------------------------------+
| LINESTRING(10 40,10 41)           |
+-----------------------------------+
```

1.2.9.3.1.16 ST_UNION

Syntax

```
ST_UNION(g1,g2)
```

Description

Returns a geometry that is the union of the geometry `g1` and geometry `g2` .

Examples

```
SET @g1 = GEOMFROMTEXT('POINT (0 2)');

SET @g2 = GEOMFROMTEXT('POINT (2 0)');

SELECT ASTEXT(ST_UNION(@g1,@g2));
+---------------------------+
| ASTEXT(ST_UNION(@g1,@g2)) |
+---------------------------+
| MULTIPOINT(2 0,0 2)       |
+---------------------------+
```

```
SET @g1 = GEOMFROMTEXT('POLYGON((0 0,0 3,3 3,3 0,0 0))');

SET @g2 = GEOMFROMTEXT('POLYGON((2 2,4 2,4 4,2 4,2 2))');

SELECT ASTEXT(ST_UNION(@g1,@g2));
+-------------------------------------------+
| ASTEXT(ST_UNION(@g1,@g2))                 |
+-------------------------------------------+
| POLYGON((0 0,0 3,2 3,2 4,4 4,4 2,3 2,3 0,0 0)) |
+-------------------------------------------+
```

1.2.9.3.2 Geometry Properties

Geometry properties

BOUNDARY
Synonym for ST_BOUNDARY.

DIMENSION
Synonym for ST_DIMENSION.

ENVELOPE
Synonym for ST_ENVELOPE.

GeometryN
Synonym for ST_GeometryN.

GeometryType
Synonym for ST_GeometryType.

IsClosed
Synonym for ST_IsClosed.

IsEmpty
Synonym for ST_IsEmpty.

IsRing
Synonym for ST_IsRing.

IsSimple
Synonym for ST_IsSimple.

NumGeometries
Synonym for ST_NumGeometries.

SRID
Synonym for ST_SRID.

ST_BOUNDARY
Returns a geometry that is the closure of a combinatorial boundary.

ST_DIMENSION
Inherent dimension of a geometry value.

ST_ENVELOPE
Returns the Minimum Bounding Rectangle for a geometry value.

ST_GEOMETRYN
Returns the N-th geometry in a GeometryCollection.

ST_GEOMETRYTYPE
Returns name of the geometry type of which a given geometry instance is a member.

ST_ISCLOSED
Returns true if a given LINESTRING's start and end points are the same.

ST_ISEMPTY
Indicated validity of geometry value.

ST_IsRing
Returns true if a given LINESTRING is both ST_IsClosed and ST_IsSimple.

ST_IsSimple
Returns true if the given Geometry has no anomalous geometric points.

ST_NUMGEOMETRIES
Number of geometries in a GeometryCollection.

ST_RELATE
Returns true if two geometries are related

ST_SRID
Returns a Spatial Reference System ID.

1.2.9.3.2.1 BOUNDARY

A synonym for ST_BOUNDARY.

1.2.9.3.2.2 DIMENSION

A synonym for ST_DIMENSION.

1.2.9.3.2.3 ENVELOPE

A synonym for ST_ENVELOPE.

1.2.9.3.2.4 GeometryN

A synonym for ST_GeometryN.

1.2.9.3.2.5 GeometryType

A synonym for ST_GeometryType.

1.2.9.3.2.6 IsClosed

A synonym for ST_IsClosed.

1.2.9.3.2.7 IsEmpty

A synonym for ST_IsEmpty.

1.2.9.3.2.8 IsRing

A synonym for ST_IsRing.

1.2.9.3.2.9 IsSimple

A synonym for ST_IsSImple.

1.2.9.3.2.10 NumGeometries

A synonym for ST_NumGeometries.

1.2.9.3.2.11 SRID

A synonym for ST_SRID.

1.2.9.3.2.12 ST_BOUNDARY

MariaDB starting with 10.1.2
The ST_BOUNDARY function was introduced in MariaDB 10.1.2

Syntax

```
ST_BOUNDARY(g)
BOUNDARY(g)
```

Description

Returns a geometry that is the closure of the combinatorial boundary of the geometry value `g`.

BOUNDARY() is a synonym.

Examples

```
SELECT ST_AsText(ST_Boundary(ST_GeomFromText('LINESTRING(3 3,0 0, -3 3)')));
+-----------------------------------------------------------------------+
| ST_AsText(ST_Boundary(ST_GeomFromText('LINESTRING(3 3,0 0, -3 3)'))) |
+-----------------------------------------------------------------------+
| MULTIPOINT(3 3,-3 3)                                                  |
+-----------------------------------------------------------------------+

SELECT ST_AsText(ST_Boundary(ST_GeomFromText('POLYGON((3 3,0 0, -3 3, 3 3))')));
+--------------------------------------------------------------------------+
| ST_AsText(ST_Boundary(ST_GeomFromText('POLYGON((3 3,0 0, -3 3, 3 3))'))) |
+--------------------------------------------------------------------------+
| LINESTRING(3 3,0 0,-3 3,3 3)                                             |
+--------------------------------------------------------------------------+
```

1.2.9.3.2.13 ST_DIMENSION

Syntax

```
ST_Dimension(g)
Dimension(g)
```

Description

Returns the inherent dimension of the geometry value `g`. The result can be

Dimension	Definition
-1	empty geometry
0	geometry with no length or area
1	geometry with no area but nonzero length
2	geometry with nonzero area

`ST_Dimension()` and `Dimension()` are synonyms.

Examples

```
SELECT Dimension(GeomFromText('LineString(1 1,2 2)'));
+-----------------------------------------------+
| Dimension(GeomFromText('LineString(1 1,2 2)')) |
+-----------------------------------------------+
|                                             1 |
+-----------------------------------------------+
```

1.2.9.3.2.14 ST_ENVELOPE

Syntax

```
ST_ENVELOPE(g)
ENVELOPE(g)
```

Description

Returns the Minimum Bounding Rectangle (MBR) for the geometry value `g`. The result is returned as a Polygon value.

The polygon is defined by the corner points of the bounding box:

```
POLYGON((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))
```

`ST_ENVELOPE()` and `ENVELOPE()` are synonyms.

Examples

```
SELECT AsText(ST_ENVELOPE(GeomFromText('LineString(1 1,4 4)')));
+-------------------------------------------------------------+
| AsText(ST_ENVELOPE(GeomFromText('LineString(1 1,4 4)'))) |
+-------------------------------------------------------------+
| POLYGON((1 1,4 1,4 4,1 4,1 1))                              |
+-------------------------------------------------------------+
```

1.2.9.3.2.15 ST_GEOMETRYN

Syntax

```
ST_GeometryN(gc,N)
GeometryN(gc,N)
```

Description

Returns the N-th geometry in the GeometryCollection `gc`. Geometries are numbered beginning with 1.

`ST_GeometryN()` and `GeometryN()` are synonyms.

Example

```
SET @gc = 'GeometryCollection(Point(1 1),LineString(12 14, 9 11))';

SELECT AsText(GeometryN(GeomFromText(@gc),1));
+--------------------------------------+
| AsText(GeometryN(GeomFromText(@gc),1)) |
+--------------------------------------+
| POINT(1 1)                           |
+--------------------------------------+
```

1.2.9.3.2.16 ST_GEOMETRYTYPE

Syntax

```
ST_GeometryType(g)
GeometryType(g)
```

Description

Returns as a string the name of the geometry type of which the geometry instance `g` is a member. The name corresponds to one of the instantiable Geometry subclasses.

`ST_GeometryType()` and `GeometryType()` are synonyms.

Examples

```
SELECT GeometryType(GeomFromText('POINT(1 1)'));
+---------------------------------------+
| GeometryType(GeomFromText('POINT(1 1)')) |
+---------------------------------------+
| POINT                                 |
+---------------------------------------+
```

1.2.9.3.2.17 ST_ISCLOSED

Syntax

```
ST_IsClosed(g)
IsClosed(g)
```

Description

Returns 1 if a given LINESTRING's start and end points are the same, or 0 if they are not the same. Before MariaDB 10.1.5, returns NULL if not given a LINESTRING. After MariaDB 10.1.5, returns -1.

`ST_IsClosed()` and `IsClosed()` are synonyms.

Examples

```
SET @ls = 'LineString(0 0, 0 4, 4 4, 0 0)';
SELECT ST_ISCLOSED(GEOMFROMTEXT(@ls));
+--------------------------------+
| ST_ISCLOSED(GEOMFROMTEXT(@ls)) |
+--------------------------------+
|                              1 |
+--------------------------------+

SET @ls = 'LineString(0 0, 0 4, 4 4, 0 1)';
SELECT ST_ISCLOSED(GEOMFROMTEXT(@ls));
+--------------------------------+
| ST_ISCLOSED(GEOMFROMTEXT(@ls)) |
+--------------------------------+
|                              0 |
+--------------------------------+
```

1.2.9.3.2.18 ST_ISEMPTY

Syntax

```
ST_IsEmpty(g)
IsEmpty(g)
```

Description

IsEmpty is a function defined by the OpenGIS specification, but is not fully implemented by MariaDB or MySQL.

Since MariaDB and MySQL do not support GIS EMPTY values such as POINT EMPTY, as implemented it simply returns `1` if the geometry value `g` is invalid, `0` if it is valid, and `NULL` if the argument is `NULL`.

`ST_IsEmpty()` and `IsEmpty()` are synonyms.

1.2.9.3.2.19 ST_IsRing

MariaDB starting with 10.1.2

The ST_IsRing function was introduced in MariaDB 10.1.2

Syntax

```
ST_IsRing(g)
IsRing(g)
```

Description

Returns true if a given LINESTRING is a ring, that is, both ST_IsClosed and ST_IsSimple. A simple curve does not pass through the same point more than once. However, see MDEV-7510.

`St_IsRing()` and `IsRing()` are synonyms.

1.2.9.3.2.20 ST_IsSimple

Syntax

```
ST_IsSimple(g)
IsSimple(g)
```

Description

Returns true if the given Geometry has no anomalous geometric points, false if it does, or NULL if given a NULL value.

ST_IsSimple() and IsSimple() are synonyms.

Examples

A POINT is always simple.

```
SET @g = 'Point(1 2)';

SELECT ST_ISSIMPLE(GEOMFROMTEXT(@g));
+------------------------------+
| ST_ISSIMPLE(GEOMFROMTEXT(@g)) |
+------------------------------+
|                            1 |
+------------------------------+
```

1.2.9.3.2.21 ST_NUMGEOMETRIES

Syntax

```
ST_NumGeometries(gc)
NumGeometries(gc)
```

Description

Returns the number of geometries in the GeometryCollection `gc`.

`ST_NumGeometries()` and `NumGeometries()` are synonyms.

Example

```
SET @gc = 'GeometryCollection(Point(1 1),LineString(2 2, 3 3))';

SELECT NUMGEOMETRIES(GeomFromText(@gc));
+----------------------------------+
| NUMGEOMETRIES(GeomFromText(@gc)) |
+----------------------------------+
|                                2 |
+----------------------------------+
```

1.2.9.3.2.22 ST_RELATE

MariaDB starting with 10.1.2

The ST_RELATE() function was introduced in MariaDB 10.1.2

Syntax

```
ST_Relate(g1, g2, i)
```

Description

Returns true if Geometry `g1` is spatially related to Geometry `g2` by testing for intersections between the interior, boundary and exterior of the two geometries as specified by the values in intersection matrix pattern `i`.

1.2.9.3.2.23 ST_SRID

Syntax

```
ST_SRID(g)
SRID(g)
```

Description

Returns an integer indicating the Spatial Reference System ID for the geometry value g.

In MariaDB, the SRID value is just an integer associated with the geometry value. All calculations are done assuming Euclidean (planar) geometry.

`ST_SRID()` and `SRID()` are synonyms.

Examples

```
SELECT SRID(GeomFromText('LineString(1 1,2 2)',101));
+-----------------------------------------------+
| SRID(GeomFromText('LineString(1 1,2 2)',101)) |
+-----------------------------------------------+
|                                           101 |
+-----------------------------------------------+
```

1.2.9.3.3 Geometry Relations

Geometry relations

CONTAINS

Whether one geometry contains another.

CROSSES

Whether two geometries spatially cross

DISJOINT

Whether the two elements do not intersect.

EQUALS

Indicates whether two geometries are spatially equal.

INTERSECTS

Indicates whether two geometries spatially intersect.

OVERLAPS

Indicates whether two elements spatially overlap.

ST_CONTAINS

Whether one geometry is contained by another.

ST_CROSSES

Whether two geometries spatially cross.

ST_DIFFERENCE

Point set difference.

ST_DISJOINT

Whether one geometry is spatially disjoint from another.

ST_DISTANCE

The distance between two geometries.

ST_DISTANCE_SPHERE

Spherical distance between two geometries (point or multipoint) on a sphere.

ST_EQUALS

Whether two geometries are spatoially equal.

ST_INTERSECTS

Whether two geometries spatially intersect.

ST_LENGTH

Length of a LineString value.

ST_OVERLAPS

Whether two geometries overlap.

ST_TOUCHES

Whether one geometry g1 spatially touches another.

ST_WITHIN

Whether one geometry is within another.

TOUCHES

Whether two geometries spatially touch.

WITHIN

Indicate whether a geographic element is spacially within another.

1.2.9.3.3.1 CONTAINS

Syntax

```
Contains(g1,g2)
```

Description

Returns `1` or `0` to indicate whether a geometry `g1` completely contains geometry `g2` . CONTAINS() is based on the original MySQL implementation and uses object bounding rectangles, while ST_CONTAINS() uses object shapes.

This tests the opposite relationship to Within().

1.2.9.3.3.2 CROSSES

Syntax

```
Crosses(g1,g2)
```

Description

Returns `1` if `g1` spatially crosses `g2` . Returns `NULL` if `g1` is a Polygon or a MultiPolygon, or if `g2` is a Point or a MultiPoint. Otherwise, returns `0` .

The term spatially crosses denotes a spatial relation between two given geometries that has the following properties:

- The two geometries intersect
- Their intersection results in a geometry that has a dimension that is one less than the maximum dimension of the two given geometries
- Their intersection is not equal to either of the two given geometries

CROSSES() is based on the original MySQL implementation, and uses object bounding rectangles, while ST_CROSSES() uses object shapes.

1.2.9.3.3.3 DISJOINT

Syntax

```
Disjoint(g1,g2)
```

Description

Returns `1` or `0` to indicate whether `g1` is spatially disjoint from (does not intersect) `g2` .

DISJOINT() tests the opposite relationship to INTERSECTS().

DISJOINT() is based on the original MySQL implementation and uses object bounding rectangles, while ST_DISJOINT() uses object shapes.

1.2.9.3.3.4 EQUALS

Syntax

```
Equals(g1,g2)
```

From MariaDB 10.2.3:

```
MBREQUALS(g1,g2)
```

Description

Returns `1` or `0` to indicate whether *g1* is spatially equal to *g2*.

EQUALS() is based on the original MySQL implementation and uses object bounding rectangles, while ST_EQUALS() uses object shapes.

From MariaDB 10.2.3, `MBREQUALS` is a synonym for `Equals`.

1.2.9.3.3.5 INTERSECTS

Syntax

```
INTERSECTS(g1,g2)
```

Description

Returns `1` or `0` to indicate whether geometry *g1* spatially intersects geometry *g2*.

INTERSECTS() is based on the original MySQL implementation and uses object bounding rectangles, while ST_INTERSECTS() uses object shapes.

INTERSECTS() tests the opposite relationship to DISJOINT().

1.2.9.3.3.6 OVERLAPS

Syntax

```
OVERLAPS(g1,g2)
```

Description

Returns `1` or `0` to indicate whether `g1` spatially overlaps `g2`. The term spatially overlaps is used if two geometries intersect and their intersection results in a geometry of the same dimension but not equal to either of the given geometries.

OVERLAPS() is based on the original MySQL implementation and uses object bounding rectangles, while ST_OVERLAPS() uses object shapes.

1.2.9.3.3.7 ST_CONTAINS

Syntax

```
ST_CONTAINS(g1,g2)
```

Description

Returns `1` or `0` to indicate whether a geometry `g1` completely contains geometry `g2`.

ST_CONTAINS() uses object shapes, while CONTAINS(), based on the original MySQL implementation, uses object bounding rectangles.

ST_CONTAINS tests the opposite relationship to ST_WITHIN().

Examples

```
SET @g1 = ST_GEOMFROMTEXT('POLYGON((175 150, 20 40, 50 60, 125 100, 175 150))');

SET @g2 = ST_GEOMFROMTEXT('POINT(174 149)');

SELECT ST_CONTAINS(@g1,@g2);
+----------------------+
| ST_CONTAINS(@g1,@g2) |
+----------------------+
|                    1 |
+----------------------+

SET @g2 = ST_GEOMFROMTEXT('POINT(175 151)');

SELECT ST_CONTAINS(@g1,@g2);
+----------------------+
| ST_CONTAINS(@g1,@g2) |
+----------------------+
|                    0 |
+----------------------+
```

1.2.9.3.3.8 ST_CROSSES

Syntax

```
ST_CROSSES(g1,g2)
```

Description

Returns `1` if geometry *`g1`* spatially crosses geometry *`g2`*. Returns `NULL` if `g1` is a Polygon or a MultiPolygon, or if `g2` is a Point or a MultiPoint. Otherwise, returns `0`.

The term spatially crosses denotes a spatial relation between two given geometries that has the following properties:

- The two geometries intersect
- Their intersection results in a geometry that has a dimension that is one less than the maximum dimension of the two given geometries
- Their intersection is not equal to either of the two given geometries

ST_CROSSES() uses object shapes, while CROSSES(), based on the original MySQL implementation, uses object bounding rectangles.

Examples

```
SET @g1 = ST_GEOMFROMTEXT('LINESTRING(174 149, 176 151)');

SET @g2 = ST_GEOMFROMTEXT('POLYGON((175 150, 20 40, 50 60, 125 100, 175 150))');

SELECT ST_CROSSES(@g1,@g2);
+---------------------+
| ST_CROSSES(@g1,@g2) |
+---------------------+
|                   1 |
+---------------------+

SET @g1 = ST_GEOMFROMTEXT('LINESTRING(176 149, 176 151)');

SELECT ST_CROSSES(@g1,@g2);
+---------------------+
| ST_CROSSES(@g1,@g2) |
+---------------------+
|                   0 |
+---------------------+
```

1.2.9.3.3.9 ST_DIFFERENCE

Syntax

```
ST_DIFFERENCE(g1,g2)
```

Description

Returns a geometry representing the point set difference of the given geometry values.

Example

```
SET @g1 = POINT(10,10), @g2 = POINT(20,20);

SELECT ST_AsText(ST_Difference(@g1, @g2));
+-----------------------------------+
| ST_AsText(ST_Difference(@g1, @g2)) |
+-----------------------------------+
| POINT(10 10)                      |
+-----------------------------------+
```

1.2.9.3.3.10 ST_DISJOINT

Syntax

```
ST_DISJOINT(g1,g2)
```

Description

Returns `1` or `0` to indicate whether geometry *`g1`* is spatially disjoint from (does not intersect with) geometry *`g2`*.

ST_DISJOINT() uses object shapes, while DISJOINT(), based on the original MySQL implementation, uses object bounding rectangles.

ST_DISJOINT() tests the opposite relationship to ST_INTERSECTS().

Examples

```
SET @g1 = ST_GEOMFROMTEXT('POINT(0 0)');

SET @g2 = ST_GEOMFROMTEXT('LINESTRING(2 0, 0 2)');

SELECT ST_DISJOINT(@g1,@g2);
+----------------------+
| ST_DISJOINT(@g1,@g2) |
+----------------------+
|                    1 |
+----------------------+

SET @g2 = ST_GEOMFROMTEXT('LINESTRING(0 0, 0 2)');

SELECT ST_DISJOINT(@g1,@g2);
+----------------------+
| ST_DISJOINT(@g1,@g2) |
+----------------------+
|                    0 |
+----------------------+
```

1.2.9.3.3.11 ST_DISTANCE

Syntax

```
ST_DISTANCE(g1,g2)
```

Description

Returns the distance between two geometries, or null if not given valid inputs.

Example

```
SELECT ST_Distance(POINT(1,2),POINT(2,2));
+-----------------------------------+
| ST_Distance(POINT(1,2),POINT(2,2)) |
+-----------------------------------+
|                                 1 |
+-----------------------------------+
```

1.2.9.3.3.12 ST_DISTANCE_SPHERE

MariaDB starting with 10.2.38

`ST_DISTANCE_SPHERE` was introduced in MariaDB 10.2.38, MariaDB 10.3.29, MariaDB 10.4.19 and MariaDB 10.5.10.

Syntax

```
ST_DISTANCE_SPHERE(g1,g2,[r])
```

Description

Returns the spherical distance between two geometries (point or multipoint) on a sphere with the optional radius *r* (default is the Earth radius if *r* is not specified), or NULL if not given valid inputs.

Example

```
set @zenica   = ST_GeomFromText('POINT(17.907743 44.203438)');
set @sarajevo = ST_GeomFromText('POINT(18.413076 43.856258)');
SELECT ST_Distance_Sphere(@zenica, @sarajevo);
55878.59337591705
```

1.2.9.3.3.13 ST_EQUALS

Syntax

```
ST_EQUALS(g1,g2)
```

Description

Returns `1` or `0` to indicate whether geometry *`g1`* is spatially equal to geometry *`g2`*.

ST_EQUALS() uses object shapes, while EQUALS(), based on the original MySQL implementation, uses object bounding rectangles.

Examples

```
SET @g1 = ST_GEOMFROMTEXT('LINESTRING(174 149, 176 151)');

SET @g2 = ST_GEOMFROMTEXT('LINESTRING(176 151, 174 149)');

SELECT ST_EQUALS(@g1,@g2);
+--------------------+
| ST_EQUALS(@g1,@g2) |
+--------------------+
|                  1 |
+--------------------+
```

```
SET @g1 = ST_GEOMFROMTEXT('POINT(0 2)');

SET @g1 = ST_GEOMFROMTEXT('POINT(2 0)');

SELECT ST_EQUALS(@g1,@g2);
+--------------------+
| ST_EQUALS(@g1,@g2) |
+--------------------+
|                  0 |
+--------------------+
```

1.2.9.3.3.14 ST_INTERSECTS

Syntax

```
ST_INTERSECTS(g1,g2)
```

Description

Returns `1` or `0` to indicate whether geometry *`g1`* spatially intersects geometry *`g2`*.

ST_INTERSECTS() uses object shapes, while INTERSECTS(), based on the original MySQL implementation, uses object bounding rectangles.

ST_INTERSECTS() tests the opposite relationship to ST_DISJOINT().

Examples

```
SET @g1 = ST_GEOMFROMTEXT('POINT(0 0)');

SET @g2 = ST_GEOMFROMTEXT('LINESTRING(0 0, 0 2)');

SELECT ST_INTERSECTS(@g1,@g2);
+------------------------+
| ST_INTERSECTS(@g1,@g2) |
+------------------------+
|                      1 |
+------------------------+
```

```
SET @g2 = ST_GEOMFROMTEXT('LINESTRING(2 0, 0 2)');

SELECT ST_INTERSECTS(@g1,@g2);
+------------------------+
| ST_INTERSECTS(@g1,@g2) |
+------------------------+
|                      0 |
+------------------------+
```

1.2.9.3.3.15 ST_LENGTH

Syntax

```
ST_LENGTH(ls)
```

Description

Returns as a double-precision number the length of the LineString value `ls` in its associated spatial reference.

Examples

```
SET @ls = 'LineString(1 1,2 2,3 3)';

SELECT ST_LENGTH(ST_GeomFromText(@ls));
+---------------------------------+
| ST_LENGTH(ST_GeomFromText(@ls)) |
+---------------------------------+
|                2.82842712474619 |
+---------------------------------+
```

1.2.9.3.3.16 ST_OVERLAPS

Syntax

```
ST_OVERLAPS(g1,g2)
```

Description

Returns `1` or `0` to indicate whether geometry `g1` spatially overlaps geometry `g2`.

The term spatially overlaps is used if two geometries intersect and their intersection results in a geometry of the same dimension but not equal to either of the given geometries.

ST_OVERLAPS() uses object shapes, while OVERLAPS(), based on the original MySQL implementation, uses object

bounding rectangles.

1.2.9.3.3.17 ST_TOUCHES

Syntax

```
ST_TOUCHES(g1,g2)
```

Description

Returns `1` or `0` to indicate whether geometry `g1` spatially touches geometry `g2`. Two geometries spatially touch if the interiors of the geometries do not intersect, but the boundary of one of the geometries intersects either the boundary or the interior of the other.

ST_TOUCHES() uses object shapes, while TOUCHES(), based on the original MySQL implementation, uses object bounding rectangles.

Examples

```
SET @g1 = ST_GEOMFROMTEXT('POINT(2 0)');

SET @g2 = ST_GEOMFROMTEXT('LINESTRING(2 0, 0 2)');

SELECT ST_TOUCHES(@g1,@g2);
+---------------------+
| ST_TOUCHES(@g1,@g2) |
+---------------------+
|                   1 |
+---------------------+

SET @g1 = ST_GEOMFROMTEXT('POINT(2 1)');

SELECT ST_TOUCHES(@g1,@g2);
+---------------------+
| ST_TOUCHES(@g1,@g2) |
+---------------------+
|                   0 |
+---------------------+
```

1.2.9.3.3.18 ST_WITHIN

Syntax

```
ST_WITHIN(g1,g2)
```

Description

Returns `1` or `0` to indicate whether geometry `g1` is spatially within geometry `g2`.

This tests the opposite relationship as `ST_CONTAINS()`.

ST_WITHIN() uses object shapes, while WITHIN(), based on the original MySQL implementation, uses object bounding rectangles.

Examples

```
SET @g1 = ST_GEOMFROMTEXT('POINT(174 149)');

SET @g2 = ST_GEOMFROMTEXT('POLYGON((175 150, 20 40, 50 60, 125 100, 175 150))');

SELECT ST_WITHIN(@g1,@g2);
+--------------------+
| ST_WITHIN(@g1,@g2) |
+--------------------+
|                  1 |
+--------------------+

SET @g1 = ST_GEOMFROMTEXT('POINT(176 151)');

SELECT ST_WITHIN(@g1,@g2);
+--------------------+
| ST_WITHIN(@g1,@g2) |
+--------------------+
|                  0 |
+--------------------+
```

1.2.9.3.3.19 TOUCHES

Syntax

```
Touches(g1,g2)
```

Description

Returns `1` or `0` to indicate whether `g1` spatially touches `g2`. Two geometries spatially touch if the interiors of the geometries do not intersect, but the boundary of one of the geometries intersects either the boundary or the interior of the other.

TOUCHES() is based on the original MySQL implementation and uses object bounding rectangles, while ST_TOUCHES() uses object shapes.

1.2.9.3.3.20 WITHIN

Syntax

```
Within(g1,g2)
```

Description

Returns `1` or `0` to indicate whether `g1` is spatially within `g2`. This tests the opposite relationship as `Contains()`.

WITHIN() is based on the original MySQL implementation, and uses object bounding rectangles, while ST_WITHIN() uses object shapes.

Examples

```
SET @g1 = GEOMFROMTEXT('POINT(174 149)');
SET @g2 = GEOMFROMTEXT('POINT(176 151)');
SET @g3 = GEOMFROMTEXT('POLYGON((175 150, 20 40, 50 60, 125 100, 175 150))');

SELECT within(@g1,@g3);
+-----------------+
| within(@g1,@g3) |
+-----------------+
|               1 |
+-----------------+

SELECT within(@g2,@g3);
+-----------------+
| within(@g2,@g3) |
+-----------------+
|               0 |
+-----------------+
```

1.2.9.3.4 LineString Properties

LineString properties

ENDPOINT

Synonym for ST_ENDPOINT.

GLENGTH

Length of a LineString value.

NumPoints

Synonym for ST_NumPoints.

PointN

Synonym for PointN.

STARTPOINT

Synonym for ST_StartPoint.

ST_ENDPOINT

Returns the endpoint of a LineString.

ST_NUMPOINTS

Returns the number of Point objects in a LineString.

ST_POINTN

Returns the N-th Point in the LineString.

ST_STARTPOINT

Returns the start point of a LineString

1.2.9.3.4.1 ENDPOINT

A synonym for ST_ENDPOINT.

1.2.9.3.4.2 GLENGTH

Syntax

```
GLength(ls)
```

Description

Returns as a double-precision number the length of the LineString value `ls` in its associated spatial reference.

Examples

```
SET @ls = 'LineString(1 1,2 2,3 3)';

SELECT GLength(GeomFromText(@ls));
+----------------------------+
| GLength(GeomFromText(@ls)) |
+----------------------------+
|          2.82842712474619 |
+----------------------------+
```

1.2.9.3.4.3 NumPoints

A synonym for ST_NumPoints.

1.2.9.3.4.4 PointN

A synonym for ST_PointN.

1.2.9.3.4.5 STARTPOINT

A synonym for ST_STARTPOINT.

1.2.9.3.4.6 ST_ENDPOINT

Syntax

```
ST_EndPoint(ls)
EndPoint(ls)
```

Description

Returns the Point that is the endpoint of the LineString value `ls` .

`ST_EndPoint()` and `EndPoint()` are synonyms.

Examples

```
SET @ls = 'LineString(1 1,2 2,3 3)';

SELECT AsText(EndPoint(GeomFromText(@ls)));
+-------------------------------------+
| AsText(EndPoint(GeomFromText(@ls))) |
+-------------------------------------+
| POINT(3 3)                          |
+-------------------------------------+
```

1.2.9.3.4.7 ST_NUMPOINTS

Syntax

```
ST_NumPoints(ls)
NumPoints(ls)
```

Description

Returns the number of Point objects in the LineString value `ls` .

`ST_NumPoints()` and `NumPoints()` are synonyms.

Examples

```
SET @ls = 'LineString(1 1,2 2,3 3)';

SELECT NumPoints(GeomFromText(@ls));
+------------------------------+
| NumPoints(GeomFromText(@ls)) |
+------------------------------+
|                            3 |
+------------------------------+
```

1.2.9.3.4.8 ST_POINTN

Syntax

```
ST_PointN(ls,N)
PointN(ls,N)
```

Description

Returns the N-th Point in the LineString value `ls` . Points are numbered beginning with `1` .

`ST_PointN()` and `PointN()` are synonyms.

Examples

```
SET @ls = 'LineString(1 1,2 2,3 3)';

SELECT AsText(PointN(GeomFromText(@ls),2));
+-------------------------------------+
| AsText(PointN(GeomFromText(@ls),2)) |
+-------------------------------------+
| POINT(2 2)                          |
+-------------------------------------+
```

1.2.9.3.4.9 ST_STARTPOINT

Syntax

```
ST_StartPoint(ls)
StartPoint(ls)
```

Description

Returns the Point that is the start point of the LineString value `ls` .

`ST_StartPoint()` and `StartPoint()` are synonyms.

Examples

```
SET @ls = 'LineString(1 1,2 2,3 3)';

SELECT AsText(StartPoint(GeomFromText(@ls)));
+--------------------------------------+
| AsText(StartPoint(GeomFromText(@ls))) |
+--------------------------------------+
| POINT(1 1)                           |
+--------------------------------------+
```

1.2.9.3.5 MBR (Minimum Bounding Rectangle)

MBR Definition

Minimum Bounding Rectangle.

MBRContains

Indicates one Minimum Bounding Rectangle contains another.

MBRDisjoint

Indicates whether the Minimum Bounding Rectangles of two geometries are disjoint.

MBREqual

Whether the Minimum Bounding Rectangles of two geometries are the same.

MBRIntersects

Indicates whether the Minimum Bounding Rectangles of the two geometries intersect.

MBROverlaps

Whether the Minimum Bounding Rectangles of two geometries overlap.

MBRTouches

Whether the Minimum Bounding Rectangles of two geometries touch.

MBRWithin

Indicates whether one Minimum Bounding Rectangle is within another

1.2.9.3.5.1 MBR Definition

Description

The MBR (Minimum Bounding Rectangle), or Envelope is the bounding geometry, formed by the minimum and maximum (X,Y) coordinates:

Examples

```
((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))
```

1.2.9.3.5.2 MBRContains

Syntax

```
MBRContains(g1,g2)
```

Description

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangle of g1 contains the Minimum Bounding Rectangle of g2. This tests the opposite relationship as MBRWithin().

Examples

```
SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');

SET @g2 = GeomFromText('Point(1 1)');

SELECT MBRContains(@g1,@g2), MBRContains(@g2,@g1);
+----------------------+----------------------+
| MBRContains(@g1,@g2) | MBRContains(@g2,@g1) |
+----------------------+----------------------+
|                    1 |                    0 |
+----------------------+----------------------+
```

1.2.9.3.5.3 MBRDisjoint

Syntax

```
MBRDisjoint(g1,g2)
```

Description

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries g1 and g2 are disjoint. Two geometries are disjoint if they do not intersect, that is touch or overlap.

Examples

```
SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
SET @g2 = GeomFromText('Polygon((4 4,4 7,7 7,7 4,4 4))');
SELECTmbrdisjoint(@g1,@g2);
+----------------------+
| mbrdisjoint(@g1,@g2) |
+----------------------+
|                    1 |
+----------------------+

SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
SET @g2 = GeomFromText('Polygon((3 3,3 6,6 6,6 3,3 3))');
SELECT mbrdisjoint(@g1,@g2);
+----------------------+
| mbrdisjoint(@g1,@g2) |
+----------------------+
|                    0 |
+----------------------+
```

1.2.9.3.5.4 MBREqual

Syntax

```
MBREqual(g1,g2)
```

Description

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries g1 and g2 are the same.

Examples

```
SET @g1=GEOMFROMTEXT('LINESTRING(0 0, 1 2)');
SET @g2=GEOMFROMTEXT('POLYGON((0 0, 0 2, 1 2, 1 0, 0 0))');
SELECT MbrEqual(@g1,@g2);
+-------------------+
| MbrEqual(@g1,@g2) |
+-------------------+
|                 1 |
+-------------------+

SET @g1=GEOMFROMTEXT('LINESTRING(0 0, 1 3)');
SET @g2=GEOMFROMTEXT('POLYGON((0 0, 0 2, 1 4, 1 0, 0 0))');
SELECT MbrEqual(@g1,@g2);
+-------------------+
| MbrEqual(@g1,@g2) |
+-------------------+
|                 0 |
+-------------------+
```

1.2.9.3.5.5 MBRIntersects

Syntax

```
MBRIntersects(g1,g2)
```

Description

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries g1 and g2 intersect.

Examples

```
SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
SET @g2 = GeomFromText('Polygon((3 3,3 6,6 6,6 3,3 3))');
SELECT mbrintersects(@g1,@g2);
+------------------------+
| mbrintersects(@g1,@g2) |
+------------------------+
|                      1 |
+------------------------+

SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
SET @g2 = GeomFromText('Polygon((4 4,4 7,7 7,7 4,4 4))');
SELECT mbrintersects(@g1,@g2);
+------------------------+
| mbrintersects(@g1,@g2) |
+------------------------+
|                      0 |
+------------------------+
```

1.2.9.3.5.6 MBROverlaps

Syntax

```
MBROverlaps(g1,g2)
```

Description

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries `g1` and `g2` overlap. The term spatially overlaps is used if two geometries intersect and their intersection results in a geometry of the same dimension but not equal to either of the given geometries.

Examples

```
SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
SET @g2 = GeomFromText('Polygon((4 4,4 7,7 7,7 4,4 4))');
SELECT mbroverlaps(@g1,@g2);
+----------------------+
| mbroverlaps(@g1,@g2) |
+----------------------+
|                    0 |
+----------------------+

SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
SET @g2 = GeomFromText('Polygon((3 3,3 6,6 6,6 3,3 3))');
SELECT mbroverlaps(@g1,@g2);
+----------------------+
| mbroverlaps(@g1,@g2) |
+----------------------+
|                    0 |
+----------------------+

SET @g1 = GeomFromText('Polygon((0 0,0 4,4 4,4 0,0 0))');
SET @g2 = GeomFromText('Polygon((3 3,3 6,6 6,6 3,3 3))');
SELECT mbroverlaps(@g1,@g2);
+----------------------+
| mbroverlaps(@g1,@g2) |
+----------------------+
|                    1 |
+----------------------+
```

1.2.9.3.5.7 MBRTouches

Syntax

```
MBRTouches(g1,g2)
```

Description

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangles of the two geometries `g1` and `g2` touch. Two geometries spatially touch if the interiors of the geometries do not intersect, but the boundary of one of the geometries intersects either the boundary or the interior of the other.

Examples

```
SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
SET @g2 = GeomFromText('Polygon((4 4,4 7,7 7,7 4,4 4))');
SELECT mbrtouches(@g1,@g2);
+---------------------+
| mbrtouches(@g1,@g2) |
+---------------------+
|                   0 |
+---------------------+

SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
SET @g2 = GeomFromText('Polygon((3 3,3 6,6 6,6 3,3 3))');
SELECT mbrtouches(@g1,@g2);
+---------------------+
| mbrtouches(@g1,@g2) |
+---------------------+
|                   1 |
+---------------------+

SET @g1 = GeomFromText('Polygon((0 0,0 4,4 4,4 0,0 0))');
SET @g2 = GeomFromText('Polygon((3 3,3 6,6 6,6 3,3 3))');
SELECT mbrtouches(@g1,@g2);
+---------------------+
| mbrtouches(@g1,@g2) |
+---------------------+
|                   0 |
+---------------------+
```

1.2.9.3.5.8 MBRWithin

Syntax

```
MBRWithin(g1,g2)
```

Description

Returns 1 or 0 to indicate whether the Minimum Bounding Rectangle of g1 is within the Minimum Bounding Rectangle of g2. This tests the opposite relationship as MBRContains().

Examples

```
SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
SET @g2 = GeomFromText('Polygon((0 0,0 5,5 5,5 0,0 0))');
SELECT MBRWithin(@g1,@g2), MBRWithin(@g2,@g1);
+--------------------+--------------------+
| MBRWithin(@g1,@g2) | MBRWithin(@g2,@g1) |
+--------------------+--------------------+
|                  1 |                  0 |
+--------------------+--------------------+
```

1.2.9.3.6 Point Properties

Point properties

ST_X

X-coordinate value for a point.

ST_Y

Y-coordinate for a point.

X

Synonym for ST_X.

Y

Synonym for ST_Y.

1.2.9.3.6.1 ST_X

Syntax

```
ST_X(p)
X(p)
```

Description

Returns the X-coordinate value for the point `p` as a double-precision number.

`ST_X()` and `X()` are synonyms.

Examples

```
SET @pt = 'Point(56.7 53.34)';

SELECT X(GeomFromText(@pt));
+----------------------+
| X(GeomFromText(@pt)) |
+----------------------+
|                 56.7 |
+----------------------+
```

1.2.9.3.6.2 ST_Y

Syntax

```
ST_Y(p)
Y(p)
```

Description

Returns the Y-coordinate value for the point p as a double-precision number.

`ST_Y()` and `Y()` are synonyms.

Examples

```
SET @pt = 'Point(56.7 53.34)';

SELECT Y(GeomFromText(@pt));
+----------------------+
| Y(GeomFromText(@pt)) |
+----------------------+
|                53.34 |
+----------------------+
```

1.2.9.3.6.3 X

A synonym for ST_X.

1.2.9.3.6.4 Y

A synonym for ST_Y.

1.2.9.3.7 Polygon Properties

Polygon properties

AREA

Synonym for ST_AREA.

CENTROID

Synonym for ST_CENTROID.

ExteriorRing

Synonym for ST_ExteriorRing.

InteriorRingN

Synonym for ST_InteriorRingN.

NumInteriorRings

Synonym for NumInteriorRings.

ST_AREA

Area of a Polygon.

ST_CENTROID

The mathematical centroid (geometric center) for a MultiPolygon.

ST_ExteriorRing

Returns the exterior ring of a Polygon as a LineString.

ST_InteriorRingN

Returns the N-th interior ring for a Polygon.

ST_NumInteriorRings

Number of interior rings in a Polygon.

1.2.9.3.7.1 AREA

A synonym for ST_AREA.

1.2.9.3.7.2 CENTROID

A synonym for ST_CENTROID.

1.2.9.3.7.3 ExteriorRing

A synonym for ST_ExteriorRing.

1.2.9.3.7.4 InteriorRingN

A synonym for ST_InteriorRingN.

1.2.9.3.7.5 NumInteriorRings

A synonym for ST_NumInteriorRings.

1.2.9.3.7.6 ST_AREA

Syntax

```
ST_Area(poly)
Area(poly)
```

Description

Returns as a double-precision number the area of the Polygon value `poly`, as measured in its spatial reference system.

`ST_Area()` and `Area()` are synonyms.

Examples

```
SET @poly = 'Polygon((0 0,0 3,3 0,0 0),(1 1,1 2,2 1,1 1))';

SELECT Area(GeomFromText(@poly));
+---------------------------+
| Area(GeomFromText(@poly)) |
+---------------------------+
|                         4 |
+---------------------------+
```

1.2.9.3.7.7 ST_CENTROID

Syntax

```
ST_Centroid(mpoly)
Centroid(mpoly)
```

Description

Returns a point reflecting the mathematical centroid (geometric center) for the MultiPolygon *mpoly*. The resulting point will not necessarily be on the MultiPolygon.

`ST_Centroid()` and `Centroid()` are synonyms.

Examples

```
SET @poly = ST_GeomFromText('POLYGON((0 0,20 0,20 20,0 20,0 0))');
SELECT ST_AsText(ST_Centroid(@poly)) AS center;
+--------------+
| center       |
+--------------+
| POINT(10 10) |
+--------------+
```

1.2.9.3.7.8 ST_ExteriorRing

Syntax

```
ST_ExteriorRing(poly)
ExteriorRing(poly)
```

Description

Returns the exterior ring of the Polygon value `poly` as a LineString.

`ST_ExteriorRing()` and `ExteriorRing()` are synonyms.

Examples

```
SET @poly = 'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))';

SELECT AsText(ExteriorRing(GeomFromText(@poly)));
+------------------------------------------+
| AsText(ExteriorRing(GeomFromText(@poly))) |
+------------------------------------------+
| LINESTRING(0 0,0 3,3 3,3 0,0 0)           |
+------------------------------------------+
```

1.2.9.3.7.9 ST_InteriorRingN

Syntax

```
ST_InteriorRingN(poly,N)
InteriorRingN(poly,N)
```

Description

Returns the N-th interior ring for the Polygon value `poly` as a LineString. Rings are numbered beginning with 1.

`ST_InteriorRingN()` and `InteriorRingN()` are synonyms.

Examples

```
SET @poly = 'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))';

SELECT AsText(InteriorRingN(GeomFromText(@poly),1));
+---------------------------------------------+
| AsText(InteriorRingN(GeomFromText(@poly),1)) |
+---------------------------------------------+
| LINESTRING(1 1,1 2,2 2,2 1,1 1)              |
+---------------------------------------------+
```

1.2.9.3.7.10 ST_NumInteriorRings

Syntax

```
ST_NumInteriorRings(poly)
NumInteriorRings(poly)
```

Description

Returns an integer containing the number of interior rings in the Polygon value `poly`.

Note that according the the OpenGIS standard, a POLYGON should have exactly one ExteriorRing and all other rings should lie within that ExteriorRing and thus be the InteriorRings. Practically, however, some systems, including MariaDB's, permit polygons to have several 'ExteriorRings'. In the case of there being multiple, non-overlapping exterior rings ST_NumInteriorRings() will return `1`.

`ST_NumInteriorRings()` and `NumInteriorRings()` are synonyms.

Examples

```
SET @poly = 'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))';

SELECT NumInteriorRings(GeomFromText(@poly));
+---------------------------------------+
| NumInteriorRings(GeomFromText(@poly)) |
+---------------------------------------+
|                                     1 |
+---------------------------------------+
```

Non-overlapping 'polygon':

```
SELECT ST_NumInteriorRings(ST_PolyFromText('POLYGON((0 0,10 0,10 10,0 10,0 0),
  (-1 -1,-5 -1,-5 -5,-1 -5,-1 -1))')) AS NumInteriorRings;
+------------------+
| NumInteriorRings |
+------------------+
|                1 |
+------------------+
```

1.2.9.3.8 WKB

WKB stands for Well-Known Binary, a standard representation for geometric values.

Well-Known Binary (WKB) Format

Well-Known Binary format for representing geometric data.

AsBinary

Synonym for ST_AsBinary.

AsWKB

Synonym for ST_AsBinary.

MLineFromWKB

Constructs a MULTILINESTRING.

MPointFromWKB

Constructs a MULTIPOINT value using its WKB representation and SRID.

MPolyFromWKB

Constructs a MULTIPOLYGON value using its WKB representation and SRID.

GeomCollFromWKB

Synonym for ST_GeomCollFromWKB.

GeometryCollectionFromWKB

Synonym for ST_GeomCollFromWKB.

GeometryFromWKB

Synonym for ST_GeomFromWKB.

GeomFromWKB

Synonym for ST_GeomFromWKB.

LineFromWKB

Synonym for ST_LineFromWKB.

LineStringFromWKB

Synonym for ST_LineFromWKB.

MultiLineStringFromWKB
A synonym for MLineFromWKB.

MultiPointFromWKB
Synonym for MPointFromWKB.

MultiPolygonFromWKB
Synonym for MPolyFromWKB.

PointFromWKB
Synonym for PointFromWKB.

PolyFromWKB
Synonym for ST_PolyFromWKB.

PolygonFromWKB
Synonym for ST_PolyFromWKB.

ST_AsBinary
Converts a value to its WKB representation.

ST_AsWKB
Synonym for ST_AsBinary.

ST_GeomCollFromWKB
Constructs a GEOMETRYCOLLECTION value from a WKB.

ST_GeometryCollectionFromWKB
Synonym for ST_GeomCollFromWKB.

ST_GeometryFromWKB
Synonym for ST_GeomFromWKB.

ST_GeomFromWKB
Constructs a geometry value using its WKB representation and SRID.

ST_LineFromWKB
Constructs a LINESTRING using its WKB and SRID.

ST_LineStringFromWKB
Synonym for ST_LineFromWKB.

ST_PointFromWKB
Constructs POINT using its WKB and SRID.

ST_PolyFromWKB
Constructs POLYGON value using its WKB representation and SRID.

ST_PolygonFromWKB
Synonym for ST_PolyFromWKB.

1.2.9.3.8.1 Well-Known Binary (WKB) Format

WKB stands for Well-Known Binary, a format for representing geographical and geometrical data.

WKB uses 1-byte unsigned integers, 4-byte unsigned integers, and 8-byte double-precision numbers.

- The first byte indicates the byte order. 00 for big endian, or 01 for little endian.
- The next 4 bytes indicate the geometry type. Values from 1 to 7 indicate whether the type is Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, or GeometryCollection respectively.
- The 8-byte floats represent the co-ordinates.

Take the following example, a sequence of 21 bytes each represented by two hex digits:

```
000000000140000000000000004010000000000000
```

- It's big endian
  - 00**00000000140000000000000004010000000000000**
- It's a POINT
  - 00**00000001**40000000000000004010000000000000
- The X co-ordinate is 2.0
  - 00000000001**40000000000000000**4010000000000000
- The Y-co-ordinate is 4.0
  - 00000000014000000000000000**4010000000000000**

1.2.9.3.8.2 AsBinary

A synonym for ST_AsBinary().

1.2.9.3.8.3 AsWKB

A synonym for ST_AsBinary().

1.2.9.3.8.4 MLineFromWKB

Syntax

```
MLineFromWKB(wkb[,srid])
MultiLineStringFromWKB(wkb[,srid])
```

Description

Constructs a MULTILINESTRING value using its WKB representation and SRID.

`MLineFromWKB()` and `MultiLineStringFromWKB()` are synonyms.

Examples

```
SET @g = ST_AsBinary(MLineFromText('MULTILINESTRING((10 48,10 21,10 0),(16 0,16 23,16 48))'));

SELECT ST_AsText(MLineFromWKB(@g));
+-----------------------------------------------------+
| ST_AsText(MLineFromWKB(@g))                         |
+-----------------------------------------------------+
| MULTILINESTRING((10 48,10 21,10 0),(16 0,16 23,16 48)) |
+-----------------------------------------------------+
```

1.2.9.3.8.5 MPointFromWKB

Syntax

```
MPointFromWKB(wkb[,srid])
MultiPointFromWKB(wkb[,srid])
```

Description

Constructs a MULTIPOINT value using its WKB representation and SRID.

`MPointFromWKB()` and `MultiPointFromWKB()` are synonyms.

Examples

```
SET @g = ST_AsBinary(MPointFromText('MultiPoint( 1 1, 2 2, 5 3, 7 2, 9 3, 8 4, 6 6, 6 9, 4 9, 1 5

SELECT ST_AsText(MPointFromWKB(@g));
+-------------------------------------------------+
| ST_AsText(MPointFromWKB(@g))                    |
+-------------------------------------------------+
| MULTIPOINT(1 1,2 2,5 3,7 2,9 3,8 4,6 6,6 9,4 9,1 5) |
+-------------------------------------------------+
```

1.2.9.3.8.6 MPolyFromWKB

Syntax

```
MPolyFromWKB(wkb[,srid])
MultiPolygonFromWKB(wkb[,srid])
```

Description

Constructs a MULTIPOLYGON value using its WKB representation and SRID.

`MPolyFromWKB()` and `MultiPolygonFromWKB()` are synonyms.

Examples

```
SET @g = ST_AsBinary(MPointFromText('MULTIPOLYGON(((28 26,28 0,84 0,84 42,28 26),(52 18,66
23,73 9,48 6,52 18)),((59 18,67 18,67 13,59 13,59 18)))'));

SELECT ST_AsText(MPolyFromWKB(@g))\G
*************************** 1. row ***************************
ST_AsText(MPolyFromWKB(@g)): MULTIPOLYGON(((28 26,28 0,84 0,84 42,28 26),(52 18,66 23,73 9,48
6,52 18)),((59 18,67 18,67 13,59 13,59 18)))
```

1.2.9.3.8.7 GeomCollFromWKB

A synonym for ST_GeomCollFromWKB.

1.2.9.3.8.8 GeometryCollectionFromWKB

A synonym for ST_GeomCollFromWKB.

1.2.9.3.8.9 GeometryFromWKB

A synonym for ST_GeomFromWKB.

1.2.9.3.8.10 GeomFromWKB

A synonym for ST_GeomFromWKB.

1.2.9.3.8.11 LineFromWKB

A synonym for ST_LineFromWKB.

1.2.9.3.8.12 LineStringFromWKB

A synonym for ST_LineFromWKB.

1.2.9.3.8.13 MultiLineStringFromWKB

A synonym for MLineFromWKB().

1.2.9.3.8.14 MultiPointFromWKB

A synonym for MPointFromWKB.

1.2.9.3.8.15 MultiPolygonFromWKB

Synonym for MPolyFromWKB.

1.2.9.3.8.16 PointFromWKB

A synonym for ST_PointFromWKB.

1.2.9.3.8.17 PolyFromWKB

A synonym for ST_PolyFromWKB.

1.2.9.3.8.18 PolygonFromWKB

A synonym for ST_PolyFromWKB.

1.2.9.3.8.19 ST_AsBinary

Syntax

```
ST_AsBinary(g)
AsBinary(g)
ST_AsWKB(g)
AsWKB(g)
```

Description

Converts a value in internal geometry format to its WKB representation and returns the binary result.

`ST_AsBinary()`, `AsBinary()`, `ST_AsWKB()` and `AsWKB()` are synonyms,

Examples

```
SET @poly = ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))');
SELECT ST_AsBinary(@poly);

SELECT ST_AsText(ST_GeomFromWKB(ST_AsWKB(@poly)));
+-------------------------------------------+
| ST_AsText(ST_GeomFromWKB(ST_AsWKB(@poly))) |
+-------------------------------------------+
| POLYGON((0 0,0 1,1 1,1 0,0 0))             |
+-------------------------------------------+
```

1.2.9.3.8.20 ST_AsWKB

A synonym for ST_AsBinary().

1.2.9.3.8.21 ST_GeomCollFromWKB

Syntax

```
ST_GeomCollFromWKB(wkb[,srid])
ST_GeometryCollectionFromWKB(wkb[,srid])
GeomCollFromWKB(wkb[,srid])
GeometryCollectionFromWKB(wkb[,srid])
```

Description

Constructs a GEOMETRYCOLLECTION value using its WKB representation and SRID.

`ST_GeomCollFromWKB()`, `ST_GeometryCollectionFromWKB()`, `GeomCollFromWKB()` and `GeometryCollectionFromWKB()` are synonyms.

Examples

```
SET @g = ST_AsBinary(ST_GeomFromText('GEOMETRYCOLLECTION(
   POLYGON((5 5,10 5,10 10,5 5)),POINT(10 10))'));

SELECT ST_AsText(ST_GeomCollFromWKB(@g));
+-----------------------------------------------------------------+
| ST_AsText(ST_GeomCollFromWKB(@g))                               |
+-----------------------------------------------------------------+
| GEOMETRYCOLLECTION(POLYGON((5 5,10 5,10 10,5 5)),POINT(10 10)) |
+-----------------------------------------------------------------+
```

1.2.9.3.8.22 ST_GeometryCollectionFromWKB

A synonym for ST_GeomCollFromWKB.

1.2.9.3.8.23 ST_GeometryFromWKB

A synonym for ST_GeomFromWKB.

1.2.9.3.8.24 ST_GeomFromWKB

Syntax

```
ST_GeomFromWKB(wkb[,srid])
ST_GeometryFromWKB(wkb[,srid])
GeomFromWKB(wkb[,srid])
GeometryFromWKB(wkb[,srid])
```

Description

Constructs a geometry value of any type using its WKB representation and SRID.

`ST_GeomFromWKB()`, `ST_GeometryFromWKB()`, `GeomFromWKB()` and `GeometryFromWKB()` are synonyms.

Examples

```
SET @g = ST_AsBinary(ST_LineFromText('LINESTRING(0 4, 4 6)'));

SELECT ST_AsText(ST_GeomFromWKB(@g));
+------------------------------+
| ST_AsText(ST_GeomFromWKB(@g)) |
+------------------------------+
| LINESTRING(0 4,4 6)          |
+------------------------------+
```

1.2.9.3.8.25 ST_LineFromWKB

Syntax

```
ST_LineFromWKB(wkb[,srid])
LineFromWKB(wkb[,srid])
ST_LineStringFromWKB(wkb[,srid])
LineStringFromWKB(wkb[,srid])
```

Description

Constructs a LINESTRING value using its WKB representation and SRID.

`ST_LineFromWKB()`, `LineFromWKB()`, `ST_LineStringFromWKB()`, and `LineStringFromWKB()` are synonyms.

Examples

```
SET @g = ST_AsBinary(ST_LineFromText('LineString(0 4,4 6)'));

SELECT ST_AsText(ST_LineFromWKB(@g)) AS l;
+---------------------+
| l                   |
+---------------------+
| LINESTRING(0 4,4 6) |
+---------------------+
```

1.2.9.3.8.26 ST_LineStringFromWKB

A synonym for ST_LineFromWKB.

1.2.9.3.8.27 ST_PointFromWKB

Syntax

```
ST_PointFromWKB(wkb[,srid])
PointFromWKB(wkb[,srid])
```

Description

Constructs a POINT value using its WKB representation and SRID.

`ST_PointFromWKB()` and `PointFromWKB()` are synonyms.

Examples

```
SET @g = ST_AsBinary(ST_PointFromText('POINT(0 4)'));

SELECT ST_AsText(ST_PointFromWKB(@g)) AS p;
+------------+
| p          |
+------------+
| POINT(0 4) |
+------------+
```

1.2.9.3.8.28 ST_PolyFromWKB

Syntax

```
ST_PolyFromWKB(wkb[,srid])
ST_PolygonFromWKB(wkb[,srid])
PolyFromWKB(wkb[,srid])
PolygonFromWKB(wkb[,srid])
```

Description

Constructs a POLYGON value using its WKB representation and SRID.

`ST_PolyFromWKB()`, `ST_PolygonFromWKB()`, `PolyFromWKB()` and `PolygonFromWKB()` are synonyms.

Examples

```
SET @g = ST_AsBinary(ST_PolyFromText('POLYGON((1 1,1 5,4 9,6 9,9 3,7 2,1 1))'));

SELECT ST_AsText(ST_PolyFromWKB(@g)) AS p;
+--------------------------------------+
| p                                    |
+--------------------------------------+
| POLYGON((1 1,1 5,4 9,6 9,9 3,7 2,1 1)) |
+--------------------------------------+
```

1.2.9.3.8.29 ST_PolygonFromWKB

A synonym for ST_PolyFromWKB.

1.2.9.3.9 WKT

The Well-Known Text (WKT) representation of Geometry is designed to exchange geometry data in ASCII form. This section has articles on WKT in MariaDB.

WKT Definition
Well-Known Text for exchanging geometry data in ASCII form.

AsText
Synonym for ST_AsText.

AsWKT
Synonym for ST_AsText.

GeomCollFromText
Synonym for ST_GeomCollFromText.

GeometryCollectionFromText
Synonym for ST_GeomCollFromText.

GeometryFromText
Synonym for ST_GeomFromText.

GeomFromText
Synonym for ST_GeomFromText.

LineFromText
Synonym for ST_LineFromText.

LineStringFromText
Synonym for ST_LineFromText.

MLineFromText
Constructs MULTILINESTRING using its WKT representation and SRID.

MPointFromText
Constructs a MULTIPOINT value using its WKT and SRID.

MPolyFromText
Constructs a MULTIPOLYGON value.

MultiLineStringFromText
Synonym for MLineFromText.

MultiPointFromText
Synonym for MPointFromText.

MultiPolygonFromText
Synonym for MPolyFromText.

PointFromText
Synonym for ST_PointFromText.

PolyFromText
Synonym for ST_PolyFromText.

PolygonFromText
Synonym for ST_PolyFromText.

ST_AsText
Converts a value to its WKT-Definition.

ST_ASWKT
Synonym for ST_ASTEXT().

ST_GeomCollFromText
Constructs a GEOMETRYCOLLECTION value.

ST_GeometryCollectionFromText
Synonym for ST_GeomCollFromText.

ST_GeometryFromText
Synonym for ST_GeomFromText.

ST_GeomFromText
Constructs a geometry value using its WKT and SRID.

ST_LineFromText
Creates a linestring value.

ST_LineStringFromText
Synonym for ST_LineFromText.

ST_PointFromText
Constructs a POINT value.

ST_PolyFromText
Constructs a POLYGON value.

ST_PolygonFromText
Synonym for ST_PolyFromText.

1.2.9.3.9.1 WKT Definition

Description

The Well-Known Text (WKT) representation of Geometry is designed to exchange geometry data in ASCII form. Examples of the basic geometry types include:

Geometry Types
POINT
LINESTRING
POLYGON
MULTIPOINT
MULTILINESTRING
MULTIPOLYGON
GEOMETRYCOLLECTION
GEOMETRY

1.2.9.3.9.2 AsText

A synonym for ST_AsText().

1.2.9.3.9.3 AsWKT

A synonym for ST_AsText().

1.2.9.3.9.4 GeomCollFromText

A synonym for ST_GeomCollFromText.

1.2.9.3.9.5 GeometryCollectionFromText

A synonym for ST_GeomCollFromText.

1.2.9.3.9.6 GeometryFromText

A synonym for ST_GeomFromText.

1.2.9.3.9.7 GeomFromText

A synonym for ST_GeomFromText.

1.2.9.3.9.8 LineFromText

A synonym for ST_LineFromText.

1.2.9.3.9.9 LineStringFromText

A synonym for ST_LineFromText.

1.2.9.3.9.10 MLineFromText

Syntax

```
MLineFromText(wkt[,srid])
MultiLineStringFromText(wkt[,srid])
```

Description

Constructs a MULTILINESTRING value using its WKT representation and SRID.

`MLineFromText()` and `MultiLineStringFromText()` are synonyms.

Examples

```
CREATE TABLE gis_multi_line (g MULTILINESTRING);
SHOW FIELDS FROM gis_multi_line;
INSERT INTO gis_multi_line VALUES
    (MultiLineStringFromText('MULTILINESTRING((10 48,10 21,10 0),(16 0,16 23,16 48))')),
    (MLineFromText('MULTILINESTRING((10 48,10 21,10 0))')),
    (MLineFromWKB(AsWKB(MultiLineString(
      LineString(Point(1, 2), Point(3, 5)),
      LineString(Point(2, 5), Point(5, 8), Point(21, 7))))));
```

1.2.9.3.9.11 MPointFromText

Syntax

```
MPointFromText(wkt[,srid])
MultiPointFromText(wkt[,srid])
```

Description

Constructs a MULTIPOINT value using its WKT representation and SRID.

`MPointFromText()` and `MultiPointFromText()` are synonyms.

Examples

```
CREATE TABLE gis_multi_point (g MULTIPOINT);
SHOW FIELDS FROM gis_multi_point;
INSERT INTO gis_multi_point VALUES
    (MultiPointFromText('MULTIPOINT(0 0,10 10,10 20,20 20)')),
    (MPointFromText('MULTIPOINT(1 1,11 11,11 21,21 21)')),
    (MPointFromWKB(AsWKB(MultiPoint(Point(3, 6), Point(4, 10)))));
```

1.2.9.3.9.12 MPolyFromText

Syntax

```
MPolyFromText(wkt[,srid])
MultiPolygonFromText(wkt[,srid])
```

Description

Constructs a MULTIPOLYGON value using its WKT representation and SRID.

`MPolyFromText()` and `MultiPolygonFromText()` are synonyms.

Examples

```
CREATE TABLE gis_multi_polygon  (g MULTIPOLYGON);
SHOW FIELDS FROM gis_multi_polygon;
INSERT INTO gis_multi_polygon VALUES
    (MultiPolygonFromText('MULTIPOLYGON(
       ((28 26,28 0,84 0,84 42,28 26),(52 18,66 23,73 9,48 6,52 18)),
       ((59 18,67 18,67 13,59 13,59 18)))')),
    (MPolyFromText('MULTIPOLYGON(
       ((28 26,28 0,84 0,84 42,28 26),(52 18,66 23,73 9,48 6,52 18)),
       ((59 18,67 18,67 13,59 13,59 18)))')),
    (MPolyFromWKB(AsWKB(MultiPolygon(Polygon(
       LineString(Point(0, 3), Point(3, 3), Point(3, 0), Point(0, 3)))))));
```

1.2.9.3.9.13 MultiLineStringFromText

A synonym for MLineFromText.

1.2.9.3.9.14 MultiPointFromText

A synonym for MPointFromText.

1.2.9.3.9.15 MultiPolygonFromText

A synonym for MPolyFromText.

1.2.9.3.9.16 PointFromText

A synonym for ST_PointFromText.

1.2.9.3.9.17 PolyFromText

A synonym for ST_PolyFromText.

1.2.9.3.9.18 PolygonFromText

A synonym for ST_PolyFromText.

1.2.9.3.9.19 ST_AsText

Syntax

```
ST_AsText(g)
AsText(g)
ST_AsWKT(g)
AsWKT(g)
```

Description

Converts a value in internal geometry format to its WKT representation and returns the string result.

`ST_AsText()`, `AsText()`, `ST_AsWKT()` and `AsWKT()` are all synonyms.

Examples

```
SET @g = 'LineString(1 1,4 4,6 6)';

SELECT ST_AsText(ST_GeomFromText(@g));
+--------------------------------+
| ST_AsText(ST_GeomFromText(@g)) |
+--------------------------------+
| LINESTRING(1 1,4 4,6 6)        |
+--------------------------------+
```

1.2.9.3.9.20 ST_ASWKT

A synonym for ST_ASTEXT().

1.2.9.3.9.21 ST_GeomCollFromText

Syntax

```
ST_GeomCollFromText(wkt[,srid])
ST_GeometryCollectionFromText(wkt[,srid])
GeomCollFromText(wkt[,srid])
GeometryCollectionFromText(wkt[,srid])
```

Description

Constructs a GEOMETRYCOLLECTION value using its WKT representation and SRID.

`ST_GeomCollFromText()`, `ST_GeometryCollectionFromText()`, `GeomCollFromText()` and `GeometryCollectionFromText()` are all synonyms.

Example

```
CREATE TABLE gis_geometrycollection  (g GEOMETRYCOLLECTION);
SHOW FIELDS FROM gis_geometrycollection;
INSERT INTO gis_geometrycollection VALUES
    (GeomCollFromText('GEOMETRYCOLLECTION(POINT(0 0), LINESTRING(0 0,10 10))')),
    (GeometryFromWKB(AsWKB(GeometryCollection(Point(44, 6), LineString(Point(3, 6), Point(7, 9))))),
    (GeomFromText('GeometryCollection()')),
    (GeomFromText('GeometryCollection EMPTY'));
```

1.2.9.3.9.22 ST_GeometryCollectionFromText

A synonym for ST_GeomCollFromText.

1.2.9.3.9.23 ST_GeometryFromText

A synonym for ST_GeomFromText.

1.2.9.3.9.24 ST_GeomFromText

Syntax

```
ST_GeomFromText(wkt[,srid])
ST_GeometryFromText(wkt[,srid])
GeomFromText(wkt[,srid])
GeometryFromText(wkt[,srid])
```

Description

Constructs a geometry value of any type using its WKT representation and SRID.

`GeomFromText()`, `GeometryFromText()`, `ST_GeomFromText()` and `ST_GeometryFromText()` are all synonyms.

Example

```
SET @g = ST_GEOMFROMTEXT('POLYGON((1 1,1 5,4 9,6 9,9 3,7 2,1 1))');
```

1.2.9.3.9.25 ST_LineFromText

Syntax

```
ST_LineFromText(wkt[,srid])
ST_LineStringFromText(wkt[,srid])
LineFromText(wkt[,srid])
LineStringFromText(wkt[,srid])
```

Description

Constructs a LINESTRING value using its WKT representation and SRID.

`ST_LineFromText()`, `ST_LineStringFromText()`, `ST_LineFromText()` and `ST_LineStringFromText()` are all synonyms.

Examples

```
CREATE TABLE gis_line  (g LINESTRING);
SHOW FIELDS FROM gis_line;
INSERT INTO gis_line VALUES
    (LineFromText('LINESTRING(0 0,0 10,10 0)')),
    (LineStringFromText('LINESTRING(10 10,20 10,20 20,10 20,10 10)')),
    (LineStringFromWKB(AsWKB(LineString(Point(10, 10), Point(40, 10)))));
```

1.2.9.3.9.26 ST_LineStringFromText

A synonym for ST_LineFromText.

1.2.9.3.9.27 ST_PointFromText

Syntax

```
ST_PointFromText(wkt[,srid])
PointFromText(wkt[,srid])
```

Description

Constructs a POINT value using its WKT representation and SRID.

`ST_PointFromText()` and `PointFromText()` are synonyms.

Examples

```
CREATE TABLE gis_point  (g POINT);
SHOW FIELDS FROM gis_point;
INSERT INTO gis_point VALUES
    (PointFromText('POINT(10 10)')),
    (PointFromText('POINT(20 10)')),
    (PointFromText('POINT(20 20)')),
    (PointFromWKB(AsWKB(PointFromText('POINT(10 20)'))));
```

1.2.9.3.9.28 ST_PolyFromText

Syntax

```
ST_PolyFromText(wkt[,srid])
ST_PolygonFromText(wkt[,srid])
PolyFromText(wkt[,srid])
PolygonFromText(wkt[,srid])
```

Description

Constructs a POLYGON value using its WKT representation and SRID.

`ST_PolyFromText()`, `ST_PolygonFromText()`, `PolyFromText()` and `ST_PolygonFromText()` are all synonyms.

Examples

```
CREATE TABLE gis_polygon   (g POLYGON);
INSERT INTO gis_polygon VALUES
    (PolygonFromText('POLYGON((10 10,20 10,20 20,10 20,10 10))')),
    (PolyFromText('POLYGON((0 0,50 0,50 50,0 50,0 0), (10 10,20 10,20 20,10 20,10 10))'));
```

1.2.9.3.9.29 ST_PolygonFromText

A synonym for ST_PolyFromText.

1.2.9.4 JSON Functions

Functions relating to JSON, such as JSON_VALUE, JSON_ARRAY etc.

Differences between JSON_QUERY and JSON_VALUE

Comparison between and examples with JSON_QUERY and JSON_VALUE.

JSONPath Expressions

MariaDB JSONPath description and definition.

JSON_ARRAY

Returns a JSON array containing the listed values.

JSON_ARRAYAGG

Returns a JSON array containing an element for each value in a given set of JSON or SQL values.

JSON_ARRAY_APPEND

Appends values to the end of the given arrays within a JSON document.

JSON_ARRAY_INSERT

Inserts a value into a JSON document.

JSON_ARRAY_INTERSECT

Finds intersection between two arrays.

JSON_COMPACT

Removes all unnecessary spaces so the json document is as short as possible.

JSON_CONTAINS

Whether a value is found in a given JSON document or at a specified path within the document.

JSON_CONTAINS_PATH

Indicates whether the given JSON document contains data at the specified path or paths.

JSON_DEPTH

Maximum depth of a JSON document.

JSON_DETAILED

Represents JSON in the most understandable way emphasizing nested structures.

JSON_EQUALS

Checks if there is equality between two json objects.

JSON_EXISTS

Determines whether a specified JSON value exists in the given data.

JSON_EXTRACT

Extracts data from a JSON document.

JSON_INSERT

Inserts data into a JSON document.

JSON_KEYS

Returns keys from top-level value of a JSON object or top-level keys from the path.

JSON_LENGTH

Returns the length of a JSON document, or the length of a value within the document.

JSON_LOOSE

Adds spaces to a JSON document to make it look more readable.

JSON_MERGE

Merges the given JSON documents.

JSON_MERGE_PATCH

RFC 7396-compliant merge of the given JSON documents.

JSON_MERGE_PRESERVE

Synonym for JSON_MERGE.

JSON_NORMALIZE

Recursively sorts keys and removes spaces, allowing comparison of json documents for equality.

JSON_OBJECT

Returns a JSON object containing the given key/value pairs.

JSON_OBJECT_FILTER_KEYS

Returns keys from the object that are present in the array.

JSON_OBJECT_TO_ARRAY

Converts JSON objects to JSON array where each item represents each key-value pair from object.

JSON_OBJECTAGG

Returns a JSON object containing key-value pairs.

JSON_OVERLAPS

Returns true if two json documents have at least one key-value pair or array element in common.

JSON_PRETTY

Alias for JSON_DETAILED.

JSON_QUERY

Given a JSON document, returns an object or array specified by the path.

JSON_QUOTE

Quotes a string as a JSON value.

JSON_REMOVE

Removes data from a JSON document.

JSON_REPLACE

Replaces existing values in a JSON document.

JSON_SCHEMA_VALID

JSON schema validation.

JSON_SEARCH

Returns the path to the given string within a JSON document.

JSON_SET

Updates or inserts data into a JSON document.

JSON_TABLE

Given data from a JSON document, returns a representation of it as a relational table.

JSON_TYPE

Returns the type of a JSON value.

JSON_UNQUOTE

Unquotes a JSON value, returning a string.

JSON_VALID

Whether a value is a valid JSON document or not.

JSON_VALUE

Given a JSON document, returns the specified scalar.

There are 6 related questions.

1.2.9.4.1 Differences between JSON_QUERY and JSON_VALUE

The primary difference between the two functions is that *JSON_QUERY* returns an object or an array, while *JSON_VALUE* returns a scalar.

Take the following JSON document as an example

```
SET @json='{ "x": [0,1], "y": "[0,1]", "z": "Monty" }';
```

Note that data member "x" is an array, and data members "y" and "z" are strings. The following examples demonstrate the differences between the two functions.

```
SELECT JSON_QUERY(@json,'$'), JSON_VALUE(@json,'$');
+-----------------------------------------+-----------------------+
| JSON_QUERY(@json,'$')                   | JSON_VALUE(@json,'$') |
+-----------------------------------------+-----------------------+
| { "x": [0,1], "y": "[0,1]", "z": "Monty" } | NULL                  |
+-----------------------------------------+-----------------------+

SELECT JSON_QUERY(@json,'$.x'), JSON_VALUE(@json,'$.x');
+-------------------------+-------------------------+
| JSON_QUERY(@json,'$.x') | JSON_VALUE(@json,'$.x') |
+-------------------------+-------------------------+
| [0,1]                   | NULL                    |
+-------------------------+-------------------------+

SELECT JSON_QUERY(@json,'$.y'), JSON_VALUE(@json,'$.y');
+-------------------------+-------------------------+
| JSON_QUERY(@json,'$.y') | JSON_VALUE(@json,'$.y') |
+-------------------------+-------------------------+
| NULL                    | [0,1]                   |
+-------------------------+-------------------------+

SELECT JSON_QUERY(@json,'$.z'), JSON_VALUE(@json,'$.z');
+-------------------------+-------------------------+
| JSON_QUERY(@json,'$.z') | JSON_VALUE(@json,'$.z') |
+-------------------------+-------------------------+
| NULL                    | Monty                   |
+-------------------------+-------------------------+

SELECT JSON_QUERY(@json,'$.x[0]'), JSON_VALUE(@json,'$.x[0]');
+----------------------------+----------------------------+
| JSON_QUERY(@json,'$.x[0]') | JSON_VALUE(@json,'$.x[0]') |
+----------------------------+----------------------------+
| NULL                       | 0                          |
+----------------------------+----------------------------+
```

1.2.9.4.2 JSONPath Expressions

Contents

A number of JSON functions accept JSON Path expressions. MariaDB defines this path as follows:

JSON Path Syntax

```
path : ['lax'] '$' [step]*
```

The path starts with an optional *path mode*. At the moment, MariaDB supports only the "lax" mode, which is also the mode that is used when it is not explicitly specified.

The `$` symbol represents the context item. The search always starts from the context item; because of that, the path always starts with `$`.

Then, it is followed by zero or more steps, which select element(s) in the JSON document. A step may be one of the following:

- Object member selector
- Array element selector
- Wildcard selector

Object Member Selector

To select member(s) in a JSON object, one can use one of the following:

- `.memberName` selects the value of the member with name memberName.
- `."memberName"` - the same as above but allows one to select a member with a name that's not a valid identifier (that is, has space, dot, and/or other characters)
- `.*` - selects the values of all members of the object.

If the current item is an array (instead of an object), nothing will be selected.

Array Element Selector

To select elements of an array, one can use one of the following:

- `[N]` selects element number N in the array. The elements are counted from zero.
- `[*]` selects all elements in the array.

If the current item is an object (instead of an array), nothing will be selected.

Starting from MariaDB server 10.9, JSON path also supports negative index in array, 'last' keyword and range notation ('to' keyword) for accessing array elements. Negative index starts from -1.

- `[-N]` selects n th element from end.
- `[last-N]` selects n th element from the last element.
- `[M to N]` selects range of elements starting from index M to N.

Example:

```
SET @json='{
            "A": [0,
                  [1, 2, 3],
                  [4, 5, 6],
                  "seven",
                   0.8,
                   true,
                   false,
                   "eleven",
                  [12, [13, 14], {"key1":"value1"},[15]],
                  true],
            "B": {"C": 1},
            "D": 2
           }';
SELECT JSON_EXTRACT(@json, '$.A[-8][1]');
+---------------------------------------------------+
| JSON_EXTRACT(@json, '$.A[-8][1]')                 |
+---------------------------------------------------+
| 5                                                 |
+---------------------------------------------------+

SELECT JSON_EXTRACT(@json, '$.A[last-7][1]');
+-----------------------------------------------+
| SELECT JSON_EXTRACT(@json, '$.A[last-7][1]'); |
+-----------------------------------------------+
| 5                                             |
+-----------------------------------------------+


SET @json= '[
             [1, {"key1": "value1"}, 3],
             [false, 5, 6],
             [7, 8, [9, {"key2": 2}, 11]],
             [15, 1.34, [14], ["string1", [16, {"key1":[1,2,3,[4,5,6]]}, 18]]],
             [19, 20],
             21, 22
            ]';

SELECT JSON_EXTRACT(@json, '$[0 to 3][2]');
+-----------------------------------------------+
| JSON_EXTRACT(@json, '$[0 to 3][2]')           |
+-----------------------------------------------+
| [3, 6, [9, {"key2": 2}, 11], [14]]            |
+-----------------------------------------------+
```

This will produce output for first index of eighth from last element of a two dimensional array.

Note: In range notation, when M > N (when M,N are greater than or equal to 0) or (size of array - M or size of array - N when M, N are less than 0), then it is treated as an impossible range and NULL is returned.

```
SET @json= '[1, 2, 3, 4, 5]';
SELECT JSON_EXTRACT(@json, '$[4 to 2]');
+----------------------------------+
| JSON_EXTRACT(@json, '$[4 to 2]') |
+----------------------------------+
| NULL                             |
+----------------------------------+
```

Wildcard

The wildcard step, `**`, recursively selects all child elements of the current element. Both array elements and object members are selected.

The wildcard step must not be the last step in the JSONPath expression. It must be followed by an array or object member selector step.

For example:

```
select json_extract(@json_doc, '$**.price');
```

will select all object members in the document that are named `price`, while

```
select json_extract(@json_doc, '$**[2]');
```

will select the second element in each of the arrays present in the document.

Compatibility

MariaDB's JSONPath syntax supports a subset of JSON Path's definition in the SQL Standard. The most notable things not supported are the `strict` mode and filters.

MariaDB's JSONPath is close to MySQL's JSONPath. The wildcard step (`**`) is a non-standard extension that has the same meaning as in MySQL. The difference between MariaDB and MySQL's JSONPath is: MySQL doesn't allow one to specify the mode explicitly (but uses `lax` mode implicitly).

1.2.9.4.3 JSON_ARRAY

Syntax

```
JSON_ARRAY([value[, value2] ...])
```

Description

Returns a JSON array containing the listed values. The list can be empty.

Example

```
SELECT Json_Array(56, 3.1416, 'My name is "Foo"', NULL);
+--------------------------------------------------+
| Json_Array(56, 3.1416, 'My name is "Foo"', NULL) |
+--------------------------------------------------+
| [56, 3.1416, "My name is \"Foo\"", null]         |
+--------------------------------------------------+
```

1.2.9.4.4 JSON_ARRAYAGG

1.2.9.4.5 JSON_ARRAY_APPEND

Syntax

```
JSON_ARRAY_APPEND(json_doc, path, value[, path, value] ...)
```

Description

Appends values to the end of the specified arrays within a JSON document, returning the result, or NULL if any of the arguments are NULL.

Evaluation is performed from left to right, with the resulting document from the previous pair becoming the new value against which the next pair is evaluated.

If the `json_doc` is not a valid JSON document, or if any of the paths are not valid, or contain a `*` or `**` wildcard, an error is returned.

Examples

```
SET @json = '[1, 2, [3, 4]]';

SELECT JSON_ARRAY_APPEND(@json, '$[0]', 5)
+-------------------------------------+
| JSON_ARRAY_APPEND(@json, '$[0]', 5) |
+-------------------------------------+
| [[1, 5], 2, [3, 4]]                 |
+-------------------------------------+

SELECT JSON_ARRAY_APPEND(@json, '$[1]', 6);
+-------------------------------------+
| JSON_ARRAY_APPEND(@json, '$[1]', 6) |
+-------------------------------------+
| [1, [2, 6], [3, 4]]                 |
+-------------------------------------+

SELECT JSON_ARRAY_APPEND(@json, '$[1]', 6, '$[2]', 7);
+-----------------------------------------------+
| JSON_ARRAY_APPEND(@json, '$[1]', 6, '$[2]', 7) |
+-----------------------------------------------+
| [1, [2, 6], [3, 4, 7]]                        |
+-----------------------------------------------+

SELECT JSON_ARRAY_APPEND(@json, '$', 5);
+----------------------------------+
| JSON_ARRAY_APPEND(@json, '$', 5) |
+----------------------------------+
| [1, 2, [3, 4], 5]                |
+----------------------------------+

SET @json = '{"A": 1, "B": [2], "C": [3, 4]}';

SELECT JSON_ARRAY_APPEND(@json, '$.B', 5);
+-----------------------------------+
| JSON_ARRAY_APPEND(@json, '$.B', 5) |
+-----------------------------------+
| {"A": 1, "B": [2, 5], "C": [3, 4]} |
+-----------------------------------+
```

1.2.9.4.6 JSON_ARRAY_INSERT

Syntax

```
JSON_ARRAY_INSERT(json_doc, path, value[, path, value] ...)
```

Description

Inserts a value into a JSON document, returning the modified document, or NULL if any of the arguments are NULL.

Evaluation is performed from left to right, with the resulting document from the previous pair becoming the new value against which the next pair is evaluated.

If the `json_doc` is not a valid JSON document, or if any of the paths are not valid, or contain a `*` or `**` wildcard, an error is returned.

Examples

```
SET @json = '[1, 2, [3, 4]]';

SELECT JSON_ARRAY_INSERT(@json, '$[0]', 5);
+-----------------------------------+
| JSON_ARRAY_INSERT(@json, '$[0]', 5) |
+-----------------------------------+
| [5, 1, 2, [3, 4]]                 |
+-----------------------------------+

SELECT JSON_ARRAY_INSERT(@json, '$[1]', 6);
+-----------------------------------+
| JSON_ARRAY_INSERT(@json, '$[1]', 6) |
+-----------------------------------+
| [1, 6, 2, [3, 4]]                 |
+-----------------------------------+

SELECT JSON_ARRAY_INSERT(@json, '$[1]', 6, '$[2]', 7);
+-----------------------------------------------+
| JSON_ARRAY_INSERT(@json, '$[1]', 6, '$[2]', 7) |
+-----------------------------------------------+
| [1, 6, 7, 2, [3, 4]]                          |
+-----------------------------------------------+
```

1.2.9.4.7 JSON_ARRAY_INTERSECT

MariaDB starting with 11.2.0

JSON_ARRAY_INTERSECT was added in MariaDB 11.2.0.

Syntax

```
JSON_ARRAY_INTERSECT(arr1, arr2)
```

Description

Finds intersection between two json arrays and returns an array of items found in both array.

Examples

```
SET @json1= '[1,2,3]';
SET @json2= '[1,2,4]';

SELECT json_array_intersect(@json1, @json2);
+-------------------------------------+
| json_array_intersect(@json1, @json2) |
+-------------------------------------+
| [1, 2]                              |
+-------------------------------------+
```

1.2.9.4.8 JSON_COMPACT

Syntax

```
JSON_COMPACT(json_doc)
```

Contents

Description

Removes all unnecessary spaces so the json document is as short as possible.

Example

```
SET @j = '{ "A": 1, "B": [2, 3]}';

SELECT JSON_COMPACT(@j), @j;
+------------------+-----------------------+
| JSON_COMPACT(@j)  | @j                    |
+------------------+-----------------------+
| {"A":1,"B":[2,3]} | { "A": 1, "B": [2, 3]} |
+------------------+-----------------------+
```

1.2.9.4.9 JSON_CONTAINS

Syntax

```
JSON_CONTAINS(json_doc, val[, path])
```

Description

Returns whether or not the specified value is found in the given JSON document or, optionally, at the specified path within the document. Returns `1` if it does, `0` if not and NULL if any of the arguments are null. An error occurs if the document or path is not valid, or contains the `*` or `**` wildcards.

Examples

```
SET @json = '{"A": 0, "B": {"C": 1}, "D": 2}';

SELECT JSON_CONTAINS(@json, '2', '$.A');
+---------------------------------+
| JSON_CONTAINS(@json, '2', '$.A') |
+---------------------------------+
|                               0 |
+---------------------------------+

SELECT JSON_CONTAINS(@json, '2', '$.D');
+---------------------------------+
| JSON_CONTAINS(@json, '2', '$.D') |
+---------------------------------+
|                               1 |
+---------------------------------+

SELECT JSON_CONTAINS(@json, '{"C": 1}', '$.A');
+-------------------------------------+
| JSON_CONTAINS(@json, '{"C": 1}', '$.A') |
+-------------------------------------+
|                                   0 |
+-------------------------------------+

SELECT JSON_CONTAINS(@json, '{"C": 1}', '$.B');
+-------------------------------------+
| JSON_CONTAINS(@json, '{"C": 1}', '$.B') |
+-------------------------------------+
|                                   1 |
+-------------------------------------+
```

1.2.9.4.10 JSON_CONTAINS_PATH

Syntax

```
JSON_CONTAINS_PATH(json_doc, return_arg, path[, path] ...)
```

Description

Indicates whether the given JSON document contains data at the specified path or paths. Returns `1` if it does, `0` if not and NULL if any of the arguments are null.

The *return_arg* can be `one` or `all`:

- `one` - Returns `1` if at least one path exists within the JSON document.
- `all` - Returns `1` only if all paths exist within the JSON document.

Examples

```
SET @json = '{"A": 1, "B": [2], "C": [3, 4]}';

SELECT JSON_CONTAINS_PATH(@json, 'one', '$.A', '$.D');
+-----------------------------------------------+
| JSON_CONTAINS_PATH(@json, 'one', '$.A', '$.D') |
+-----------------------------------------------+
|                                             1 |
+-----------------------------------------------+
1 row in set (0.00 sec)

SELECT JSON_CONTAINS_PATH(@json, 'all', '$.A', '$.D');
+-----------------------------------------------+
| JSON_CONTAINS_PATH(@json, 'all', '$.A', '$.D') |
+-----------------------------------------------+
|                                             0 |
+-----------------------------------------------+
```

1.2.9.4.11 JSON_DEPTH

Syntax

```
JSON_DEPTH(json_doc)
```

Description

Returns the maximum depth of the given JSON document, or NULL if the argument is null. An error will occur if the argument is an invalid JSON document.

- Scalar values or empty arrays or objects have a depth of 1.
- Arrays or objects that are not empty but contain only elements or member values of depth 1 will have a depth of 2.
- In other cases, the depth will be greater than 2.

Examples

```
SELECT JSON_DEPTH('[]'), JSON_DEPTH('true'), JSON_DEPTH('{}');
+------------------+-------------------+------------------+
| JSON_DEPTH('[]') | JSON_DEPTH('true') | JSON_DEPTH('{}') |
+------------------+-------------------+------------------+
|                1 |                 1 |                1 |
+------------------+-------------------+------------------+

SELECT JSON_DEPTH('[1, 2, 3]'), JSON_DEPTH('[[], {}, []]');
+-------------------------+----------------------------+
| JSON_DEPTH('[1, 2, 3]') | JSON_DEPTH('[[], {}, []]') |
+-------------------------+----------------------------+
|                       2 |                          2 |
+-------------------------+----------------------------+

SELECT JSON_DEPTH('[1, 2, [3, 4, 5, 6], 7]');
+---------------------------------------+
| JSON_DEPTH('[1, 2, [3, 4, 5, 6], 7]') |
+---------------------------------------+
|                                     3 |
+---------------------------------------+
```

1.2.9.4.12 JSON_DETAILED

Syntax

```
JSON_DETAILED(json_doc[, tab_size])
JSON_PRETTY(json_doc[, tab_size])
```

Contents

Description

Represents JSON in the most understandable way emphasizing nested structures.

`JSON_PRETTY` was added as an alias for `JSON_DETAILED` in MariaDB 10.10.3, MariaDB 10.9.5, MariaDB 10.8.7, MariaDB 10.7.8, MariaDB 10.6.12, MariaDB 10.5.19 and MariaDB 10.4.28.

Example

```
SET @j = '{ "A":1,"B":[2,3]}';

SELECT @j;
+--------------------+
| @j                 |
+--------------------+
| { "A":1,"B":[2,3]} |
+--------------------+

SELECT JSON_DETAILED(@j);
+------------------------------------------------------------+
| JSON_DETAILED(@j)                                          |
+------------------------------------------------------------+
| {
    "A": 1,
    "B":
    [
        2,
        3
    ]
} |
+------------------------------------------------------------+
```

1.2.9.4.13 JSON_EQUALS

MariaDB starting with 10.7.0
JSON_EQUALS was added in MariaDB 10.7.0

Syntax

```
JSON_EQUALS(json1, json2)
```

Description

Checks if there is equality between two json objects. Returns `1` if it there is, `0` if not, or NULL if any of the arguments are null.

Examples

```
SELECT JSON_EQUALS('{"a"   :[1, 2, 3],"b":[4]}', '{"b":[4],"a":[1, 2, 3.0]}');
+-----------------------------------------------------------------------+
| JSON_EQUALS('{"a"   :[1, 2, 3],"b":[4]}', '{"b":[4],"a":[1, 2, 3.0]}') |
+-----------------------------------------------------------------------+
|                                                                     1 |
+-----------------------------------------------------------------------+

SELECT JSON_EQUALS('{"a":[1, 2, 3]}', '{"a":[1, 2, 3.01]}');
+-------------------------------------------------------+
| JSON_EQUALS('{"a":[1, 2, 3]}', '{"a":[1, 2, 3.01]}') |
+-------------------------------------------------------+
|                                                     0 |
+-------------------------------------------------------+
```

1.2.9.4.14 JSON_EXISTS

Syntax

Description

Determines whether a specified JSON value exists in the given data. Returns `1` if found, `0` if not, or `NULL` if any of the inputs were NULL.

Examples

```
SELECT JSON_EXISTS('{"key1":"xxxx", "key2":[1, 2, 3]}', "$.key2");
+-----------------------------------------------------------+
| JSON_EXISTS('{"key1":"xxxx", "key2":[1, 2, 3]}', "$.key2") |
+-----------------------------------------------------------+
|                                                         1 |
+-----------------------------------------------------------+

SELECT JSON_EXISTS('{"key1":"xxxx", "key2":[1, 2, 3]}', "$.key3");
+-----------------------------------------------------------+
| JSON_EXISTS('{"key1":"xxxx", "key2":[1, 2, 3]}', "$.key3") |
+-----------------------------------------------------------+
|                                                         0 |
+-----------------------------------------------------------+

SELECT JSON_EXISTS('{"key1":"xxxx", "key2":[1, 2, 3]}', "$.key2[1]");
+-------------------------------------------------------------+
| JSON_EXISTS('{"key1":"xxxx", "key2":[1, 2, 3]}', "$.key2[1]") |
+-------------------------------------------------------------+
|                                                           1 |
+-------------------------------------------------------------+

SELECT JSON_EXISTS('{"key1":"xxxx", "key2":[1, 2, 3]}', "$.key2[10]");
+--------------------------------------------------------------+
| JSON_EXISTS('{"key1":"xxxx", "key2":[1, 2, 3]}', "$.key2[10]") |
+--------------------------------------------------------------+
|                                                            0 |
+--------------------------------------------------------------+
```

1.2.9.4.15 JSON_EXTRACT

Contents

Syntax

```
JSON_EXTRACT(json_doc, path[, path] ...)
```

Description

Extracts data from a JSON document. The extracted data is selected from the parts matching the path arguments. Returns all matched values; either as a single matched value, or, if the arguments could return multiple values, a result autowrapped as an array in the matching order.

Returns NULL if no paths match or if any of the arguments are NULL.

An error will occur if any path argument is not a valid path, or if the json_doc argument is not a valid JSON document.

The path expression be a JSONPath expression as supported by MariaDB

Examples

```
SET @json = '[1, 2, [3, 4]]';

SELECT JSON_EXTRACT(@json, '$[1]');
+----------------------------+
| JSON_EXTRACT(@json, '$[1]') |
+----------------------------+
| 2                          |
+----------------------------+

SELECT JSON_EXTRACT(@json, '$[2]');
+----------------------------+
| JSON_EXTRACT(@json, '$[2]') |
+----------------------------+
| [3, 4]                     |
+----------------------------+

SELECT JSON_EXTRACT(@json, '$[2][1]');
+-------------------------------+
| JSON_EXTRACT(@json, '$[2][1]') |
+-------------------------------+
| 4                             |
+-------------------------------+
```

1.2.9.4.16 JSON_INSERT

Syntax

```
JSON_INSERT(json_doc, path, val[, path, val] ...)
```

Contents

Description

Inserts data into a JSON document, returning the resulting document or NULL if either of the *json_doc* or *path* arguments are null.

An error will occur if the JSON document is invalid, or if any of the paths are invalid or contain a `*` or `**` wildcard.

JSON_INSERT can only insert data while JSON_REPLACE can only update. JSON_SET can update or insert data.

Examples

```
SET @json = '{ "A": 0, "B": [1, 2]}';

SELECT JSON_INSERT(@json, '$.C', '[3, 4]');
+-------------------------------------+
| JSON_INSERT(@json, '$.C', '[3, 4]')  |
+-------------------------------------+
| { "A": 0, "B": [1, 2], "C":"[3, 4]"} |
+-------------------------------------+
```

1.2.9.4.17 JSON_KEYS

Syntax

```
JSON_KEYS(json_doc[, path])
```

Description

Returns the keys as a JSON array from the top-level value of a JSON object or, if the optional path argument is provided, the top-level keys from the path.

Excludes keys from nested sub-objects in the top level value. The resulting array will be empty if the selected object is empty.

Returns NULL if any of the arguments are null, a given path does not locate an object, or if the json_doc argument is not an object.

An error will occur if JSON document is invalid, the path is invalid or if the path contains a `*` or `**` wildcard.

Examples

```
SELECT JSON_KEYS('{"A": 1, "B": {"C": 2}}');
+-------------------------------------+
| JSON_KEYS('{"A": 1, "B": {"C": 2}}') |
+-------------------------------------+
| ["A", "B"]                          |
+-------------------------------------+

SELECT JSON_KEYS('{"A": 1, "B": 2, "C": {"D": 3}}', '$.C');
+-----------------------------------------------------+
| JSON_KEYS('{"A": 1, "B": 2, "C": {"D": 3}}', '$.C') |
+-----------------------------------------------------+
| ["D"]                                               |
+-----------------------------------------------------+
```

1.2.9.4.18 JSON_LENGTH

Syntax

```
JSON_LENGTH(json_doc[, path])
```

Description

Returns the length of a JSON document, or, if the optional path argument is given, the length of the value within the document specified by the path.

Returns NULL if any of the arguments argument are null or the path argument does not identify a value in the document.

An error will occur if the JSON document is invalid, the path is invalid or if the path contains a `*` or `**` wildcard.

Length will be determined as follow:

- A scalar's length is always 1.
- If an array, the number of elements in the array.
- If an object, the number of members in the object.

The length of nested arrays or objects are not counted.

Examples

```
```

1.2.9.4.19 JSON_LOOSE

Syntax

```
JSON_LOOSE(json_doc)
```

Description

Adds spaces to a JSON document to make it look more readable.

Example

```
SET @j = '{ "A":1,"B":[2,3]}';

SELECT JSON_LOOSE(@j), @j;
+----------------------+--------------------+
| JSON_LOOSE(@j)       | @j                 |
+----------------------+--------------------+
| {"A": 1, "B": [2, 3]} | { "A":1,"B":[2,3]} |
+----------------------+--------------------+
```

1.2.9.4.20 JSON_MERGE

Syntax

```
JSON_MERGE(json_doc, json_doc[, json_doc] ...)
```

Description

Merges the given JSON documents.

Returns the merged result,or NULL if any argument is NULL.

An error occurs if any of the arguments are not valid JSON documents.

`JSON_MERGE` has been deprecated since MariaDB 10.2.25, MariaDB 10.3.16 and MariaDB 10.4.5. JSON_MERGE_PATCH is an RFC 7396-compliant replacement, and JSON_MERGE_PRESERVE is a synonym.

Example

```
SET @json1 = '[1, 2]';
SET @json2 = '[3, 4]';

SELECT JSON_MERGE(@json1,@json2);
+---------------------------+
| JSON_MERGE(@json1,@json2) |
+---------------------------+
| [1, 2, 3, 4]              |
+---------------------------+
```

1.2.9.4.21 JSON_MERGE_PATCH

Syntax

```
JSON_MERGE_PATCH(json_doc, json_doc[, json_doc] ...)
```

Description

Merges the given JSON documents, returning the merged result, or NULL if any argument is NULL.

`JSON_MERGE_PATCH` is an RFC 7396-compliant replacement for JSON_MERGE, which has been deprecated.

Unlike JSON_MERGE_PRESERVE, members with duplicate keys are not preserved.

Example

```
SET @json1 = '[1, 2]';
SET @json2 = '[2, 3]';
SELECT JSON_MERGE_PATCH(@json1,@json2),JSON_MERGE_PRESERVE(@json1,@json2);
+--------------------------------+-----------------------------------+
| JSON_MERGE_PATCH(@json1,@json2) | JSON_MERGE_PRESERVE(@json1,@json2) |
+--------------------------------+-----------------------------------+
| [2, 3]                         | [1, 2, 2, 3]                      |
+--------------------------------+-----------------------------------+
```

1.2.9.4.22 JSON_MERGE_PRESERVE

Syntax

```
JSON_MERGE_PRESERVE(json_doc, json_doc[, json_doc] ...)
```

Description

Merges the given JSON documents, returning the merged result, or NULL if any argument is NULL.

`JSON_MERGE_PRESERVE` was introduced as a synonym for JSON_MERGE, which has been deprecated.

Unlike JSON_MERGE_PATCH, members with duplicate keys are preserved.

Example

```
SET @json1 = '[1, 2]';
SET @json2 = '[2, 3]';
SELECT JSON_MERGE_PATCH(@json1,@json2),JSON_MERGE_PRESERVE(@json1,@json2);
+--------------------------------+-----------------------------------+
| JSON_MERGE_PATCH(@json1,@json2) | JSON_MERGE_PRESERVE(@json1,@json2) |
+--------------------------------+-----------------------------------+
| [2, 3]                         | [1, 2, 2, 3]                      |
+--------------------------------+-----------------------------------+
```

1.2.9.4.23 JSON_NORMALIZE

MariaDB starting with 10.7.0

JSON_NORMALIZE was added in MariaDB 10.7.0.

Syntax

```
JSON_NORMALIZE(json)
```

Description

Recursively sorts keys and removes spaces, allowing comparison of json documents for equality.

Examples

We may wish our application to use the database to enforce a unique constraint on the JSON contents, and we can do so using the JSON_NORMALIZE function in combination with a unique key.

For example, if we have a table with a JSON column:

```
CREATE TABLE t1 (
 id BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
 val JSON,
 /* other columns here */
 PRIMARY KEY (id)
);
```

Add a unique constraint using JSON_NORMALIZE like this:

```
ALTER TABLE t1
   ADD COLUMN jnorm JSON AS (JSON_NORMALIZE(val)) VIRTUAL,
   ADD UNIQUE KEY (jnorm);
```

We can test this by first inserting a row as normal:

```
INSERT INTO t1 (val) VALUES ('{"name":"alice","color":"blue"}');
```

And then seeing what happens with a different string which would produce the same JSON object:

```
INSERT INTO t1 (val) VALUES ('{ "color": "blue", "name": "alice" }');
ERROR 1062 (23000): Duplicate entry '{"color":"blue","name":"alice"}' for key 'jnorm'
```

1.2.9.4.24 JSON_OBJECT

Syntax

```
JSON_OBJECT([key, value[, key, value] ...])
```

Description

Returns a JSON object containing the given key/value pairs. The key/value list can be empty.

An error will occur if there are an odd number of arguments, or any key name is NULL.

Example

```
SELECT JSON_OBJECT("id", 1, "name", "Monty");
+---------------------------------------+
| JSON_OBJECT("id", 1, "name", "Monty") |
+---------------------------------------+
| {"id": 1, "name": "Monty"}            |
+---------------------------------------+
```

1.2.9.4.25 JSON_OBJECT_FILTER_KEYS

MariaDB starting with 11.2.0

JSON_OBJECT_FILTER_KEYS was added in MariaDB 11.2.0.

Syntax

```
JSON_OBJECT_FILTER_KEYS(obj, array_keys)
```

Description

`JSON_OBJECT_FILTER_KEYS` returns a JSON object with keys from the object that are also present in the array as string. It is used when one wants to get key-value pair such that the keys are common but the values may not be common.

Example

```
SET @obj1= '{ "a": 1, "b": 2, "c": 3}';
SET @obj2= '{"b" : 10, "c": 20, "d": 30}';
SELECT JSON_OBJECT_FILTER_KEYS (@obj1, JSON_ARRAY_INTERSECT(JSON_KEYS(@obj1), JSON_KEYS(@obj2)));
+-----------------------------------------------------------------------------------------+
| JSON_OBJECT_FILTER_KEYS (@obj1, JSON_ARRAY_INTERSECT(JSON_KEYS(@obj1), JSON_KEYS(@obj2))) |
+-----------------------------------------------------------------------------------------+
| {"b": 2, "c": 3}                                                                        |
+-----------------------------------------------------------------------------------------+
```

1.2.9.4.26 JSON_OBJECT_TO_ARRAY

MariaDB starting with 11.2.0
JSON_OBJECT_TO_ARRAY was added in MariaDB 11.2.0.

Syntax

```
JSON_OBJECT_TO_ARRAY(Obj)
```

Description

It is used to convert all JSON objects found in a JSON document to JSON arrays where each item in the outer array represents a single key-value pair from the object. It is used when we want not just common keys, but also common values. It can be used in conjunction with JSON_ARRAY_INTERSECT().

Examples

```
SET @obj1= '{ "a": [1, 2, 3], "b": { "key1":"val1", "key2": {"key3":"val3"} }}';

SELECT JSON_OBJECT_TO_ARRAY(@obj1);
+-----------------------------------------------------------------------+
| JSON_OBJECT_TO_ARRAY(@obj1)                                           |
+-----------------------------------------------------------------------+
| [["a", [1, 2, 3]], ["b", {"key1": "val1", "key2": {"key3": "val3"}}]] |
+-----------------------------------------------------------------------+
```

1.2.4.10 JSON_OBJECTAGG

1.2.9.4.28 JSON_OVERLAPS

MariaDB starting with 10.9
JSON_OVERLAPS was added in MariaDB 10.9.

Syntax

```
JSON_OVERLAPS(json_doc1, json_doc2)
```

Description

JSON_OVERLAPS() compares two json documents and returns true if they have at least one common key-value pair between two objects, array element common between two arrays, or array element common with scalar if one of the arguments is a scalar and other is an array. If two json documents are scalars, it returns true if they have same type and

value.

If none of the above conditions are satisfied then it returns false.

Examples

```
SELECT JSON_OVERLAPS('false', 'false');
+---------------------------------+
| JSON_OVERLAPS('false', 'false') |
+---------------------------------+
| 1                               |
+---------------------------------+

SELECT JSON_OVERLAPS('true', '["abc", 1, 2, true, false]');
+-----------------------------------------------------+
| JSON_OVERLAPS('true','["abc", 1, 2, true, false]') |
+-----------------------------------------------------+
| 1                                                   |
+-----------------------------------------------------+

SELECT JSON_OVERLAPS('{"A": 1, "B": {"C":2}}', '{"A": 2, "B": {"C":2}}') AS is_overlap;
+---------------------+
| is_overlap          |
+---------------------+
| 1                   |
+---------------------+
```

Partial match is considered as no-match.

Examples

```
SELECT JSON_OVERLAPS('[1, 2, true, false, null]', '[3, 4, [1]]') AS is_overlap;
+---------------------- +
| is_overlap           |
+----------------------+
| 0                    |
+----------------------+
```

1.2.9.4.29 JSON_PRETTY

`JSON_PRETTY` was added as an alias for JSON_DETAILED in MariaDB 10.10.3, MariaDB 10.9.5, MariaDB 10.8.7, MariaDB 10.7.8, MariaDB 10.6.12, MariaDB 10.5.19 and MariaDB 10.4.28.

1.2.9.4.30 JSON_QUERY

Syntax

```
JSON_QUERY(json_doc, path)
```

Description

Given a JSON document, returns an object or array specified by the path. Returns NULL if not given a valid JSON document, or if there is no match.

Examples

```
select json_query('{"key1":{"a":1, "b":[1,2]}}', '$.key1');
+-----------------------------------------------------+
| json_query('{"key1":{"a":1, "b":[1,2]}}', '$.key1') |
+-----------------------------------------------------+
| {"a":1, "b":[1,2]}                                  |
+-----------------------------------------------------+

select json_query('{"key1":123, "key1": [1,2,3]}', '$.key1');
+-----------------------------------------------------+
| json_query('{"key1":123, "key1": [1,2,3]}', '$.key1') |
+-----------------------------------------------------+
| [1,2,3]                                             |
+-----------------------------------------------------+
```

1.2.9.4.31 JSON_QUOTE

Syntax

```
JSON_QUOTE(json_value)
```

Description

Quotes a string as a JSON value, usually for producing valid JSON string literals for inclusion in JSON documents. Wraps the string with double quote characters and escapes interior quotes and other special characters, returning a utf8mb4 string.

Returns NULL if the argument is NULL.

Examples

```
SELECT JSON_QUOTE('A'), JSON_QUOTE("B"), JSON_QUOTE('"C"');
+-----------------+-----------------+-------------------+
| JSON_QUOTE('A') | JSON_QUOTE("B") | JSON_QUOTE('"C"') |
+-----------------+-----------------+-------------------+
| "A"             | "B"             | "\"C\""           |
+-----------------+-----------------+-------------------+
```

1.2.9.4.32 JSON_REMOVE

Syntax

```
JSON_REMOVE(json_doc, path[, path] ...)
```

Contents

Description

Removes data from a JSON document returning the result, or NULL if any of the arguments are null. If the element does not exist in the document, no changes are made.

The function returns NULL and throws a warning if the JSON document is invalid, the path is invalid, contains a range, or contains a `*` or `**` wildcard.

Path arguments are evaluated from left to right, with the result from the earlier evaluation being used as the value for the next.

Examples

```
SELECT JSON_REMOVE('{"A": 1, "B": 2, "C": {"D": 3}}', '$.C');
+-----------------------------------------------------+
| JSON_REMOVE('{"A": 1, "B": 2, "C": {"D": 3}}', '$.C') |
+-----------------------------------------------------+
| {"A": 1, "B": 2}                                    |
+-----------------------------------------------------+

SELECT JSON_REMOVE('["A", "B", ["C", "D"], "E"]', '$[1]');
+-------------------------------------------------+
| JSON_REMOVE('["A", "B", ["C", "D"], "E"]', '$[1]') |
+-------------------------------------------------+
| ["A", ["C", "D"], "E"]                          |
+-------------------------------------------------+
```

1.2.9.4.33 JSON_REPLACE

Syntax

```
JSON_REPLACE(json_doc, path, val[, path, val] ...)
```

Contents

Description

Replaces existing values in a JSON document, returning the result, or NULL if any of the arguments are NULL.

An error will occur if the JSON document is invalid, the path is invalid or if the path contains a `*` or `**` wildcard.

Paths and values are evaluated from left to right, with the result from the earlier evaluation being used as the value for the next.

JSON_REPLACE can only update data, while JSON_INSERT can only insert. JSON_SET can update or insert data.

Examples

```
SELECT JSON_REPLACE('{ "A": 1, "B": [2, 3]}', '$.B[1]', 4);
+-----------------------------------------------------+
| JSON_REPLACE('{ "A": 1, "B": [2, 3]}', '$.B[1]', 4) |
+-----------------------------------------------------+
| { "A": 1, "B": [2, 4]}                              |
+-----------------------------------------------------+
```

1.2.9.4.34 JSON_SCHEMA_VALID

MariaDB starting with 11.1

`JSON_SCHEMA_VALID` was introduced in MariaDB 11.1.

Syntax

```
JSON_SCHEMA_VALID(schema, json);
```

Contents

Description

`JSON_SCHEMA_VALID` allows MariaDB to support JSON schema validation. If a given json is valid against a schema it returns true. When JSON does not validate against the schema, it does not return a message about which keyword it failed against and only returns false.

The function supports JSON Schema Draft 2020 with a few exceptions:

- External resources are not supported
- Hyper schema keywords are not supported
- Formats like date, email etc are treated as annotations.

Examples

To create validation rules for json field

```
CREATE TABLE obj_table(val_obj JSON CHECK(JSON_SCHEMA_VALID('{
  "type":"object",
    "properties": {
      "number1":{
        "type":"number",
        "maximum":5,
        "const":4
      },
      "string1":{
        "type":"string",
        "maxLength":5,
        "minLength":3
      },
    "object1":{
      "type":"object",
       "properties":{
         "key1": {"type":"string"},
         "key2":{"type":"array"},
         "key3":{"type":"number", "minimum":3}
       },
       "dependentRequired": { "key1":["key3"] }
     }
  },
  "required":["number1","object1"]
  }', val_obj)));

INSERT INTO obj_table VALUES(
  '{"number1":4, "string1":"abcd",
  "object1":{"key1":"val1", "key2":[1,2,3, "string1"], "key3":4}}'
);

INSERT INTO obj_table VALUES(
  '{"number1":3, "string1":"abcd",
  "object1":{"key1":"val1", "key2":[1,2,3, "string1"], "key3":4}}'
);
ERROR 4025 (23000): CONSTRAINT `obj_table.val_obj` failed for `test`.`obj_table`

SELECT * FROM obj_table;
+-----------------------------------------------------------------------------------------------------------+
| val_obj                                                                                                   |
+-----------------------------------------------------------------------------------------------------------+
| {"number1":4, "string1":"abcd", "object1":{"key1":"val1", "key2":[1,2,3, "string1"], "key3":4}}           |
+-----------------------------------------------------------------------------------------------------------+

SET @schema= '{
  "properties" : {
    "number1":{ "maximum":10 },
    "string1" : { "maxLength": 3}
  }
}';

SELECT JSON_SCHEMA_VALID(@schema, '{ "number1":25, "string1":"ab" }');
+-----------------------------------------------------------------+
| JSON_SCHEMA_VALID(@schema, '{ "number1":25, "string1":"ab" }') |
+-----------------------------------------------------------------+
|                                                               0 |
+-----------------------------------------------------------------+

SELECT JSON_SCHEMA_VALID(@schema, '{ "number1":10, "string1":"ab" }');
+-----------------------------------------------------------------+
| JSON_SCHEMA_VALID(@schema, '{ "number1":10, "string1":"ab" }') |
+-----------------------------------------------------------------+
|                                                               1 |
+-----------------------------------------------------------------+
```

1.2.9.4.35 JSON_SEARCH

Syntax

```
JSON_SEARCH(json_doc, return_arg, search_str[, escape_char[, path] ...])
```

Description

Returns the path to the given string within a JSON document, or NULL if any of *json_doc*, *search_str* or a path argument is NULL; if the search string is not found, or if no path exists within the document.

A warning will occur if the JSON document is not valid, any of the path arguments are not valid, if *return_arg* is neither *one* nor *all*, or if the escape character is not a constant. NULL will be returned.

return_arg can be one of two values:

- `'one` : Terminates after finding the first match, so will return one path string. If there is more than one match, it is undefined which is considered first.
- `all` : Returns all matching path strings, without duplicates. Multiple strings are autowrapped as an array. The order is undefined.

Examples

```
SET @json = '["A", [{"B": "1"}], {"C":"AB"}, {"D":"BC"}]';

SELECT JSON_SEARCH(@json, 'one', 'AB');
+--------------------------------+
| JSON_SEARCH(@json, 'one', 'AB') |
+--------------------------------+
| "$[2].C"                       |
+--------------------------------+
```

1.2.9.4.36 JSON_SET

Syntax

```
JSON_SET(json_doc, path, val[, path, val] ...)
```

Description

Updates or inserts data into a JSON document, returning the result, or NULL if any of the arguments are NULL or the optional path fails to find an object.

An error will occur if the JSON document is invalid, the path is invalid or if the path contains a * or **wildcard.**

JSON_SET can update or insert data, while JSON_REPLACE can only update, and JSON_INSERT only insert.

Examples

```
SELECT JSON_SET(Priv, '$.locked', 'true') FROM mysql.global_priv
```

1.2.9.4.37 JSON_TABLE

MariaDB starting with 10.6.0

JSON_TABLE was added in MariaDB 10.6.0.

JSON_TABLE is a table function that converts JSON data into a relational form.

Syntax

```
JSON_TABLE(json_doc,
          context_path COLUMNS (column_list)
) [AS] alias
```

```
column_list:
    column[, column][, ...]
```

```
column:
    name FOR ORDINALITY
    |  name type PATH path_str [on_empty] [on_error]
    |  name type EXISTS PATH path_str
    |  NESTED PATH path_str COLUMNS (column_list)
```

```
on_empty:
    {NULL | DEFAULT string | ERROR} ON EMPTY
```

```
on_error:
    {NULL | DEFAULT string | ERROR} ON ERROR
```

Contents

Description

JSON_TABLE can be used in contexts where a table reference can be used; in the FROM clause of a SELECT statement, and in multi-table UPDATE/DELETE statements.

`json_doc` is the JSON document to extract data from. In the simplest case, it is a string literal containing JSON. In more complex cases it can be an arbitrary expression returning JSON. The expression may have references to columns of other tables. However, one can only refer to tables that precede this JSON_TABLE invocation. For RIGHT JOIN, it is assumed that its outer side precedes the inner. All tables in outer selects are also considered preceding.

`context_path` is a JSON Path expression pointing to a collection of nodes in `json_doc` that will be used as the source of rows.

The `COLUMNS` clause declares the names and types of the columns that JSON_TABLE returns, as well as how the values of the columns are produced.

Column Definitions

The following types of columns are supported:

Path Columns

```
name type PATH path_str [on_empty] [on_error]
```

Locates the JSON node pointed to by `path_str` and returns its value. The path_str is evaluated using the current row source node as the context node.

```
set @json='
[
  {"name":"Laptop", "color":"black", "price":"1000"},
  {"name":"Jeans",  "color":"blue"}
]';

select * from json_table(@json, '$[*]'
  columns(
   name  varchar(10) path '$.name',
   color varchar(10) path '$.color',
   price decimal(8,2) path '$.price' )
) as jt;
+--------+-------+---------+
| name   | color | price   |
+--------+-------+---------+
| Laptop | black | 1000.00 |
| Jeans  | blue  |    NULL |
+--------+-------+---------+
```

The `on_empty` and `on_error` clauses specify the actions to be performed when the value was not found or there was an error condition. See the ON EMPTY and ON ERROR clauses section for details.

ORDINALITY Columns

```
name FOR ORDINALITY
```

Counts the rows, starting from 1.

Example:

```
set @json='
[
  {"name":"Laptop", "color":"black"},
  {"name":"Jeans",  "color":"blue"}
]';

select * from json_table(@json, '$[*]'
  columns(
   id for ordinality,
   name  varchar(10) path '$.name')
) as jt;
+------+--------+
| id   | name   |
+------+--------+
|    1 | Laptop |
|    2 | Jeans  |
+------+--------+
```

EXISTS PATH Columns

```
name type EXISTS PATH path_str
```

Checks whether the node pointed to by `value_path` exists. The `value_path` is evaluated using the current row source node as the context node.

```
set @json='
[
  {"name":"Laptop", "color":"black", "price":1000},
  {"name":"Jeans",  "color":"blue"}
]';

select * from json_table(@json, '$[*]'
  columns(
   name  varchar(10) path '$.name',
   has_price integer exists path '$.price')
) as jt;
+--------+-----------+
| name   | has_price |
+--------+-----------+
| Laptop |         1 |
| Jeans  |         0 |
+--------+-----------+
```

NESTED PATHs

NESTED PATH converts nested JSON structures into multiple rows.

```
NESTED PATH path COLUMNS (column_list)
```

It finds the sequence of JSON nodes pointed to by `path` and uses it to produce rows. For each found node, a row is generated with column values as specified by the NESTED PATH's COLUMNS clause. If `path` finds no nodes, only one row is generated with all columns having NULL values.

For example, consider a JSON document that contains an array of items, and each item, in turn, is expected to have an array of its available sizes:

```
set @json='
[
  {"name":"Jeans",  "sizes": [32, 34, 36]},
  {"name":"T-Shirt", "sizes":["Medium", "Large"]},
  {"name":"Cellphone"}
]';
```

NESTED PATH allows one to produce a separate row for each size each item has:

```
select * from json_table(@json, '$[*]'
  columns(
    name  varchar(10) path '$.name',
    nested path '$.sizes[*]' columns (
      size varchar(32) path '$'
    )
  )
) as jt;
+-----------+--------+
| name      | size   |
+-----------+--------+
| Jeans     | 32     |
| Jeans     | 34     |
| Jeans     | 36     |
| T-Shirt   | Medium |
| T-Shirt   | Large  |
| Cellphone | NULL   |
+-----------+--------+
```

NESTED PATH clauses can be nested within one another. They can also be located next to each other. In that case, the nested path clauses will produce records one at a time. The ones that are not producing records will have all columns set to NULL.

Example:

```
set @json='
[
  {"name":"Jeans",  "sizes": [32, 34, 36], "colors":["black", "blue"]}
]';

select * from json_table(@json, '$[*]'
  columns(
    name  varchar(10) path '$.name',
    nested path '$.sizes[*]' columns (
      size varchar(32) path '$'
    ),
    nested path '$.colors[*]' columns (
      color varchar(32) path '$'
    )
  )
) as jt;

+-------+------+-------+
| name  | size | color |
+-------+------+-------+
| Jeans | 32   | NULL  |
| Jeans | 34   | NULL  |
| Jeans | 36   | NULL  |
| Jeans | NULL | black |
| Jeans | NULL | blue  |
+-------+------+-------+
```

ON EMPTY and ON ERROR Clauses

The ON EMPTY clause specifies what will be done when the element specified by the search path is missing in the JSON document.

```
on_empty:
    {NULL | DEFAULT string | ERROR} ON EMPTY
```

When `ON EMPTY` clause is not present, `NULL ON EMPTY` is implied.

```
on_error:
    {NULL | DEFAULT string | ERROR} ON ERROR
```

The ON ERROR clause specifies what should be done if a JSON structure error occurs when trying to extract the value pointed to by the path expression. A JSON structure error here occurs only when one attempts to convert a JSON non-scalar (array or object) into a scalar value. When the `ON ERROR` clause is not present, `NULL ON ERROR` is implied.

Note: A datatype conversion error (e.g. attempt to store a non-integer value into an integer field, or a varchar column being truncated) is not considered a JSON error and so will not trigger the `ON ERROR` behavior. It will produce warnings, in the same way as CAST(value AS datatype) would.

Replication

In the current code, evaluation of JSON_TABLE is deterministic, that is, for a given input string JSON_TABLE will always produce the same set of rows in the same order. However, one can think of JSON documents that one can consider identical which will produce different output. In order to be future-proof and withstand changes like:

- sorting JSON object members by name (like MySQL does)
- changing the way duplicate object members are handled the function is marked as unsafe for statement-based replication.

Extracting a Subdocument into a Column

MariaDB starting with 10.6.9

Prior to MariaDB 10.6.9, JSON_TABLE did not allow one to extract a JSON "subdocument" into a JSON column.

```
SELECT * FROM JSON_TABLE('{"foo": [1,2,3,4]}','$' columns( jscol json path '$.foo') ) AS T;
+--------+
| jscol |
+--------+
| NULL  |
+--------+
```

This is supported from MariaDB 10.6.9:

```
SELECT * FROM JSON_TABLE('{"foo": [1,2,3,4]}','$' columns( jscol json path '$.foo') ) AS T;
+-----------+
| jscol     |
+-----------+
| [1,2,3,4] |
+-----------+
```

1.2.9.4.38 JSON_TYPE

Syntax

```
JSON_TYPE(json_val)
```

Description

Returns the type of a JSON value (as a string), or NULL if the argument is null.

An error will occur if the argument is an invalid JSON value.

The following is a complete list of the possible return types:

Return type	Value	Example
ARRAY	JSON array	[1, 2, {"key": "value"}]
OBJECT	JSON object	{"key":"value"}
BOOLEAN	JSON true/false literals	true, false
DOUBLE	A number with at least one floating point decimal.	1.2
INTEGER	A number without a floating point decimal.	1
NULL	JSON null literal (this is returned as a string, not to be confused with the SQL NULL value!)	null
STRING	JSON String	"a sample string"

Examples

```
SELECT JSON_TYPE('{"A": 1, "B": 2, "C": 3}');
+---------------------------------------+
| JSON_TYPE('{"A": 1, "B": 2, "C": 3}') |
+---------------------------------------+
| OBJECT                                |
+---------------------------------------+
```

1.2.9.4.39 JSON_UNQUOTE

Syntax

```
JSON_UNQUOTE(val)
```

Description

Unquotes a JSON value, returning a string, or NULL if the argument is null.

An error will occur if the given value begins and ends with double quotes and is an invalid JSON string literal.

If the given value is not a JSON string, value is passed through unmodified.

Certain character sequences have special meanings within a string. Usually, a backslash is ignored, but the escape sequences in the table below are recognised by MariaDB, unless the SQL Mode is set to NO_BACKSLASH_ESCAPES SQL.

Escape sequence	Character
`\"`	Double quote (")
`\b`	Backslash
`\f`	Formfeed
`\n`	Newline (linefeed)
`\r`	Carriage return
`\t`	Tab
`\\`	Backslash (\)
`\uXXXX`	UTF-8 bytes for Unicode value XXXX

Examples

```
SELECT JSON_UNQUOTE('"Monty"');
+-------------------------+
| JSON_UNQUOTE('"Monty"') |
+-------------------------+
| Monty                   |
+-------------------------+
```

With the default SQL Mode:

```
SELECT JSON_UNQUOTE('Si\bng\ting');
+-----------------------------+
| JSON_UNQUOTE('Si\bng\ting') |
+-----------------------------+
| Sng ing                     |
+-----------------------------+
```

Setting NO_BACKSLASH_ESCAPES:

```
SET @@sql_mode = 'NO_BACKSLASH_ESCAPES';

SELECT JSON_UNQUOTE('Si\bng\ting');
+-----------------------------+
| JSON_UNQUOTE('Si\bng\ting') |
+-----------------------------+
| Si\bng\ting                 |
+-----------------------------+
```

1.2.9.4.40 JSON_VALID

Syntax

```
JSON_VALID(value)
```

Contents

Description

Indicates whether the given value is a valid JSON document or not. Returns `1` if valid, `0` if not, and NULL if the argument is NULL.

From MariaDB 10.4.3, the JSON_VALID function is automatically used as a CHECK constraint for the JSON data type alias in order to ensure that a valid json document is inserted.

Examples

```
SELECT JSON_VALID('{"id": 1, "name": "Monty"}');
+---------------------------------------+
| JSON_VALID('{"id": 1, "name": "Monty"}') |
+---------------------------------------+
|                                     1 |
+---------------------------------------+

SELECT JSON_VALID('{"id": 1, "name": "Monty", "oddfield"}');
+-----------------------------------------------------+
| JSON_VALID('{"id": 1, "name": "Monty", "oddfield"}') |
+-----------------------------------------------------+
|                                                   0 |
+-----------------------------------------------------+
```

1.2.9.4.41 JSON_VALUE

Syntax

```
JSON_VALUE(json_doc, path)
```

Description

Given a JSON document, returns the scalar specified by the path. Returns NULL if not given a valid JSON document, or if there is no match.

Examples

```
select json_value('{"key1":123}', '$.key1');
+--------------------------------------+
| json_value('{"key1":123}', '$.key1') |
+--------------------------------------+
| 123                                  |
+--------------------------------------+

select json_value('{"key1": [1,2,3], "key1":123}', '$.key1');
+-----------------------------------------------------+
| json_value('{"key1": [1,2,3], "key1":123}', '$.key1') |
+-----------------------------------------------------+
| 123                                                 |
+-----------------------------------------------------+
```

In the SET statement below, two escape characters are needed, as a single escape character would be applied by the SQL parser in the SET statement, and the escaped character would not form part of the saved value.

```
SET @json = '{"key1":"60\\" Table", "key2":"1"}';

SELECT JSON_VALUE(@json,'$.key1') AS Name , json_value(@json,'$.key2') as ID;
+-----------+------+
| Name      | ID   |
+-----------+------+
| 60" Table | 1    |
+-----------+------+
```

1.1.6.6 SEQUENCE Functions

1.2.9.6 Spider Functions

The following UDFs are available with the Spider Storage Engine.

SPIDER_BG_DIRECT_SQL

Background SQL execution

SPIDER_COPY_TABLES

Copy table data

SPIDER_DIRECT_SQL

Execute SQL on the remote server

SPIDER_FLUSH_TABLE_MON_CACHE

Refreshing Spider monitoring server information

1.2.9.6.1 SPIDER_BG_DIRECT_SQL

Syntax

```
SPIDER_BG_DIRECT_SQL('sql', 'tmp_table_list', 'parameters')
```

Description

Executes the given SQL statement in the background on the remote server, as defined in the parameters listing. If the query returns a result-set, it sttores the results in the given temporary table. When the given SQL statement executes successfully, this function returns the number of called UDF's. It returns `0` when the given SQL statement fails.

This function is a UDF installed with the Spider storage engine.

Examples

```
SELECT SPIDER_BG_DIRECT_SQL('SELECT * FROM example_table',  '',
   'srv "node1", port "8607"') AS "Direct Query";
+--------------+
| Direct Query |
+--------------+
|            1 |
+--------------+
```

Parameters

`error_rw_mode`

- **Description:** Returns empty results on network error.
  - `0` : Return error on getting network error.
  - `1` : Return 0 records on getting network error.

- **Default Table Value:** `0`
- **DSN Parameter Name:** `erwm`

1.2.9.6.2 SPIDER_COPY_TABLES

Syntax

```
SPIDER_COPY_TABLES(spider_table_name,
  source_link_id, destination_link_id_list [,parameters])
```

Description

A UDF installed with the Spider Storage Engine, this function copies table data from `source_link_id` to `destination_link_id_list`. The service does not need to be stopped in order to copy.

If the Spider table is partitioned, the name must be of the format `table_name#P#partition_name`. The partition name can be viewed in the `mysql.spider_tables` table, for example:

```
SELECT table_name FROM mysql.spider_tables;
+-------------+
| table_name  |
+-------------+
| spt_a#P#pt1 |
| spt_a#P#pt2 |
| spt_a#P#pt3 |
+-------------+
```

Returns `1` if the data was copied successfully, or `0` if copying the data failed.

1.2.9.6.3 SPIDER_DIRECT_SQL

Syntax

```
SPIDER_DIRECT_SQL('sql', 'tmp_table_list', 'parameters')
```

Description

A UDF installed with the Spider Storage Engine, this function is used to execute the SQL string `sql` on the remote server, as defined in `parameters`. If any resultsets are returned, they are stored in the `tmp_table_list`.

The function returns `1` if the SQL executes successfully, or `0` if it fails.

Examples

```
SELECT SPIDER_DIRECT_SQL('SELECT * FROM s', '', 'srv "node1", port "8607"');
+---------------------------------------------------------------------------+
| SPIDER_DIRECT_SQL('SELECT * FROM s', '', 'srv "node1", port "8607"') |
+---------------------------------------------------------------------------+
|                                                                         1 |
+---------------------------------------------------------------------------+
```

1.2.9.6.4 SPIDER_FLUSH_TABLE_MON_CACHE

Syntax

```
SPIDER_FLUSH_TABLE_MON_CACHE()
```

Description

A UDF installed with the Spider Storage Engine, this function is used for refreshing monitoring server information. It returns a value of `1`.

Examples

```
SELECT SPIDER_FLUSH_TABLE_MON_CACHE();
+--------------------------------+
| SPIDER_FLUSH_TABLE_MON_CACHE() |
+--------------------------------+
|                              1 |
+--------------------------------+
```

1.2.9.7 Window Functions

Window functions perform calculations across a set of rows related to the current row.

Window Functions Overview

Window functions perform calculations across a set of rows related to the current row.

AVG

Returns the average value.

BIT_AND

Bitwise AND.

BIT_OR

Bitwise OR.

BIT_XOR

Bitwise XOR.

COUNT

Returns count of non-null values.

CUME_DIST

Window function that returns the cumulative distribution of a given row.

DENSE_RANK

Rank of a given row with identical values receiving the same result, no skipping.

FIRST_VALUE

Returns the first result from an ordered set.

JSON_ARRAYAGG

Returns a JSON array containing an element for each value in a given set of JSON or SQL values.

JSON_OBJECTAGG

Returns a JSON object containing key-value pairs.

LAG

Accesses data from a previous row in the same result set without the need for a self-join.

LAST_VALUE

Returns the last value in a list or set of values.

LEAD

Accesses data from a following row in the same result set without the need for a self-join.

MAX
Returns the maximum value.

MEDIAN
Window function that returns the median value of a range of values.

MIN
Returns the minimum value.

NTH_VALUE
Returns the value evaluated at the specified row number of the window frame.

NTILE
Returns an integer indicating which group a given row falls into.

PERCENT_RANK
Window function that returns the relative percent rank of a given row.

PERCENTILE_CONT
Continuous percentile.

PERCENTILE_DISC
Discrete percentile.

RANK
Rank of a given row with identical values receiving the same result.

ROW_NUMBER
Row number of a given row with identical values receiving a different result.

STD
Population standard deviation.

STDDEV
Population standard deviation.

STDDEV_POP
Returns the population standard deviation.

STDDEV_SAMP
Standard deviation.

SUM
Sum total.

VAR_POP
Population standard variance.

VAR_SAMP
Returns the sample variance.

VARIANCE
Population standard variance.

Aggregate Functions as Window Functions
It is possible to use aggregate functions as window functions.

ColumnStore Window Functions
Summary of window function use with the ColumnStore engine

Window Frames
Some window functions operate on window frames.

1.2.9.7.1 Window Functions Overview

Contents

Introduction

Window functions allow calculations to be performed across a set of rows related to the current row.

Syntax

```
function (expression) OVER (
  [ PARTITION BY expression_list ]
  [ ORDER BY order_list [ frame_clause ] ] )

function:
  A valid window function

expression_list:
  expression | column_name [, expr_list ]

order_list:
  expression | column_name [ ASC | DESC ]
  [, ... ]

frame_clause:
  {ROWS | RANGE} {frame_border | BETWEEN frame_border AND frame_border}

frame_border:
  | UNBOUNDED PRECEDING
  | UNBOUNDED FOLLOWING
  | CURRENT ROW
  | expr PRECEDING
  | expr FOLLOWING
```

Description

In some ways, window functions are similar to aggregate functions in that they perform calculations across a set of rows. However, unlike aggregate functions, the output is not grouped into a single row.

Non-aggregate window functions include

- CUME_DIST
- DENSE_RANK
- FIRST_VALUE
- LAG
- LAST_VALUE
- LEAD
- MEDIAN
- NTH_VALUE
- NTILE
- PERCENT_RANK
- PERCENTILE_CONT
- PERCENTILE_DISC
- RANK, ROW_NUMBER

Aggregate functions that can also be used as window functions include

- AVG
- BIT_AND
- BIT_OR
- BIT_XOR
- COUNT

- MAX
- MIN
- STD
- STDDEV
- STDDEV_POP
- STDDEV_SAMP
- SUM
- VAR_POP
- VAR_SAMP
- VARIANCE

Window function queries are characterised by the OVER keyword, following which the set of rows used for the calculation is specified. By default, the set of rows used for the calculation (the "window) is the entire dataset, which can be ordered with the ORDER BY clause. The PARTITION BY clause is used to reduce the window to a particular group within the dataset.

For example, given the following data:

```
CREATE TABLE student (name CHAR(10), test CHAR(10), score TINYINT);

INSERT INTO student VALUES
  ('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),
  ('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),
  ('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),
  ('Tatiana', 'SQL', 87), ('Tatiana', 'Tuning', 83);
```

the following two queries return the average partitioned by test and by name respectively:

```
SELECT name, test, score, AVG(score) OVER (PARTITION BY test)
  AS average_by_test FROM student;
+---------+--------+-------+-----------------+
| name    | test   | score | average_by_test |
+---------+--------+-------+-----------------+
| Chun    | SQL    |    75 |         65.2500 |
| Chun    | Tuning |    73 |         68.7500 |
| Esben   | SQL    |    43 |         65.2500 |
| Esben   | Tuning |    31 |         68.7500 |
| Kaolin  | SQL    |    56 |         65.2500 |
| Kaolin  | Tuning |    88 |         68.7500 |
| Tatiana | SQL    |    87 |         65.2500 |
| Tatiana | Tuning |    83 |         68.7500 |
+---------+--------+-------+-----------------+

SELECT name, test, score, AVG(score) OVER (PARTITION BY name)
  AS average_by_name FROM student;
+---------+--------+-------+-----------------+
| name    | test   | score | average_by_name |
+---------+--------+-------+-----------------+
| Chun    | SQL    |    75 |         74.0000 |
| Chun    | Tuning |    73 |         74.0000 |
| Esben   | SQL    |    43 |         37.0000 |
| Esben   | Tuning |    31 |         37.0000 |
| Kaolin  | SQL    |    56 |         72.0000 |
| Kaolin  | Tuning |    88 |         72.0000 |
| Tatiana | SQL    |    87 |         85.0000 |
| Tatiana | Tuning |    83 |         85.0000 |
+---------+--------+-------+-----------------+
```

It is also possible to specify which rows to include for the window function (for example, the current row and all preceding rows). See Window Frames for more details.

Scope

Window functions were introduced in SQL:2003, and their definition was expanded in subsequent versions of the standard. The last expansion was in the latest version of the standard, SQL:2011.

Most database products support a subset of the standard, they implement some functions defined as late as in SQL:2011, and at the same time leave some parts of SQL:2008 unimplemented.

MariaDB:

- Supports ROWS and RANGE-type frames

    - All kinds of frame bounds are supported, including `RANGE PRECEDING|FOLLOWING n` frame bounds (unlike PostgreSQL or MS SQL Server)
    - Does not yet support DATE[TIME] datatype and arithmetic for RANGE-type frames (MDEV-9727)
- Does not support GROUPS-type frames (it seems that no popular database supports it, either)
- Does not support frame exclusion (no other database seems to support it, either) (MDEV-9724)
- Does not support explicit `NULLS FIRST` or `NULLS LAST` .
- Does not support nested navigation in window functions (this is `VALUE_OF(expr AT row_marker [, default_value)` syntax)
- The following window functions are supported:
    - "Streamable" window functions: ROW_NUMBER, RANK, DENSE_RANK,
    - Window functions that can be streamed once the number of rows in partition is known: PERCENT_RANK, CUME_DIST, NTILE
- Aggregate functions that are currently supported as window functions are: COUNT, SUM, AVG, BIT_OR, BIT_AND, BIT_XOR.
- Aggregate functions with the `DISTINCT` specifier (e.g. `COUNT( DISTINCT x)`) are not supported as window functions.

Links

- MDEV-6115 is the main jira task for window functions development. Other tasks are are attached as sub-tasks
- bb-10.2-mdev9543 is the feature tree for window functions. Development is ongoing, and this tree has the newest changes.
- Testcases are in `mysql-test/t/win*.test`

Examples

Given the following sample data:

```
CREATE TABLE users (
  email VARCHAR(30),
  first_name VARCHAR(30),
  last_name VARCHAR(30),
  account_type VARCHAR(30)
);

INSERT INTO users VALUES
  ('admin@boss.org', 'Admin', 'Boss', 'admin'),
  ('bob.carlsen@foo.bar', 'Bob', 'Carlsen', 'regular'),
  ('eddie.stevens@data.org', 'Eddie', 'Stevens', 'regular'),
  ('john.smith@xyz.org', 'John', 'Smith', 'regular'),
  ('root@boss.org', 'Root', 'Chief', 'admin')
```

First, let's order the records by email alphabetically, giving each an ascending *rnum* value starting with 1. This will make use of the ROW_NUMBER window function:

```
SELECT row_number() OVER (ORDER BY email) AS rnum,
    email, first_name, last_name, account_type
FROM users ORDER BY email;
+------+-------------------------+------------+-----------+--------------+
| rnum | email                   | first_name | last_name | account_type |
+------+-------------------------+------------+-----------+--------------+
|    1 | admin@boss.org          | Admin      | Boss      | admin        |
|    2 | bob.carlsen@foo.bar     | Bob        | Carlsen   | regular      |
|    3 | eddie.stevens@data.org  | Eddie      | Stevens   | regular      |
|    4 | john.smith@xyz.org      | John       | Smith     | regular      |
|    5 | root@boss.org           | Root       | Chief     | admin        |
+------+-------------------------+------------+-----------+--------------
```

We can generate separate sequences based on account type, using the PARTITION BY clause:

```
SELECT row_number() OVER (PARTITION BY account_type ORDER BY email) AS rnum,
  email, first_name, last_name, account_type
FROM users ORDER BY account_type,email;
+------+-----------------------+------------+-----------+--------------+
| rnum | email                 | first_name | last_name | account_type |
+------+-----------------------+------------+-----------+--------------+
|    1 | admin@boss.org        | Admin      | Boss      | admin        |
|    2 | root@boss.org         | Root       | Chief     | admin        |
|    1 | bob.carlsen@foo.bar   | Bob        | Carlsen   | regular      |
|    2 | eddie.stevens@data.org | Eddie      | Stevens   | regular      |
|    3 | john.smith@xyz.org    | John       | Smith     | regular      |
+------+-----------------------+------------+-----------+--------------+
```

Given the following structure and data, we want to find the top 5 salaries from each department.

```
CREATE TABLE employee_salaries (dept VARCHAR(20), name VARCHAR(20), salary INT(11));

INSERT INTO employee_salaries VALUES
('Engineering', 'Dharma', 3500),
('Engineering', 'Binh', 3000),
('Engineering', 'Adalynn', 2800),
('Engineering', 'Samuel', 2500),
('Engineering', 'Cveta', 2200),
('Engineering', 'Ebele', 1800),
('Sales', 'Carbry', 500),
('Sales', 'Clytemnestra', 400),
('Sales', 'Juraj', 300),
('Sales', 'Kalpana', 300),
('Sales', 'Svantepolk', 250),
('Sales', 'Angelo', 200);
```

We could do this without using window functions, as follows:

```
select dept, name, salary
from employee_salaries as t1
where (select count(t2.salary)
       from employee_salaries as t2
       where t1.name != t2.name and
             t1.dept = t2.dept and
             t2.salary > t1.salary) < 5
order by dept, salary desc;

+-------------+--------------+--------+
| dept        | name         | salary |
+-------------+--------------+--------+
| Engineering | Dharma       |   3500 |
| Engineering | Binh         |   3000 |
| Engineering | Adalynn      |   2800 |
| Engineering | Samuel       |   2500 |
| Engineering | Cveta        |   2200 |
| Sales       | Carbry       |    500 |
| Sales       | Clytemnestra |    400 |
| Sales       | Juraj        |    300 |
| Sales       | Kalpana      |    300 |
| Sales       | Svantepolk   |    250 |
+-------------+--------------+--------+
```

This has a number of disadvantages:

- if there is no index, the query could take a long time if the employee_salary_table is large
- Adding and maintaining indexes adds overhead, and even with indexes on *dept* and *salary*, each subquery execution adds overhead by performing a lookup through the index.

Let's try achieve the same with window functions. First, generate a rank for all employees, using the RANK function.

```
select rank() over (partition by dept order by salary desc) as ranking,
    dept, name, salary
    from employee_salaries
    order by dept, ranking;
+---------+-------------+--------------+--------+
| ranking | dept        | name         | salary |
+---------+-------------+--------------+--------+
|       1 | Engineering | Dharma       |   3500 |
|       2 | Engineering | Binh         |   3000 |
|       3 | Engineering | Adalynn      |   2800 |
|       4 | Engineering | Samuel       |   2500 |
|       5 | Engineering | Cveta        |   2200 |
|       6 | Engineering | Ebele        |   1800 |
|       1 | Sales       | Carbry       |    500 |
|       2 | Sales       | Clytemnestra |    400 |
|       3 | Sales       | Juraj        |    300 |
|       3 | Sales       | Kalpana      |    300 |
|       5 | Sales       | Svantepolk   |    250 |
|       6 | Sales       | Angelo       |    200 |
+---------+-------------+--------------+--------+
```

Each department has a separate sequence of ranks due to the *PARTITION BY* clause. This particular sequence of values for *rank()* is given by the *ORDER BY* clause inside the window function's *OVER* clause. Finally, to get our results in a readable format we order the data by *dept* and the newly generated *ranking* column.

Now, we need to reduce the results to find only the top 5 per department. Here is a common mistake:

```
select
rank() over (partition by dept order by salary desc) as ranking,
dept, name, salary
from employee_salaries
where ranking <= 5
order by dept, ranking;

ERROR 1054 (42S22): Unknown column 'ranking' in 'where clause'
```

Trying to filter only the first 5 values per department by putting a where clause in the statement does not work, due to the way window functions are computed. The computation of window functions happens after all WHERE, GROUP BY and HAVING clauses have been completed, right before ORDER BY, so the WHERE clause has no idea that the ranking column exists. It is only present after we have filtered and grouped all the rows.

To counteract this problem, we need to wrap our query into a derived table. We can then attach a where clause to it:

```
select *from (select rank() over (partition by dept order by salary desc) as ranking,
  dept, name, salary
from employee_salaries) as salary_ranks
where (salary_ranks.ranking <= 5)
  order by dept, ranking;
+---------+-------------+--------------+--------+
| ranking | dept        | name         | salary |
+---------+-------------+--------------+--------+
|       1 | Engineering | Dharma       |   3500 |
|       2 | Engineering | Binh         |   3000 |
|       3 | Engineering | Adalynn      |   2800 |
|       4 | Engineering | Samuel       |   2500 |
|       5 | Engineering | Cveta        |   2200 |
|       1 | Sales       | Carbry       |    500 |
|       2 | Sales       | Clytemnestra |    400 |
|       3 | Sales       | Juraj        |    300 |
|       3 | Sales       | Kalpana      |    300 |
|       5 | Sales       | Svantepolk   |    250 |
+---------+-------------+--------------+--------+
```

1.2.4.2 AVG

1.2.4.3 BIT_AND

1.2.4.4 BIT_OR

1.2.4.5 BIT_XOR

1.2.4.6 COUNT

1.2.9.7.7 CUME_DIST

Contents

Syntax

```
CUME_DIST() OVER (
  [ PARTITION BY partition_expression ]
  [ ORDER BY order_list ]
)
```

Description

CUME_DIST() is a window function that returns the cumulative distribution of a given row. The following formula is used to calculate the value:

```
(number of rows <= current row) / (total rows)
```

Examples

```
create table t1 (
  pk int primary key,
  a int,
  b int
);


insert into t1 values
( 1 , 0, 10),
( 2 , 0, 10),
( 3 , 1, 10),
( 4 , 1, 10),
( 8 , 2, 10),
( 5 , 2, 20),
( 6 , 2, 20),
( 7 , 2, 20),
( 9 , 4, 20),
(10 , 4, 20);

select pk, a, b,
    rank() over (order by a) as rank,
    percent_rank() over (order by a) as pct_rank,
    cume_dist() over (order by a) as cume_dist
from t1;
+----+------+------+------+--------------+--------------+
| pk | a    | b    | rank | pct_rank     | cume_dist    |
+----+------+------+------+--------------+--------------+
|  1 |    0 |   10 |    1 | 0.0000000000 | 0.2000000000 |
|  2 |    0 |   10 |    1 | 0.0000000000 | 0.2000000000 |
|  3 |    1 |   10 |    3 | 0.2222222222 | 0.4000000000 |
|  4 |    1 |   10 |    3 | 0.2222222222 | 0.4000000000 |
|  5 |    2 |   20 |    5 | 0.4444444444 | 0.8000000000 |
|  6 |    2 |   20 |    5 | 0.4444444444 | 0.8000000000 |
|  7 |    2 |   20 |    5 | 0.4444444444 | 0.8000000000 |
|  8 |    2 |   10 |    5 | 0.4444444444 | 0.8000000000 |
|  9 |    4 |   20 |    9 | 0.8888888889 | 1.0000000000 |
| 10 |    4 |   20 |    9 | 0.8888888889 | 1.0000000000 |
+----+------+------+------+--------------+--------------+
```

```
select pk, a, b,
       percent_rank() over (order by pk) as pct_rank,
       cume_dist() over (order by pk) as cume_dist
from t1 order by pk;
+----+------+------+--------------+--------------+
| pk | a    | b    | pct_rank     | cume_dist    |
+----+------+------+--------------+--------------+
|  1 |    0 |   10 | 0.0000000000 | 0.1000000000 |
|  2 |    0 |   10 | 0.1111111111 | 0.2000000000 |
|  3 |    1 |   10 | 0.2222222222 | 0.3000000000 |
|  4 |    1 |   10 | 0.3333333333 | 0.4000000000 |
|  5 |    2 |   20 | 0.4444444444 | 0.5000000000 |
|  6 |    2 |   20 | 0.5555555556 | 0.6000000000 |
|  7 |    2 |   20 | 0.6666666667 | 0.7000000000 |
|  8 |    2 |   10 | 0.7777777778 | 0.8000000000 |
|  9 |    4 |   20 | 0.8888888889 | 0.9000000000 |
| 10 |    4 |   20 | 1.0000000000 | 1.0000000000 |
+----+------+------+--------------+--------------+

select pk, a, b,
        percent_rank() over (partition by a order by a) as pct_rank,
        cume_dist() over (partition by a order by a) as cume_dist
from t1;
+----+------+------+--------------+--------------+
| pk | a    | b    | pct_rank     | cume_dist    |
+----+------+------+--------------+--------------+
|  1 |    0 |   10 | 0.0000000000 | 1.0000000000 |
|  2 |    0 |   10 | 0.0000000000 | 1.0000000000 |
|  3 |    1 |   10 | 0.0000000000 | 1.0000000000 |
|  4 |    1 |   10 | 0.0000000000 | 1.0000000000 |
|  5 |    2 |   20 | 0.0000000000 | 1.0000000000 |
|  6 |    2 |   20 | 0.0000000000 | 1.0000000000 |
|  7 |    2 |   20 | 0.0000000000 | 1.0000000000 |
|  8 |    2 |   10 | 0.0000000000 | 1.0000000000 |
|  9 |    4 |   20 | 0.0000000000 | 1.0000000000 |
| 10 |    4 |   20 | 0.0000000000 | 1.0000000000 |
+----+------+------+--------------+--------------+
```

1.2.9.7.8 DENSE_RANK

Contents

Syntax

```
DENSE_RANK() OVER (
  [ PARTITION BY partition_expression ]
  [ ORDER BY order_list ]
)
```

Description

DENSE_RANK() is a window function that displays the number of a given row, starting at one and following the ORDER BY sequence of the window function, with identical values receiving the same result. Unlike the RANK() function, there are no skipped values if the preceding results are identical. It is also similar to the ROW_NUMBER() function except that in that function, identical values will receive a different row number for each result.

Examples

The distinction between DENSE_RANK(), RANK() and ROW_NUMBER():

```
CREATE TABLE student(course VARCHAR(10), mark int, name varchar(10));

INSERT INTO student VALUES
  ('Maths', 60, 'Thulile'),
  ('Maths', 60, 'Pritha'),
  ('Maths', 70, 'Voitto'),
  ('Maths', 55, 'Chun'),
  ('Biology', 60, 'Bilal'),
   ('Biology', 70, 'Roger');

SELECT
  RANK() OVER (PARTITION BY course ORDER BY mark DESC) AS rank,
  DENSE_RANK() OVER (PARTITION BY course ORDER BY mark DESC) AS dense_rank,
  ROW_NUMBER() OVER (PARTITION BY course ORDER BY mark DESC) AS row_num,
  course, mark, name
FROM student ORDER BY course, mark DESC;
+------+------------+---------+---------+------+---------+
| rank | dense_rank | row_num | course  | mark | name    |
+------+------------+---------+---------+------+---------+
|    1 |          1 |       1 | Biology |   70 | Roger   |
|    2 |          2 |       2 | Biology |   60 | Bilal   |
|    1 |          1 |       1 | Maths   |   70 | Voitto  |
|    2 |          2 |       2 | Maths   |   60 | Thulile |
|    2 |          2 |       3 | Maths   |   60 | Pritha  |
|    4 |          3 |       4 | Maths   |   55 | Chun    |
+------+------------+---------+---------+------+---------+
```

1.2.9.7.9 FIRST_VALUE

Contents

Syntax

```
FIRST_VALUE(expr) OVER (
  [ PARTITION BY partition_expression ]
  [ ORDER BY order_list ]
)
```

Description

`FIRST_VALUE` returns the first result from an ordered set, or NULL if no such result exists.

Examples

```
CREATE TABLE t1 (
  pk int primary key,
  a int,
  b int,
  c char(10),
  d decimal(10, 3),
  e real
);

INSERT INTO t1 VALUES
( 1, 0, 1,    'one',    0.1,  0.001),
( 2, 0, 2,    'two',    0.2,  0.002),
( 3, 0, 3,    'three',  0.3,  0.003),
( 4, 1, 2,    'three',  0.4,  0.004),
( 5, 1, 1,    'two',    0.5,  0.005),
( 6, 1, 1,    'one',    0.6,  0.006),
( 7, 2, NULL, 'n_one',  0.5,  0.007),
( 8, 2, 1,    'n_two',  NULL, 0.008),
( 9, 2, 2,    NULL,     0.7,  0.009),
(10, 2, 0,    'n_four', 0.8,  0.010),
(11, 2, 10,   NULL,     0.9,  NULL);

SELECT pk, FIRST_VALUE(pk) OVER (ORDER BY pk) AS first_asc,
           LAST_VALUE(pk) OVER (ORDER BY pk) AS last_asc,
           FIRST_VALUE(pk) OVER (ORDER BY pk DESC) AS first_desc,
           LAST_VALUE(pk) OVER (ORDER BY pk DESC) AS last_desc
FROM t1
ORDER BY pk DESC;

+----+-----------+----------+------------+-----------+
| pk | first_asc | last_asc | first_desc | last_desc |
+----+-----------+----------+------------+-----------+
| 11 |         1 |       11 |         11 |        11 |
| 10 |         1 |       10 |         11 |        10 |
|  9 |         1 |        9 |         11 |         9 |
|  8 |         1 |        8 |         11 |         8 |
|  7 |         1 |        7 |         11 |         7 |
|  6 |         1 |        6 |         11 |         6 |
|  5 |         1 |        5 |         11 |         5 |
|  4 |         1 |        4 |         11 |         4 |
|  3 |         1 |        3 |         11 |         3 |
|  2 |         1 |        2 |         11 |         2 |
|  1 |         1 |        1 |         11 |         1 |
+----+-----------+----------+------------+-----------+
```

```
CREATE OR REPLACE TABLE t1 (i int);
INSERT INTO t1 VALUES (1),(2),(3),(4),(5),(6),(7),(8),(9),(10);

SELECT i,
  FIRST_VALUE(i) OVER (ORDER BY i ROWS BETWEEN CURRENT ROW and 1 FOLLOWING) AS f_1f,
  LAST_VALUE(i) OVER (ORDER BY i ROWS BETWEEN CURRENT ROW and 1 FOLLOWING) AS l_1f,
  FIRST_VALUE(i) OVER (ORDER BY i ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS f_1p1f,
  LAST_VALUE(i) OVER (ORDER BY i ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS l_1p1f,
  FIRST_VALUE(i) OVER (ORDER BY i ROWS BETWEEN 2 PRECEDING AND 1 PRECEDING) AS f_2p1p,
  LAST_VALUE(i) OVER (ORDER BY i ROWS BETWEEN 2 PRECEDING AND 1 PRECEDING) AS l_2p1p,
  FIRST_VALUE(i) OVER (ORDER BY i ROWS BETWEEN 1 FOLLOWING AND 2 FOLLOWING) AS f_1f2f,
  LAST_VALUE(i) OVER (ORDER BY i ROWS BETWEEN 1 FOLLOWING AND 2 FOLLOWING) AS l_1f2f
FROM t1;

+------+------+------+--------+--------+--------+--------+--------+--------+
| i    | f_1f | l_1f | f_1p1f | l_1p1f | f_2p1p | l_2p1p | f_1f2f | l_1f2f |
+------+------+------+--------+--------+--------+--------+--------+--------+
|    1 |    1 |    2 |      1 |      2 |   NULL |   NULL |      2 |      3 |
|    2 |    2 |    3 |      1 |      3 |      1 |      1 |      3 |      4 |
|    3 |    3 |    4 |      2 |      4 |      1 |      2 |      4 |      5 |
|    4 |    4 |    5 |      3 |      5 |      2 |      3 |      5 |      6 |
|    5 |    5 |    6 |      4 |      6 |      3 |      4 |      6 |      7 |
|    6 |    6 |    7 |      5 |      7 |      4 |      5 |      7 |      8 |
|    7 |    7 |    8 |      6 |      8 |      5 |      6 |      8 |      9 |
|    8 |    8 |    9 |      7 |      9 |      6 |      7 |      9 |     10 |
|    9 |    9 |   10 |      8 |     10 |      7 |      8 |     10 |     10 |
|   10 |   10 |   10 |      9 |     10 |      8 |      9 |   NULL |   NULL |
+------+------+------+--------+--------+--------+--------+--------+--------+
```

1.2.4.9 JSON_ARRAYAGG

1.2.4.10 JSON_OBJECTAGG

1.2.9.7.12 LAG

Contents

Syntax

```
LAG (expr[, offset]) OVER (
  [ PARTITION BY partition_expression ]
  < ORDER BY order_list >
)
```

Description

The *LAG* function accesses data from a previous row according to the ORDER BY clause without the need for a self-join. The specific row is determined by the *offset* (default *1*), which specifies the number of rows behind the current row to use. An offset of *0* is the current row.

Examples

```
CREATE TABLE t1 (pk int primary key, a int, b int, c char(10), d decimal(10, 3), e real);

INSERT INTO t1 VALUES
 ( 1, 0, 1,    'one',    0.1,  0.001),
 ( 2, 0, 2,    'two',    0.2,  0.002),
 ( 3, 0, 3,    'three',  0.3,  0.003),
 ( 4, 1, 2,    'three',  0.4,  0.004),
 ( 5, 1, 1,    'two',    0.5,  0.005),
 ( 6, 1, 1,    'one',    0.6,  0.006),
 ( 7, 2, NULL, 'n_one',  0.5,  0.007),
 ( 8, 2, 1,    'n_two',  NULL, 0.008),
 ( 9, 2, 2,    NULL,     0.7,  0.009),
 (10, 2, 0,    'n_four', 0.8,  0.010),
 (11, 2, 10,   NULL,     0.9,  NULL);

SELECT pk, LAG(pk) OVER (ORDER BY pk) AS l,
  LAG(pk,1) OVER (ORDER BY pk) AS l1,
  LAG(pk,2) OVER (ORDER BY pk) AS l2,
  LAG(pk,0) OVER (ORDER BY pk) AS l0,
  LAG(pk,-1) OVER (ORDER BY pk) AS lm1,
  LAG(pk,-2) OVER (ORDER BY pk) AS lm2
FROM t1;
+----+------+------+------+------+------+------+
| pk | l    | l1   | l2   | l0   | lm1  | lm2  |
+----+------+------+------+------+------+------+
|  1 | NULL | NULL | NULL |    1 |    2 |    3 |
|  2 |    1 |    1 | NULL |    2 |    3 |    4 |
|  3 |    2 |    2 |    1 |    3 |    4 |    5 |
|  4 |    3 |    3 |    2 |    4 |    5 |    6 |
|  5 |    4 |    4 |    3 |    5 |    6 |    7 |
|  6 |    5 |    5 |    4 |    6 |    7 |    8 |
|  7 |    6 |    6 |    5 |    7 |    8 |    9 |
|  8 |    7 |    7 |    6 |    8 |    9 |   10 |
|  9 |    8 |    8 |    7 |    9 |   10 |   11 |
| 10 |    9 |    9 |    8 |   10 |   11 | NULL |
| 11 |   10 |   10 |    9 |   11 | NULL | NULL |
+----+------+------+------+------+------+------+
```

1.2.8.3.14 LAST_VALUE

1.2.9.7.14 LEAD

Contents

Syntax

```
LEAD (expr[, offset]) OVER (
  [ PARTITION BY partition_expression ]
  [ ORDER BY order_list ]
)
```

Description

The *LEAD* function accesses data from a following row in the same result set without the need for a self-join. The specific row is determined by the *offset* (default *1*), which specifies the number of rows ahead the current row to use. An offset of *0* is the current row.

Example

```
CREATE TABLE t1 (pk int primary key, a int, b int, c char(10), d decimal(10, 3), e real);

INSERT INTO t1 VALUES
 ( 1, 0, 1,    'one',    0.1,  0.001),
 ( 2, 0, 2,    'two',    0.2,  0.002),
 ( 3, 0, 3,    'three',  0.3,  0.003),
 ( 4, 1, 2,    'three',  0.4,  0.004),
 ( 5, 1, 1,    'two',    0.5,  0.005),
 ( 6, 1, 1,    'one',    0.6,  0.006),
 ( 7, 2, NULL, 'n_one',  0.5,  0.007),
 ( 8, 2, 1,    'n_two',  NULL, 0.008),
 ( 9, 2, 2,    NULL,     0.7,  0.009),
 (10, 2, 0,    'n_four', 0.8,  0.010),
 (11, 2, 10,   NULL,     0.9,  NULL);

SELECT pk, LEAD(pk) OVER (ORDER BY pk) AS l,
  LEAD(pk,1) OVER (ORDER BY pk) AS l1,
  LEAD(pk,2) OVER (ORDER BY pk) AS l2,
  LEAD(pk,0) OVER (ORDER BY pk) AS l0,
  LEAD(pk,-1) OVER (ORDER BY pk) AS lm1,
  LEAD(pk,-2) OVER (ORDER BY pk) AS lm2
FROM t1;
+----+------+------+------+------+------+------+
| pk | l    | l1   | l2   | l0   | lm1  | lm2  |
+----+------+------+------+------+------+------+
|  1 |    2 |    2 |    3 |    1 | NULL | NULL |
|  2 |    3 |    3 |    4 |    2 |    1 | NULL |
|  3 |    4 |    4 |    5 |    3 |    2 |    1 |
|  4 |    5 |    5 |    6 |    4 |    3 |    2 |
|  5 |    6 |    6 |    7 |    5 |    4 |    3 |
|  6 |    7 |    7 |    8 |    6 |    5 |    4 |
|  7 |    8 |    8 |    9 |    7 |    6 |    5 |
|  8 |    9 |    9 |   10 |    8 |    7 |    6 |
|  9 |   10 |   10 |   11 |    9 |    8 |    7 |
| 10 |   11 |   11 | NULL |   10 |    9 |    8 |
| 11 | NULL | NULL | NULL |   11 |   10 |    9 |
+----+------+------+------+------+------+------+
```

1.2.4.11 MAX

1.2.9.7.16 MEDIAN

Contents

Syntax

```
MEDIAN(median expression) OVER (
  [ PARTITION BY partition_expression ]
)
```

Description

MEDIAN() is a window function that returns the median value of a range of values.

It is a specific case of PERCENTILE_CONT, with an argument of 0.5 and the ORDER BY column the one in `MEDIAN` 's argument.

```
MEDIAN(<median-arg>) OVER ( [ PARTITION BY partition_expression] )
```

Is equivalent to:

```
PERCENTILE_CONT(0.5) WITHIN
  GROUP (ORDER BY <median-arg>) OVER ( [ PARTITION BY partition_expression ])
```

Examples

```
CREATE TABLE book_rating (name CHAR(30), star_rating TINYINT);

INSERT INTO book_rating VALUES ('Lord of the Ladybirds', 5);
INSERT INTO book_rating VALUES ('Lord of the Ladybirds', 3);
INSERT INTO book_rating VALUES ('Lady of the Flies', 1);
INSERT INTO book_rating VALUES ('Lady of the Flies', 2);
INSERT INTO book_rating VALUES ('Lady of the Flies', 5);

SELECT name, median(star_rating) OVER (PARTITION BY name) FROM book_rating;
+----------------------+-----------------------------------------------+
| name                 | median(star_rating) OVER (PARTITION BY name)  |
+----------------------+-----------------------------------------------+
| Lord of the Ladybirds |                                  4.0000000000 |
| Lord of the Ladybirds |                                  4.0000000000 |
| Lady of the Flies     |                                  2.0000000000 |
| Lady of the Flies     |                                  2.0000000000 |
| Lady of the Flies     |                                  2.0000000000 |
+----------------------+-----------------------------------------------+
```

1.2.4.12 MIN

1.2.9.7.18 NTH_VALUE

Syntax

```
NTH_VALUE (expr[, num_row]) OVER (
  [ PARTITION BY partition_expression ]
  [ ORDER BY order_list ]
)
```

Contents

Description

The `NTH_VALUE` function returns the value evaluated at row number `num_row` of the window frame, starting from 1, or NULL if the row does not exist.

1.2.9.7.19 NTILE

Contents

Syntax

```
NTILE (expr) OVER (
  [ PARTITION BY partition_expression ]
  [ ORDER BY order_list ]
)
```

Description

NTILE() is a window function that returns an integer indicating which group a given row falls into. The number of groups is specified in the argument (*expr*), starting at one. Ordered rows in the partition are divided into the specified number of groups with as equal a size as possible.

Examples

```
create table t1 (
    pk int primary key,
    a int,
    b int
  );

insert into t1 values
    (11 , 0, 10),
    (12 , 0, 10),
    (13 , 1, 10),
    (14 , 1, 10),
    (18 , 2, 10),
    (15 , 2, 20),
    (16 , 2, 20),
    (17 , 2, 20),
    (19 , 4, 20),
    (20 , 4, 20);

select pk, a, b,
    ntile(1) over (order by pk)
  from t1;
+----+------+------+-----------------------------+
| pk | a    | b    | ntile(1) over (order by pk) |
+----+------+------+-----------------------------+
| 11 |    0 |   10 |                           1 |
| 12 |    0 |   10 |                           1 |
| 13 |    1 |   10 |                           1 |
| 14 |    1 |   10 |                           1 |
| 15 |    2 |   20 |                           1 |
| 16 |    2 |   20 |                           1 |
| 17 |    2 |   20 |                           1 |
| 18 |    2 |   10 |                           1 |
| 19 |    4 |   20 |                           1 |
| 20 |    4 |   20 |                           1 |
+----+------+------+-----------------------------+

select pk, a, b,
    ntile(4) over (order by pk)
 from t1;
+----+------+------+-----------------------------+
| pk | a    | b    | ntile(4) over (order by pk) |
+----+------+------+-----------------------------+
| 11 |    0 |   10 |                           1 |
| 12 |    0 |   10 |                           1 |
| 13 |    1 |   10 |                           1 |
| 14 |    1 |   10 |                           2 |
| 15 |    2 |   20 |                           2 |
| 16 |    2 |   20 |                           2 |
| 17 |    2 |   20 |                           3 |
| 18 |    2 |   10 |                           3 |
| 19 |    4 |   20 |                           4 |
| 20 |    4 |   20 |                           4 |
+----+------+------+-----------------------------+
```

1.2.9.7.20 PERCENT_RANK

Contents

Syntax

```
PERCENT_RANK() OVER (
  [ PARTITION BY partition_expression ]
  [ ORDER BY order_list ]
)
```

Description

PERCENT_RANK() is a window function that returns the relative percent rank of a given row. The following formula is used to calculate the percent rank:

```
(rank - 1) / (number of rows in the window or partition - 1)
```

Examples

```
create table t1 (
  pk int primary key,
  a int,
  b int
);


insert into t1 values
( 1 , 0, 10),
( 2 , 0, 10),
( 3 , 1, 10),
( 4 , 1, 10),
( 8 , 2, 10),
( 5 , 2, 20),
( 6 , 2, 20),
( 7 , 2, 20),
( 9 , 4, 20),
(10 , 4, 20);

select pk, a, b,
    rank() over (order by a) as rank,
    percent_rank() over (order by a) as pct_rank,
    cume_dist() over (order by a) as cume_dist
from t1;
+----+------+------+------+--------------+--------------+
| pk | a    | b    | rank | pct_rank     | cume_dist    |
+----+------+------+------+--------------+--------------+
|  1 |    0 |   10 |    1 | 0.0000000000 | 0.2000000000 |
|  2 |    0 |   10 |    1 | 0.0000000000 | 0.2000000000 |
|  3 |    1 |   10 |    3 | 0.2222222222 | 0.4000000000 |
|  4 |    1 |   10 |    3 | 0.2222222222 | 0.4000000000 |
|  5 |    2 |   20 |    5 | 0.4444444444 | 0.8000000000 |
|  6 |    2 |   20 |    5 | 0.4444444444 | 0.8000000000 |
|  7 |    2 |   20 |    5 | 0.4444444444 | 0.8000000000 |
|  8 |    2 |   10 |    5 | 0.4444444444 | 0.8000000000 |
|  9 |    4 |   20 |    9 | 0.8888888889 | 1.0000000000 |
| 10 |    4 |   20 |    9 | 0.8888888889 | 1.0000000000 |
+----+------+------+------+--------------+--------------+

select pk, a, b,
       percent_rank() over (order by pk) as pct_rank,
       cume_dist() over (order by pk) as cume_dist
from t1 order by pk;
+----+------+------+--------------+--------------+
| pk | a    | b    | pct_rank     | cume_dist    |
+----+------+------+--------------+--------------+
|  1 |    0 |   10 | 0.0000000000 | 0.1000000000 |
|  2 |    0 |   10 | 0.1111111111 | 0.2000000000 |
|  3 |    1 |   10 | 0.2222222222 | 0.3000000000 |
|  4 |    1 |   10 | 0.3333333333 | 0.4000000000 |
|  5 |    2 |   20 | 0.4444444444 | 0.5000000000 |
|  6 |    2 |   20 | 0.5555555556 | 0.6000000000 |
|  7 |    2 |   20 | 0.6666666667 | 0.7000000000 |
```

```
|  8 |    2 |   10 | 0.7777777778 | 0.8000000000 |
|  9 |    4 |   20 | 0.8888888889 | 0.9000000000 |
| 10 |    4 |   20 | 1.0000000000 | 1.0000000000 |
+----+------+------+--------------+--------------+

select pk, a, b,
       percent_rank() over (partition by a order by a) as pct_rank,
       cume_dist() over (partition by a order by a) as cume_dist
from t1;
+----+------+------+--------------+--------------+
| pk | a    | b    | pct_rank     | cume_dist    |
+----+------+------+--------------+--------------+
|  1 |    0 |   10 | 0.0000000000 | 1.0000000000 |
|  2 |    0 |   10 | 0.0000000000 | 1.0000000000 |
|  3 |    1 |   10 | 0.0000000000 | 1.0000000000 |
|  4 |    1 |   10 | 0.0000000000 | 1.0000000000 |
|  5 |    2 |   20 | 0.0000000000 | 1.0000000000 |
|  6 |    2 |   20 | 0.0000000000 | 1.0000000000 |
|  7 |    2 |   20 | 0.0000000000 | 1.0000000000 |
|  8 |    2 |   10 | 0.0000000000 | 1.0000000000 |
|  9 |    4 |   20 | 0.0000000000 | 1.0000000000 |
| 10 |    4 |   20 | 0.0000000000 | 1.0000000000 |
+----+------+------+--------------+--------------+
```

1.2.9.7.21 PERCENTILE_CONT

MariaDB starting with 10.3.3

The PERCENTILE_CONT() window function was first introduced with in MariaDB 10.3.3.

Contents

Syntax

Description

`PERCENTILE_CONT()` (standing for continuous percentile) is a window function which returns a value which corresponds to the given fraction in the sort order. If required, it will interpolate between adjacent input items.

Essentially, the following process is followed to find the value to return:

- Get the number of rows in the partition, denoted by N
- RN = p*(N-1), where p denotes the argument to the PERCENTILE_CONT function
- calculate the FRN(floor row number) and CRN(column row number for the group(FRN= floor(RN) and CRN = ceil(RN))
- look up rows FRN and CRN
- If (CRN = FRN = RN) then the result is (value of expression from row at RN)
- Otherwise the result is
- (CRN - RN) * (value of expression for row at FRN) +
- (RN - FRN) * (value of expression for row at CRN)

The MEDIAN function is a specific case of `PERCENTILE_CONT`, equivalent to `PERCENTILE_CONT(0.5)`.

Examples

```
CREATE TABLE book_rating (name CHAR(30), star_rating TINYINT);

INSERT INTO book_rating VALUES ('Lord of the Ladybirds', 5);
INSERT INTO book_rating VALUES ('Lord of the Ladybirds', 3);
INSERT INTO book_rating VALUES ('Lady of the Flies', 1);
INSERT INTO book_rating VALUES ('Lady of the Flies', 2);
INSERT INTO book_rating VALUES ('Lady of the Flies', 5);

SELECT name, PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY star_rating)
  OVER (PARTITION BY name) AS pc
  FROM book_rating;
+----------------------+--------------+
| name                 | pc           |
+----------------------+--------------+
| Lord of the Ladybirds | 4.0000000000 |
| Lord of the Ladybirds | 4.0000000000 |
| Lady of the Flies     | 2.0000000000 |
| Lady of the Flies     | 2.0000000000 |
| Lady of the Flies     | 2.0000000000 |
+----------------------+--------------+

SELECT name, PERCENTILE_CONT(1) WITHIN GROUP (ORDER BY star_rating)
  OVER (PARTITION BY name) AS pc
  FROM book_rating;
+----------------------+--------------+
| name                 | pc           |
+----------------------+--------------+
| Lord of the Ladybirds | 5.0000000000 |
| Lord of the Ladybirds | 5.0000000000 |
| Lady of the Flies     | 5.0000000000 |
| Lady of the Flies     | 5.0000000000 |
| Lady of the Flies     | 5.0000000000 |
+----------------------+--------------+

SELECT name, PERCENTILE_CONT(0) WITHIN GROUP (ORDER BY star_rating)
  OVER (PARTITION BY name) AS pc
  FROM book_rating;
+----------------------+--------------+
| name                 | pc           |
+----------------------+--------------+
| Lord of the Ladybirds | 3.0000000000 |
| Lord of the Ladybirds | 3.0000000000 |
| Lady of the Flies     | 1.0000000000 |
| Lady of the Flies     | 1.0000000000 |
| Lady of the Flies     | 1.0000000000 |
+----------------------+--------------+

SELECT name, PERCENTILE_CONT(0.6) WITHIN GROUP (ORDER BY star_rating)
  OVER (PARTITION BY name) AS pc
  FROM book_rating;
+----------------------+--------------+
| name                 | pc           |
+----------------------+--------------+
| Lord of the Ladybirds | 4.2000000000 |
| Lord of the Ladybirds | 4.2000000000 |
| Lady of the Flies     | 2.6000000000 |
| Lady of the Flies     | 2.6000000000 |
| Lady of the Flies     | 2.6000000000 |
+----------------------+--------------+
```

1.2.9.7.22 PERCENTILE_DISC

MariaDB starting with 10.3.3

The PERCENTILE_DISC() window function was first introduced with in MariaDB 10.3.3.

Syntax

Contents

Description

`PERCENTILE_DISC()` (standing for discrete percentile) is a window function which returns the first value in the set whose ordered position is the same or more than the specified fraction.

Essentially, the following process is followed to find the value to return:

- Get the number of rows in the partition.
- Walk through the partition, in order, until finding the the first row with `CUME_DIST()` >= function_argument.

Examples

```
CREATE TABLE book_rating (name CHAR(30), star_rating TINYINT);

INSERT INTO book_rating VALUES ('Lord of the Ladybirds', 5);
INSERT INTO book_rating VALUES ('Lord of the Ladybirds', 3);
INSERT INTO book_rating VALUES ('Lady of the Flies', 1);
INSERT INTO book_rating VALUES ('Lady of the Flies', 2);
INSERT INTO book_rating VALUES ('Lady of the Flies', 5);

SELECT name, PERCENTILE_DISC(0.5) WITHIN GROUP (ORDER BY star_rating)
  OVER (PARTITION BY name) AS pc FROM book_rating;
+----------------------+------+
| name                 | pc   |
+----------------------+------+
| Lord of the Ladybirds |    3 |
| Lord of the Ladybirds |    3 |
| Lady of the Flies     |    2 |
| Lady of the Flies     |    2 |
| Lady of the Flies     |    2 |
+----------------------+------+
5 rows in set (0.000 sec)

SELECT name, PERCENTILE_DISC(0) WITHIN GROUP (ORDER BY star_rating)
 OVER (PARTITION BY name) AS pc FROM book_rating;
+----------------------+------+
| name                 | pc   |
+----------------------+------+
| Lord of the Ladybirds |    3 |
| Lord of the Ladybirds |    3 |
| Lady of the Flies     |    1 |
| Lady of the Flies     |    1 |
| Lady of the Flies     |    1 |
+----------------------+------+
5 rows in set (0.000 sec)

SELECT name, PERCENTILE_DISC(1) WITHIN GROUP (ORDER BY star_rating)
  OVER (PARTITION BY name) AS pc FROM book_rating;
+----------------------+------+
| name                 | pc   |
+----------------------+------+
| Lord of the Ladybirds |    5 |
| Lord of the Ladybirds |    5 |
| Lady of the Flies     |    5 |
| Lady of the Flies     |    5 |
| Lady of the Flies     |    5 |
+----------------------+------+
5 rows in set (0.000 sec)

SELECT name, PERCENTILE_DISC(0.6) WITHIN GROUP (ORDER BY star_rating)
  OVER (PARTITION BY name) AS pc FROM book_rating;
+----------------------+------+
| name                 | pc   |
+----------------------+------+
| Lord of the Ladybirds |    5 |
| Lord of the Ladybirds |    5 |
| Lady of the Flies     |    2 |
| Lady of the Flies     |    2 |
| Lady of the Flies     |    2 |
+----------------------+------
```

1.2.9.7.23 RANK

Contents

Syntax

```
RANK() OVER (
  [ PARTITION BY partition_expression ]
  [ ORDER BY order_list ]
)
```

Description

RANK() is a window function that displays the number of a given row, starting at one and following the ORDER BY sequence of the window function, with identical values receiving the same result. It is similar to the ROW_NUMBER() function except that in that function, identical values will receive a different row number for each result.

Examples

The distinction between DENSE_RANK(), RANK() and ROW_NUMBER():

```
CREATE TABLE student(course VARCHAR(10), mark int, name varchar(10));

INSERT INTO student VALUES
  ('Maths', 60, 'Thulile'),
  ('Maths', 60, 'Pritha'),
  ('Maths', 70, 'Voitto'),
  ('Maths', 55, 'Chun'),
  ('Biology', 60, 'Bilal'),
   ('Biology', 70, 'Roger');

SELECT
  RANK() OVER (PARTITION BY course ORDER BY mark DESC) AS rank,
  DENSE_RANK() OVER (PARTITION BY course ORDER BY mark DESC) AS dense_rank,
  ROW_NUMBER() OVER (PARTITION BY course ORDER BY mark DESC) AS row_num,
  course, mark, name
FROM student ORDER BY course, mark DESC;
+------+------------+---------+---------+------+---------+
| rank | dense_rank | row_num | course  | mark | name    |
+------+------------+---------+---------+------+---------+
|    1 |          1 |       1 | Biology |   70 | Roger   |
|    2 |          2 |       2 | Biology |   60 | Bilal   |
|    1 |          1 |       1 | Maths   |   70 | Voitto  |
|    2 |          2 |       2 | Maths   |   60 | Thulile |
|    2 |          2 |       3 | Maths   |   60 | Pritha  |
|    4 |          3 |       4 | Maths   |   55 | Chun    |
+------+------------+---------+---------+------+---------+
```

1.2.9.7.24 ROW_NUMBER

Contents

Syntax

```
ROW_NUMBER() OVER (
  [ PARTITION BY partition_expression ]
  [ ORDER BY order_list ]
)
```

Description

ROW_NUMBER() is a window function that displays the number of a given row, starting at one and following the ORDER BY sequence of the window function, with identical values receiving different row numbers. It is similar to the RANK() and DENSE_RANK() functions except that in that function, identical values will receive the same rank for each result.

Examples

The distinction between DENSE_RANK(), RANK() and ROW_NUMBER():

```
CREATE TABLE student(course VARCHAR(10), mark int, name varchar(10));

INSERT INTO student VALUES
  ('Maths', 60, 'Thulile'),
  ('Maths', 60, 'Pritha'),
  ('Maths', 70, 'Voitto'),
  ('Maths', 55, 'Chun'),
  ('Biology', 60, 'Bilal'),
   ('Biology', 70, 'Roger');

SELECT
  RANK() OVER (PARTITION BY course ORDER BY mark DESC) AS rank,
  DENSE_RANK() OVER (PARTITION BY course ORDER BY mark DESC) AS dense_rank,
  ROW_NUMBER() OVER (PARTITION BY course ORDER BY mark DESC) AS row_num,
  course, mark, name
FROM student ORDER BY course, mark DESC;
+------+------------+---------+---------+------+---------+
| rank | dense_rank | row_num | course  | mark | name    |
+------+------------+---------+---------+------+---------+
|    1 |          1 |       1 | Biology |   70 | Roger   |
|    2 |          2 |       2 | Biology |   60 | Bilal   |
|    1 |          1 |       1 | Maths   |   70 | Voitto  |
|    2 |          2 |       2 | Maths   |   60 | Thulile |
|    2 |          2 |       3 | Maths   |   60 | Pritha  |
|    4 |          3 |       4 | Maths   |   55 | Chun    |
+------+------------+---------+---------+------+---------+
```

1.2.4.13 STD

1.2.4.14 STDDEV

1.2.4.15 STDDEV_POP

1.2.4.16 STDDEV_SAMP

1.2.4.17 SUM

1.2.4.18 VARIANCE

1.2.4.19 VAR_POP

1.2.4.20 VAR_SAMP

1.2.9.7.33 Aggregate Functions as Window Functions

It is possible to use aggregate functions as window functions. An aggregate function used as a window function must have the `OVER` clause. For example, here's COUNT() used as a window function:

```
select COUNT(*) over  (order by column) from table;
```

MariaDB currently allows these aggregate functions to be used as window functions:

- AVG
- BIT_AND
- BIT_OR

- BIT_XOR
- COUNT
- JSON_ARRAYAGG
- JSON_OBJECTAGG
- MAX
- MIN
- STD
- STDDEV
- STDDEV_POP
- STDDEV_SAMP
- SUM
- VAR_POP
- VAR_SAMP
- VARIANCE

1.2.9.7.34 ColumnStore Window Functions

Introduction

MariaDB ColumnStore provides support for window functions broadly following the SQL 2003 specification. A window function allows for calculations relating to a window of data surrounding the current row in a result set. This capability provides for simplified queries in support of common business questions such as cumulative totals, rolling averages, and top 10 lists.

Aggregate functions are utilized for window functions however differ in behavior from a group by query because the rows remain ungrouped. This provides support for cumulative sums and rolling averages, for example.

Two key concepts for window functions are Partition and Frame:

- A Partition is a group of rows, or window, that have the same value for a specific column, for example a Partition can be created over a time period such as a quarter or lookup values.
- The Frame for each row is a subset of the row's Partition. The frame typically is dynamic allowing for a sliding frame of rows within the Partition. The Frame determines the range of rows for the windowing function. A Frame could be defined as the last X rows and next Y rows all the way up to the entire Partition.

Window functions are applied after joins, group by, and having clauses are calculated.

Syntax

A window function is applied in the select clause using the following syntax:

```
function_name ([expression [, expression ... ]]) OVER ( window_definition )
```

where *window_definition* is defined as:

```
[ PARTITION BY expression [, ...] ]
[ ORDER BY expression [ ASC | DESC ] [ NULLS { FIRST | LAST } ] [, ...] ]
[ frame_clause ]
```

PARTITION BY:

- Divides the window result set into groups based on one or more *expressions*.
- An expression may be a constant, column, and non window function expressions.
- A query is not limited to a single partition by clause. Different partition clauses can be used across different window function applications.
- The partition by columns do not need to be in the select list but do need to be available from the query result set.
- If there is no PARTITION BY clause, all rows of the result set define the group.

ORDER BY

- Defines the ordering of values within the partition.
- Can be ordered by multiple keys which may be a constant, column or non window function expression.
- The order by columns do not need to be in the select list but need to be available from the query result set.
- Use of a select column alias from the query is not supported.
- ASC (default) and DESC options allow for ordering ascending or descending.
- NULLS FIRST and NULL_LAST options specify whether null values come first or last in the ordering sequence. NULLS_FIRST is the default for ASC order, and NULLS_LAST is the default for DESC order.

and the optional *frame_clause* is defined as:

```
{ RANGE | ROWS } frame_start
{ RANGE | ROWS } BETWEEN frame_start AND frame_end
```

and the optional *frame_start* and *frame_end* are defined as (value being a numeric expression):

```
UNBOUNDED PRECEDING
value PRECEDING
CURRENT ROW
value FOLLOWING
UNBOUNDED FOLLOWING
```

RANGE/ROWS:

- Defines the windowing clause for calculating the set of rows that the function applies to for calculating a given rows window function result.
- Requires an ORDER BY clause to define the row order for the window.
- ROWS specify the window in physical units, i.e. result set rows and must be a constant or expression evaluating to a positive numeric value.
- RANGE specifies the window as a logical offset. If the the expression evaluates to a numeric value then the ORDER BY expression must be a numeric or DATE type. If the expression evaluates to an interval value then the ORDER BY expression must be a DATE data type.
- UNBOUNDED PRECEDING indicates the window starts at the first row of the partition.
- UNBOUNDED FOLLOWING indicates the window ends at the last row of the partition.
- CURRENT ROW specifies the window start or ends at the current row or value.
- If omitted, the default is ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW.

Supported Functions

Function	Description
AVG()	The average of all input values.
COUNT()	Number of input rows.
CUME_DIST()	Calculates the cumulative distribution, or relative rank, of the current row to other rows in the same partition. Number of peer or preceding rows / number of rows in partition.
DENSE_RANK()	Ranks items in a group leaving no gaps in ranking sequence when there are ties.
FIRST_VALUE()	The value evaluated at the row that is the first row of the window frame (counting from 1); null if no such row.
LAG()	The value evaluated at the row that is offset rows before the current row within the partition; if there is no such row, instead return default. Both offset and default are evaluated with respect to the current row. If omitted, offset defaults to 1 and default to null. LAG provides access to more than one row of a table at the same time without a self-join. Given a series of rows returned from a query and a position of the cursor, LAG provides access to a row at a given physical offset prior to that position.
LAST_VALUE()	The value evaluated at the row that is the last row of the window frame (counting from 1); null if no such row.
LEAD()	Provides access to a row at a given physical offset beyond that position. Returns value evaluated at the row that is offset rows after the current row within the partition; if there is no such row, instead return default. Both offset and default are evaluated with respect to the current row. If omitted, offset defaults to 1 and default to null.
MAX()	Maximum value of expression across all input values.
MEDIAN()	An inverse distribution function that assumes a continuous distribution model. It takes a numeric or datetime value and returns the middle value or an interpolated value that would be the middle value once the values are sorted. Nulls are ignored in the calculation.
MIN()	Minimum value of expression across all input values.
NTH_VALUE()	The value evaluated at the row that is the nth row of the window frame (counting from 1); null if no such row.

NTILE()	Divides an ordered data set into a number of buckets indicated by expr and assigns the appropriate bucket number to each row. The buckets are numbered 1 through expr. The expr value must resolve to a positive constant for each partition. Integer ranging from 1 to the argument value, dividing the partition as equally as possible.
PERCENT_RANK()	relative rank of the current row: (rank - 1) / (total rows - 1).
PERCENTILE_CONT()	An inverse distribution function that assumes a continuous distribution model. It takes a percentile value and a sort specification, and returns an interpolated value that would fall into that percentile value with respect to the sort specification. Nulls are ignored in the calculation.
PERCENTILE_DISC()	An inverse distribution function that assumes a discrete distribution model. It takes a percentile value and a sort specification and returns an element from the set. Nulls are ignored in the calculation.
RANK()	rank of the current row with gaps; same as row_number of its first peer.
ROW_NUMBER()	number of the current row within its partition, counting from 1
STDDEV() STDDEV_POP()	Computes the population standard deviation and returns the square root of the population variance.
STDDEV_SAMP()	Computes the cumulative sample standard deviation and returns the square root of the sample variance.
SUM()	Sum of expression across all input values.
VARIANCE() VAR_POP()	Population variance of the input values (square of the population standard deviation).
VAR_SAMP()	Sample variance of the input values (square of the sample standard deviation).

Examples

Example Schema

The examples are all based on the following simplified sales opportunity table:

```
create table opportunities (
id int,
accountName varchar(20),
name varchar(128),
owner varchar(7),
amount decimal(10,2),
closeDate date,
stageName varchar(11)
) engine=columnstore;
```

Some example values are (thanks to https://www.mockaroo.com for sample data generation):

id	accountName	name	owner	amount	closeDate	stageName
1	Browseblab	Multi-lateral executive function	Bob	26444.86	2016-10-20	Negotiating
2	Mita	Organic demand-driven benchmark	Maria	477878.41	2016-11-28	ClosedWon
3	Miboo	De-engineered hybrid groupware	Olivier	80181.78	2017-01-05	ClosedWon
4	Youbridge	Enterprise-wide bottom-line Graphic Interface	Chris	946245.29	2016-07-02	ClosedWon
5	Skyba	Reverse-engineered fresh-thinking standardization	Maria	696241.82	2017-02-17	Negotiating
6	Eayo	Fundamental well-modulated artificial intelligence	Bob	765605.52	2016-08-27	Prospecting
7	Yotz	Extended secondary infrastructure	Chris	319624.20	2017-01-06	ClosedLost
8	Oloo	Configurable web-enabled data-warehouse	Chris	321016.26	2017-03-08	ClosedLost
9	Kaymbo	Multi-lateral web-enabled definition	Bob	690881.01	2017-01-02	Developing
10	Rhyloo	Public-key coherent infrastructure	Chris	965477.74	2016-11-07	Prospecting

The schema, sample data, and queries are available as an attachment to this article.

Cumulative Sum and Running Max Example

Window functions can be used to achieve cumulative / running calculations on a detail report. In this case a won opportunity report for a 7 day period adds columns to show the accumulated won amount as well as the current highest opportunity amount in preceding rows.

```
select owner,
accountName,
CloseDate,
amount,
sum(amount) over (order by CloseDate rows between unbounded preceding and current row) cumeWon,

max(amount) over (order by CloseDate rows between unbounded preceding and current row) runningM
ax
from opportunities
where stageName='ClosedWon'
and closeDate >= '2016-10-02' and closeDate <= '2016-10-09'
order by CloseDate;
```

with example results:

owner	accountName	CloseDate	amount	cumeWon	runningMax
Bill	Babbleopia	2016-10-02	437636.47	437636.47	437636.47
Bill	Thoughtworks	2016-10-04	146086.51	583722.98	437636.47
Olivier	Devpulse	2016-10-05	834235.93	1417958.91	834235.93
Chris	Linkbridge	2016-10-07	539977.45	2458738.65	834235.93
Olivier	Trupe	2016-10-07	500802.29	1918761.20	834235.93
Bill	Latz	2016-10-08	857254.87	3315993.52	857254.87
Chris	Avamm	2016-10-09	699566.86	4015560.38	857254.87

Partitioned Cumulative Sum and Running Max Example

The above example can be partitioned, so that the window functions are over a particular field grouping such as owner and accumulate within that grouping. This is achieved by adding the syntax "partition by <columns>" in the window function clause.

```
select owner,
accountName,
CloseDate,
amount,
sum(amount) over (partition by owner order by CloseDate rows between unbounded preceding and cu
rrent row) cumeWon,
max(amount) over (partition by owner order by CloseDate rows between unbounded preceding and cu
rrent row) runningMax
from opportunities
where stageName='ClosedWon'
and closeDate >= '2016-10-02' and closeDate <= '2016-10-09'
order by owner, CloseDate;
```

with example results:

owner	accountName	CloseDate	amount	cumeWon	runningMax
Bill	Babbleopia	2016-10-02	437636.47	437636.47	437636.47
Bill	Thoughtworks	2016-10-04	146086.51	583722.98	437636.47
Bill	Latz	2016-10-08	857254.87	1440977.85	857254.87
Chris	Linkbridge	2016-10-07	539977.45	539977.45	539977.45
Chris	Avamm	2016-10-09	699566.86	1239544.31	699566.86
Olivier	Devpulse	2016-10-05	834235.93	834235.93	834235.93
Olivier	Trupe	2016-10-07	500802.29	1335038.22	834235.93

Ranking / Top Results

The rank window function allows for ranking or assigning a numeric order value based on the window function definition.

Using the Rank() function will result in the same value for ties / equal values and the next rank value skipped. The Dense_Rank() function behaves similarly except the next consecutive number is used after a tie rather than skipped. The Row_Number() function will provide a unique ordering value. The example query shows the Rank() function being applied to rank sales reps by the number of opportunities for Q4 2016.

```
select owner,
wonCount,
rank() over (order by wonCount desc) rank
from (
  select owner,
  count(*) wonCount
  from opportunities
  where stageName='ClosedWon'
  and closeDate >= '2016-10-01' and closeDate < '2016-12-31'
  group by owner
) t
order by rank;
```

with example results (note the query is technically incorrect by using closeDate < '2016-12-31' however this creates a tie scenario for illustrative purposes):

owner	wonCount	rank
Bill	19	1
Chris	15	2
Maria	14	3
Bob	14	3
Olivier	10	5

If the dense_rank function is used the rank values would be 1,2,3,3,4 and for the row_number function the values would be 1,2,3,4,5.

First and Last Values

The first_value and last_value functions allow determining the first and last values of a given range. Combined with a group by this allows summarizing opening and closing values. The example shows a more complex case where detailed information is presented for first and last opportunity by quarter.

```
select a.year,
a.quarter,
f.accountName firstAccountName,
f.owner firstOwner,
f.amount firstAmount,
l.accountName lastAccountName,
l.owner lastOwner,
l.amount lastAmount
from (
  select year,
  quarter,
  min(firstId) firstId,
  min(lastId) lastId
  from (
    select year(closeDate) year,
    quarter(closeDate) quarter,
    first_value(id) over (partition by year(closeDate), quarter(closeDate) order by closeDate r
ows between unbounded preceding and current row) firstId,
    last_value(id) over (partition by year(closeDate), quarter(closeDate) order by closeDate ro
ws between current row and unbounded following) lastId
    from opportunities  where stageName='ClosedWon'
  ) t
  group by year, quarter order by year,quarter
) a
join opportunities f on a.firstId = f.id
join opportunities l on a.lastId = l.id
order by year, quarter;
```

with example results:

year	quarter	firstAccountName	firstOwner	firstAmount	lastAccountName	lastOwner	lastAmount

2016	3	Skidoo	Bill	523295.07	Skipstorm	Bill	151420.86
2016	4	Skimia	Chris	961513.59	Avamm	Maria	112493.65
2017	1	Yombu	Bob	536875.51	Skaboo	Chris	270273.08

Prior and Next Example

Sometimes it useful to understand the previous and next values in the context of a given row. The lag and lead window functions provide this capability. By default the offset is one providing the prior or next value but can also be provided to get a larger offset. The example query is a report of opportunities by account name showing the opportunity amount, and the prior and next opportunity amount for that account by close date.

```
select accountName,
closeDate,
amount currentOppAmount,
lag(amount) over (partition by accountName order by closeDate) priorAmount, lead(amount) over (
partition by accountName order by closeDate) nextAmount
from opportunities
order by accountName, closeDate
limit 9;
```

with example results:

accountName	closeDate	currentOppAmount	priorAmount	nextAmount
Abata	2016-09-10	645098.45	NULL	161086.82
Abata	2016-10-14	161086.82	645098.45	350235.75
Abata	2016-12-18	350235.75	161086.82	878595.89
Abata	2016-12-31	878595.89	350235.75	922322.39
Abata	2017-01-21	922322.39	878595.89	NULL
Abatz	2016-10-19	795424.15	NULL	NULL
Agimba	2016-07-09	288974.84	NULL	914461.49
Agimba	2016-09-07	914461.49	288974.84	176645.52
Agimba	2016-09-20	176645.52	914461.49	NULL

Quartiles Example

The NTile window function allows for breaking up a data set into portions assigned a numeric value to each portion of the range. NTile(4) breaks the data up into quartiles (4 sets). The example query produces a report of all opportunities summarizing the quartile boundaries of amount values.

```
select t.quartile,
min(t.amount) min,
max(t.amount) max
from (
  select amount,
  ntile(4) over (order by amount asc) quartile
  from opportunities
  where closeDate >= '2016-10-01' and closeDate <= '2016-12-31'
  ) t
group by quartile
order by quartile;
```

With example results:

quartile	min	max
1	6337.15	287634.01
2	288796.14	539977.45
3	540070.04	748727.51
4	753670.77	998864.47

Percentile Example

The percentile functions have a slightly different syntax from other window functions as can be seen in the example below. These functions can be only applied against numeric values. The argument to the function is the percentile to evaluate. Following 'within group' is the sort expression which indicates the sort column and optionally order. Finally after 'over' is an optional partition by clause, for no partition clause use 'over ()'. The example below utilizes the value 0.5 to calculate the median opportunity amount in the rows. The values differ sometimes because percentile_cont will return the average of the 2 middle rows for an even data set while percentile_desc returns the first encountered in the sort.

```
select owner,
accountName,
CloseDate,
amount,
percentile_cont(0.5) within group (order by amount) over (partition by owner) pct_cont,
percentile_disc(0.5) within group (order by amount) over (partition by owner) pct_disc
from opportunities
where stageName='ClosedWon'
and closeDate >= '2016-10-02' and closeDate <= '2016-10-09'
order by owner, CloseDate;
```

With example results:

owner	accountName	CloseDate	amount	pct_cont	pct_disc
Bill	Babbleopia	2016-10-02	437636.47	437636.4700000000	437636.47
Bill	Thoughtworks	2016-10-04	146086.51	437636.4700000000	437636.47
Bill	Latz	2016-10-08	857254.87	437636.4700000000	437636.47
Chris	Linkbridge	2016-10-07	539977.45	619772.1550000000	539977.45
Chris	Avamm	2016-10-09	699566.86	619772.1550000000	539977.45
Olivier	Devpulse	2016-10-05	834235.93	667519.1100000000	500802.29
Olivier	Trupe	2016-10-07	500802.29	667519.1100000000	500802.29

1.2.9.7.35 Window Frames

Syntax

```
frame_clause:
  {ROWS | RANGE} {frame_border | BETWEEN frame_border AND frame_border}

frame_border:
  | UNBOUNDED PRECEDING
  | UNBOUNDED FOLLOWING
  | CURRENT ROW
  | expr PRECEDING
  | expr FOLLOWING
```

Description

A basic overview of window functions is described in Window Functions Overview. Window frames expand this functionality by allowing the function to include a specified a number of rows around the current row.

These include:

- All rows before the current row (UNBOUNDED PRECEDING), for example `RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`
- All rows after the current row (UNBOUNDED FOLLOWING), for example `RANGE BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING`
- A set number of rows before the current row (expr PRECEDING) for example `RANGE BETWEEN 6 PRECEDING AND CURRENT ROW`
- A set number of rows after the current row (expr PRECEDING AND expr FOLLOWING) for example `RANGE BETWEEN CURRENT ROW AND 2 FOLLOWING`
- A specified number of rows both before and after the current row, for example `RANGE BETWEEN 6 PRECEDING AND`

```
    3 FOLLOWING
```

The following functions operate on window frames:

- AVG
- BIT_AND
- BIT_OR
- BIT_XOR
- COUNT
- LEAD
- MAX
- MIN
- NTILE
- STD
- STDDEV
- STDDEV_POP
- STDDEV_SAMP
- SUM
- VAR_POP
- VAR_SAMP
- VARIANCE

Window frames are determined by the *frame_clause* in the window function request.

Take the following example:

```
CREATE TABLE `student_test` (
  name char(10),
  test char(10),
  score tinyint(4)
);

INSERT INTO student_test VALUES
    ('Chun', 'SQL', 75), ('Chun', 'Tuning', 73),
    ('Esben', 'SQL', 43), ('Esben', 'Tuning', 31),
    ('Kaolin', 'SQL', 56), ('Kaolin', 'Tuning', 88),
    ('Tatiana', 'SQL', 87);

SELECT name, test, score, SUM(score)
  OVER () AS total_score
  FROM student_test;
+---------+--------+-------+-------------+
| name    | test   | score | total_score |
+---------+--------+-------+-------------+
| Chun    | SQL    |    75 |         453 |
| Chun    | Tuning |    73 |         453 |
| Esben   | SQL    |    43 |         453 |
| Esben   | Tuning |    31 |         453 |
| Kaolin  | SQL    |    56 |         453 |
| Kaolin  | Tuning |    88 |         453 |
| Tatiana | SQL    |    87 |         453 |
+---------+--------+-------+-------------+
```

By not specifying an OVER clause, the SUM function is run over the entire dataset. However, if we specify an ORDER BY condition based on score (and order the entire result in the same way for clarity), the following result is returned:

```
SELECT name, test, score, SUM(score)
  OVER (ORDER BY score) AS total_score
  FROM student_test ORDER BY score;
+---------+--------+-------+-------------+
| name    | test   | score | total_score |
+---------+--------+-------+-------------+
| Esben   | Tuning |    31 |          31 |
| Esben   | SQL    |    43 |          74 |
| Kaolin  | SQL    |    56 |         130 |
| Chun    | Tuning |    73 |         203 |
| Chun    | SQL    |    75 |         278 |
| Tatiana | SQL    |    87 |         365 |
| Kaolin  | Tuning |    88 |         453 |
+---------+--------+-------+-------------+
```

The total_score column represents a running total of the current row, and all previous rows. The window frame in this

example expands as the function proceeds.

The above query makes use of the default to define the window frame. It could be written explicitly as follows:

```
SELECT name, test, score, SUM(score)
  OVER (ORDER BY score RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS total_score
  FROM student_test ORDER BY score;
+---------+--------+-------+-------------+
| name    | test   | score | total_score |
+---------+--------+-------+-------------+
| Esben   | Tuning |    31 |          31 |
| Esben   | SQL    |    43 |          74 |
| Kaolin  | SQL    |    56 |         130 |
| Chun    | Tuning |    73 |         203 |
| Chun    | SQL    |    75 |         278 |
| Tatiana | SQL    |    87 |         365 |
| Kaolin  | Tuning |    88 |         453 |
+---------+--------+-------+-------------+
```

Let's look at some alternatives:

Firstly, applying the window function to the current row and all following rows can be done with the use of UNBOUNDED FOLLOWING:

```
SELECT name, test, score, SUM(score)
  OVER (ORDER BY score RANGE BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING) AS total_score
  FROM student_test ORDER BY score;
+---------+--------+-------+-------------+
| name    | test   | score | total_score |
+---------+--------+-------+-------------+
| Esben   | Tuning |    31 |         453 |
| Esben   | SQL    |    43 |         422 |
| Kaolin  | SQL    |    56 |         379 |
| Chun    | Tuning |    73 |         323 |
| Chun    | SQL    |    75 |         250 |
| Tatiana | SQL    |    87 |         175 |
| Kaolin  | Tuning |    88 |          88 |
+---------+--------+-------+-------------+
```

It's possible to specify a number of rows, rather than the entire unbounded following or preceding set. The following example takes the current row, as well as the previous row:

```
SELECT name, test, score, SUM(score)
  OVER (ORDER BY score ROWS BETWEEN 1 PRECEDING AND CURRENT ROW) AS total_score
  FROM student_test ORDER BY score;
+---------+--------+-------+-------------+
| name    | test   | score | total_score |
+---------+--------+-------+-------------+
| Esben   | Tuning |    31 |          31 |
| Esben   | SQL    |    43 |          74 |
| Kaolin  | SQL    |    56 |          99 |
| Chun    | Tuning |    73 |         129 |
| Chun    | SQL    |    75 |         148 |
| Tatiana | SQL    |    87 |         162 |
| Kaolin  | Tuning |    88 |         175 |
+---------+--------+-------+-------------+
```

The current row and the following row:

```
SELECT name, test, score, SUM(score)
  OVER (ORDER BY score ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS total_score
  FROM student_test ORDER BY score;
+---------+--------+-------+-------------+
| name    | test   | score | total_score |
+---------+--------+-------+-------------+
| Esben   | Tuning |    31 |          74 |
| Esben   | SQL    |    43 |         130 |
| Kaolin  | SQL    |    56 |         172 |
| Chun    | Tuning |    73 |         204 |
| Chun    | SQL    |    75 |         235 |
| Tatiana | SQL    |    87 |         250 |
| Kaolin  | Tuning |    88 |         175 |
+---------+--------+-------+-------------+
```

1.3 Clients & Utilities

mariadb Client

The mariadb command-line client.

Aria Clients and Utilities

Clients and utilities for working with Aria tables

Backup, Restore and Import Clients

Clients for taking backups or importing/restoring data

Graphical and Enhanced Clients

Incomplete list of graphical clients

MyISAM Clients and Utilities

Clients and utilities for working with MyISAM tables

dbdeployer

Installing and testing multiple MariaDB versions in isolation.

dbForge Studio for MySQL/MariaDB

IDE for the development, management, and administration of MariaDB & MySQL databases.

EXPLAIN Analyzer

The EXPLAIN Analyzer is no longer active. The EXPLAIN Analyzer was an onlin...

EXPLAIN Analyzer API

The online EXPLAIN Analyzer tool has an open API to allow client applicatio...

innochecksum

Tool for printing checksums for InnoDB files.

msql2mysql

Description Initially, the MySQL C API was developed to be very similar to ...

my_print_defaults

Displays the options from option groups of option files

mariadb-binlog

mariadb-binlog utility for processing binary log files.

mariadb-stress-test

Perl script that performs stress-testing of the MariaDB server

mariadb-test

Testing utility

perror

Display descriptions for system or storage engine error codes

replace Utility

The replace utility program changes strings in place infiles or on the standard input

resolveip

Resolves IP addresses to host names and vice versa

resolve_stack_dump

Resolve numeric stack strace dump into symbols

xtstat

Used to monitor all internal activity of PBXT

mariadb-access

Tool for checking access privileges.

mariadb-admin

Admin tool for monitoring, creating/dropping databases, stopping MariaDB etc.

mariadb-check

Tool for checking, repairing, analyzing and optimizing tables.

mariadb-conv

Character set conversion utility for MariaDB.

mariadb-convert-table-format

Convert tables to use a particular storage engine by default.

mariadb-dumpslow

Display data from the slow query log.

mariadb-embedded

mariadb client statically linked to libmariadbd, the embedded server.

mariadb-find-rows

Read files containing SQL statements and extract statements that match a pattern.

mariadb-fix-extensions

Converts the extensions for MyISAM (or ISAM) table files to their canonical forms.

mariadb-install-db

Tool for creating the system tables in the mysql database.

mariadb-plugin

Tool for enabling or disabling plugins.

mariadb-report

Creates a friendly report of important MariaDB status values.

mariadb-secure-installation

Improve the security of a MariaDB installation.

mariadb-setpermission

Helps add users or databases or change passwords in MariaDB.

mariadb-show

Shows database structure.

mariadb-slap

Tool for load-testing MariaDB.

mariadb-tzinfo-to-sql

Load time zones into the time zone tables.

mariadb-upgrade

Update to the latest MariaDB Server version.

mariadb-waitpid

Terminate processes.

Legacy Clients and Utilities

Removed, deprecated or unmaintained MariaDB clients and utilities.

There are 8 related questions.

1.3.1 mariadb Client

The mariadb command-line client.

Previously, the client was called `mysql` which, from MariaDB 10.5, is still a symlink.

mariadb Command-Line Client

mariadb is a simple SQL shell with GNU readline capabilities.

Delimiters

How to change the delimiter for the mariadb client.

mysql Command-line Client

Symlink or old name for mariadb, the command-line client.

There are 4 related questions.

1.3.2 mariadb Command-Line Client

mariadb is a simple SQL shell (with GNU readline capabilities).

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadb` is a symlink to `mysql`, the command-line client.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mariadb` is the name of the command-line client, with `mysql` a symlink.

Contents

16. -D, --database=name
17. -# [options], --debug[=options]
18. --debug-check
19. -T, --debug-info
20. --default-auth=plugin
21. --default-character-set=name
22. --defaults-extra-file=file
23. --defaults-file=file
24. --defaults-group-suffix=suffix
25. --delimiter=name
26. --enable-cleartext-plugin
27. -e, --execute=name
28. -f, --force
29. -h, --host=name
30. -H, --html
31. -U, --i-am-a-dummy
32. -i, --ignore-spaces
33. --init-command=str
34. --line-numbers
35. --local-infile
36. --max-allowed-packet=num
37. --max-join-size=num
38. -G, --named-commands
39. --net-buffer-length=num
40. -b, --no-beep
41. --no-defaults
42. -o, --one-database
43. --pager[=name]
44. -p, --password[=name]
45. --plugin-dir=name
46. -P, --port=num
47. --print-defaults
48. --progress-reports
49. --prompt=name
50. --protocol=name
51. -q, --quick
52. -r, --raw
53. --reconnect
54. -U, --safe-updates
55. --secure-auth
56. --select-limit=num
57. --server-arg=name
58. --shared-memory-base-name=name
59. --show-warnings
60. --sigint-ignore
61. -s, --silent
62. --skip-auto-rehash
63. -N, --skip-column-names
64. --skip-comments
65. -L, --skip-line-numbers
66. --skip-progress-reports
67. --skip-reconnect
68. -S, --socket=name
69. --ssl
70. --ssl-ca=name
71. --ssl-capath=name
72. --ssl-cert=name
73. --ssl-cipher=name
74. --ssl-crl=name
75. --ssl-crlpath=name
76. --ssl-key=name
77. --ssl-verify-server-cert
78. -t, --table
79. --tee=name
80. --tls-version=name
81. --tls-fp=name
82. --tls-fplist=name

About the mariadb Command-Line Client

mariadb supports interactive and non-interactive use. When used interactively, query results are presented in an ASCII-table format. When used non-interactively (for example, as a filter), the result is presented in tab-separated format. The output format can be changed using command options.

If you have problems due to insufficient memory for large result sets, use the `--quick` option. This forces mariadb to retrieve results from the server a row at a time rather than retrieving the entire result set and buffering it in memory before displaying it. This is done by returning the result set using the `mysql_use_result()` C API function in the client/server library rather than `mysql_store_result()`.

Using mariadb is very easy. Invoke it from the prompt of your command interpreter as follows:

```
mariadb db_name
```

Or:

```
mariadb --user=user_name --password=your_password db_name
```

Then type an SQL statement, end it with “;”, \g, or \G and press Enter.

Typing Control-C causes mariadb to attempt to kill the current statement. If this cannot be done, or Control-C is typed again before the statement is killed, mariadb exits.

You can execute SQL statements in a script file (batch file) like this:

```
mariadb db_name < script.sql > output.tab
```

Using mariadb

The command to use `mariadb` and the general syntax is:

```
mariadb <options>
```

Options

`mariadb` supports the following options:

`-?, --help`

Display help and exit.

`-I, --help`

Synonym for `-?`

`--abort-source-on-error`

Abort 'source filename' operations in case of errors.

`--auto-rehash`

Enable automatic rehashing. This option is on by default, which enables database, table, and column name completion. Use `--disable-auto-rehash`, `--no-auto-rehash` or `skip-auto-rehash` to disable rehashing. That causes mariadb to start faster, but you must issue the rehash command if you want to use name completion. To complete a name, enter the first part and press Tab. If the name is unambiguous, mariadb completes it. Otherwise, you can press Tab again to see the possible names that begin with what you have typed so far. Completion does not occur if there is no default database.

`-A, --no-auto-rehash`

No automatic rehashing. One has to use 'rehash' to get table and field completion. This gives a quicker start of mariadb and disables rehashing on reconnect.

`--auto-vertical-output`

Automatically switch to vertical output mode if the result is wider than the terminal width.

`-B, --batch`

Print results using tab as the column separator, with each row on a new line. With this option, mariadb does not use the history file. Batch mode results in nontabular output format and escaping of special characters. Escaping may be disabled by using raw mode; see the description for the `--raw` option. (Enables `--silent`.)

`--binary-mode`

By default, ASCII '\0' is disallowed and '\r\n' is translated to '\n'. This switch turns off both features, and also turns off parsing of all client commands except \C and DELIMITER, in non-interactive mode (for input piped to mariadb or loaded using the 'source' command). This is necessary when processing output from mariadb-binlog that may contain blobs.

`--character-sets-dir=name`

Directory for character set files.

`--column-names`

Write column names in results. (Defaults to on; use `--skip-column-names` to disable.)

`--column-type-info`

Display column type information.

`-c, --comments`

Preserve comments. Send comments to the server. The default is `--skip-comments` (discard comments), enable with `--comments`.

`-C, --compress`

Compress all information sent between the client and the server if both support compression.

`--connect-expired-password`

Notify the server that this client is prepared to handle expired password sandbox mode even if `--batch` was specified. From MariaDB 10.4.3.

`--connect-timeout=num`

Number of seconds before connection timeout. Defaults to zero.

`-D, --database=name`

Database to use.

`-# [options], --debug[=options]`

On debugging builds, write a debugging log. A typical debug_options string is `d:t:o,file_name`. The default is `d:t:o,/tmp/mysql.trace`.

`--debug-check`

Check memory and open file usage at exit.

`-T, --debug-info`

Print some debug info at exit.

`--default-auth=plugin`

Default authentication client-side plugin to use.

`--default-character-set=name`

Set the default character set. A common issue that can occur when the operating system uses utf8 or another multibyte character set is that output from the mariadb client is formatted incorrectly, due to the fact that the MariaDB client uses the latin1 character set by default. You can usually fix such issues by using this option to force the client to use the system character set instead. If set to `auto` the character set is taken from the client environment (`LC_CTYPE` on Unix).

`--defaults-extra-file=file`

Read this file after the global files are read. Must be given as the first option.

`--defaults-file=file`

Only read default options from the given *file*. Must be given as the first option.

`--defaults-group-suffix=suffix`

In addition to the given groups, also read groups with this suffix.

`--delimiter=name`

Delimiter to be used. The default is the semicolon character (";").

`--enable-cleartext-plugin`

Obsolete option. Exists only for MySQL compatibility. From MariaDB 10.3.36.

`-e, --execute=name`

Execute statement and quit. Disables `--force` and history file. The default output format is like that produced with `--batch`.

`-f, --force`

Continue even if we get an SQL error. Sets `--abort-source-on-error` to 0.

`-h, --host=name`

Connect to host.

`-H, --html`

Produce HTML output.

`-U, --i-am-a-dummy`

Synonym for option `--safe-updates`, `-U`.

`-i, --ignore-spaces`

Ignore space after function names. Allows one to have spaces (including tab characters and new line characters) between function name and '('. The drawback is that this causes built in functions to become reserved words.

`--init-command=str`

SQL Command to execute when connecting to the MariaDB server. Will automatically be re-executed when reconnecting.

`--line-numbers`

Write line numbers for errors. (Defaults to on; use `--skip-line-numbers` to disable.)

`--local-infile`

Enable or disable LOCAL capability for LOAD DATA INFILE. With no value, the option enables LOCAL. The option may be given as `--local-infile=0` or `--local-infile=1` to explicitly disable or enable LOCAL. Enabling LOCAL has no effect if the server does not also support it.

`--max-allowed-packet=num`

The maximum packet length to send to or receive from server. The default is 16MB, the maximum 1GB.

`--max-join-size=num`

Automatic limit for rows in a join when using `--safe-updates`. Default is 1000000.

`-G, --named-commands`

Enable named commands. Named commands mean mariadb's internal commands (see below) . When enabled, the named commands can be used from any line of the query, otherwise only from the first line, before an enter. Long-format commands are allowed, not just short-format commands. For example, `quit` and `\q` are both recognized. Disable with `--disable-named-commands`. This option is disabled by default.

`--net-buffer-length=num`

The buffer size for TCP/IP and socket communication. Default is 16KB.

`-b, --no-beep`

Turn off beep on error.

`--no-defaults`

Don't read default options from any option file. Must be given as the first option.

`-o, --one-database`

Ignore statements except those those that occur while the default database is the one named on the command line. This filtering is limited, and based only on USE statements. This is useful for skipping updates to other databases in the binary log.

`--pager[=name]`

Pager to use to display results (Unix only). If you don't supply an option, the default pager is taken from your ENV variable PAGER. Valid pagers are *less*, *more*, *cat [> filename]*, etc. See interactive help (\h) also. This option does not work in batch mode. Disable with `--disable-pager`. This option is disabled by default.

`-p, --password[=name]`

Password to use when connecting to server. If you use the short option form (-p), you cannot have a space between the

option and the password. If you omit the password value following the `--password` or `-p` option on the command line, mariadb prompts for one. Specifying a password on the command line should be considered insecure. You can use an option file to avoid giving the password on the command line.

`--plugin-dir=name`

Directory for client-side plugins.

`-P, --port=num`

Port number to use for connection or 0 for default to, in order of preference, my.cnf, $MYSQL_TCP_PORT, /etc/services, built-in default (3306).

`--print-defaults`

Print the program argument list and exit. Must be given as the first option.

`--progress-reports`

Get progress reports for long running commands (such as ALTER TABLE). (Defaults to on; use `--skip-progress-reports` to disable.)

`--prompt=name`

Set the mariadb prompt to this value. See prompt command for options.

`--protocol=name`

The protocol to use for connection (tcp, socket, pipe, memory).

`-q, --quick`

Don't cache result, print it row by row. This may slow down the server if the output is suspended. Doesn't use history file.

`-r, --raw`

For tabular output, the “boxing” around columns enables one column value to be distinguished from another. For nontabular output (such as is produced in batch mode or when the `--batch` or `--silent` option is given), special characters are escaped in the output so they can be identified easily. Newline, tab, NUL, and backslash are written as `\n`, `\t`, `\0`, and . The `--raw` option disables this character escaping.

`--reconnect`

Reconnect if the connection is lost. This option is enabled by default. Disable with `--disable-reconnect` or `skip-reconnect`.

`-U, --safe-updates`

Allow only those UPDATE and DELETE statements that specify which rows to modify by using key values. If you have set this option in an option file, you can override it by using `--safe-updates` on the command line. See using the --safe-updates option for more.

`--secure-auth`

Refuse client connecting to server if it uses old (pre-MySQL4.1.1) protocol. Defaults to false.

`--select-limit=num`

Automatic limit for SELECT when using --safe-updates. Default 1000.

`--server-arg=name`

Send embedded server this as a parameter.

`--shared-memory-base-name=name`

Shared-memory name to use for Windows connections using shared memory to a local server (started with the --shared-memory option). Case-sensitive.

`--show-warnings`

Show warnings after every statement. Applies to interactive and batch mode.

`--sigint-ignore`

Ignore SIGINT signals (usually CTRL-C).

`-s, --silent`

Be more silent. This option can be given multiple times to produce less and less output. This option results in nontabular output format and escaping of special characters. Escaping may be disabled by using raw mode; see the description for the `--raw` option.

`--skip-auto-rehash`

Disable automatic rehashing. See `--auto-rehash`.

`-N, --skip-column-names`

Don't write column names in results. See `--column-names`.

`--skip-comments`

Discard comments. Set by default, see `--comments` to enable.

`-L, --skip-line-numbers`

Don't write line number for errors. See `--line-numbers`.

`--skip-progress-reports`

Disables getting progress reports for long running commands. See `--progress-reports`.

`--skip-reconnect`

Don't reconnect if the connection is lost. See `--reconnect`.

`-S, --socket=name`

For connections to localhost, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

`--ssl`

Enables TLS. TLS is also enabled even without setting this option when certain other TLS options are set. The `--ssl` option does not enable verifying the server certificate by default. In order to verify the server certificate, the user must specify the `--ssl-verify-server-cert` option. Set by default from MariaDB 10.10.

`--ssl-ca=name`

Defines a path to a PEM file that should contain one or more X509 certificates for trusted Certificate Authorities (CAs) to use for TLS. This option requires that you use the absolute path, not a relative path. See Secure Connections Overview: Certificate Authorities (CAs) for more information. This option implies the `--ssl` option.

`--ssl-capath=name`

Defines a path to a directory that contains one or more PEM files that should each contain one X509 certificate for a trusted Certificate Authority (CA) to use for TLS. This option requires that you use the absolute path, not a relative path. The directory specified by this option needs to be run through the `openssl rehash` command. See Secure Connections Overview: Certificate Authorities (CAs) for more information. This option is only supported if the client was built with

OpenSSL or yaSSL. If the client was built with GnuTLS or Schannel, then this option is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms. This option implies the `--ssl` option.

`--ssl-cert=name`

Defines a path to the X509 certificate file to use for TLS. This option requires that you use the absolute path, not a relative path. This option implies the `--ssl` option.

`--ssl-cipher=name`

List of permitted ciphers or cipher suites to use for TLS. This option implies the `--ssl` option.

`--ssl-crl=name`

Defines a path to a PEM file that should contain one or more revoked X509 certificates to use for TLS. This option requires that you use the absolute path, not a relative path. See Secure Connections Overview: Certificate Revocation Lists (CRLs) for more information. This option is only supported if the client was built with OpenSSL or Schannel. If the client was built with yaSSL or GnuTLS, then this option is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.

`--ssl-crlpath=name`

Defines a path to a directory that contains one or more PEM files that should each contain one revoked X509 certificate to use for TLS. This option requires that you use the absolute path, not a relative path. The directory specified by this option needs to be run through the `openssl rehash` command. See Secure Connections Overview: Certificate Revocation Lists (CRLs) for more information. This option is only supported if the client was built with OpenSSL. If the client was built with yaSSL, GnuTLS, or Schannel, then this option is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.

`--ssl-key=name`

Defines a path to a private key file to use for TLS. This option requires that you use the absolute path, not a relative path. This option implies the `--ssl` option.

`--ssl-verify-server-cert`

Enables server certificate verification. Prior to MariaDB 11.3, this option is disabled by default, otherwise enabled. Use `--disable-ssl` or `--disable-ssl-verify-server-cert` to revert to the pre-11.3 behavior.

`-t, --table`

Display output in table format. This is the default for interactive use, but can be used to produce table output in batch mode.

`--tee=name`

Append everything into outfile. See interactive help (\h) also. Does not work in batch mode. Disable with `--disable-tee`. This option is disabled by default.

`--tls-version=name`

This option accepts a comma-separated list of TLS protocol versions. A TLS protocol version will only be enabled if it is present in this list. All other TLS protocol versions will not be permitted. See Secure Connections Overview: TLS Protocol Versions for more information. This option was added in MariaDB 10.4.6.

`--tls-fp=name`

Server certificate fingerprint (implies --ssl). Added in MariaDB 11.3.0.

`--tls-fplist=name`

File with accepted server certificate fingerprints, one per line (implies --ssl). Added in MariaDB 11.3.0.

`-n, --unbuffered`

Flush buffer after each query.

`-u, --user=name`

User for login if not current user.

`-v, --verbose`

Write more. (-v -v -v gives the table output format).

`-V, --version`

Output version information and exit.

`-E, --vertical`

Print the output of a query (rows) vertically. Use the `\G` delimiter to apply to a particular statement if this option is not enabled.

`-w, --wait`

If the connection cannot be established, wait and retry instead of aborting.

`-X, --xml`

Produce XML output. See the mariadb-dump --xml option for more.

Option Files

In addition to reading options from the command-line, `mariadb` can also read options from option files. If an unknown option is provided to `mariadb` in an option file, then it is ignored.

The following options relate to how MariaDB command-line tools handles option files. They must be given as the first argument on the command-line:

Option	Description
`--print-defaults`	Print the program argument list and exit.
`--no-defaults`	Don't read default options from any option file.
`--defaults-file=#`	Only read default options from the given file #.
`--defaults-extra-file=#`	Read this file after the global files are read.
`--defaults-group-suffix=#`	In addition to the default option groups, also read option groups with this suffix.

`mariadb` is linked with MariaDB Connector/C. However, MariaDB Connector/C does not yet handle the parsing of option files for this client. That is still performed by the server option file parsing code. See MDEV-19035 for more information.

Option Groups

`mariadb` reads options from the following option groups from option files:

Group	Description
`[mysql]`	Options read by `mysql`, which includes both MariaDB Server and MySQL Server.
`[mariadb-client]`	Options read by `mariadb`. Available starting with MariaDB 10.4.6.
`[client]`	Options read by all MariaDB and MySQL client programs, which includes both MariaDB and MySQL clients. For example, `mysqldump`.
`[client-server]`	Options read by all MariaDB client programs and the MariaDB Server. This is useful for options like socket and port, which is common between the server and the clients.
`[client-mariadb]`	Options read by all MariaDB client programs.

How to Specify Which Protocol to Use When Connecting to the Server

You can force which protocol to be used to connect to the `mariadbd` server by giving the `protocol` option one of the following values: `tcp`, `socket`, `pipe` or `memory`.

If `protocol` is not specified, before MariaDB 10.6.1, command line connection properties that do not force protocol are ignored.

From MariaDB 10.6.1, a connection property specified via the command line (e.g. `--port=3306`) will force its type. The protocol that matches the respective connection property is used, e.g. a TCP/IP connection is created when `--port` is specified.

If multiple or no connection properties are specified via the command-line, then the following happens:

Linux/Unix

- If `hostname` is not specified or `hostname` is `localhost`, then Unix sockets are used.
- In other cases (`hostname` is given and it's not `localhost`) then a TCP/IP connection through the `port` option is used.

Note that `localhost` is a special value. Using 127.0.0.1 is not the same thing. The latter will connect to the mariadbd server through TCP/IP.

Windows

- If `shared-memory-base-name` is specified and `hostname` is not specified or `hostname` is `localhost`, then the connection will happen through shared memory.
- If `shared-memory-base-name` is not specified and `hostname` is not specified or `hostname` is `localhost`, then the connection will happen through windows named pipes.
- Named pipes will also be used if the `libmysql` / `libmariadb` client library detects that the client doesn't support TCP/IP.
- In other cases then a TCP/IP connection through the `port` option is used.

How to Test Which Protocol is Used

The `status` command shows you information about which protocol is used:

```
shell> mariadb test

Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 10
Server version: 10.2.2-MariaDB-valgrind-max-debug Source distribution

Copyright (c) 2000, 2016, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [test]> status;
--------------
mysql  Ver 15.1 Distrib 10.0.25-MariaDB, for Linux (x86_64) using readline 5.2

Connection id:          10
Current database:       test
Current user:           monty@localhost
...
Connection:             Localhost via UNIX socket
...
UNIX socket:            /tmp/mysql-dbug.sock
```

mariadb Commands

There are also a number of commands that can be run inside the client. Note that all text commands must be first on line and end with ';'

Command	Description

`?`, `\?`	Synonym for `help'.
`clear`, `\c`	Clear the current input statement.
`connect`, `\r`	Reconnect to the server. Optional arguments are db and host.
`delimiter`, `\d`	Set statement delimiter.
`edit`, `\e`	Edit command with $EDITOR.
`ego`, `\G`	Send command to mariadb server, display result vertically.
`exit`, `\q`	Exit mariadb. Same as quit.
`go`, `\g`	Send command to mariadb server.
`help`, `\h`	Display this help.
`nopager`, `\n`	Disable pager, print to stdout.
`notee`, `\t`	Don't write into outfile.
`pager`, `\P`	Set PAGER [to_pager]. Print the query results via PAGER.
`print`, `\p`	Print current command.
`prompt`, `\R`	Change your mariadb prompt. See prompt command for options.
`quit`, `\q`	Quit mariadb.
`rehash`, `\#`	Rebuild completion hash.
`source`, `\.`	Execute an SQL script file. Takes a file name as an argument.
`status`, `\s`	Get status information from the server.
`system`, `\!`	Execute a system shell command. Only works in Unix-like systems.
`tee`, `\T`	Set outfile [to_outfile]. Append everything into given outfile.
`use`, `\u`	Use another database. Takes database name as argument.
`charset`, `\C`	Switch to another charset. Might be needed for processing binlog with multi-byte charsets.
`warnings`, `\W`	Show warnings after every statement.
`nowarning`, `\w`	Don't show warnings after every statement.

The mysql_history File

On Unix, the mariadb client writes a record of executed statements to a history file. By default, this file is named `.mysql_history` and is created in your home directory. To specify a different file, set the value of the MYSQL_HISTFILE environment variable.

The .mysql_history file should be protected with a restrictive access mode because sensitive information might be written to it, such as the text of SQL statements that contain passwords.

If you do not want to maintain a history file, first remove .mysql_history if it exists, and then use either of the following techniques:

- Set the MYSQL_HISTFILE variable to /dev/null. To cause this setting to take effect each time you log in, put the setting in one of your shell's startup files.
- Create .mysql_history as a symbolic link to /dev/null:

```
shell> ln -s /dev/null $HOME/.mysql_history
```

You need do this only once.

prompt Command

The prompt command reconfigures the default prompt `\N [\d]>`. The string for defining the prompt can contain the following special sequences.

Option	Description

`\c`	A counter that increments for each statement you issue.
`\D`	The full current date.
`\d`	The default database.
`\h`	The server host.
`\l`	The current delimiter.
`\m`	Minutes of the current time.
`\n`	A newline character.
`\O`	The current month in three-letter format (Jan, Feb, ...).
`\o`	The current month in numeric format.
`\P`	am/pm.
`\p`	The current TCP/IP port or socket file.
`\R`	The current time, in 24-hour military time (0–23).
`\r`	The current time, standard 12-hour time (1–12).
`\S`	Semicolon.
`\s`	Seconds of the current time.
`\t`	A tab character.
`\U`	Your full user_name@host_name account name.
`\u`	Your user name.
`\v`	The server version.
`\w`	The current day of the week in three-letter format (Mon, Tue, ...).
`\Y`	The current year, four digits.
`\y`	The current year, two digits.
`\_`	A space.
`\ `	A space (a space follows the backslash).
`\'`	Single quote.
`\"`	Double quote.
`\ \`	A literal “\” backslash character.
`\x`	x, for any “x” not listed above.

mariadb Tips

This section describes some techniques that can help you use **`mariadb`** more effectively.

Displaying Query Results Vertically

Some query results are much more readable when displayed vertically, instead of in the usual horizontal table format. Queries can be displayed vertically by terminating the query with \G instead of a semicolon. For example, longer text values that include newlines often are much easier to read with vertical output:

```
mariadb> SELECT * FROM mails WHERE LENGTH(txt) < 300 LIMIT 300,1\G
*************************** 1. row ***************************
  msg_nro: 3068
     date: 2000-03-01 23:29:50
time_zone: +0200
mail_from: Monty
    reply: monty@no.spam.com
  mail_to: "Thimble Smith" <tim@no.spam.com>
      sbj: UTF-8
      txt: >>>>> "Thimble" == Thimble Smith writes:
Thimble> Hi.  I think this is a good idea.  Is anyone familiar
Thimble> with UTF-8 or Unicode? Otherwise, I´ll put this on my
Thimble> TODO list and see what happens.
Yes, please do that.
Regards,
Monty
    file: inbox-jani-1
    hash: 190402944
1 row in set (0.09 sec)
```

Using the --safe-updates Option

For beginners, a useful startup option is `--safe-updates` (or `--i-am-a-dummy`, which has the same effect). It is helpful for cases when you might have issued a `DELETE FROM tbl_name` statement but forgotten the `WHERE` clause. Normally, such a statement deletes all rows from the table. With `--safe-updates`, you can delete rows only by specifying the key values that identify them. This helps prevent accidents.

When you use the `--safe-updates` option, mariadb issues the following statement when it connects to the MariaDB server:

```
SET sql_safe_updates=1, sql_select_limit=1000, sql_max_join_size=1000000;
```

The SET statement has the following effects:

- You are not allowed to execute an UPDATE or DELETE statement unless you specify a key constraint in the WHERE clause or provide a LIMIT clause (or both). For example:

```
UPDATE tbl_name SET not_key_column=val WHERE key_column=val;
UPDATE tbl_name SET not_key_column=val LIMIT 1;
```

- The server limits all large `SELECT` results to 1,000 rows unless the statement includes a `LIMIT` clause.
- The server aborts multiple-table `SELECT` statements that probably need to examine more than 1,000,000 row combinations.

To specify limits different from 1,000 and 1,000,000, you can override the defaults by using the `--select_limit` and `--max_join_size` options:

```
mariadb --safe-updates --select_limit=500 --max_join_size=10000
```

Disabling mariadb Auto-Reconnect

If the mariadb client loses its connection to the server while sending a statement, it immediately and automatically tries to reconnect once to the server and send the statement again. However, even if mariadb succeeds in reconnecting, your first connection has ended and all your previous session objects and settings are lost: temporary tables, the autocommit mode, and user-defined and session variables. Also, any current transaction rolls back. This behavior may be dangerous for you, as in the following example where the server was shut down and restarted between the first and second statements without you knowing it:

```
mariadb> SET @a=1;
Query OK, 0 rows affected (0.05 sec)
mariadb> INSERT INTO t VALUES(@a);
ERROR 2006: MySQL server has gone away
No connection. Trying to reconnect...
Connection id:    1
Current database: test
Query OK, 1 row affected (1.30 sec)
mariadb> SELECT * FROM t;
+------+
| a    |
+------+
| NULL |
+------+
```

The @a user variable has been lost with the connection, and after the reconnection it is undefined. If it is important to have mariadb terminate with an error if the connection has been lost, you can start the mariadb client with the `--skip-reconnect` option.

1.3.3 Delimiters

The default delimiter in the mariadb client is the semicolon.

When creating stored programs from the command-line, it is likely you will need to differentiate between the regular delimiter and a delimiter inside a BEGIN END block. To understand better, consider the following example:

```
CREATE FUNCTION FortyTwo() RETURNS TINYINT DETERMINISTIC
BEGIN
 DECLARE x TINYINT;
 SET x = 42;
 RETURN x;
END;
```

If you enter the above line by line, the mariadb client will treat the first semicolon, at the end of the `DECLARE x TINYINT` line, as the end of the statement. Since that's only a partial definition, it will throw a syntax error, as follows:

```
CREATE FUNCTION FortyTwo() RETURNS TINYINT DETERMINISTIC
BEGIN
DECLARE x TINYINT;
ERROR 1064 (42000): You have an error in your SQL syntax;
check the manual that corresponds to your MariaDB server version
 for the right syntax to use near '' at line 3
```

The solution is to specify a distinct delimiter for the duration of the process, using the DELIMITER command. The delimiter can be any set of characters you choose, but it needs to be a distinctive set of characters that won't cause further confusion. `//` is a common choice, and used throughout the Knowledge Base.

Here's how the function could be successfully entered from the mariadb client with the new delimiter.

```
DELIMITER //

CREATE FUNCTION FortyTwo() RETURNS TINYINT DETERMINISTIC
BEGIN
  DECLARE x TINYINT;
  SET x = 42;
  RETURN x;
END

//

DELIMITER ;
```

At the end, the delimiter is restored to the default semicolon. The `\g` and `\G` delimiters can always be used, even when a custom delimiter is specified.

1.3.4 mysql Command-line Client

mysql is a simple SQL shell (with GNU readline capabilities).

MariaDB starting with 10.4.6

From MariaDB 10.4.6, mariadb is a symlink to `mysql`.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, mariadb is the name of the script, with `mysql` a symlink.

See mariadb for details.

1.3.5 Aria Clients and Utilities

Clients and utilities for working with the Aria storage engine

aria_chk

Used for checking, repairing, optimizing and sorting Aria tables.

aria_pack

Tool for compressing Aria tables.

aria_read_log

Tool for displaying and applying log records from an Aria transaction log.

aria_s3_copy

Copies an Aria table to and from S3.

There are 4 related questions.

1.3.5.1 aria_chk

Contents

`aria_chk` is used to check, repair, optimize, sort and get information about Aria tables.

With the MariaDB server you can use CHECK TABLE, REPAIR TABLE and OPTIMIZE TABLE to do similar things.

Note: `aria_chk` should not be used when MariaDB is running. MariaDB assumes that no one is changing the tables it's using!

Usage:

```
aria_chk [OPTIONS] aria_tables[.MAI]
```

Aria table information is stored in 2 files: the `.MAI` file contains base table information and the index and the `.MAD` file contains the data. `aria_chk` takes one or more `.MAI` files as arguments.

The following groups are read from the my.cnf files:

- `[maria_chk]`
- `[aria_chk]`

Options and Variables

Global Options

The following options to handle option files may be given as the first argument:

Option	Description
`--print-defaults`	Print the program argument list and exit.
`--no-defaults`	Don't read default options from any option file.
`--defaults-file=#`	Only read default options from the given file #.
`--defaults-extra-file=#`	Read this file after the global files are read.

Main Arguments

Option	Description
`-#`, `--debug=...`	Output debug log. Often this is 'd:t:o,filename'.
`-H`, `--HELP`	Display this help and exit.
`-?`, `--help`	Display this help and exit.
`--datadir=path`	Path for control file (and logs if `--logdir` not used).
`--ignore-control-file`	Don't open the control file. Only use this if you are sure the tables are not used by another program
`--logdir=path`	Path for log files.
`--require-control-file`	Abort if we can't find/read the maria_log_control file
`-s`, `--silent`	Only print errors. One can use two -s to make aria_chk very silent.
`-t`, `--tmpdir=path`	Path for temporary files. Multiple paths can be specified, separated by colon (:) on Unix or semicolon (;) on Windows. They will be used in a round-robin fashion.
`-v`, `--verbose`	Print more information. This can be used with `--description` and `--check`. Use many -v for more verbosity.
`-V`, `--version`	Print version and exit.
`-w`, `--wait`	Wait if table is locked.

Check Options (--check is the Default Action for aria_chk):

Option	Description
`-c`, `--check`	Check table for errors.
`-e`, `--extend-check`	Check the table VERY throughly. Only use this in extreme cases as aria_chk should normally be able to find out if the table is ok even without this switch.
`-F`, `--fast`	Check only tables that haven't been closed properly.
`-C`, `--check-only-changed`	Check only tables that have changed since last check.
`-f`, `--force`	Restart with '`-r`' if there are any errors in the table. States will be updated as with '`--update-state`'.
`-i`, `--information`	Print statistics information about table that is checked.

`-m`, `--medium-check`	Faster than extend-check, and finds 99.99% of all errors. Should be good enough for most cases.
`-U`, `--update-state`	Mark tables as crashed if any errors were found and clean if check didn't find any errors but table was marked as 'not clean' before. This allows one to get rid of warnings like 'table not properly closed'. If table was updated, update also the timestamp for when the check was made. This option is on by default! Use `--skip-update-state` to disable.
`-T`, `--read-only`	Don't mark table as checked.

Recover (Repair) Options (When Using '--recover' or '--safe-recover'):

Option	Description
`-B`, `--backup`	Make a backup of the .MAD file as 'filename-time.BAK'.
`--correct-checksum`	Correct checksum information for table.
`-D`, `--data-file-length=#`	Max length of data file (when recreating data file when it's full).
`-e`, `--extend-check`	Try to recover every possible row from the data file Normally this will also find a lot of garbage rows; Don't use this option if you are not totally desperate.
`-f`, `--force`	Overwrite old temporary files.
`-k`, `--keys-used=#`	Tell MARIA to update only some specific keys. # is a bit mask of which keys to use. This can be used to get faster inserts.
`--max-record-length=#`	Skip rows bigger than this if aria_chk can't allocate memory to hold it.
`-r`, `--recover`	Can fix almost anything except unique keys that aren't unique.
`-n`, `--sort-recover`	Forces recovering with sorting even if the temporary file would be very big.
`-p`, `--parallel-recover`	Uses the same technique as '-r' and '-n', but creates all the keys in parallel, in different threads.
`-o`, `--safe-recover`	Uses old recovery method; Slower than '-r' but can handle a couple of cases where '-r' reports that it can't fix the data file.
`--transaction-log`	Log repair command to transaction log. This is needed if one wants to use the maria_read_log to repeat the repair.
`--character-sets-dir=...`	Directory where character sets are.
`--set-collation=name`	Change the collation used by the index.
`-q`, `--quick`	Faster repair by not modifying the data file. One can give a second '`-q`' to force aria_chk to modify the original datafile in case of duplicate keys. NOTE: Tables where the data file is currupted can't be fixed with this option.
`-u`, `--unpack`	Unpack file packed with aria_pack.

Other Options

Option	Description
`-a`, `--analyze`	Analyze distribution of keys. Will make some joins in MariaDB faster. You can check the calculated distribution by using '`--description --verbose table_name`'.

`--stats_method=name`	Specifies how index statistics collection code should treat NULLs. Possible values of name are "nulls_unequal" (default for 4.1/5.0), "nulls_equal" (emulate 4.0), and "nulls_ignored".
`-d, --description`	Prints some information about table.
`-A, --set-auto-increment[=value]`	Force auto_increment to start at this or higher value If no value is given, then sets the next auto_increment value to the highest used value for the auto key + 1.
`-S, --sort-index`	Sort index blocks. This speeds up 'read-next' in applications.
`-R, --sort-records=#`	Sort records according to an index. This makes your data much more localized and may speed up things (It may be VERY slow to do a sort the first time!).
`-b, --block-search=#`	Find a record, a block at given offset belongs to.
`-z, --zerofill`	Remove transaction id's from the data and index files and fills empty space in the data and index files with zeroes. Zerofilling makes it possible to move the table from one system to another without the server having to do an automatic zerofill. It also allows one to compress the tables better if one want to archive them.
`--zerofill-keep-lsn`	Like `--zerofill` but does not zero out LSN of data/index pages.

Variables

Option	Description
`page_buffer_size`	Size of page buffer. Used by `--safe-repair`
`read_buffer_size`	Read buffer size for sequential reads during scanning
`write_buffer_size`	Write buffer size for sequential writes during repair of fixed size or dynamic size rows
`sort_buffer_size`	Size of sort buffer. Used by `--recover`
`sort_key_blocks`	Internal buffer for sorting keys; Don't touch :)

Usage

One main usage of `aria_chk` is when you want to do a fast check of all Aria tables in your system. This is faster than doing it in MariaDB as you can allocate all free memory to the buffers.

Assuming you have a bit more than 2G free memory.

The following commands, run in the MariaDB data directory, check all your tables and repairs only those that have an error:

```
aria_chk --check --sort_order --force --sort_buffer_size=1G */*.MAI
```

If you want to optimize all your tables: (The `--zerofill` is used here to fill up empty space with `\0` which can speed up compressed backups).

```
aria_chk --analyze --sort-index --page_buffer_size=1G --zerofill */*.MAI
```

In case you have a serious problem and have to use `--safe-recover`:

```
aria_chk --safe-recover --zerofill --page_buffer_size=2G */*.MAI
```

1.3.5.2 aria_pack

Contents

aria_pack is a tool for compressing Aria tables. The resulting table are read-only, and usually about 40% to 70% smaller.

aria_pack is run as follows

```
aria_pack [options] file_name [file_name2...]
```

The file name is the .MAI index file. The extension can be omitted, although keeping it permits wildcards, such as

```
aria_pack *.MAI
```

to compress all the files.

aria_pack compresses each column separately, and, when the resulting data is read, only the individual rows and columns required need to be decompressed, allowing for quicker reading.

Once a table has been packed, use aria_chk -rq (the quick and recover options) to rebuild its indexes.

Options

The following variables can be set while passed as commandline options to aria_pack, or set in the [ariapack] section in your my.cnf file.

Option	Description
-b, --backup	Make a backup of the table as table_name.OLD.
--character-sets-dir=name	Directory where character sets are.
-h, --datadir	Path for control file (and logs if `--logdir` not used). From MariaDB 10.5.3
-#, --debug[=name]	Output debug log. Often this is 'd:t:o,filename'.
-?, --help	Display help and exit.
-f, --force	Force packing of table even if it gets bigger or if tempfile exists.
--ignore-control-file	Ignore the control file. From MariaDB 10.5.3.
-j, --join=name	Join all given tables into 'new_table_name'. All tables MUST have identical layouts.
--require-control-file	Abort if cannot find control file. From MariaDB 10.5.3.
-s, --silent	Only write output when an error occurs.
-t, --test	Don't pack table, only test packing it.
-T, --tmpdir=name	Use temporary directory to store temporary table.
-v, --verbose	Write info about progress and packing result. Use many -v for more verbosity!
-V, --version	Output version information and exit.
-w, --wait	Wait and retry if table is in use.

Unpacking

To unpack a table compressed with aria_pack, use the aria_chk -u option.

Example

```
> aria_pack /my/data/test/posts
Compressing /my/data/test/posts.MAD: (1690 records)
- Calculating statistics
- Compressing file
37.71%
> aria_chk -rq --ignore-control-file /my/data/test/posts
- check record delete-chain
- recovering (with keycache) Aria-table '/my/data/test/posts'
Data records: 1690
State updated
```

1.3.5.3 aria_read_log

aria_read_log is a tool for displaying and applying log records from an Aria transaction log.

Note: Aria is compiled without -DIDENTICAL_PAGES_AFTER_RECOVERY which means that the table files are not byte-to-byte identical to files created during normal execution. This should be ok, except for test scripts that try to compare files before and after recovery.

Usage:

```
aria_read_log OPTIONS
```

You need to use one of `-d` or `-a` .

Options

The following variables can be set while passed as commandline options to aria_read_log, or set in the `[aria_read_log]` section in your my.cnf file.

Option	Description
-a, --apply	Apply log to tables: modifies tables! you should make a backup first! Displays a lot of information if not run with --silent.
--character-sets-dir=name	Directory where character sets are.
-c, --check	if --display-only, check if record is fully readable (for debugging).
-?, --help	Display help and exit.
-d, --display-only	Display brief info read from records' header.
-e, --end-lsn=#	Stop applying at this lsn. If end-lsn is used, UNDO:s will not be applied
-h, --aria-log-dir-path=name	Path to the directory where to store transactional log
-P, --page-buffer-size=#	The size of the buffer used for index blocks for Aria tables.
-l, --print-log-control-file	Print the content of the aria_log_control_file. From MariaDB 10.4.1.
-o, --start-from-lsn=#	Start reading log from this lsn.
-C, --start-from-checkpoint	Start applying from last checkpoint.
-s, --silent	Print less information during apply/undo phase.
-T, --tables-to-redo=name	List of comma-separated tables that we should apply REDO on. Use this if you only want to recover some tables.
-t, --tmpdir=name	Path for temporary files. Multiple paths can be specified, separated by colon (:)
--translog-buffer-size=#	The size of the buffer used for transaction log for Aria tables.
-u, --undo	Apply UNDO records to tables. (disable with --disable-undo) (Defaults to on; use --skip-undo to disable.)
-v, --verbose	Print more information during apply/undo phase.
-V, --version	Print version and exit.

5.3.16.4 aria_s3_copy

1.3.6 Backup, Restore and Import Clients

Clients for taking backups or importing/restoring data

Mariabackup

Physical backups, supports Data-at-Rest and InnoDB compression.

mariadb-dump

Dump a database or a collection of databases in a portable format.

mariadb-hotcopy

Fast backup program on local machine. Deprecated.

mariadb-import

Loads tables from text files in various formats.

Backup/Restore + Data Export/Import via dbForge Studio

The fastest and easiest way to perform these operations with MariaDB databases.

There are 5 related questions.

2.3.4 Mariabackup

1.3.6.2 mariadb-dump

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadb-dump` is a symlink to `mysqldump`, the backup tool.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mariadb-dump` is the name of the tool, with `mysqldump` a symlink.

MariaDB starting with 11.0.1

From MariaDB 11.0.1, `mysqldump` (the symlink) is deprecated and removed from the `mariadb` Docker Official Image.

The `mariadb-dump` client (originally called `mysqldump`)is a backup program originally written by Igor Romanenko. It can be used to dump a database or a collection of databases for backup or transfer to another database server (not necessarily MariaDB or MySQL). The dump typically contains SQL statements to create the table, populate it, or both. However, `mariadb-dump` can also be used to generate files in CSV, other delimited text, or XML format.

If you are doing a backup on the server and your tables all are MyISAM tables, consider using mariadb-hotcopy instead because it can accomplish faster backups and faster restores.

mariadb-dump dumps triggers along with tables, as these are part of the table definition. However, stored procedures, views, and events are not, and need extra parameters to be recreated explicitly (for example, `--routines` and `--events`). Procedures and functions are however also part of the system tables (for example mysql.proc).

`mariadb-dump` supports the enhancements for START TRANSACTION WITH CONSISTENT SNAPSHOT.

Performance

mariadb-dump doesn't usually consume much CPU resources on modern hardware as by default it uses a single thread. This method is good for a heavily loaded server.

Disk input/outputs per second (IOPS), can however increase for multiple reasons. When you back-up on the same device as the database, this produces unnecessary random IOPS. The dump is done sequentially, on a per table basis, causing a full-table scan and many buffer page misses on tables that are not fully cached in memory.

It's recommended that you back-up from a network location to remove disk IOPS on the database server, but it is vital to use a separate network card to keep network bandwidth available for regular traffic.

Although mariadb-dump will by default preserve your resources for regular spindle disks and low-core hardware, this doesn't mean that concurrent dumps cannot benefit from hardware architecture like SAN, flash storage, low write workload. The back-up time would benefit from a tool such as MyDumper.

Usage

There are four general ways to invoke `mariadb-dump`:

```
shell> mariadb-dump [options] db_name [tbl_name ...]
shell> mariadb-dump [options] --databases db_name ...
shell> mariadb-dump [options] --all-databases
shell> mariadb-dump [options] --system=[option_list]
```

If you do not name any tables following db_name or if you use the `--databases` or `--all-databases` option, entire databases are dumped.

`mariadb-dump` does not dump the INFORMATION_SCHEMA (or PERFORMANCE_SCHEMA, if enabled) database by default. MariaDB dumps the `INFORMATION_SCHEMA` if you name it explicitly on the command line, although currently you must also use the `--skip-lock-tables` option.

To see a list of the options your version of `mariadb-dump` supports, execute `mariadb-dump --help`.

Row by Row vs. Buffering

`mariadb-dump` can retrieve and dump table contents row by row, or it can retrieve the entire content from a table and buffer it in memory before dumping it. Buffering in memory can be a problem if you are dumping large tables. To dump tables row by row, use the `--quick` option (or `--opt`, which enables `--quick`). The `--opt` option (and hence `--quick`) is enabled by default, so to enable memory buffering, use `--skip-quick`.

mariadb-dump and the mysql.transaction_registry_table

`mariadb-dump` includes logic to cater for the mysql.transaction_registry table. `mysqldump` from an earlier MariaDB release cannot be used on MariaDB 10.3 and beyond.

mariadb-dump and Old Versions of MySQL

If you are using a recent version of `mariadb-dump` to generate a dump to be reloaded into a very old MySQL server, you should not use the `--opt` or `--extended-insert` option. Use `--skip-opt` instead.

Options

`mariadb-dump` supports the following options:

Option	Description
`--all`	Deprecated. Use `--create-options` instead.
`-A`, `--all-databases`	Dump all the databases. This will be same as `--databases` with all databases selected.
`-Y`, `--all-tablespaces`	Dump all the tablespaces.
`-y`, `--no-tablespaces`	Do not dump any tablespace information.
`--add-drop-database`	Add a DROP DATABASE before each create. Typically used in conjunction with the `--all-databases` or `--databases` option because no CREATE DATABASE statements are written unless one of those options is specified.
`--add-drop-table`	Add a DROP TABLE before each create.
`--add-drop-trigger`	Add a DROP TRIGGER statement before each CREATE TRIGGER. From MariaDB 10.2.6.
`--add-locks`	Add locks around INSERT statements, which results in faster inserts when the dump file is reloaded. Use `--skip-add-locks` to disable.
`--allow-keywords`	Allow creation of column names that are keywords. This works by prefixing each column name with the table name.
`--apply-slave-statements`	Adds STOP SLAVE prior to CHANGE MASTER and START SLAVE to bottom of dump.
`--as-of`	Dump system versioned table as of specified timestamp. From MariaDB 10.7.0.
`--character-sets-dir=name`	Directory for character set files.
`-i`, `--comments`	Write additional information in the dump file such as program version, server version, and host. Disable with `--skip-comments`.
`--compact`	Give less verbose output (useful for debugging). Disables structure comments and header/footer constructs. Enables the `--skip-add-drop-table`, `--skip-add-locks`, `--skip-comments`, `--skip-disable-keys`, and `--skip-set-charset` options.
`--compatible=name`	Change the dump to be compatible with a given mode. By default tables are dumped in a format optimized for MariaDB and MySQL. Legal modes are: `ansi`, `mysql323`, `mysql40`, `postgresql`, `oracle`, `mssql`, `db2`, `maxdb`, `no_key_options`, `no_table_options`, and `no_field_options`. One can use several modes separated by commas. This option does not guarantee compatibility with other servers. It only enables those SQL mode values that are currently available for making dump output more compatible. For example, `--compatible=oracle` does not map data types to Oracle types or use Oracle comment syntax.

`-c , --complete-insert`	Use complete INSERT statements that include column names.
`-C , --compress`	Use compression in server/client protocol. Both client and server must support compression for this to work.
`--copy-s3-tables`	By default S3 tables are ignored. With this option set, the result file will contain a CREATE statement for a similar Aria table, followed by the table data and ending with an `ALTER TABLE xxx ENGINE=S3` . From MariaDB 10.5.0.
`-a , --create-options`	Include all MariaDB and/or MySQL specific create options in `CREATE TABLE` statements. Use `--skip-create-options` to disable.
`-B , --databases`	Dump several databases. Normally, `mariadb-dump` treats the first name argument on the command line as a database name and following names as table names. With this option, it treats all name arguments as database names. CREATE DATABASE and USE statements are included in the output before each new database.
`-#, --debug[=#]`	If using a debug version of MariaDB, write a debugging log. A typical debug_options string is ´d:t:o,file_name´. The default value is ´d:t:o,/tmp/mysqldump.trace´. If using a non-debug version, `mariadb-dump` will catch this and exit.
`--debug-check`	Check memory and open file usage at exit.
`--debug-info`	Print some debug info at exit.
`--default-auth=name`	Default authentication client-side plugin to use.
`--default-character-set=name`	Set the default character set to *name*. If no character set is specified *mariadb-dump* uses utf8mb4.
`--defaults-extra-file=name`	Read the file *name* after the global files are read. Must be given as the first argument.
`--defaults-file=name`	Only read default options from the given file *name*. Must be given as the first argument.
`--defaults-group-suffix=str`	Also read groups with a suffix of *str*. For example, since *mariadb-dump* normally reads the [client] and [mariadb-dump] (or [mysqldump]) groups, --defaults-group-suffix=x would cause it to also read the groups [mariadb-dump_x] (or [mysqldump_x]) and [client_x].
`--delayed-insert`	Insert rows with INSERT DELAYED instead of INSERT.
`--delete-master-logs`	On a primary replication server, delete the binary logs by sending a PURGE BINARY LOGS statement to the server after performing the dump operation. This option automatically enables `--master-data` .
`-K , --disable-keys`	`'/*!40000 ALTER TABLE tb_name DISABLE KEYS */;` and `'/*!40000 ALTER TABLE tb_name ENABLE KEYS */;` will be put in the output. This makes loading the dump file faster because the indexes are created after all rows are inserted. This option is effective only for non-unique indexes of MyISAM tables. Disable with `--skip-disable-keys` .
`--dump-date`	If the `--comments` option and this option are given, `mariadb-dump` produces a comment at the end of the dump of the following form: `-- Dump completed on DATE` However, the date causes dump files taken at different times to appear to be different, even if the data are otherwise identical. `--dump-date` and `--skip-dump-date` control whether the date is added to the comment. The default is `--dump-date` (include the date in the comment). `--skip-dump-date` suppresses date printing.
`-H , --dump-history`	Dump tables with history. From MariaDB 10.11.0. Until this option, `mariadb-dump` could not read historical rows from versioned tables, and so historical data would not be backed up.
`--dump-slave[=value]`	Used for producing a dump file from a replica server that can be used to set up another replica server with the same primary. Causes the binary log position and filename of the primary to be appended to the dumped data output. Setting the value to `1` (the default) will print it as a CHANGE MASTER command in the dumped data output; if set to `2` , that command will be prefixed with a comment symbol. This option will turn `--lock-all-tables` on, unless `--single-transaction` is specified too (in which case a global read lock is only taken a short time at the beginning of the dump - don't forget to read about `--single-transaction` below). In all cases any action on logs will happen at the exact moment of the dump. Option automatically turns `--lock-tables` off. Using this option causes mariadb-dump to stop the replica SQL thread before beginning the dump, and restart it again after completion.
`-E, --events`	Include Event Scheduler events for the dumped databases in the output.
`-e , --extended-insert`	Use multiple-row INSERT syntax that include several `VALUES` lists. This results in a smaller dump file and speeds up inserts when the file is reloaded. Defaults to on; use `--skip-extended-insert` to disable.
`--fields-terminated-by=name`	Fields in the output file are terminated by the given string. Used with the `--tab` option and has the same meaning as the corresponding `FIELDS` clause for LOAD DATA INFILE.
`--fields-enclosed-by=name`	Fields in the output file are enclosed by the given character. Used with the `--tab` option and has the same meaning as the corresponding `FIELDS` clause for LOAD DATA INFILE.
`--fields-optionally-enclosed-by=name`	Fields in the output file are optionally enclosed by the given character. Used with the `--tab` option and has the same meaning as the corresponding `FIELDS` clause for LOAD DATA INFILE.
`--fields-escaped-by=name`	Fields in the output file are escaped by the given character. Used with the `--tab` option and has the same meaning as the corresponding `FIELDS` clause for LOAD DATA INFILE.
`--first-slave`	Removed in MariaDB 5.5. Use `--lock-all-tables` instead.
`-F , --flush-logs`	Flush the MariaDB server log files before starting the dump. This option requires the `RELOAD` privilege. If you use this option in combination with the `--databases=` or `--all-databases` option, the logs are flushed for each database dumped. The exception is when using `--lock-all-tables` or `--master-data` : In this case, the logs are flushed only once, corresponding to the moment all tables are locked. If you want your dump and the log flush to happen at the same exact moment, you should use `--flush-logs` together with either `--lock-all-tables` or `--master-data` .
`--flush-privileges`	Send a FLUSH PRIVILEGES statement to the server after dumping the mysql database. This option should be used any time the dump contains the mysql database and any other database that depends on the data in the mysql database for proper restoration.
`-f , --force`	Continue even if an SQL error occurs during a table dump. One use for this option is to cause *mariadb-dump* to continue executing even when it encounters a view that has become invalid because the definition refers to a table that has been dropped. Without `--force` in this example, `mariadb-dump` exits with an error message. With `--force` , *mariadb-dump* prints the error message, but it also writes an SQL comment containing the view definition to the dump output and continues executing.

`--gtid`	Used together with `--master-data` and `--dump-slave` to more conveniently set up a new GTID replica. It causes those options to output SQL statements that configure the replica to use the global transaction ID to connect to the primary instead of old-style filename/offset positions. The old-style positions are still included in comments when `--gtid` is used; likewise the GTID position is included in comments even if `--gtid` is not used.
`-?` , `--help`	Display a help message and exit.
`--hex-blob`	Dump binary strings in hexadecimal format (for example, `´abc´` becomes `0x616263`). The affected data types are BINARY, VARBINARY, the BLOB types, and BIT.
`-h name` , `--host=name`	Connect to and dump data from the MariaDB or MySQL server on the given host. The default host is `localhost` .
`--ignore-database=name`	Do not dump the specified database. To specify more than one database to ignore, use the directive multiple times, once for each database. Only takes effect when used together with `--all-databases` or `-A` . Added in MariaDB 10.3.7.
`--ignore-table=name`	Do not dump the specified table. To specify more than one table to ignore, use the directive multiple times, once for each table. Each table must be specified with both database and table names, e.g., `--ignore-table=database.table` . This option also can be used to ignore views.
`--ignore-table-data=name`	Do not dump the specified table data (only the structure). To specify more than one table to ignore, use the directive multiple times, once for each table. Each table must be specified with both database and table names. From MariaDB 10.1.46, MariaDB 10.2.33, MariaDB 10.3.24, MariaDB 10.4.14 and MariaDB 10.5.3. See also `--no-data` .
`--include-master-host-port`	Add the `MASTER_HOST` and `MASTER_PORT` options for the CHANGE MASTER TO statement when using the `--dump-slave` option for a replica dump.
`--insert-ignore`	Insert rows with `INSERT IGNORE` instead of `INSERT` .
`--lines-terminated-by=name`	Lines in the output file are terminated by the given string. This option is used with the `--tab` option and has the same meaning as the corresponding `LINES` clause for LOAD DATA INFILE.
`-x` , `--lock-all-tables`	Lock all tables across all databases. This is achieved by acquiring a global read lock for the duration of the whole dump by executing FLUSH TABLES WITH READ LOCK. This option automatically turns off `--single-transaction` and `--lock-tables` .
`-l` , `--lock-tables`	For each dumped database, lock all tables to be dumped before dumping them. The tables are locked with `READ LOCAL` to allow concurrent inserts in the case of MyISAM tables. For transactional tables such as InnoDB, `--single-transaction` is a much better option than `--lock-tables` because it does not need to lock the tables at all. Because `--lock-tables` locks tables for each database separately, this option does not guarantee that the tables in the dump file are logically consistent between databases. Tables in different databases may be dumped in completely different states. Use `--skip-lock-tables` to disable.
`--log-error=name`	Log warnings and errors by appending them to the named file. The default is to do no logging.
`--log-queries`	When restoring the dump, the server will, if logging is turned on, log the queries to the general and slow query log. Defaults to on; use --skip-log-queries to disable. Added in MariaDB 10.1.1.
`--master-data[=#]`	Causes the binary log position and filename to be appended to the output, useful for dumping a primary replication server to produce a dump file that can be used to set up another server as a replica of the primary. These are the primary server coordinates from which the replica should start replicating after you load the dump file into the replica. If the option is set to 1 (the default), will print it as a CHANGE MASTER command; if set to 2, that command will be prefixed with a comment symbol. This `--master-data` option will turn `--lock-all-tables` on, unless `--single-transaction` is specified too. Before MariaDB 5.3 this would take a global read lock for a short time at the beginning of the dump - see Enhancements for START TRANSACTION WITH CONSISTENT SNAPSHOT and the `--single-transaction` option below). In all cases, any action on logs will happen at the exact moment of the dump. This option automatically turns `--lock-tables` off. In all cases, any action on logs happens at the exact moment of the dump. It is also possible to set up a replica by dumping an existing replica of the primary. To do this, use the following procedure on the existing replica: 1. Stop the replica's SQL thread and get its current status: `mariadb> STOP SLAVE SQL_THREAD;` `mariadb> SHOW SLAVE STATUS;` 2. From the output of the SHOW SLAVE STATUS statement, the binary log coordinates of the primary server from which the new replica should start replicating are the values of the Relay_Master_Log_File and Exec_Master_Log_Pos fields. Denote those values as file_name and file_pos. 2. Dump the replica server: `shell> mariadb-dump --master-data=2 --all-databases > dumpfile` 3. Restart the replica: `mariadb> START SLAVE;` 4. On the new replica, load the dump file: `shell> mariadb < dumpfile` 5. On the new replica, set the replication coordinates to those of the primary server obtained earlier: `mariadb> CHANGE MASTER TO MASTER_LOG_FILE = ´file_name´, MASTER_LOG_POS = file_pos;` The `CHANGE MASTER` TO statement might also need other parameters, such as MASTER_HOST to point the replica to the correct primary server host. Add any such parameters as necessary.
`--max-allowed-packet=#`	The maximum packet length to send to or receive from server. The maximum is 1GB.
`--max-statement-time=#`	Sets the maximum time any statement can run before being timed out by the server. (Default value is 0 (no limit))
`--net-buffer-length=#`	The initial buffer size for client/server TCP/IP and socket communication. This can be used to limit the size of rows in the dump. When creating multiple-row `INSERT` statements (as with the `--extended-insert` or `--opt` option), `mariadb-dump` creates rows up to `net_buffer_length` length.
`--no-autocommit`	Enclose the INSERT statements for each dumped table within SET autocommit = 0 and COMMIT statements.
`-n` , `--no-create-db`	This option suppresses the CREATE DATABASE ... IF EXISTS statement that normally is output for each dumped database if `--all-databases` or `--databases` is given.

`-t , --no-create-info`	Do not write CREATE TABLE statements which re-create each dumped table.
`-d , --no-data`	Do not write any table row information (that is, do not dump table contents). This is useful if you want to dump only the CREATE TABLE statement for the table (for example, to create an empty copy of the table by loading the dump file). See also --ignore-table-data .
`--no-data-med`	Do not dump rows for engines that manage external data (i.e.MRG_MyISAM, MRG_ISAM, CONNECT, OQGRAPH, Spider, VP, Federated). This option is enabled by default. If you want to dump data for these engines, then you would need to set `--no-data-med=0` .
`--no-defaults`	Don't read default options from any option file. Must be given as the first argument.
`-N , --no-set-names`	Suppress the `SET NAMES` statement. This has the same effect as `--skip-set-charset` .
`--opt`	This option is shorthand. It is the same as specifying `--add-drop-table` , `--add-locks` , `--create-options` , `--quick` , `--extended-insert` , `--lock-tables` , `--set-charset` , and `--disable-keys` . Enabled by default, disable with `--skip-opt` . It should give you a fast dump operation and produce a dump file that can be reloaded into a MariaDB server quickly. The `--opt` option is enabled by default. Use `--skip-opt` to disable it. See the discussion at the beginning of this section for information about selectively enabling or disabling a subset of the options affected by `--opt` .
`--order-by-primary`	Sorts each table's rows by primary key, or first unique key, if such a key exists. This is useful when dumping a MyISAM table to be loaded into an InnoDB table, but will make the dump itself take considerably longer.
`--order-by-size`	Dump each table according to their size, smallest first. Useful when using `--single-transaction` on tables which get truncated/altered often. The assumption here is that smaller tables get truncated more often, and by dumping those first, this reduces the chance that a `--single-transaction` dump will fail with with 'Table definition has changed, please retry transaction'. From MariaDB 10.9.1.
`-p[passwd] , --password[=passwd`	The password to use when connecting to the server. If you use the short option form (`-p`), you cannot have a space between the option and the password. If you omit the password value following the `--password` or `-p` option on the command line, *mariadb-dump* prompts for one. Specifying a password on the command line should be considered insecure. You can use an option file to avoid giving the password on the command line.
`-W , --pipe`	On Windows, connect to the server via a named pipe. This option applies only if the server supports named-pipe connections.
`--plugin-dir`	Directory for client-side plugins.
`-P num , --port=num`	The TCP/IP port number to use for the connection.
`--print-defaults`	Print the program argument list and exit. Must be given as the first argument.
`--protocol=name`	The connection protocol to use for connecting to the server (TCP, SOCKET, PIPE, MEMORY). It is useful when the other connection parameters normally would cause a protocol to be used other than the one you want.
`-q , --quick`	This option is useful for dumping large tables. It forces *mariadb-dump* to retrieve rows for a table from the server a row at a time and to then dump the results directly to stdout rather than retrieving the entire row set and buffering it in memory before writing it out. Defaults to on, use `--skip-quick` to disable.
`-Q , --quote-names`	Quote identifiers (such as database, table, and column names) within backtick (`` ` ``) characters. If the `ANSI_QUOTES` SQL mode is enabled, identifiers are quoted within (`"`) characters. This option is enabled by default. It can be disabled with `--skip-quote-names` , but this option should be given after any option such as `--compatible` that may enable `--quote-names` .
`--replace`	Use REPLACE INTO statements instead of INSERT INTO statements.
`-r , --result-file=name`	Direct output to a given file. This option should be used on Windows to prevent newline `"\n"` characters from being converted to `"\r\n"` carriage return/newline sequences. The result file is created and its previous contents overwritten, even if an error occurs while generating the dump.
`-R , --routines`	Include stored routines (procedures and functions) for the dumped databases in the output. Use of this option requires the `SELECT` privilege for the mysql.proc table. The output generated using `--routines` contains CREATE PROCEDURE and CREATE FUNCTION statements to re-create the routines. However, these statements do not include attributes such as the routine creation and modification timestamps. This means that when the routines are reloaded, they will be created with the timestamps equal to the reload time. If you require routines to be re-created with their original timestamp attributes, do not use `--routines` . Instead, dump and reload the contents of the mysql.proc table directly, using a MariaDB account which has appropriate privileges for the mysql database.
`set-charset`	Add `'SET NAMES default_character_set'` to the output in order to set the character set. Enabled by default; suppress with `--skip-set-charset` .
`-O, --set-variable=name`	Change the value of a variable. Please note that this option is deprecated; you can set variables directly with `--variable-name=value` .
`--shared-memory-base-name`	Shared-memory name to use for Windows connections using shared memory to a local server (started with the `--shared-memory` option). Case-sensitive. Defaults to `MYSQL` .

`--single-transaction`	This option sends a START TRANSACTION SQL statement to the server before dumping data. It is useful only with transactional tables such as InnoDB, because then it dumps the consistent state of the database at the time when `BEGIN` was issued without blocking any applications. When using this option, you should keep in mind that only InnoDB tables are dumped in a consistent state. The single-transaction feature depends not only on the engine being transactional and capable of REPEATABLE-READ, but also on START TRANSACTION WITH CONSISTENT SNAPSHOT. The dump is **not** guaranteed to be consistent for other storage engines. For example, any TokuDB, MyISAM or MEMORY tables dumped while using this option may still change state. While a `--single-transaction` dump is in process, to ensure a valid dump file (correct table contents and binary log coordinates), no other connection should use the following statements: ALTER TABLE, CREATE TABLE, DROP TABLE, RENAME TABLE, or TRUNCATE TABLE. A consistent read is not isolated from those statements, so use of them on a table to be dumped can cause the `SELECT` (performed by *mariadb-dump* to retrieve the table contents) to obtain incorrect contents or fail. The `--single-transaction` option and the `--lock-tables` option are mutually exclusive because LOCK TABLES causes any pending transactions to be committed implicitly. So this option automatically turns off `--lock-tables` To dump large tables, you should combine the `--single-transaction` option with `--quick`.
`--skip-add-locks`	Disable the `--add-locks` option.
`--skip-comments`	Disable the `--comments` option.
`--skip-disable-keys`	Disable the `--disable-keys` option.
`--skip-extended-insert`	Disable the `--extended-insert` option.
`--skip-opt`	Disable the `--opt` option (disables `--add-drop-table`, `--add-locks`, `--create-options`, `--quick`, `--extended-insert`, `--lock-tables`, `--set-charset`, and `--disable-keys`).
`--skip-quick`	Disable the `--quick` option.
`--skip-quote-name`	Disable the `--quote-names` option.
`--skip-set-charset`	Disable the `--set-charset` option.
`--skip-triggers`	Disable the `--triggers` option.
`--skip-tz-utc`	Disable the `--tz-utc` option.
`-S name`, `--socket=name`	For connections to localhost, the Unix socket file to use, or, on Windows, the name of the named pipe to use.
`--ssl`	Enables TLS. TLS is also enabled even without setting this option when certain other TLS options are set. Starting with MariaDB 10.2, the `--ssl` option will not enable verifying the server certificate by default. In order to verify the server certificate, the user must specify the `--ssl-verify-server-cert` option.
`--ssl-ca=name`	Defines a path to a PEM file that should contain one or more X509 certificates for trusted Certificate Authorities (CAs) to use for TLS. This option requires that you use the absolute path, not a relative path. See Secure Connections Overview: Certificate Authorities (CAs) for more information. This option implies the `--ssl` option.
`--ssl-capath=name`	Defines a path to a directory that contains one or more PEM files that should each contain one X509 certificate for a trusted Certificate Authority (CA) to use for TLS. This option requires that you use the absolute path, not a relative path. The directory specified by this option needs to be run through the `openssl rehash` command. See Secure Connections Overview: Certificate Authorities (CAs) for more information. This option is only supported if the client was built with OpenSSL or yaSSL. If the client was built with GnuTLS or Schannel, then this option is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms. This option implies the `--ssl` option.
`--ssl-cert=name`	Defines a path to the X509 certificate file to use for TLS. This option requires that you use the absolute path, not a relative path. This option implies the `--ssl` option.
`--ssl-cipher=name`	List of permitted ciphers or cipher suites to use for TLS. This option implies the `--ssl` option.
`--ssl-crl=name`	Defines a path to a PEM file that should contain one or more revoked X509 certificates to use for TLS. This option requires that you use the absolute path, not a relative path. See Secure Connections Overview: Certificate Revocation Lists (CRLs) for more information. This option is only supported if the client was built with OpenSSL or Schannel. If the client was built with yaSSL or GnuTLS, then this option is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.
`--ssl-crlpath=name`	Defines a path to a directory that contains one or more PEM files that should each contain one revoked X509 certificate to use for TLS. This option requires that you use the absolute path, not a relative path. The directory specified by this option needs to be run through the `openssl rehash` command. See Secure Connections Overview: Certificate Revocation Lists (CRLs) for more information. This option is only supported if the client was built with OpenSSL. If the client was built with yaSSL, GnuTLS, or Schannel, then this option is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.
`--ssl-key=name`	Defines a path to a private key file to use for TLS. This option requires that you use the absolute path, not a relative path. This option implies the `--ssl` option.
`--ssl-verify-server-cert`	Enables server certificate verification. This option is disabled by default.
`--system=option[,option]`	Dump the database's system tables in a logical form. With this option, the mysql database tables are dumped as CREATE USER, CREATE SERVER and other forms of logical portable SQL statements. The option values here are from the set of `all`, `users`, `plugins`, `udfs`, `servers`, `stats`, `timezones`. From MariaDB 10.2.37, MariaDB 10.3.28, MariaDB 10.4.18 and MariaDB 10.5.9.

`-T`, `--tab=name`	Produce tab-separated text-format data files. With this option, for each dumped table `mariadb-dump` will create a `tbl_name.sql` file containing the `CREATE TABLE` statement that creates the table, and a `tbl_name.txt` file containing the table's data. The option value is the directory in which to write the files. **Note:** *This option can only be used when* mariadb-dump/ is run on the same machine as the mysqld server. You must have the `FILE` privilege, and the server must have permission to write files in the directory that you specify. *By default, the `.txt` data files are formatted using tab characters between column values and a newline at the end of each line. The format can be specified explicitly using the `--fields-xxx` and `--lines-terminated-by` options.* *Column values are converted to the character set specified by the `--default-character-set` option.*
`--tables`	This option overrides the `--databases` (`-B`) option. *mariadb-dump* regards all name arguments following the option as table names.
`--tls-version=name`	This option accepts a comma-separated list of TLS protocol versions. A TLS protocol version will only be enabled if it is present in this list. All other TLS protocol versions will not be permitted. See Secure Connections Overview: TLS Protocol Versions for more information. This option was added in MariaDB 10.4.6.
`--triggers`	Include triggers for each dumped table in the output. This option is enabled by default; disable it with `--skip-triggers`.
`--tz-utc`	This option enables TIMESTAMP columns to be dumped and reloaded between servers in different time zones. *mariadb-dump* sets its connection time zone to UTC and adds `SET TIME_ZONE='+00:00'` to the dump file. Without this option, TIMESTAMP columns are dumped and reloaded in the time zones local to the source and destination servers, which can cause the values to change if the servers are in different time zones. `--tz-utc` also protects against changes due to daylight saving time. `--tz-utc` is enabled by default. To disable it, use `--skip-tz-utc`.
`-u name`, `--user=name`	The MariaDB user name to use when connecting to the server.
`-v`, `--verbose`	Verbose mode. Print more information about what the program is doing during various stages.
`-V`, `--version`	Output version information and exit.
`-w cond`, `--where=cond`	Dump only rows selected by the given `WHERE` condition *cond*. Quotes around the condition are mandatory if it contains spaces or other characters that are special to your command interpreter. Examples: `--where="user='jimf'"` `-w"userid>1"` `-w"userid<1"`
`-X`, `--xml`	Dump a database as well formed XML.

Group Options

Some `mariadb-dump` options are shorthand for groups of other options:

- Use of `--opt` is the same as specifying `--add-drop-table`, `--add-locks`, `--create-options`, `--disable-keys`, `--extended-insert`, `--lock-tables`, `--quick`, and `--set-charset`. All of the options that `--opt` stands for also are on by default because `--opt` is on by default.
- Use of `--compact` is the same as specifying `--skip-add-drop-table`, `--skip-add-locks`, `--skip-comments`, `--skip-disable-keys`, and `--skip-set-charset` options.

To reverse the effect of a group option, uses its `--skip-xxx` form (`--skip-opt` or `--skip-compact`). It is also possible to select only part of the effect of a group option by following it with options that enable or disable specific features. Here are some examples:

- To select the effect of `--opt` except for some features, use the `--skip` option for each feature. To disable extended inserts and memory buffering, use `--opt --skip-extended-insert --skip-quick`. (Actually, `--skip-extended-insert --skip-quick` is sufficient because `--opt` is on by default.)
- To reverse `--opt` for all features except index disabling and table locking, use `--skip-opt --disable-keys --lock-tables`.

When you selectively enable or disable the effect of a group option, order is important because options are processed first to last. For example, `--disable-keys --lock-tables --skip-opt` would not have the intended effect; it is the same as `--skip-opt` by itself.

Special Characters in Option Values

Some options, like `--lines-terminated-by`, accept a string. The string can be quoted, if necessary. For example, on Unix systems this is the option to enclose fields within double quotes:

```
--fields-enclosed-by='"'
```

An alternative to specify the hexadecimal value of a character. For example, the following syntax works on any platform:

```
--fields-enclosed-by=0x22
```

Option Files

In addition to reading options from the command-line, `mariadb-dump` can also read options from option files. If an unknown option is provided to `mariadb-dump` in an option file, then it is ignored.

The following options relate to how MariaDB command-line tools handles option files. They must be given as the first argument on the command-line:

Option	Description
`--print-defaults`	Print the program argument list and exit.
`--no-defaults`	Don't read default options from any option file.
`--defaults-file=#`	Only read default options from the given file #.
`--defaults-extra-file=#`	Read this file after the global files are read.
`--defaults-group-suffix=#`	In addition to the default option groups, also read option groups with this suffix.

`mariadb-dump` is linked with MariaDB Connector/C. However, MariaDB Connector/C does not yet handle the parsing of option files for this client. That is still performed by the server option file parsing code. See MDEV-19035 for more information.

Option Groups

`mariadb-dump` reads options from the following option groups from option files:

Group	Description
`[mysqldump]`	Options read by *mariadb-dump*, which includes both MariaDB Server and MySQL Server.
`[mariadb-dump]`	Options read by *mariadb-dump*. Available starting with MariaDB 10.4.6.
`[client]`	Options read by all MariaDB and MySQL client programs, which includes both MariaDB and MySQL clients. For example, `mysqldump` .
`[client-server]`	Options read by all MariaDB client programs and the MariaDB Server. This is useful for options like socket and port, which is common between the server and the clients.
`[client-mariadb]`	Options read by all MariaDB client programs.

NULL, ´NULL´, and Empty Values in XML

For a column named `column_name` , the `NULL` value, an empty string, and the string value `´NULL´` are distinguished from one another in the output generated by this option as follows.

Value	XML Representation
NULL (unknown value)	`<field name="column_name" xsi:nil="true" />`
´´ (empty string)	`<field name="column_name"></field>`
´NULL´ (string value)	`<field name="column_name">NULL</field>`

The output from the mariadb client when run using the `--xml` option also follows the preceding rules.

XML output from mariadb-dump includes the XML namespace, as shown here :

```
shell> mariadb-dump --xml -u root world City
<?xml version="1.0"?>
<mariadb-dump xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<database name="world">
<table_structure name="City">
<field Field="ID" Type="int(11)" Null="NO" Key="PRI" Extra="auto_increment" />
<field Field="Name" Type="char(35)" Null="NO" Key="" Default="" Extra="" />
<field Field="CountryCode" Type="char(3)" Null="NO" Key="" Default="" Extra="" />
<field Field="District" Type="char(20)" Null="NO" Key="" Default="" Extra="" />
<field Field="Population" Type="int(11)" Null="NO" Key="" Default="0" Extra="" />
<key Table="City" Non_unique="0" Key_name="PRIMARY" Seq_in_index="1" Column_name="ID"
Collation="A" Cardinality="4079" Null="" Index_type="BTREE" Comment="" />
<options Name="City" Engine="MyISAM" Version="10" Row_format="Fixed" Rows="4079"
Avg_row_length="67" Data_length="273293" Max_data_length="18858823439613951"
Index_length="43008" Data_free="0" Auto_increment="4080"
Create_time="2007-03-31 01:47:01" Update_time="2007-03-31 01:47:02"
Collation="latin1_swedish_ci" Create_options="" Comment="" />
</table_structure>
<table_data name="City">
<row>
<field name="ID">1</field>
<field name="Name">Kabul</field>
<field name="CountryCode">AFG</field>
<field name="District">Kabol</field>
<field name="Population">1780000</field>
</row>
...
<row>
<field name="ID">4079</field>
<field name="Name">Rafah</field>
<field name="CountryCode">PSE</field>
<field name="District">Rafah</field>
<field name="Population">92020</field>
</row>
</table_data>
</database>
</mariadb-dump>
```

Restoring

To restore a backup created with mariadb-dump, use the mariadb client to import the dump, for example:

```
mariadb db_name < backup-file.sql
```

Variables

You can also set the following variables (`--variable-name=value`) and boolean options `{FALSE|TRUE}` by using:

Name	Default Values
all	TRUE
all-databases	FALSE
all-tablespaces	FALSE
no-tablespaces	FALSE
add-drop-database	FALSE
add-drop-table	TRUE
add-drop-trigger	FALSE
add-locks	TRUE
allow-keywords	FALSE
apply-slave-statements	FALSE
as-of	*(No default value)*

character-sets-dir	*(No default value)*
comments	TRUE
compatible	*(No default value)*
compact	FALSE
complete-insert	FALSE
compress	FALSE
copy-s3-tables	FALSE
create-options	TRUE
databases	FALSE
debug-check	FALSE
debug-info	FALSE
default-character-set	utf8mb4
delayed-insert	FALSE
delete-master-logs	FALSE
disable-keys	TRUE
events	FALSE
extended-insert	TRUE
fields-terminated-by	*(No default value)*
fields-enclosed-by	*(No default value)*
fields-optionally-enclosed-by	*(No default value)*
fields-escaped-by	*(No default value)*
flush-logs	FALSE
flush-privileges	FALSE
force	FALSE
hex-blob	FALSE
host	*(No default value)*
include-master-host-port	FALSE
insert-ignore	FALSE
lines-terminated-by	*(No default value)*
lock-all-tables	FALSE
lock-tables	TRUE
log-error	*(No default value)*
log-queries	TRUE
master-data	0
max_allowed_packet	16777216
net-buffer-length	1046528
no-autocommit	FALSE
no-create-db	FALSE
no-create-info	FALSE
no-data	FALSE
no-data-med	TRUE

order-by-primary	FALSE
port	0
quick	TRUE
quote-names	TRUE
replace	FALSE
routines	FALSE
set-charset	TRUE
single-transaction	FALSE
dump-date	TRUE
socket	*No default value)*
ssl	FALSE
ssl-ca	*(No default value)*
ssl-capath	*(No default value)*
ssl-cert	*(No default value)*
ssl-cipher	*(No default value)*
ssl-key	*(No default value)*
ssl-verify-server-cert	FALSE
system	*(No default value)*
tab	*(No default value)*
triggers	TRUE
tz-utc	TRUE
user	*(No default value)*
verbose	FALSE
where	*(No default value)*
plugin-dir	*(No default value)*
default-auth	*(No default value)*

Examples

A common use of `mariadb-dump` is for making a backup of an entire database:

```
shell> mariadb-dump db_name > backup-file.sql
```

You can load the dump file back into the server like this:

```
shell> mariadb db_name < backup-file.sql
```

Or like this:

```
shell> mariadb -e "source /path-to-backup/backup-file.sql" db_name
```

`mariadb-dump` is also very useful for populating databases by copying data from one MariaDB server to another:

```
shell> mariadb-dump --opt db_name | mariadb --host=remote_host -C db_name
```

It is possible to dump several databases with one command:

```
shell> mariadb-dump --databases db_name1 [db_name2 ...] > my_databases.sql
```

To dump all databases, use the `--all-databases` option:

```
shell> mariadb-dump --all-databases > all_databases.sql
```

For InnoDB tables, `mariadb-dump` provides a way of making an online backup:

```
shell> mariadb-dump --all-databases --single-transaction all_databases.sql
```

This backup acquires a global read lock on all tables (using `FLUSH TABLES WITH READ LOCK`) at the beginning of the dump. As soon as this lock has been acquired, the binary log coordinates are read and the lock is released. If long updating statements are running when the FLUSH statement is issued, the MariaDB server may get stalled until those statements finish. After that, the dump becomes lock free and does not disturb reads and writes on the tables. If the update statements that the MariaDB server receives are short (in terms of execution time), the initial lock period should not be noticeable, even with many updates.

For point-in-time recovery (also known as "roll-forward," when you need to restore an old backup and replay the changes that happened since that backup), it is often useful to rotate the binary log or at least know the binary log coordinates to which the dump corresponds:

```
shell> mariadb-dump --all-databases --master-data=2 > all_databases.sql
```

Or:

```
shell> mariadb-dump --all-databases --flush-logs --master-data=2 > all_databases.sql
```

The `--master-data` and `--single-transaction` options can be used simultaneously, which provides a convenient way to make an online backup suitable for use prior to point-in-time recovery if tables are stored using the InnoDB storage engine.

1.3.6.3 mariadb-hotcopy

mariadb-hotcopy is deprecated and may be removed in a future release.

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadb-hotcopy` is a symlink to `mysqlhotcopy`, the deprecated backup script.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mariadb-hotcopy` is the name of the script, with `mysqlhotcopy` a symlink .

`mariadb-hotcopy` is a Perl script that was originally written (as `mysqlhotcopy`) and contributed by Tim Bunce. It uses FLUSH TABLES, LOCK TABLES, and cp or scp to make a database backup. It is a fast way to make a backup of the database or single tables, but it can be run only on the same machine where the database directories are located. `mariadb-hotcopy` > works only for backing up MyISAM and ARCHIVE tables. It runs on Unix and NetWare.

To use `mariadb-hotcopy`, you must have read access to the files for the tables that you are backing up, the SELECT privilege for those tables, the RELOAD privilege (to be able to execute FLUSH TABLES), and the LOCK TABLES privilege (to be able to lock the tables).

```
shell> mariadb-hotcopy db_name [/path/to/new_directory]
shell> mariadb-hotcopy db_name_1 ... db_name_n /path/to/new_directory
```

Back up tables in the given database that match a regular expression:

```
shell> mariadb-hotcopy db_name./regex/
```

The regular expression for the table name can be negated by prefixing it with a tilde (" `~` "):

```
shell> mariadb-hotcopy db_name./~regex/
```

`mariadb-hotcopy` supports the following options, which can be specified on the command line or in the [`mariadb-`

`hotcopy`] and [`client`] option file groups.

Option	Description
`--help`, `-?`	Display a help message and exit.
`--addtodest`	Do not rename target directory (if it exists); merely add files to it.
`--allowold`	Do not abort if a target exists; rename it by adding an _old suffix.
`--checkpoint=db_name.tbl_name`	Insert checkpoint entries into the specified database `db_name` and table `tbl_name`.
`--chroot=path`	Base directory of the chroot jail in which mysqld operates. The path value should match that of the `--chroot` option given to mysqld.
`--debug`	Enable debug output.
`--dryrun`, `-n`	Report actions without performing them.
`--flushlog`	Flush logs after all tables are locked.
`--host=host_name`, `-h host_name`	The host name of the local host to use for making a TCP/IP connection to the local server. By default, the connection is made to localhost using a Unix socket file.
`--keepold`	Do not delete previous (renamed) target when done.
`--method=command`	The method for copying files (cp or scp). The default is cp.
`--noindices`	Do not include full index files for MyISAM tables in the backup. This makes the backup smaller and faster. The indexes for reloaded tables can be reconstructed later with myisamchk -rq.
`--old-server`	Connect to old MySQL-server (before v5.5) which doesn't have FLUSH TABLES WITH READ LOCK fully implemented.
`--password=password`, `-ppassword`	The password to use when connecting to the server. The password value is not optional for this option, unlike for other MariaDB programs. Specifying a password on the command line should be considered insecure. You can use an option file to avoid giving the password on the command line.
`--port=port_num`, `-P port_num`	The TCP/IP port number to use when connecting to the local server.
`--quiet`, `-q`	Be silent except for errors.
`--record_log_pos=db_name.tbl_name`	Record master and slave status in the specified database db_name and table tbl_name.
`--regexp=expr`	Copy all databases with names that match the given regular expression.
`--resetmaster`	Reset the binary log after locking all the tables.
`--resetslave`	Reset the master.info file after locking all the tables.
`--socket=path`, `-S path`	The Unix socket file to use for connections to localhost.
`--suffix=str`	The suffix to use for names of copied databases.
`--tmpdir=path`	The temporary directory. The default is /tmp.
`--user=username`, `-u username`	The MariaDB username to use when connecting to the server.

Use perldoc for additional `mariadb-hotcopy` documentation, including information about the structure of the tables needed for the `--checkpoint` and `--record_log_pos` options:

```
shell> perldoc mariadb-hotcopy
```

1.3.6.4 mariadb-import

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadb-import` is a symlink to `mysqlimport`, the old name of tool for loading tables from text files in various formats.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mariadb-import` is the name of the script, with `mysqlimport` a symlink .

`mariadb-import` loads tables from text files in various formats. The base name of the text file must be the name of the table that should be used. If one uses sockets to connect to the MariaDB server, the server will open and read the text file directly. In other cases the client will open the text file. The SQL command LOAD DATA INFILE is used to import the rows.

Using mariadb-import

The command to use `mariadb-import` and the general syntax is:

```
mariadb-import [OPTIONS] database textfile1 [textfile2 ...]
```

Options

`mariadb-import` supports the following options:

variable	Description
`--character-sets-dir=name`	Directory for character set files.
`-c cols , --columns=cols`	Use only these columns to import the data to. Give the column names in a comma separated list. This is same as giving columns to LOAD DATA INFILE.
`-C , --compress`	Use compression in server/client protocol.
`-# [options] , --debug[=options]`	Output debug log. Often this is `d:t:o,filename` . The default is `d:t:o` .
`--debug-check`	Check memory and open file usage at exit.
`--debug-info`	Print some debug info at exit.
`--default-auth=plugin`	Default authentication client-side plugin to use.
`--default-character-set=name`	Set the default character set.
`--defaults-extra-file=name`	Read this file after the global files are read. Must be given as the first option.
`--defaults-file=name`	Only read default options from the given file *name* Must be given as the first option.
`--defaults-group-suffix=name`	In addition to the given groups, also read groups with this suffix.
`-d , --delete`	First delete all rows from table.
`--fields-terminated-by=name`	Fields in the input file are terminated by the given string.
`--fields-enclosed-by=name`	Fields in the import file are enclosed by the given character.
`--fields-optionally-enclosed-by=name`	Fields in the input file are optionally enclosed by the given character.
`--fields-escaped-by=name`	Fields in the input file are escaped by the given character.
`-f , --force`	Continue even if we get an SQL error.
`-? , --help`	Displays this help and exits.

`-h name , --host=name`	Connect to host.
`-i , --ignore`	If duplicate unique key was found, keep old row.
`k , --ignore-foreign-keys`	Disable foreign key checks while importing the data. From MariaDB 10.3.16, MariaDB 10.2.25 and MariaDB 10.1.41.
`--ignore-lines=n`	Ignore first *n* lines of data infile.
`--lines-terminated-by=name`	Lines in the input file are terminated by the given string.
`-L , --local`	Read all files through the client.
`-l , --lock-tables`	Lock all tables for write (this disables threads).
`--low-priority`	Use LOW_PRIORITY when updating the table.
`--no-defaults`	Don't read default options from any option file. Must be given as the first option.
`-p[passwd] , --password[=passwd]`	Password to use when connecting to server. If password is not given it's asked from the terminal. Specifying a password on the command line should be considered insecure. You can use an option file to avoid giving the password on the command line.
`--pipe , -W`	On Windows, connect to the server via a named pipe. This option applies only if the server supports named-pipe connections.
`--plugin-dir`	Directory for client-side plugins.
`-P num , --port=num`	Port number to use for connection or 0 for default to, in order of preference, my.cnf, the MYSQL_TCP_PORT environment variable, /etc/services, built-in default (3306).
`--print-defaults`	Print the program argument list and exit. Must be given as the first option.
`--protocol=name`	The protocol to use for connection (tcp, socket, pipe, memory).
`-r , --replace`	If duplicate unique key was found, replace old row.
`--shared-memory-base-name`	Shared-memory name to use for Windows connections using shared memory to a local server (started with the `--shared-memory` option). Case-sensitive.
`-s , --silent`	Silent mode. Produce output only when errors occur.
`-S , --socket=name`	For connections to localhost, the Unix socket file to use, or, on Windows, the name of the named pipe to use.
`--ssl`	Enables TLS. TLS is also enabled even without setting this option when certain other TLS options are set. Starting with MariaDB 10.2, the `--ssl` option will not enable verifying the server certificate by default. In order to verify the server certificate, the user must specify the `--ssl-verify-server-cert` option.
`--ssl-ca=name`	Defines a path to a PEM file that should contain one or more X509 certificates for trusted Certificate Authorities (CAs) to use for TLS. This option requires that you use the absolute path, not a relative path. See Secure Connections Overview: Certificate Authorities (CAs) for more information. This option implies the `--ssl` option.
`--ssl-capath=name`	Defines a path to a directory that contains one or more PEM files that should each contain one X509 certificate for a trusted Certificate Authority (CA) to use for TLS. This option requires that you use the absolute path, not a relative path. The directory specified by this option needs to be run through the `openssl rehash` command. See Secure Connections Overview: Certificate Authorities (CAs) for more information. This option is only supported if the client was built with OpenSSL or yaSSL. If the client was built with GnuTLS or Schannel, then this option is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms. This option implies the `--ssl` option.
`--ssl-cert=name`	Defines a path to the X509 certificate file to use for TLS. This option requires that you use the absolute path, not a relative path. This option implies the `--ssl` option.
`--ssl-cipher=name`	List of permitted ciphers or cipher suites to use for TLS. This option implies the `--ssl` option.

`--ssl-crl=name`	Defines a path to a PEM file that should contain one or more revoked X509 certificates to use for TLS. This option requires that you use the absolute path, not a relative path. See Secure Connections Overview: Certificate Revocation Lists (CRLs) for more information. This option is only supported if the client was built with OpenSSL or Schannel. If the client was built with yaSSL or GnuTLS, then this option is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.
`--ssl-crlpath=name`	Defines a path to a directory that contains one or more PEM files that should each contain one revoked X509 certificate to use for TLS. This option requires that you use the absolute path, not a relative path. The directory specified by this option needs to be run through the `openssl rehash` command. See Secure Connections Overview: Certificate Revocation Lists (CRLs) for more information. This option is only supported if the client was built with OpenSSL. If the client was built with yaSSL, GnuTLS, or Schannel, then this option is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.
`--ssl-key=name`	Defines a path to a private key file to use for TLS. This option requires that you use the absolute path, not a relative path. This option implies the `--ssl` option.
`--ssl-verify-server-cert`	Enables server certificate verification. This option is disabled by default.
`--tls-version=name`	This option accepts a comma-separated list of TLS protocol versions. A TLS protocol version will only be enabled if it is present in this list. All other TLS protocol versions will not be permitted. See Secure Connections Overview: TLS Protocol Versions for more information. This option was added in MariaDB 10.4.6.
`--use-threads=num`	Load files in parallel. The argument is the number of threads to use for loading data.
`-u name`, `--user=name`	User for login if not current user.
`-v`, `--verbose`	Print info about the various stages.
`-V`, `--version`	Output version information and exit.

Option Files

In addition to reading options from the command-line, `mariadb-import` can also read options from option files. If an unknown option is provided to `mariadb-import` in an option file, then it is ignored.

The following options relate to how MariaDB command-line tools handles option files. They must be given as the first argument on the command-line:

Option	Description
`--print-defaults`	Print the program argument list and exit.
`--no-defaults`	Don't read default options from any option file.
`--defaults-file=#`	Only read default options from the given file #.
`--defaults-extra-file=#`	Read this file after the global files are read.

In MariaDB 10.2 and later, `mariadb-import` is linked with MariaDB Connector/C. Therefore, it may be helpful to see Configuring MariaDB Connector/C with Option Files for more information on how MariaDB Connector/C handles option files.

Option Groups

`mariadb-import` reads options from the following option groups from option files:

Group	Description
`[mysqlimport]`	Options read by `mysqlimport`, which includes both MariaDB Server and MySQL Server.
`[mariadb-import]`	Options read by `mysqlimport`. Available starting with MariaDB 10.4.6.
`[client]`	Options read by all MariaDB and MySQL client programs, which includes both MariaDB and MySQL clients. For example, `mysqldump`.

`[client-server]`	Options read by all MariaDB client programs and the MariaDB Server. This is useful for options like socket and port, which is common between the server and the clients.
`[client-mariadb]`	Options read by all MariaDB client programs.

Default Values

Variables (`--variable-name=value`) and boolean options `{FALSE | TRUE}`	Value (after reading options)
`character-sets-dir`	*(No default value)*
`default-character-set`	latin1
`columns`	*(No default value)*
`compress`	FALSE
`debug-check`	FALSE
`debug-info`	FALSE
`delete`	FALSE
`fields-terminated-by`	*(No default value)*
`fields-enclosed-by`	*(No default value)*
`fields-optionally-enclosed-by`	*(No default value)*
`fields-escaped-by`	*(No default value)*
`force`	FALSE
`host`	*(No default value)*
`ignore`	FALSE
`ignore-lines`	0
`lines-terminated-by`	*(No default value)*
`local`	FALSE
`lock-tables`	FALSE
`low-priority`	FALSE
`port`	3306
`replace`	FALSE
`silent`	FALSE
`socket`	/var/run/mysqld/mysqld.sock
`ssl`	FALSE
`ssl-ca`	*(No default value)*
`ssl-capath`	*(No default value)*
`ssl-cert`	*(No default value)*
`ssl-cipher`	*(No default value)*
`ssl-key`	*(No default value)*
`ssl-verify-server-cert`	FALSE
`use-threads`	0
`user`	*(No default value)*
`verbose`	FALSE

1.3.7 Graphical and Enhanced Clients

This list is incomplete - most MySQL tools will work with MariaDB. See also a list of projects that officially work with MariaDB.

dbForge Studio for MariaDB
Universal GUI Tool for Management & Administration, Development for MariaDB and MySQL

dbForge Edge
dbForge Edge is a versatile software solution designed to meet the needs of...

DBeaver
Free convenient cross-platform and cross-database Java GUI client

ERBuilder Data Modeler
A data modeling tool for multiple databases platforms including MariaDB, MySQL, and more ...

SQLyog: Community Edition
SQLyog Community Edition

HeidiSQL
Windows GUI client for MariaDB and MySQL.

Navicat
Graphical front-end for MariaDB

Querious
Mac OS X tool for database administration

TablePlus
A modern, native GUI client for multiple databases

Database Workbench
Database development environment for multiple database systems including MySQL and MariaDB

Luna Modeler
Luna Modeler is a database design tool for MariaDB. Draw ER diagrams, visua...

SQL Diagnostic Manager & SQLyog
Graphical MariaDB manager and monitor

mycli
Command line interface with auto-completion and syntax highlighting

ocelotgui
Linux client for MySQL and MariaDB

phpMyAdmin
Web-based MariaDB administration tool

Sequel Pro
Database management tool running on Mac

SQLTool Pro Database Editor
Android SQL client

dbForge Data Compare
A tool for MariaDB & MySQL data comparison and synchronization of data betw...

dbForge Data Generator
A tool for generation of large volumes of meaningful test table data.

dbForge Documenter for MariaDB and MySQL
dbForge Documenter is a useful tool for MariaDB and MySQL database for the ...

dbForge Fusion: MySQL & MariaDB Plugin for VS

Visual Studio plugin designed to simplify database development and management.

dbForge Query Builder for MySQL & MariaDB

A tool for visual query creation without code typing.

dbForge Schema Compare for MariaDB & MySQL

A tool for comparison and synchronization of DDL differences between database objects.

DbSchema

Mariadb Diagram Designer & Admin GUI Tool

Improved SQL Document Parser Performance in Updated dbForge Tools for MySQL and MariaDB

Devart has upgraded dbForge Tools for MySQL and MariaDB with improved SQL d...

OmniDB

Browser-based IDE for MariaDB Administration

TOAD Edge

Windows GUI for MySQL. SQL Syntax Check. Freeware (Basic Features) & Payware (Extended Features).

TOAD for MySQL

Windows GUI for MySQL. Compatible with MariaDB. Freeware. SQL syntax check.

SQLPro Studio

SQLPro Studio is a fully native database client for macOS and iOS.

SB Data Generator

A tool to generate and populate selected tables or entire databases with realistic test data.

Beekeeper Studio

Open source and free GUI with a focus on usability. Mac, Linux, and Windows

LibreOffice Base

An open source RDBMS front-end tool to create and manage various databases

Valentina Studio

Free, advanced MariaDB GUI native on macOS, Windows & Linux, with advanced commercial version

DbVisualizer

Cross-platform universal database tool supporting MariaDB, PostgreSQL, MySQL and more

KS DB Merge Tools for MySQL and MariaDB

Windows GUI to compare and synchronize MySQL and MariaDB schema and data, freemium

There are 5 related questions.

1.3.8 MyISAM Clients and Utilities

Clients and utilities for working with the MyISAM storage engine

myisamchk

Utility for checking, repairing and optimizing MyISAM tables.

Memory and Disk Use With myisamchk

myisamchk's performance can be dramatically enhanced for larger tables

myisamchk Table Information

myisamchk can be used to obtain information about MyISAM tables

myisamlog

Process the MyISAM log

myisampack

Tool for compressing MyISAM tables

myisam_ftdump

A tool for displaying information on MyISAM FULLTEXT indexes.

1.3.8.1 myisamchk

Contents

myisamchk is a commandline tool for checking, repairing and optimizing non-partitioned MyISAM tables.

myisamchk is run from the commandline as follows:

```
myisamchk [OPTIONS] tables[.MYI]
```

The full list of options are listed below. One or more MyISAM tables can be specified. MyISAM tables have an associated .MYI index file, and the table name can either be specified with or without the .MYI extension. Referencing it with the extension allows you to use wildcards, so it's possible to run myisamchk on *all* the MyISAM tables in the database with `*.MYI`.

The path to the files must also be specified if they're not in the current directory.

> myisamchk should not be run while anyone is accessing any of the affected tables. It is also best to make a backup before running.

With no options, myisamchk simply checks your table as the default operation.

The following options can be set while passed as commandline options to myisamchk, or set with a [myisamchk] section in your my.cnf file.

General Options

Option	Description
-H, --HELP	Display help and exit. Options are presented in a single list.
-?, --help	Display help and exit. Options are grouped by type of operation.
-debug=options, -# options	Write a debugging log. A typical debug_options string is ´d:t:o,file_name´. The default is ´d:t:o,/tmp/myisamchk.trace´. (Available in debug builds only)
-t path, --tmpdir=path	Path for temporary files. Multiple paths can be specified, separated by colon (:) on Unix and semicolon (;) on Windows. They will be used in a round-robin fashion. If not set, the TMPDIR environment variable is used.
-s, --silent	Only print errors. One can use two -s (-ss) to make myisamchk very silent.
-v, --verbose	Print more information. This can be used with --description and --check. Use many -v for more verbosity.
-V, --version	Print version and exit.
-w, --wait	If table is locked, wait instead of returning an error.
--print-defaults	Print the program argument list and exit.
--no-defaults	Don't read default options from any option file.

--defaults-file=filename	Only read default options from the given file *filename*, which can be the full path, or the path relative to the current directory.
--defaults-extra-file=filename	Read the file *filename*, which can be the full path, or the path relative to the current directory, after the global files are read.
--defaults-group-suffix=str	Also read groups with a suffix of *str*. For example, `--defaults-group-suffix=x` would read the groups [myisamchk] and [myisamchk_x]

The following variables can also be set by using *--var_name=value*, for example *--ft_min_word_len=5*

Variable	Default Value
decode_bits	9
ft_max_word_len	version-dependent
ft_min_word_len	4
ft_stopword_file	built-in list
key_buffer_size	1044480
key_cache_block_size	1024
myisam_block_size	1024
myisam_sort_buffer_size	134216704
myisam_sort_key_blocks	16
read_buffer_size	262136
sort_buffer_size	134216704
sort_key_blocks	16
stats_method	nulls_unequal
write_buffer_size	262136

Checking Tables

If no option is provided, myisamchk will perform a check table. It is possible to check MyISAM tables without shutting down or restricting access to the server by using CHECK TABLE instead.

The following check options are available:

Option	Description
-c, --check	Check table for errors. This is the default operation if you specify no option that selects an operation type explicitly.
-e, --extend-check	Check the table VERY throughly. Only use this in extreme cases as it may be slow, and myisamchk should normally be able to find out if the table has errors even without this switch. Increasing the *key_buffer_size* can help speed the process up.
-F, --fast	Check only tables that haven't been closed properly.
-C, --check-only-changed	Check only tables that have changed since last check.
-f, --force	Restart with '-r' (recover) if there are any errors in the table. States will be updated as with '--update-state'.
-i, --information	Print statistics information about the table that is checked.
-m, --medium-check	Faster than *extend-check*, but only finds 99.99% of all errors. Should be good enough for most cases.
-U --update-state	Mark tables as crashed if you find any errors. This should be used to get the full benefit of the *--check-only-changed* option, but you shouldn´t use this option if the mysqld server is using the table and you are running it with external locking disabled.

-T, --read-only	Don't mark table as checked. This is useful if you use myisamchk to check a table that is in use by some other application that does not use locking, such as mysqld when run with external locking disabled.

Repairing Tables

It is also possible to repair MyISAM tables by using REPAIR TABLE.

The following repair options are available, and are applicable when using '-r' or '-o':

Option	Description
-B, --backup	Make a backup of the .MYD file as 'filename-time.BAK'.
--correct-checksum	Correct the checksum information for table.
-D len, --data-file-length=#	Max length of data file (when recreating data file when it's full).
-e, --extend-check	Try to recover every possible row from the data file. Normally this will also find a lot of garbage rows; Don't use this option if you are not totally desperate.
-f, --force	Overwrite old temporary files. Add another --force to avoid 'myisam_sort_buffer_size is too small' errors. In this case we will attempt to do the repair with the given myisam_sort_buffer_size and dynamically allocate as many management buffers as needed.
-k val, --keys-used=#	Specify which keys to update. The value is a bit mask of which keys to use. Each binary bit corresponds to a table index, with the first index being bit 0. *0* disables all index updates, useful for faster inserts. Deactivated indexes can be reactivated by using *myisamchk -r*.
--create-missing-keys	Create missing keys. This assumes that the data file is correct and that the number of rows stored in the index file is correct. Enables *--quick*
--max-record-length=#	Skip rows larger than this if myisamchk can't allocate memory to hold them.
-r, --recover	Can fix almost anything except unique keys that aren't unique (a rare occurrence). Usually this is the best option to try first. Increase *myisam_sort_buffer_size* for better performance.
-n, --sort-recover	Forces recovering with sorting even if the temporary file would be very large.
-p, --parallel-recover	Uses the same technique as '-r' and '-n', but creates all the keys in parallel, in different threads.
-o, --safe-recover	Uses old recovery method; Slower than '-r' but uses less disk space and can handle a couple of cases where '-r' reports that it can't fix the data file. Increase *key_buffer_size* for better performance.
--character-sets-dir=directory_name	Directory where the character sets are installed.
--set-collation=name	Change the collation (and by implication, the character set) used by the index.
-q, --quick	Faster repair by not modifying the data file. One can give a second '-q' to force myisamchk to modify the original datafile in case of duplicate keys. NOTE: Tables where the data file is corrupted can't be fixed with this option.
-u, --unpack	Unpack file packed with myisampack.

Other Actions

Option	Description
-a, --analyze	Analyze distribution of keys. Will make some joins faster as the join optimizer can better choose the order in which to join the tables and which indexes to use. You can check the calculated distribution by using '--description --verbose table_name' or SHOW INDEX FROM table_name.
--stats_method=name	Specifies how index statistics collection code should treat NULLs. Possible values of *name* are "nulls_unequal" (default), "nulls_equal" (emulate MySQL 4.0 behavior), and "nulls_ignored".
-d, --description	Print some descriptive information about the table. Specifying the *--verbose* option once or twice produces additional information.

-A [value], --set-auto-increment[=value]	Force auto_increment to start at this or higher value. If no value is given, then sets the next auto_increment value to the highest used value for the auto key + 1.
-S, --sort-index	Sort the index tree blocks in high-low order. This optimizes seeks and makes table scans that use indexes faster.
-R index_num, --sort-records=#	Sort records according to the given index (as specified by the index number). This makes your data much more localized and may speed up range-based SELECTs and ORDER BYs using this index. It may be VERY slow to do a sort the first time! To see the index numbers, SHOW INDEX displays table indexes in the same order that myisamchk sees them. The first index is 1.
-b offset, --block-search=offset	Find the record to which a block at the given *offset* belongs.

For more, see Memory and Disk Use With myisamchk.

Examples

Check all the MyISAM tables in the current directory:

```
myisamchk *.MYI
```

If you are not in the database directory, you can check all the tables there by specifying the path to the directory:

```
myisamchk /path/to/database_dir/*.MYI
```

Check all tables in all databases by specifying a wildcard with the path to the MariaDB data directory:

```
myisamchk /path/to/datadir/*/*.MYI
```

The recommended way to quickly check all MyISAM tables:

```
myisamchk --silent --fast /path/to/datadir/*/*.MYI
```

Check all MyISAM tables and repair any that are corrupted:

```
myisamchk --silent --force --fast --update-state \
  --key_buffer_size=64M --sort_buffer_size=64M \
  --read_buffer_size=1M --write_buffer_size=1M \
  /path/to/datadir/*/*.MYI
```

1.3.8.2 Memory and Disk Use With myisamchk

myisamchk's performance can be dramatically enhanced for larger tables by making sure that its memory-related variables are set to an optimum level.

By default, myisamchk will use very little memory (about 3MB is allocated), but can temporarily use a lot of disk space. If disk space is a limitation when repairing, the *--safe-recover* option should be used instead of *--recover*. However, if TMPDIR points to a memory file system, an out of memory error can easily be caused, as myisamchk places temporary files in TMPDIR. The *--tmpdir=path* option should be used in this case to specify a directory on disk.

myisamchk has the following requirements for disk space:

- When repairing, space for twice the size of the data file, available in the same directory as the original file. This is for the original file as well as a copy. This space is not required if the *--quick* option is used, in which case only the index file is re-created.
- Disk space in the temporary directory (TMPDIR or the *tmpdir=path* option) is needed for sorting if the *--recover* or *--sort-recover* options are used when not using *--safe-recover*). The space required will be approximately *(largest_key + row_pointer_length) * number_of_rows * 2*. To get information about the length of the keys as well as the row pointer length, use *myisamchk -dv table_name*.
- Space for a new index file to replace the existing one. The old index is first truncated, so unless the old index file is not present or is smaller for some reason, no significant extra space will be needed.

There are a number of system variables that are useful to adjust when running myisamchk. They will increase memory usage, and since some are per-session variables, you don't want to increase the general value, but you can either pass an

increased value to myisamchk as a commandline option, or with a [myisamchk] section in your my.cnf file.

- sort_buffer_size. By default this is 4M, but it's very useful to increase to make myisamchk sorting much faster. Since the server won't be running when you run myisamchk, you can increase substantially. 16M is usually a minimum, but values such as 256M are not uncommon if memory is available.
- key_buffer_size (which particularly helps with the *--extend-check* and *--safe-recover* options.
- read_buffer_size
- write_buffer_size

For example, if you have more than 512MB available to allocate to the process, the following settings could be used:

```
myisamchk
  --myisam_sort_buffer_size=256M
  --key_buffer_size=512M
  --read_buffer_size=64M
  --write_buffer_size=64M
...
```

1.3.8.3 myisamchk Table Information

Contents

myisamchk can be used to obtain information about MyISAM tables, particularly with the *-d*, *-e*, *-i* and *-v* options.

Common options for gathering information include:

- myisamchk -d
- myisamchk -dv
- myisamchk -dvv
- myisamchk -ei
- myisamchk -eiv

The *-d* option returns a short description of the table and its keys. Running the option while the table is being updated, and with external locking disabled, may result in an error, but no damage will be done to the table. Each extra *v* adds more output. *-e* checks the table thoroughly (but slowly), and the *-i* options adds statistical information,

-dvv output

The following table describes the output from the running myisamchk with the *-dvv* option:

Heading	Description
MyISAM file	Name and path of the MyISAM index file (without the extension)
Record format	Storage format. One of *packed* (dynamic), *fixed* or *compressed*.
Chararacter set	Default character set for the table.
File-version	Always 1.
Creation time	Time the data file was created
Recover time	Most recent time the file was reconstructed.
Status	Table status. One or more of *analyzed*, *changed*, *crashed*, *open*, *optimized keys* and *sorted index pages*.
Auto increment key	Index number of the table's auto-increment column. Not shown if no auto-increment column exists.
Last value	Most recently generated auto-increment value. Not shown if no auto-increment column exists.
Data records	Number of records in the table.
Deleted blocks	Number of deleted blocks that are still reserving space. Use OPTIMIZE TABLE to defragment.

Datafile parts	For dynamic tables, the number of data blocks. If the table is optimized, this will match the number of data records.
Deleted data	Number of bytes of unreclaimed deleted data, Use OPTIMIZE TABLE to reclaim the space.
Datafile pointer	Size in bytes of the data file pointer. The size of the data file pointer, in bytes.
Keyfile pointer	Size in bytes of the index file pointer.
Max datafile length	Maximum length, in bytes, that the data file could become.
Max keyfile length	Maximum length, in bytes, that the index file could become.
Recordlength	Space, in bytes, that each row takes.
table description	Description of all indexes in the table, followed by all columns
Key	Index number, starting with one. If not shown, the index is part of a multiple-column index.
Start	Where the index part starts in the row.
Len	Length of the index or index part. The length of a multiple-column index is the sum of the component lengths. Indexes of string columns will be shorter than the full column length if only a string prefix is indexed.
Index	Whether an index value is unique or not. Either *multip.* or *unique*.
Type	Data type of the index of index part.
Rec/key	Record of the number of rows per value for the index or index part. Used by the optimizer to calculate query plans. Can be updated with myisamchk-a. If not present, defaults to *30*.
Root	Root index block address.
Blocksize	Index block size, in bytes.
Field	Column number, starting with one. The first line will contain the position and number of bytes used to store NULL flags, if any (see *Nullpos* and *Nullbit*, below).
Start	Column's byte position within the table row.
Length	Column length, in bytes.
Nullpos	Byte containing the flag for NULL values. Empty if column cannot be NULL.
Nullbit	Bit containing the flag for NULL values. Empty if column cannot be NULL.
Type	Data type - see the table below for a list of possible values.
Huff tree	Only present for packed tables, contains the Huffman tree number associated with the column.
Bits	Only present for packed tables, contains the number of bits used in the Huffman tree.

Data type	Description
constant	All rows contain the same value.
no endspace	No endspace is stored.
no endspace, not_always	No endspace is stored, and endspace compression is not always performed for all values.
no endspace, no empty	No endspace is stored, no empty values are stored.
table-lookup	Column was converted to an ENUM.
zerofill(N)	Most significant *N* bytes of the value are not stored, as they are always zero.
no zeros	Zeros are not stored.
always zero	Zero values are stored with one bit.

-eiv output

The following table describes the output from the running myisamchk with the *-eiv* option:

Heading	Description
Data records	Number of records in the table.
Deleted blocks	Number of deleted blocks that are still reserving space. Use OPTIMIZE TABLE to defragment.
Key	Index number, starting with one.
Keyblocks used	Percentage of the keyblocks that are used. Percentages will be higher for optimized tables.
Packed	Percentage space saved from packing key values with a common suffix.
Max levels	Depth of the b-tree index for the key. Larger tables and longer key values result in higher values.
Records	Number of records in the table.
M.recordlength	Average row length. For fixed rows, will be the actual length of each row.
Packed	Percentage saving from stripping spaces from the end of strings.
Recordspace used	Percentage of the data file used.
Empty space	Percentage of the data file unused.
Blocks/Record	Average number of blocks per record. Values higher than one indicate fragmentation. Use OPTIMIZE TABLE to defragment.
Recordblocks	Number of used blocks. Will match the number of rows for fixed or optimized tables.
Deleteblocks	Number of deleted blocks
Recorddata	Used bytes in the data file.
Deleted data	Unused bytes in the data file.
Lost space	Total bytes lost, such as when a row is updated to a shorter length.
Linkdata	Sum of the bytes used for pointers linking disconnected blocks. Each is four to seven bytes in size.

Examples

```
myisamchk -d /var/lib/mysql/test/posts

MyISAM file:         /var/lib/mysql/test/posts
Record format:       Compressed
Character set:       utf8mb4_unicode_ci (224)
Data records:                1680  Deleted blocks:                0
Recordlength:                2758
Using only keys '0' of 5 possibly keys

table description:
Key Start Len Index   Type
1   1     8   unique  ulonglong
2   2265  80  multip. varchar prefix
    63    80          varchar
    17    5           binary
    1     8           ulonglong
3   1231  8   multip. ulonglong
4   9     8   multip. ulonglong
5   387   764 multip. ? prefix
```

```
myisamchk -dvv /var/lib/mysql/test/posts

MyISAM file:         /var/lib/mysql/test/posts
Record format:       Compressed
Character set:       utf8mb4_unicode_ci (224)
File-version:        1
Creation time:       2015-08-10 16:26:54
Recover time:        2015-08-10 16:26:54
Status:              checked,analyzed,optimized keys
Auto increment key:              1  Last value:                  1811
Checksum:               2299272165
Data records:                 1680  Deleted blocks:                 0
Datafile parts:               1680  Deleted data:                   0
Datafile pointer (bytes):        6  Keyfile pointer (bytes):        6
Datafile length:           4298092  Keyfile length:            156672
Max datafile length: 281474976710654  Max keyfile length: 288230376151710719
Recordlength:                 2758
Using only keys '0' of 5 possibly keys

table description:
Key Start Len Index   Type                     Rec/key         Root  Blocksize
1   1     8   unique  ulonglong                      1                     1024
2   2265  80  multip. varchar prefix               336                     1024
    63    80          varchar                      187
    17    5           binary                         1
    1     8           ulonglong                      1
3   1231  8   multip. ulonglong                     10                     1024
4   9     8   multip. ulonglong                    840                     1024
5   387   764 multip. ? prefix                       1                     4096

Field Start Length Nullpos Nullbit Type                          Huff tree  Bits
1     1     8                     zerofill(6)                            1     9
2     9     8                     zerofill(7)                            1     9
3     17    5                                                            1     9
4     22    5                                                            1     9
5     27    12                    blob                                   2     9
6     39    10                    blob                                   3     9
7     49    4                     always zero                            1     9
8     53    10                    blob                                   1     9
9     63    81                    varchar                                4     9
10    144   81                    varchar                                5     5
11    225   81                    varchar                                5     5
12    306   81                    varchar                                1     9
13    387   802                   varchar                                6     9
14    1189  10                    blob                                   1     9
15    1199  10                    blob                                   7     9
16    1209  5                                                            1     9
17    1214  5                                                            1     9
18    1219  12                    blob                                   1     9
19    1231  8                     no zeros, zerofill(6)                  1     9
20    1239  1022                  varchar                                7     9
21    2261  4                     always zero                            1     9
22    2265  81                    varchar                                8     8
23    2346  402                   varchar                                2     9
24    2748  8                     no zeros, zerofill(7)                  1     9
```

```
myisamchk -eiv /var/lib/mysql/test/posts
Checking MyISAM file: /var/lib/mysql/test/posts
Data records:    1680   Deleted blocks:       0
- check file-size
- check record delete-chain
No recordlinks
- check key delete-chain
block_size 1024:
block_size 2048:
block_size 3072:
block_size 4096:
- check index reference
- check data record references index: 1
Key:  1:  Keyblocks used:  92%  Packed:    0%  Max levels:  2
- check data record references index: 2
Key:  2:  Keyblocks used:  93%  Packed:   90%  Max levels:  2
- check data record references index: 3
Key:  3:  Keyblocks used:  92%  Packed:    0%  Max levels:  2
- check data record references index: 4
Key:  4:  Keyblocks used:  92%  Packed:    0%  Max levels:  2
- check data record references index: 5
Key:  5:  Keyblocks used:  88%  Packed:   97%  Max levels:  2
Total:    Keyblocks used:  91%  Packed:   91%

- check records and index references
Records:              1680    M.recordlength:     4102   Packed:             0%
Recordspace used:      100%   Empty space:           0%  Blocks/Record:   1.00
Record blocks:        1680    Delete blocks:         0
Record data:       6892064    Deleted data:          0
Lost space:           1284    Linkdata:           6264

User time 0.11, System time 0.00
Maximum resident set size 3036, Integral resident set size 0
Non-physical pagefaults 925, Physical pagefaults 0, Swaps 0
Blocks in 0 out 0, Messages in 0 out 0, Signals 0
Voluntary context switches 0, Involuntary context switches 74
```

1.3.8.4 myisamlog

`myisamlog` processes and returns the contents of a MyISAM log file.

Invoke `myisamlog` like this:

```
shell> myisamlog [options] [log_file [tbl_name] ...]
shell> isamlog [options] [log_file [tbl_name] ...]
```

The default operation is update (`-u`). If a recovery is done (`-r`), all writes and possibly updates and deletes are done and errors are only counted. The default log file name is `myisam.log` for `myisamlog` and `isam.log` for `isamlog` if no `log_file` argument is given. If tables are named on the command line, only those tables are updated.

`myisamlog` supports the following options:

Option	Description
`-?` , `-I`	Display a help message and exit.
`-c N`	Execute only `N` commands.
`-f N`	Specify the maximum number of open files.
`-i`	Display extra information before exiting.
`-o offset`	Specify the starting offset.
`-p N`	Remove `N` components from path.
`-r`	Perform a recovery operation.
`-R record_pos_file record_pos`	Specify record position file and record position.

`-u`	Displays update operations.
`-v`	Verbose mode. Print more output about what the program does. This option can be given multiple times (*-vv*, *-vvv*) to produce more and more output.
`-w write_file`	Specify the write file.
`-V`	Display version information.

1.3.8.5 myisampack

Contents

`myisampack` is a tool for compressing MyISAM tables. The resulting tables are read-only, and usually about 40% to 70% smaller. It is run as follows:

```
myisampack [options] file_name [file_name2...]
```

The `file_name` is the `.MYI` index file. The extension can be omitted, although keeping it permits wildcards, such as:

```
myisampack *.MYI
```

...to compress all the files.

`myisampack` compresses each column separately, and, when the resulting data is read, only the individual rows and columns required need to be decompressed, allowing for quicker reading.

Once a table has been packed, use `myisamchk -rq` (the quick and recover options) to rebuild its indexes.

`myisampack` does not support partitioned tables.

> Do not run myisampack if the tables could be updated during the operation, and skip_external_locking has been set.

Options

The following variables can be set while passed as commandline options to `myisampack`, or set with a `[myisampack]` section in your my.cnf file.

Option	Description
`-b`, `--backup`	Make a backup of the table as `table_name.OLD`.
`--character-sets-dir=name`	Directory where character sets are.
`-#`, `--debug[=name]`	Output debug log. Often this is `'d:t:o,filename'`.
`-f`, `--force`	Force packing of table even if it gets bigger or if tempfile exists.
`-j`, `--join=name`	Join all given tables into `'new_table_name'`. All tables **must** have identical layouts.
`-?`, `--help`	Display help and exit.
`-s`, `--silent`	Only write output when an error occurs
`-T`, `--tmpdir=name`	Use temporary directory to store temporary table.
`-t`, `--test`	Don't pack table, only test packing it.
`-v`, `--verbose`	Write info about progress and packing result. Use multiple `-v` flags for more verbosity.
`-V`, `--version`	Output version information and exit.
`-w`, `--wait`	Wait and retry if table is in use.

Uncompressing

To uncompress a table compressed with `myisampack`, use the `myisamchk -u` option.

Examples

```
> myisampack /var/lib/mysql/test/posts
Compressing /var/lib/mysql/test/posts.MYD: (1680 records)
- Calculating statistics
- Compressing file
37.71%
> myisamchk -rq /var/lib/mysql/test/posts
- check record delete-chain
- recovering (with sort) MyISAM-table '/var/lib/mysql/test/posts'
Data records: 1680
- Fixing index 1
- Fixing index 2
```

1.3.8.6 myisam_ftdump

myisam_ftdump is a utility for displaying information about MyISAM FULLTEXT indexes. It will scan and dump the entire index, and can be a lengthy process.

If the server is running, make sure you run a FLUSH TABLES statement first.

Usage

```
myisam_ftdump <table_name> <index_num>
```

The table_name can be specified with or without the .MYI index extension.

The index number refers to the number of the index when the table was defined, starting at zero. For example, take the following table definition:

```
CREATE TABLE IF NOT EXISTS `employees_example` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `first_name` varchar(30) NOT NULL,
  `last_name` varchar(40) NOT NULL,
  `position` varchar(25) NOT NULL,
  `home_address` varchar(50) NOT NULL,
  `home_phone` varchar(12) NOT NULL,
  `employee_code` varchar(25) NOT NULL,
  `bio` text NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `employee_code` (`employee_code`),
  FULLTEXT (`bio`)
) ENGINE=MyISAM;
```

The fulltext index will be `2`. The primary key is index `0`, and the unique key index `1`.

You can use *myisam_ftdump* to generate a list of index entries in order of frequency of occurrence as follows:

```
myisam_ftdump -c mytexttable 1 | sort -r
```

Options

Option	Description
-h, --help	Display help and exit.
-?, --help	Synonym for -h.
-c, --count	Calculate per-word stats (counts and global weights).
-d, --dump	Dump index (incl. data offsets and word weights).
-l, --length	Report length distribution.

-s, --stats	Report global stats.
-v, --verbose	Be verbose.

1.3.9 dbdeployer

dbdeployer is a tool for installing multiple versions of MariaDB and/or MySQL in isolation from each other. It is primarily used for easily testing different server versions. It is written in Go, and is a replacement for MySQL Sandbox.

Visit https://www.dbdeployer.com for details on how to install and use it.

1.3.10 EXPLAIN Analyzer

> The EXPLAIN Analyzer is no longer active.

The EXPLAIN Analyzer was an online tool for analyzing and optionally sharing the output of both EXPLAIN and `EXPLAIN EXTENDED`.

Using the Analyzer

Using the analyzer is very simple.

1. In the mariadb client, run `EXPLAIN` on a query and copy the output. For example:

```
EXPLAIN SELECT * FROM t1 INNER JOIN t2 INNER JOIN t3 WHERE t1.a=t2.a AND t2.a=t3.a;
+------+-------------+-------+------+---------------+------+---------+------+------+---------
---------------------------------------------+
| id   | select_type | table | type | possible_keys | key  | key_len | ref  | rows | Extra
|
+------+-------------+-------+------+---------------+------+---------+------+------+---------
---------------------------------------------+
|    1 | SIMPLE      | t1    | ALL  | NULL          | NULL | NULL    | NULL |    3 |
|
|    1 | SIMPLE      | t2    | ALL  | NULL          | NULL | NULL    | NULL |    3 | Using
where; Using join buffer (flat, BNL join)        |
|    1 | SIMPLE      | t3    | ALL  | NULL          | NULL | NULL    | NULL |    3 | Using
where; Using join buffer (incremental, BNL join) |
+------+-------------+-------+------+---------------+------+---------+------+------+---------
---------------------------------------------+
3 rows in set (0.00 sec)
```

2. Paste the output into the `EXPLAIN` Analyzer input box and click the "Analyze Explain" button.
3. The formatted `EXPLAIN` will be shown. You can now click on various part to get more information about them.

Some Notes:

- As you can see in the example above, you don't need to chop off the query line or the command prompt.
- To save the EXPLAIN, so you can share it, or just for future reference, click the "Save Explain for analysis and sharing" button and then click the "Analyze Explain" button. You will be given a link which leads to your saved `EXPLAIN`. For example, the above explain can be viewed here: https://mariadb.org/explain_analyzer/analyze/
- Some of the elements in the formatted `EXPLAIN` are clickable. Clicking on them will show pop-up help related to that element.

Clients which integrate with the Explain Analyzer

The Analyzer has an API that client programs can use to send EXPLAINs. If you are a client application developer, see the EXPLAIN Analyzer API page for details.

The following clients have support for the EXPLAIN Analyzer built in:

HeidiSQL

HeidiSQL has a button when viewing a query that sends the query to the explain analyzer.

1.3.11 EXPLAIN Analyzer API

The online EXPLAIN Analyzer tool has an open API to allow client applications to send it EXPLAINs.

Sending EXPLAINs to the EXPLAIN Analyzer

To send an EXPLAIN to the EXPLAIN Analyzer, simply POST or GET to the following address:

```
mariadb.org/explain_analyzer/api/1/?raw_explain=EXPLAIN&client=CLIENT
```

Replace "EXPLAIN" with the output of the EXPLAIN command and "CLIENT" with the name of your client.

Client Banner

If you like, you can have a banner promoting your client appear at the bottom of the page. Once you've added support for the EXPLAIN Analyzer to your client application, just send a logo, the name of your client, and what you want the name and logo to link to to bryan AT montyprogram DOT com

1.3.12 innochecksum

Contents

innochecksum is a tool for printing checksums for InnoDB files.

Usage

```
innochecksum [options] file_name
```

Description

It reads an InnoDB tablespace file, calculates the checksum for each page, compares the calculated checksum to the stored checksum, and reports mismatches, which indicate damaged pages. It was originally developed to speed up verifying the integrity of tablespace files after power outages but can also be used after file copies. Because checksum mismatches will cause InnoDB to deliberately shut down a running server, it can be preferable to use innochecksum rather than waiting for a server in production usage to encounter the damaged pages.

Multiple filenames can be specified by a wildcard on non-Windows systems only.

innochecksum works with compressed pages, and also includes options to analyze leaf pages to estimate how fragmented an index is and how much benefit can be gained from defragmentation.

> innochecksum cannot be used on tablespace files that the server already has open. For such files, you should use CHECK TABLE to check tables within the tablespace. If checksum mismatches are found, you would normally restore the tablespace from backup or start the server and attempt to use mariadb-dump to make a backup of the tables within the tablespace.

Options

innochecksum supports the following options. For options that refer to page numbers, the numbers are zero-based.

Option	Description
-a, --allow-mismatches=#	Maximum checksum mismatch allowed before innochecksum terminates. Defaults to 0, which terminates on the first mismatch.
-c, --count	Print a count of the number of pages in the file.

-e num, --end-page=#	End at this page number (0-based).
-?, --help	Displays help and exits.
-I, --info	Synonym for --help.
-f, --leaf	Examine leaf index pages. Until MariaDB 10.2.4, the short code was `-l`, but this was changed to avoid confusion with the `--log` option.
-l fn, --log=fn	Log output to the specified filename `fn`.
-m num, --merge=#	Leaf page count if merge given number of consecutive pages.
-n, --no-check	Ignore the checksum verification. Until MariaDB 10.6, must be used with the `--write` option.
-p num, --page=#	Check only this page number (0-based).
-D, --page-type-dump=name	Dump the page type info for each page in a tablespace.
-S, --page-type-summary	Display a count of each page type in a tablespace
-i, --per-page-details	Print out per-page detail information.
-u, --skip-corrupt	Skip corrupt pages.
-s num, --start-page=#	Start at this page number (0-based).
-C, --strict-check=name	Specify the strict checksum algorithm. One of: `crc32`, `innodb`, `none`. If not specified, validates against `innodb`, `crc32` and `none`. `full_crc32` is not supported. See also innodb_checksum_algorithm. Removed in MariaDB 10.6.0
-v, --verbose	Verbose mode; print a progress indicator every five seconds.
-V, --version	Displays version information and exits.
-w, --write=name	Rewrite the checksum algorithm. One of crc32, innodb, none. An exclusive lock is obtained during use. Use in conjunction with the `-no-check` option to rewrite an invalid checksum. Removed in MariaDB 10.6.0

Examples

Rewriting a crc32 checksum to replace an invalid checksum:

```
innochecksum --no-check --write crc32 tablename.ibd
```

A count of each page type:

```
innochecksum --page-type-summary data/mysql/gtid_slave_pos.ibd

File::data/mysql/gtid_slave_pos.ibd
================PAGE TYPE SUMMARY==============
#PAGE_COUNT PAGE_TYPE
===============================================
       1 Index page
       0 Undo log page
       1 Inode page
       0 Insert buffer free list page
       2 Freshly allocated page
       1 Insert buffer bitmap
       0 System page
       0 Transaction system page
       1 File Space Header
       0 Extent descriptor page
       0 BLOB page
       0 Compressed BLOB page
       0 Page compressed page
       0 Page compressed encrypted page
       0 Other type of page

===============================================
Additional information:
Undo page type: 0 insert, 0 update, 0 other
Undo page state: 0 active, 0 cached, 0 to_free, 0 to_purge, 0 prepared, 0 other
index_id #pages  #leaf_pages #recs_per_page #bytes_per_page
24  1  1  0  0

index_id page_data_bytes_histgram(empty,...,oversized)
24  1 0 0 0 0 0 0 0 0 0 0 0
```

1.3.13 msql2mysql

Description

Initially, the MySQL C API was developed to be very similar to that of the mSQL database system.

Because of this, mSQL programs often can be converted relatively easily for use with MySQL by changing the names of their C API functions.

The `msql2mysql` utility performs the conversion of mSQL C API function calls to their MySQL equivalents.

Warning: `msql2mysql` converts the input file in place, so make a copy of the original before converting it.

Example

```
shell> cp client-prog.c client-prog.c.orig
shell> msql2mysql client-prog.c
client-prog.c converted
```

After conversion, examine `client-prog.c` and make any necessary post-conversion revisions.

`msql2mysql` uses the `replace` utility to make the function name substitutions.

1.3.14 my_print_defaults

`my_print_defaults` displays the options from option groups of option files. It is useful to see which options a particular tool will use.

Output is one option per line, displayed in the form in which they would be specified on the command line.

Usage

```
my_print_defaults [OPTIONS] [groups]
```

Options

Option	Description
`-c , --config-file=name`	Deprecated, please use --defaults-file instead. Name of config file to read; if no extension is given, default extension (e.g., .ini or .cnf) will be added.
`-# , --debug[=#]`	In debug versions, write a debugging log. A typical debug_options string is `d:t:o,file_name` . The default is `d:t:o,/tmp/my_print_defaults.trace` .
`-c, --defaults-file=name`	Like --config-file, except: if first option, then read this file only, do not read global or per-user config files; should be the first option. Removed in MariaDB 10.8.0.
`-e , --defaults-extra-file=name`	Read this file after the global config file and before the config file in the users home directory; should be the first option. Removed in MariaDB 10.8.0.
`-g , --defaults-group-suffix=name`	In addition to the given groups, read also groups with this suffix. Removed in MariaDB 10.8.0.
`-e , --extra-file=name`	Deprecated. Synonym for --defaults-extra-file.
`--mysqld`	Read the same set of groups that the mysqld binary does.
`-n , --no-defaults`	Return an empty string (useful for scripts).
`? , --help`	Display this help message and exit.
`-v , --verbose`	Increase the output level.
`-V , --version`	Output version information and exit.

Examples

```
my_print_defaults --defaults-file=example.cnf client client-server mysql
```

mariadb-check reads from the [mariadb-check] and [client] sections, so the following would display the mariadb-check options.

```
my_print_defaults mariadb-check client
```

1.3.15 mysqladmin

mysqladmin is an administration program for the mysqld daemon.

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadb-admin` is a symlink to `mysqladmin` .

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mariadb-admin` is the name of the script, with `mysqladmin` a symlink .

See mariadb-admin for details.

1.3.16 mariadb-binlog

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadb-binlog` is a symlink to `mysqlbinlog` .

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mariadb-binlog` is the name of the tool, with `mysqlbinlog` a symlink .

`mariadb-binlog` is a utility included with MariaDB for processing binary log and relay log files.

The MariaDB server's binary log is a set of files containing "events" which represent modifications to the contents of a MariaDB database. These events are written in a binary (i.e. non-human-readable) format. The mariadb-binlog utility is used to view these events in plain text.

Using mariadb-binlog

Viewing the binary log with mariadb-binlog.

mariadb-binlog Options

Options supported by mariadb-binlog.

Annotate_rows_log_event

Annotate_rows events accompany row events and describe the query which caused the row event.

mysqlbinlog

Symlink or old name for mariadb-binlog.

There are 3 related questions.

1.3.16.1 Using mariadb-binlog

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadb-binlog` is a symlink to `mysqlbinlog`.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mariadb-binlog` is the name of the tool, with `mysqlbinlog` a symlink.

The MariaDB server's binary log is a set of files containing "events" which represent modifications to the contents of a MariaDB database. These events are written in a binary (i.e. non-human-readable) format. The *mariadb-binlog* utility is used to view these events in plain text.

Run mariadb-binlog from a command-line like this:

```
shell> mariadb-binlog [options] log_file ...
```

See mariadb-binlog Options for details on the available options.

As an example, here is how you could display the contents of a binary log file named "mariadb-bin.000152":

```
shell> mariadb-binlog mariadb-bin.000152
```

The logging format is determined by the value of the binlog_format system variable. If you are using statement-based logging, the output includes the SQL statement, the ID of the server the statement was executed on, a timestamp, and how much time the statement took to execute. If you are using row-based logging the output of an event will not include an SQL statement but will instead output how individual rows were changed.

The output from mariadb-binlog can be used as input to the mariadb client to redo the statements contained in a binary log. This is useful for recovering after a server crash. Here is an example:

```
shell> mariadb-binlog binlog-filenames | mysql -u root -p
```

If you would like to view and possibly edit the file before applying it to your database, use the '-r' flag to redirect the output to a file:

```
shell> mariadb-binlog -r filename binlog-filenames
```

You can then open the file and view it and delete any statements you don't want executed (such as an accidental DROP DATABASE). Once you are satisfied with the contents you can execute it with:

```
shell> mariadb -u root -p < filename
```

Be careful to process multiple log files in a single connection, especially if one or more of them have any `CREATE TEMPORARY TABLE ...` statements. Temporary tables are dropped when the mariadb client terminates, so if you are processing multiple log files one at a time (i.e. multiple connections) and one log file creates a temporary table and then a subsequent log file refers to the table you will get an 'unknown table' error.

To execute multiple logfiles using a single connection, list them all on the mariadb-binlog command line:

```
shell> mariadb-binlog mariadb-bin.000001 mariadb-bin.000002 | mariadb -u root -p
```

If you need to manually edit the binlogs before executing them, combine them all into a single file before processing. Here is an example:

```
shell> mariadb-binlog mariadb-bin.000001 > /tmp/mariadb-bin.sql
shell> mariadb-binlog mariadb-bin.000002 >> /tmp/mariadb-bin.sql
shell> # make any edits
shell> mysql -u root -p -e "source /tmp/mariadb-bin.sql"
```

1.3.16.2 mariadb-binlog Options

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadb-binlog` is a symlink to `mysqlbinlog`.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mariadb-binlog` is the name of the tool, with `mysqlbinlog` a symlink.

Contents

mariadb-binlog is a utility included with MariaDB for processing binary log and relay log files.

Options

The following options are supported by mariadb-binlog. They can be specified on the command line or in option files:

Option	default value	Description	Introduced
`-?`, `--help`		Display a help statement.	
`--base64-output=name` (>= MariaDB 10.6.1, MariaDB 10.5.11)	`auto`	Determine when the output statements should be base64-encoded BINLOG statements. Options (case-insensitive) include `auto`, `unspec`, `never` and `decode-rows`. `never` neither prints base64 encodings nor verbose event data, and will exit on error if a row-based event is found. This option is useful for binlogs that are entirely statement based. `decode-rows` decodes row events into commented SQL statements if the `--verbose` option is also given. It can enhance the debugging experience with large binary log files, as the raw data will be omitted. Unlike *never*, *mariadb-binlog* will not exit with an error if a row event is found. `auto` (synonymous with *unspec*) outputs base64 encoded entries for row-based and format description events; it should be used when ROW-format events are processed for re-executing on the MariaDB server. This behavior is presumed, such that `auto` is the default value when no option specification is provided. The other option values are intended only for debugging or testing purposes because they may produce output that does not include all events in executable form.	
`--base64-output[=name]` (<= MariaDB 10.6.0, MariaDB 10.5.10)	*(No default Value)*	Determine when the output statements should be base64-encoded BINLOG statements. Options (case-insensitive) include `auto`, `unspec`, `always` (deprecated), `never` and `decode-rows`. `never` disables it and works only for binlogs without row-based events; `decode-rows` decodes row events into commented SQL statements if the `--verbose` option is also given; Unlike *never*, *mariadb-binlog* does not exit with an error if a row event is found `auto` or `unspec`, the default, prints base64 only when necessary (i.e., for row-based events and format description events), and is the only safe behavior if you intend to use the output of mariadb-binlog to re-execute binary log file contents. The other option values are intended only for debugging or testing purposes because they may produce output that does not include all events in executable form.; `always` prints base64 whenever possible, and is for debugging only and should not be used in a production system. If this option is not given, the default is `auto`; if it is given with no argument, `always` is used.	
`--binlog-row-event-max-size=val`	4294967040 (4GB)	The maximum size in bytes of a row-based binary log event. Should be a multiple of 256. Minimum 256, maximum 18446744073709547520.	

`--character-sets-dir=name`	*(No default value)*	Directory where the character sets are.	
`-d , --database=name`	*(No default value)*	Output entries from the binary log (local log only) that occur while *name* has been selected as the default database by USE. Only one database can be specified. The effects depend on whether the statement-based or row-based logging format is in use. For statement-based logging, the server will only log statements where the default database is *name*. The default database is set with the USE statement. For row-based logging, the server will log any updates to any tables in the named database, irrespective of the current database. Ignored in `--raw` mode.	
`-# [options] , --debug[=options]`	`d:t:o,/tmp/mariadb-binlog.trace`	In a debug build, write a debugging log. A typical debug options string is `d:t:o,file_name` .	
`--debug-check`	FALSE	Print some debug info at exit..	
`--debug-info`	FALSE	Print some debug info and memory and CPU info at exit.	
`--default-auth=name`		Default authentication client-side plugin to use.	
`--defaults-extra-file=name`		Read the file *name*, which can be the full path, or the path relative to the current directory, after the global files are read.	
`--defaults-file=name`		Only read default options from the given file name , which can be the full path, or the path relative to the current directory.	
`--defaults-group-suffix=str`		Also read groups with a suffix of *str*. For example, since *mariadb-binlog* normally reads the [client] and [mysqlbinlog] groups, `--defaults-group-suffix=x` would cause it to also read the groups [mysqlbinlog_x] and [client_x].	
`-D , --disable-log-bin`	FALSE	Disable binary log. This is useful, if you enabled `--to-last-log` and are sending the output to the same MariaDB server. This way you could avoid an endless loop. You would also like to use it when restoring after a crash to avoid duplication of the statements you already have. NOTE: you will need a SUPER privilege to use this option.	
`--do-domain-ids=name`	*(No default value)*	A list of positive integers, separated by commas, that form a whitelist of domain ids. Any log event with a GTID that originates from a domain id specified in this list is displayed. Cannot be used with `--ignore-domain-ids` . When used with `--ignore-server-ids` or `--do-server-ids` , the result is the intersection between the two datasets	MariaDB 10.9.0
`--do-server-ids=name`	*(No default value)*	A list of positive integers, separated by commas, that form a whitelist of server ids. Any log event originating from a server id specified in this list is displayed. Cannot be used with `--ignore-server-ids` . When used with `--ignore-domain-ids` or `do-domain-ids` , the result is the intersection between the two datasets. Alias for `--server-id` .	MariaDB 10.9.0
`-B , --flashback`	FALSE	Support flashback mode.	MariaDB 10.2.4
`-F , --force-if-open`	TRUE	Force if binlog was not closed properly. Defaults to ON; use `--skip-force-if-open` to disable.	
`-f , --force-read`	FALSE	If *mariadb-binlog* reads a binary log event that it does not recognize, it prints a warning, ignores the event, and continues. Without this option, mariadb-binlog stops if it reads such an event.	
`--gtid-strict-mode`	TRUE	Process binlog according to gtid-strict-mode specification. The start, stop positions are verified to satisfy start < stop comparison condition. Sequence numbers of any gtid domain must comprise monotically growing sequence, Defaults to on; use `--skip-gtid-strict-mode` to disable.	MariaDB 10.8.0
`-H , --hexdump`	FALSE	Augment output with hexadecimal and ASCII event dump.	
`-h , --host=name`	*(No default value)*	Get the binlog from the MariaDB server on the given host.	
`--ignore-domain-ids=name`	*(No default value)*	A list of positive integers, separated by commas, that form a blacklist of domain ids. Any log event with a GTID that originates from a domain id specified in this list is hidden. Cannot be used with `--do-domain-ids` . When used with `--ignore-server-ids` or `--do-server-ids` , the result is the intersection between the two datasets.	MariaDB 10.9.0
`--ignore-server-ids=name`	*(No default value)*	A list of positive integers, separated by commas, that form a blacklist of server ids. Any log event originating from a server id specified in this list is hidden. Cannot be used with `--do-server-ids` . When used with `--ignore-domain-ids` or `--do-domain-ids` , the result is the intersection between the two datasets.	MariaDB 10.9.0
`-l path , --local-load=path`	*(No default value)*	Prepare local temporary files for LOAD DATA INFILE in the specified directory. The temporary files are not automatically removed.	
`--no-defaults`		Don't read default options from any option file	
`-o value , --offset=value`	0	Skip the first *value* entries in the log.	
`--open_files_limit=#`	64	Reserve file descriptors for usage by *mariadb-binlog*.	
`-p[passwd] , --password[=passwd]`	*(No default value)*	Password to connect to the remote server. The password can be omitted allow it be entered from the prompt, or an option file can be used to avoid the security risk of passing a password on the commandline.	
`--plugin-dir=dir_name ,`		Directory for client-side plugins.	
`-P num , --port=num`	0	Port number to use for connection or 0 for default to, in order of preference, my.cnf, $MYSQL_TCP_PORT, /etc/services, built-in default (3306).	

`--position=#`	4	Removed in MariaDB 5.5. Use `--start-position` instead.	
`--print-defaults`		Print the program argument list from all option files and exit.	
`--print-row-count`	TRUE	Print row counts for each row events. (Defaults to on; use `--skip-print-row-count` to disable.)	MariaDB 10.3
`--print-row-event-positions`	TRUE	Print row event positions. Defaults to on; use `--skip-print-row-event-positions` to disable.)	MariaDB 10.3
`print-table-metadata`		Print metadata stored in Table_map_log_event.	MariaDB 10.5.0
`--protocol=name`	*(No default value)*	The protocol of the connection (tcp,socket,pipe,memory).	
`--raw`		Requires -R. Output raw binlog data instead of SQL statements. Output files named after server logs.	MariaDB 10.2.0
`-R`, `--read-from-remote-server`	FALSE	Read binary logs from a remote MariaDB server rather than reading a local log file. Any connection parameter options are ignored unless this option is given as well. These options are `--host`, `--password`, `--port`, `--protocol`, `--socket`, and `--user`. This option requires that the remote server be running. It works only for binary log files on the remote server, not relay log files.	
`-r name`, `--result-file=name`	*(No default value)*	Direct output to a given file. With --raw this is a prefix for the file names.	
`--rewrite-db=name`	*(No default value)*	Updates to a database with a different name than the original. Example: `rewrite-db='from->to'` For events that are binlogged as statements, rewriting the database constitutes changing a statement's default database from `db1` to `db2`. There is no statement analysis or rewrite of any kind, that is, if one specifies "`db1.tbl`" in the statement explicitly, that occurrence won't be changed to "`db2.tbl`". Row-based events are rewritten correctly to use the new database name. Filtering (e.g. with `--database=name`) happens *before* the database rewrites have been performed. If you use this option on the command line and "`>`" has a special meaning to your command interpreter, quote the value (e.g. `--rewrite-db="oldname->newname"`).	
`--server-id=idnum`	0	Extract only binlog entries created by the server having the given id. From MariaDB 10.9.0, alias for --do-server-ids.	
`--set-charset=character_set`	*(No default value)*	Add '`SET NAMES character_set`' to the output to specify the character set to be used for processing log files.	
`--shared-memory-base-name=name`	MYSQL	Shared-memory name to use for Windows connections using shared memory to a local server (started with the `--shared-memory` option). Case-sensitive.	
`-s`, `--short-form`	FALSE	Just show regular queries: no extra info and no row-based events. This is for testing only, and should not be used in production systems. If you want to suppress base64-output, consider using `--base64-output=never` instead.	
`--skip-annotate-row-events`		Skip all Annotate_rows events in the mariadb-binlog output (by default, mariadb-binlog prints Annotate_rows events, if the binary log does contain them).	
`-S`, `--socket=name`	*(No default value)*	For connections to localhost, the Unix socket file to use, or, on Windows, the name of the named pipe to use.	
`--ssl`	`FALSE`	Enables TLS. TLS is also enabled even without setting this option when certain other TLS options are set. Starting with MariaDB 10.2, the `--ssl` option will not enable verifying the server certificate by default. In order to verify the server certificate, the user must specify the `--ssl-verify-server-cert` option.	
`--ssl-ca=name`		Defines a path to a PEM file that should contain one or more X509 certificates for trusted Certificate Authorities (CAs) to use for TLS. This option requires that you use the absolute path, not a relative path. See Secure Connections Overview: Certificate Authorities (CAs) for more information. This option implies the `--ssl` option.	
`--ssl-capath=name`		Defines a path to a directory that contains one or more PEM files that should each contain one X509 certificate for a trusted Certificate Authority (CA) to use for TLS. This option requires that you use the absolute path, not a relative path. The directory specified by this option needs to be run through the `openssl rehash` command. See Secure Connections Overview: Certificate Authorities (CAs) for more information. This option is only supported if the client was built with OpenSSL or yaSSL. If the client was built with GnuTLS or Schannel, then this option is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms. This option implies the `--ssl` option.	
`--ssl-cert=name`		Defines a path to the X509 certificate file to use for TLS. This option requires that you use the absolute path, not a relative path. This option implies the `--ssl` option.	
`--ssl-cipher=name`		List of permitted ciphers or cipher suites to use for TLS. This option implies the `--ssl` option.	

`--ssl-crl=name`		Defines a path to a PEM file that should contain one or more revoked X509 certificates to use for TLS. This option requires that you use the absolute path, not a relative path. See Secure Connections Overview: Certificate Revocation Lists (CRLs) for more information. This option is only supported if the client was built with OpenSSL or Schannel. If the client was built with yaSSL or GnuTLS, then this option is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.	
`--ssl-crlpath=name`		Defines a path to a directory that contains one or more PEM files that should each contain one revoked X509 certificate to use for TLS. This option requires that you use the absolute path, not a relative path. The directory specified by this option needs to be run through the `openssl rehash` command. See Secure Connections Overview: Certificate Revocation Lists (CRLs) for more information. This option is only supported if the client was built with OpenSSL. If the client was built with yaSSL, GnuTLS, or Schannel, then this option is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.	
`--ssl-key=name`		Defines a path to a private key file to use for TLS. This option requires that you use the absolute path, not a relative path. This option implies the `--ssl` option.	
`--ssl-verify-server-cert`	`FALSE`	Enables server certificate verification. This option is disabled by default.	
`--start-datetime=datetime`	*(No default value)*	Start reading the binlog at first event having a datetime equal to or later than the argument; the argument must be a date and time in the local time zone, in any format accepted by the MariaDB server for DATETIME and TIMESTAMP types, for example: 2014-12-25 11:25:56 (you should probably use quotes for your shell to set it properly). This option is useful for point-in-time recovery.	
`-j pos`, `--start-position=pos`	4	Start reading the binlog at this position. Type can either be a positive integer or, from MariaDB 10.8.0, a GTID list. When using a positive integer, the value only applies to the first binlog passed on the command line. In GTID mode, multiple GTIDs can be passed as a comma separated list, where each must have a unique domain id. The list represents the gtid binlog state that the client (another "replica" server) is aware of. Therefore, each GTID is exclusive; only events after a given sequence number will be printed to allow users to receive events after their current state.	
`--stop-datetime=name`	*(No default value)*	Stop reading the binlog at first event having a datetime equal or posterior to the argument; the argument must be a date and time in the local time zone, in any format accepted by the MariaDB server for DATETIME and TIMESTAMP types, for example: 2014-12-25 11:25:56 (you should probably use quotes for your shell to set it properly). Ignored in `--raw` mode.	
`--stop-never`		Wait for more data from the server instead of stopping at the end of the last log. Implies `--to-last-log`.	MariaDB 10.2.0
`--stop-never-slave-server-id`		The slave server_id used for `--read-from-remote-server --stop-never`.	MariaDB 10.2.0
`--stop-position=pos`	18446744073709551615	Stop reading the binlog at this position. Type can either be a positive integer or, from MariaDB 10.8, a GTID list. When using a positive integer, the value only applies to the last binlog passed on the command line. In GTID mode, multiple GTIDs can be passed as a comma separated list, where each must have a unique domain id. Each GTID is inclusive; only events up to the given sequence numbers are printed. Ignored in `--raw` mode.	
`-T`, `--table`		List entries for just this table (affects only row events).	MariaDB 10.2.4
`--tls-version=name`	TLSv1.1,TLSv1.2,TLSv1.3	This option accepts a comma-separated list of TLS protocol versions. A TLS protocol version will only be enabled if it is present in this list. All other TLS protocol versions will not be permitted. See Secure Connections Overview: TLS Protocol Versions for more information.	MariaDB 10.4.6
`-t`, `--to-last-log`	FALSE	Requires `-R` or `--read-from-remote-server`. Will not stop at the end of the requested binlog but rather continue printing until the end of the last binlog of the MariaDB server. If you send the output to the same MariaDB server, that may lead to an endless loop.	
`-u`, `--user=username`	*(No default value)*	Connect to the remote server as username.	
`-v`, `--verbose`		Reconstruct SQL statements out of row events. -v -v adds comments on column data types	
`-V`, `--version`		Print version and exit.	
`--verify-binlog-checksum`		Verify binlog event checksums when reading a binlog file.	

Option Files

In addition to reading options from the command-line, `mariadb-binlog` can also read options from option files. If an unknown option is provided to `mariadb-binlog` in an option file, then it is ignored.

The following options relate to how MariaDB command-line tools handles option files. They must be given as the first argument on the command-line:

Option	Description

`--print-defaults`	Print the program argument list and exit.
`--no-defaults`	Don't read default options from any option file.
`--defaults-file=#`	Only read default options from the given file #.
`--defaults-extra-file=#`	Read this file after the global files are read.
`--defaults-group-suffix=#`	In addition to the default option groups, also read option groups with this suffix.

In MariaDB 10.2 and later, `mariadb-binlog` is linked with MariaDB Connector/C. However, MariaDB Connector/C does not yet handle the parsing of option files for this client. That is still performed by the server option file parsing code. See MDEV-19035 for more information.

Option Groups

`mariadb-binlog` reads options from the following option groups from option files:

Group	Description
`[mysqlbinlog]`	Options read by `mariadb-binlog`, which includes both MariaDB Server and MySQL Server.
`[mariadb-binlog]`	Options read by `mariadb-binlog`. Available starting with MariaDB 10.4.6.
`[client]`	Options read by all MariaDB and MySQL client programs, which includes both MariaDB and MySQL clients. For example, `mysqldump`.
`[client-server]`	Options read by all MariaDB client programs and the MariaDB Server. This is useful for options like socket and port, which is common between the server and the clients.
`[client-mariadb]`	Options read by all MariaDB client programs.

`mariadb-binlog` is linked with MariaDB Connector/C. However, MariaDB Connector/C does not yet handle the parsing of option files for this client. That is still performed by the server option file parsing code. See MDEV-19035 for more information.

1.3.16.3 Annotate_rows_log_event

The terms *master* and *slave* have historically been used in replication, but the terms terms *primary* and *replica* are now preferred. The old terms are used still used in parts of the documentation, and in MariaDB commands, although MariaDB 10.5 has begun the process of renaming. The documentation process is ongoing. See MDEV-18777 to follow progress on this effort.

Contents

`Annotate_rows` events accompany `row` events and describe the query which caused the row event.

Until MariaDB 10.2.4, the binlog event type `Annotate_rows_log_event` was off by default (so as not to change the binary log format and to allow one to replicate MariaDB 5.3 to MySQL/MariaDB 5.1). You can enable this with `--binlog-annotate-row-events`.

In the binary log, each `Annotate_rows` event precedes the corresponding Table map event or the first of the Table map events, if there are more than one (e.g. in a case of multi-delete or insert delayed).

`Annotate_rows` Example

```
master> DROP DATABASE IF EXISTS test;
master> CREATE DATABASE test;
master> USE test;
```

```
master> CREATE TABLE t1(a int);
master> INSERT INTO t1 VALUES (1), (2), (3);
master> CREATE TABLE t2(a int);
master> INSERT INTO t2 VALUES (1), (2), (3);
master> CREATE TABLE t3(a int);
master> INSERT DELAYED INTO t3 VALUES (1), (2), (3);
master> DELETE t1, t2 FROM t1 INNER JOIN t2 INNER JOIN t3
    ->        WHERE t1.a=t2.a AND t2.a=t3.a;

master> SHOW BINLOG EVENTS IN 'master-bin.000001';
+------------------+------+--------------+-----------+-------------+----------------------
-------------------------------------------------------+
| Log_name         | Pos  | Event_type   | Server_id | End_log_pos | Info
|
+------------------+------+--------------+-----------+-------------+----------------------
-------------------------------------------------------+
| master-bin.000001 |    4 | Format_desc  |       100 |         240 | Server ver:
5.5.20-MariaDB-mariadb1~oneiric-log, Binlog ver: 4                |
| master-bin.000001 |  240 | Query        |       100 |         331 | DROP DATABASE IF EXISTS
test                                                  |
| master-bin.000001 |  331 | Query        |       100 |         414 | CREATE DATABASE test
|
| master-bin.000001 |  414 | Query        |       100 |         499 | use `test`; CREATE
TABLE t1(a int)                                               |
| master-bin.000001 |  499 | Query        |       100 |         567 | BEGIN
|
| master-bin.000001 |  567 | Annotate_rows |       100 |         621 | INSERT INTO t1 VALUES
(1), (2), (3)                                            |
| master-bin.000001 |  621 | Table_map    |       100 |         662 | table_id: 16 (test.t1)
|
| master-bin.000001 |  662 | Write_rows   |       100 |         706 | table_id: 16 flags:
STMT_END_F                                                |
| master-bin.000001 |  706 | Query        |       100 |         775 | COMMIT
|
| master-bin.000001 |  775 | Query        |       100 |         860 | use `test`; CREATE
TABLE t2(a int)                                               |
| master-bin.000001 |  860 | Query        |       100 |         928 | BEGIN
|
| master-bin.000001 |  928 | Annotate_rows |       100 |         982 | INSERT INTO t2 VALUES
(1), (2), (3)                                            |
| master-bin.000001 |  982 | Table_map    |       100 |        1023 | table_id: 17 (test.t2)
|
| master-bin.000001 | 1023 | Write_rows   |       100 |        1067 | table_id: 17 flags:
STMT_END_F                                                |
| master-bin.000001 | 1067 | Query        |       100 |        1136 | COMMIT
|
| master-bin.000001 | 1136 | Query        |       100 |        1221 | use `test`; CREATE
TABLE t3(a int)                                               |
| master-bin.000001 | 1221 | Query        |       100 |        1289 | BEGIN
|
| master-bin.000001 | 1289 | Annotate_rows |       100 |        1351 | INSERT DELAYED INTO t3
VALUES (1), (2), (3)                                      |
| master-bin.000001 | 1351 | Table_map    |       100 |        1392 | table_id: 18 (test.t3)
|
| master-bin.000001 | 1392 | Write_rows   |       100 |        1426 | table_id: 18 flags:
STMT_END_F                                                |
| master-bin.000001 | 1426 | Table_map    |       100 |        1467 | table_id: 18 (test.t3)
|
| master-bin.000001 | 1467 | Write_rows   |       100 |        1506 | table_id: 18 flags:
STMT_END_F                                                |
| master-bin.000001 | 1506 | Query        |       100 |        1575 | COMMIT
|
| master-bin.000001 | 1575 | Query        |       100 |        1643 | BEGIN
|
| master-bin.000001 | 1643 | Annotate_rows |       100 |        1748 | DELETE t1, t2 FROM t1
INNER JOIN t2 INNER JOIN t3 WHERE t1.a=t2.a AND t2.a=t3.a |
| master-bin.000001 | 1748 | Table_map    |       100 |        1789 | table_id: 16 (test.t1)
|
| master-bin.000001 | 1789 | Table_map    |       100 |        1830 | table_id: 17 (test.t2)
|
| master-bin.000001 | 1830 | Delete_rows  |       100 |        1874 | table_id: 16
|
| master-bin.000001 | 1874 | Delete_rows  |       100 |        1918 | table_id: 17 flags:
STMT_END_F                                                |
| master-bin.000001 | 1918 | Query        |       100 |        1987 | COMMIT
```

```
|
+------------------+------+--------------+----------+------------+----------------------
------------------------------------------------------+
```

Options Related to Annotate_rows_log_event

The following options have been added to control the behavior of `Annotate_rows_log_event`:

Master Option: `--binlog-annotate-row-events`

This option tells the master to write `Annotate_rows` events to the binary log. See binlog_annotate_row_events for a detailed description of the variable.

Session values allow you to annotate only some selected statements:

```
...
SET SESSION binlog_annotate_row_events=ON;
... statements to be annotated ...
SET SESSION binlog_annotate_row_events=OFF;
... statements not to be annotated ...
```

Slave Option: `--replicate-annotate-row-events`

This option tells the slave to reproduce `Annotate_row` events received from the master in its own binary log (sensible only when used in tandem with the `log-slave-updates` option).

See replicate_annotate_row_events for a detailed description of the variable.

mariadb-binlog Option: `--skip-annotate-row-events`

This option tells mariadb-binlog to skip all `Annotate_row` events in its output (by default, mariadb-binlog prints `Annotate_row` events, if the binary log contains them).

Example of mariadb-binlog Output

```
...> mariadb-binlog.exe -vv -R --user=root --port=3306 --host=localhost master-bin.000001

/*!40019 SET @@session.max_insert_delayed_threads=0*/;
/*!50003 SET @OLD_COMPLETION_TYPE=@@COMPLETION_TYPE,COMPLETION_TYPE=0*/;
DELIMITER /*!*/;
# at 4
#100516 15:36:00 server id 100  end_log_pos 240         Start: binlog v 4, server v 5.1.44-
debug-log created 100516
 15:36:00 at startup
ROLLBACK/*!*/;
BINLOG '
oNjvSw9kAAAA7AAAAPAAAAAAAAQANS4xLjQ0LWRlYnVnLWxvZwAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAACg2O9LEzgNAAgAEgAEBAQEEgAA2QAEGggAAAAICAgCAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAA=
'/*!*/;
# at 240
#100516 15:36:18 server id 100  end_log_pos 331         Query   thread_id=1     exec_time=0
error_code=0
SET TIMESTAMP=1274009778/*!*/;
SET @@session.pseudo_thread_id=1/*!*/;
SET @@session.foreign_key_checks=1, @@session.sql_auto_is_null=1, @@session.unique_checks=1,
@@session.autocommit=1
/*!*/;
SET @@session.sql_mode=0/*!*/;
SET @@session.auto_increment_increment=1, @@session.auto_increment_offset=1/*!*/;
/*!\C latin1 *//*!*/;
SET
@@session.character_set_client=8,@@session.collation_connection=8,@@session.collation_server=8/
*!*/;
SET @@session.lc_time_names=0/*!*/;
```

```
SET @@session.collation_database=DEFAULT/*!*/;
DROP DATABASE IF EXISTS test
/*!*/;
# at 331
#100516 15:36:18 server id 100  end_log_pos 414         Query   thread_id=1     exec_time=0
error_code=0
SET TIMESTAMP=1274009778/*!*/;
CREATE DATABASE test
/*!*/;
# at 414
#100516 15:36:18 server id 100  end_log_pos 499         Query   thread_id=1     exec_time=0
error_code=0
use test/*!*/;
SET TIMESTAMP=1274009778/*!*/;
CREATE TABLE t1(a int)
/*!*/;
# at 499
#100516 15:36:18 server id 100  end_log_pos 567         Query   thread_id=1     exec_time=0
error_code=0
SET TIMESTAMP=1274009778/*!*/;
BEGIN
/*!*/;
# at 567
# at 621
# at 662
#100516 15:36:18 server id 100  end_log_pos 621         Annotate_rows:
#Q> INSERT INTO t1 VALUES (1), (2), (3)
#100516 15:36:18 server id 100  end_log_pos 662         Table_map: `test`.`t1` mapped to number
16
#100516 15:36:18 server id 100  end_log_pos 706         Write_rows: table id 16 flags:
STMT_END_F

BINLOG '
stjvSxNkAAAAKQAAAJYCAAAAABAAAAAAAAAABHRlc3QAAnQxAAEDAAE=
stjvSxdkAAAALAAAAMICAAAQABAAAAAAAAAEAAf/+AQAAAP4CAAAA/gMAAAA=
'/*!*/;
### INSERT INTO test.t1
### SET
###   @1=1 /* INT meta=0 nullable=1 is_null=0 */
### INSERT INTO test.t1
### SET
###   @1=2 /* INT meta=0 nullable=1 is_null=0 */
### INSERT INTO test.t1
### SET
###   @1=3 /* INT meta=0 nullable=1 is_null=0 */
# at 706
#100516 15:36:18 server id 100  end_log_pos 775         Query   thread_id=1     exec_time=0
error_code=0
SET TIMESTAMP=1274009778/*!*/;
COMMIT
/*!*/;
# at 775
#100516 15:36:18 server id 100  end_log_pos 860         Query   thread_id=1     exec_time=0
error_code=0
SET TIMESTAMP=1274009778/*!*/;
CREATE TABLE t2(a int)
/*!*/;
# at 860
#100516 15:36:18 server id 100  end_log_pos 928         Query   thread_id=1     exec_time=0
error_code=0
SET TIMESTAMP=1274009778/*!*/;
BEGIN
/*!*/;
# at 928
# at 982
# at 1023
#100516 15:36:18 server id 100  end_log_pos 982         Annotate_rows:
#Q> INSERT INTO t2 VALUES (1), (2), (3)
#100516 15:36:18 server id 100  end_log_pos 1023        Table_map: `test`.`t2` mapped to number
17
#100516 15:36:18 server id 100  end_log_pos 1067        Write_rows: table id 17 flags:
STMT_END_F

BINLOG '
stjvSxNkAAAAKQAAAP8DAAAAABEAAAAAAAAABHRlc3QAAnQyAAEDAAE=
```

```
stjvSxdkAAAALAAAACsEAAAQABEAAAAAAAAEAAf/+AQAAAP4CAAAA/gMAAAA=
'/*!*/;
### INSERT INTO test.t2
### SET
###   @1=1 /* INT meta=0 nullable=1 is_null=0 */
### INSERT INTO test.t2
### SET
###   @1=2 /* INT meta=0 nullable=1 is_null=0 */
### INSERT INTO test.t2
### SET
###   @1=3 /* INT meta=0 nullable=1 is_null=0 */
# at 1067
#100516 15:36:18 server id 100  end_log_pos 1136        Query   thread_id=1     exec_time=0
error_code=0
SET TIMESTAMP=1274009778/*!*/;
COMMIT
/*!*/;
# at 1136
#100516 15:36:18 server id 100  end_log_pos 1221        Query   thread_id=1     exec_time=0
error_code=0
SET TIMESTAMP=1274009778/*!*/;
CREATE TABLE t3(a int)
/*!*/;
# at 1221
#100516 15:36:18 server id 100  end_log_pos 1289        Query   thread_id=2     exec_time=0
error_code=0
SET TIMESTAMP=1274009778/*!*/;
BEGIN
/*!*/;
# at 1289
# at 1351
# at 1392
#100516 15:36:18 server id 100  end_log_pos 1351        Annotate_rows:
#Q> INSERT DELAYED INTO t3 VALUES (1), (2), (3)
#100516 15:36:18 server id 100  end_log_pos 1392        Table_map: `test`.`t3` mapped to number
18
#100516 15:36:18 server id 100  end_log_pos 1426        Write_rows: table id 18 flags:
STMT_END_F

BINLOG '
stjvSxNkAAAAKQAAAHAFAAAAABIAAAAAAAAAABHRlc3QAAnQzAAEDAAE=
stjvSxdkAAAAIgAAAJIFAAAQABIAAAAAAAEAAf/+AQAAAA==
'/*!*/;
### INSERT INTO test.t3
### SET
###   @1=1 /* INT meta=0 nullable=1 is_null=0 */
# at 1426
# at 1467
#100516 15:36:18 server id 100  end_log_pos 1467        Table_map: `test`.`t3` mapped to number
18
#100516 15:36:18 server id 100  end_log_pos 1506        Write_rows: table id 18 flags:
STMT_END_F

BINLOG '
stjvSxNkAAAAKQAAALsFAAAAABIAAAAAAAAAABHRlc3QAAnQzAAEDAAE=
stjvSxdkAAAAJwAAAOIFAAAQABIAAAAAAAEAAf/+AgAAAP4DAAAA
'/*!*/;
### INSERT INTO test.t3
### SET
###   @1=2 /* INT meta=0 nullable=1 is_null=0 */
### INSERT INTO test.t3
### SET
###   @1=3 /* INT meta=0 nullable=1 is_null=0 */
# at 1506
#100516 15:36:18 server id 100  end_log_pos 1575        Query   thread_id=2     exec_time=0
error_code=0
SET TIMESTAMP=1274009778/*!*/;
COMMIT
/*!*/;
# at 1575
#100516 15:36:18 server id 100  end_log_pos 1643        Query   thread_id=1     exec_time=0
error_code=0
SET TIMESTAMP=1274009778/*!*/;
BEGIN
/*!*/;
```

```
# at 1643
# at 1748
# at 1789
# at 1830
# at 1874
#100516 15:36:18 server id 100  end_log_pos 1748        Annotate_rows:
#Q> DELETE t1, t2 FROM t1 INNER JOIN t2 INNER JOIN t3
#Q>        WHERE t1.a=t2.a AND t2.a=t3.
#100516 15:36:18 server id 100  end_log_pos 1789        Table_map: `test`.`t1` mapped to number
16
#100516 15:36:18 server id 100  end_log_pos 1830        Table_map: `test`.`t2` mapped to number
17
#100516 15:36:18 server id 100  end_log_pos 1874        Delete_rows: table id 16
#100516 15:36:18 server id 100  end_log_pos 1918        Delete_rows: table id 17 flags:
STMT_END_F

BINLOG '
stjvSxNkAAAAKQAAAP0GAAAAABAAAAAAAAAABHRlc3QAAnQxAAEDAAE=
stjvSxNkAAAAKQAAACYHAAAAABEAAAAAAAAABHRlc3QAAnQyAAEDAAE=
stjvSxlkAAAALAAAAFIHAAAAABAAAAAAAAAAf/+AQAAAP4CAAAA/gMAAAA=
### DELETE FROM test.t1
### WHERE
###   @1=1 /* INT meta=0 nullable=1 is_null=0 */
### DELETE FROM test.t1
### WHERE
###   @1=2 /* INT meta=0 nullable=1 is_null=0 */
### DELETE FROM test.t1
### WHERE
###   @1=3 /* INT meta=0 nullable=1 is_null=0 */
stjvSxlkAAAALAAAAH4HAAAQABEAAAAAAAEAAf/+AQAAAP4CAAAA/gMAAAA=
'/*!*/;
### DELETE FROM test.t2
### WHERE
###   @1=1 /* INT meta=0 nullable=1 is_null=0 */
### DELETE FROM test.t2
### WHERE
###   @1=2 /* INT meta=0 nullable=1 is_null=0 */
### DELETE FROM test.t2
### WHERE
###   @1=3 /* INT meta=0 nullable=1 is_null=0 */
# at 1918
#100516 15:36:18 server id 100  end_log_pos 1987        Query   thread_id=1     exec_time=0
error_code=0
SET TIMESTAMP=1274009778/*!*/;
COMMIT
/*!*/;
DELIMITER ;
# End of log file
ROLLBACK /* added by mariadb-binlog */;
/*!50003 SET COMPLETION_TYPE=@OLD_COMPLETION_TYPE*/;
```

1.3.16.4 mysqlbinlog

MariaDB starting with 10.4.6

From MariaDB 10.4.6, mariadb-binlog is a symlink to `mysqlbinlog`.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, mariadb-binlog is the name of the tool, with `mysqlbinlog` a symlink.

See mariadb-binlog for details.

1.3.17 mariadb-stress-test

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadb-stress-test` is a symlink to `mysql-stress-test`, the script for assisting with adding users or databases or changing passwords in MariaDB.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mysql-stress-test` is the symlink, and `mariadb-stress-test` the binary name.

Contents

mariadb-stress-test.pl is a Perl script that performs stress-testing of the MariaDB server. It requires a version of Perl that has been built with threads support.

Syntax

```
mariadb-stress-test.pl [options]
```

Options

Option	Description
`--help`	Display a help message and exit.
`--abort-on-error=N`	Causes the program to abort if an error with severity less than or equal to N was encountered. Set to 1 to abort on any error.
`--check-tests-file`	Periodically check the file that lists the tests to be run. If it has been modified, reread the file. This can be useful if you update the list of tests to be run during a stress test.
`--cleanup`	Force cleanup of the working directory.
`--log-error-details`	Log error details in the global error log file.
`--loop-count=N`	In sequential test mode, the number of loops to execute before exiting.
`--mysqltest=path`	The path name to the mysqltest program.
`--server-database=db_name`	The database to use for the tests. The default is test.
`--server-host=host_name`	he host name of the local host to use for making a TCP/IP connection to the local server. By default, the connection is made to localhost using a Unix socket file.
`--server-logs-dir=path`	This option is required. path is the directory where all client session logs will be stored. Usually this is the shared directory that is associated with the server used for testing.
`--server-password=password`	The password to use when connecting to the server.
`--server-port=port_num`	The TCP/IP port number to use for connecting to the server. The default is 3306.
`--server-socket=file_name`	For connections to localhost, the Unix socket file to use, or, on Windows, the name of the named pipe to use. The default is `/tmp/mysql.sock`.
`--server-user=user_name`	The MariaDB user name to use when connecting to the server. The default is root.
`--sleep-time=N`	The delay in seconds between test executions.
`--stress-basedir=path`	This option is required and specified the path is the working directory for the test run. It is used as the temporary location for result tracking during testing.
`--stress-datadir=path`	The directory of data files to be used during testing. The default location is the data directory under the location given by the `--stress-suite-basedir` option.
`--stress-init-file[=path]`	*file_name* is the location of the file that contains the list of tests to be run once to initialize the database for the testing. If missing, the default file is `stress_init.txt` in the test suite directory.

`--stress-mode=mode`	This option indicates the test order in stress-test mode. The mode value is either `random` to select tests in random order or `seq` to run tests in each thread in the order specified in the test list file. The default mode is `random`.
`--stress-suite-basedir=path`	This option is required and specifies the directory that has the *t* and *r* subdirectories containing the test case and result files. This directory is also the default location of the `stress-test.txt` file that contains the list of tests. (A different location can be specified with the `--stress-tests-file` option.)
`--stress-tests-file[=file_name]`	Use this option to run the stress tests. *file_name* is the location of the file that contains the list of tests. If omitted, the default file is `stress-test.txt` in the stress suite directory. (See `--stress-suite-basedir`.)
`--suite=suite_name`	Run the named test suite. The default name is `main` (the regular test suite located in the `mysql-test` directory).
`--test-count=N`	The number of tests to execute before exiting.
`--test-duration=N`	The duration of stress testing in seconds.
`--threads=N`	The number of threads. The default is 1.
`--verbose`	Verbose mode. Print more information about what the program does

1.3.18 mariadb-test

MariaDB uses **mariadb-test** to test functionality. It is an all-in-one test framework, doing unit, regression, and conformance testing. The framework was inherited from MySQL, but is greatly enhanced, optimized, and extended in MariaDB.

mariadb-test Overview

Overview of mariadb-test.

mariadb-test Auxiliary Files

Besides test and result files, many other files that affect the testing process in mariadb-test

mariadb-test-run.pl Options

Run test cases.

Pausing mariadb-test-run.pl

Working while your computer is busy running mariadb-test-run.pl.

mariadb-test and mariadb-test-embedded

Runs a test case against a MariaDB server, optionally comparing the output with a result file.

New Features for mysqltest in MariaDB

MariaDB added a number of new options and commands to mysqltest.

Debugging MariaDB With a Debugger

If MariaDB is compiled for debugging, you can both use it in a debugger, an...

The Debug Sync Facility

DEBUG_SYNC synchronization points in server code

Code Coverage with dgcov

The dgcov tool helps you check the coverage for new code.

Installing MinIO for Usage With mariadb-test-run

Easiest way to access to Amazon S3 compatible storage.

1.3.18.1 mariadb-test Overview

Contents

The Basics

At its core, mariadb-test is very simple. The client program `mariadb-test` executes a *test file* and compares the produced output with the *result file*. If the files match, the test is passed; otherwise the test has failed. This approach can be used to test any SQL statement, as well as other executables (with the `exec` command).

The complete process of testing is governed and monitored by the *mariadb-test-run.pl* driver script, or *mtr* for short (for convenience, `mtr` is created as a symbolic link to `mariadb-test-run.pl`). The mtr script is responsible for preparing the test environment, creating a list of all tests to run, running them, and producing the report at the end. It can run many tests in parallel, execute tests in an order which minimizes server restarts (as they are slow), run tests in a debugger or under valgrind or strace, and so on.

Test files are located in *suites*. A *suite* is a directory which contains test files, result files, and optional configuration files. The mtr looks for suites in the `mariadb-test/suite` directory, and in the `mariadb-test` subdirectories of plugins and storage engine directories. For example, the following are all valid suite paths:

```
mariadb-test/suite/rpl
```

```
mariadb-test/suite/handler
```

```
storage/example/mariadb-test/demo
```

```
plugin/auth_pam/mariadb-test/pam
```

In almost all cases, the suite directory name is the suite name. A notable historical exception is the *main* suite, which is located directly in the `mariadb-test` directory.

Test files have a `.test` extension and can be placed directly in the suite directory (for example, `mariadb-test/suite/handler/interface.test`) or in the `t` subdirectory (e.g. `mariadb-test/suite/rpl/t/rpl_alter.test` or `mariadb-test/t/grant.test`). Similarly, result files have the `.result` extension and can be placed either in the suite directory or in the `r` subdirectory.

A test file can include other files (with the `source` command). These included files can have any name and may be placed anywhere, but customarily they have a `.inc` extension and are located either in the suite directory or in the `inc` or `include` subdirectories (for example, `mariadb-test/suite/handler/init.inc` or `mariadb-test/include/start_slave.inc`).

Other files which affect testing, while not being tests themselves, are:

- `disabled.def`
- `suite.opt`
- other `*.opt` files
- `my.cnf`
- other `*.cnf` files
- `combinations`
- other `*.combinations` files
- `suite.pm`
- `*.sh` files
- `*.require` files
- `*.rdiff` files
- `valgrind.supp`

See Auxiliary files for details on these.

Overlays

In addition to regular suite directories, mtr supports *overlays*. An *overlay* is a directory with the same name as an existing suite, but which is located in a storage engine or plugin directory. For example, `storage/myisam/mariadb-test/rpl` could be a *myisam* overlay of the *rpl* suite in `mariadb-test/suite/rpl`. And `plugin/daemon_example/mariadb-test/demo` could be a *daemon_example* overlay of the *demo* suite in `storage/example/mariadb-test/demo`. As a special exception, an overlay of the main suite, should be called `main`, as in `storage/pbxt/mariadb-test/main`.

An overlay is like a second transparent layer in a graphics editor. It can obscure, extend, or modify the background image. Also, one may notice that an overlay is very close to a *UnionFS*, but implemented in perl inside mtr.

An overlay can replace almost any file in the overlaid suite, or add new files. For example, if some overlay of the main suite contains a `include/have_innodb.inc` file, then all tests that include it will see and use the overlaid version. Or, an overlay can create a `t/create.opt` file (even though the main suite does not have such a file), and `create.test` will be executed with the specified additional options.

But adding an overlay never affects how the original suite is executed. That is, mtr always executes the original suite as if no overlay was present. And then, additionally, it executes a combined "union" of the overlay and the original suite. When doing that, mtr takes care to avoid re-executing tests that are not changed in the overlay. For example, creating `t/create.opt` in the overlay of the main suite will only cause `create.test` to be executed in the overlay. But creating `suite.opt` affects all tests — and it will cause all tests to be re-executed with the new options.

Combinations

In certain cases it makes sense to run a specific test or a group of tests several times with different server settings. This can be done using so-called *combinations*. Combinations are groups of settings that are used alternatively. A combinations file defines these alternatives using `my.cnf` syntax, for example

```
[row]
binlog-format=row

[stmt]
binlog-format=statement

[mix]
binlog-format=mixed
```

And all tests where this combinations file applies will be run three times: once for the combination called "row", and `--binlog-format=row` on the server command line, once for the "stmt" combination, and once for the "mix" combination.

More than one combinations file may be applicable to a given test file. In this case, mtr will run the test for all possible combinations of the given combinations. A test that uses replication (three combinations as above) and innodb (two combinations - innodb and xtradb), will be run six times.

Sample Output

Typical mtr output looks like this

```
==========================================================================
TEST                                   WORKER RESULT   TIME (ms) or COMMENT
--------------------------------------------------------------------------
rpl.rpl_row_find_row_debug                [ skipped ]  Requires debug build
main-pbxt.connect                         [ skipped ]  No PBXT engine
main-pbxt.mysqlbinlog_row                 [ disabled ]  test expects a non-transactional engine
rpl.rpl_savepoint 'mix,xtradb'            w2 [ pass ]    238
rpl.rpl_stm_innodb 'innodb_plugin,row'    w1 [ skipped ]  Neither MIXED nor STATEMENT binlog form
binlog.binlog_sf 'stmt'                   w2 [ pass ]      7
unit.dbug                                 w2 [ pass ]      1
maria.small_blocksize                     w1 [ pass ]     23
sys_vars.autocommit_func3 'innodb_plugin' w1 [ pass ]      5
sys_vars.autocommit_func3 'xtradb'        w1 [ pass ]      6
main.ipv6                                 w1 [ pass ]    131
...
```

Every test is printed as "suitename.testname", and a suite name may include an overlay name (like in `main-pbxt`). After the test name, mtr prints combinations that were applied to this test, if any.

A similar syntax can be used on the mtr command line to specify what tests to run:

`$ ./mtr innodb`	search for *innodb* test in every suite from the default list, and run all that was found.
`$ ./mtr main.innodb`	run the *innodb* test from the *main* suite
`$ ./mtr main-pbxt.innodb`	run the *innodb* test from the *pbxt* overlay of the *main* suite
`$ ./mtr main-.innodb`	run the *innodb* test from the *main* suite and all its overlays.
`$ ./mtr main.innodb,xtradb`	run the *innodb* test from the *main* suite, only in the *xtradb* combination

Plugin Support

The mtr driver has special support for MariaDB plugins.

First, on startup it copies or symlinks all dynamically-built plugins into `var/plugins`. This allows one to have many plugins loaded at the same time. For example, one can load Federated and InnoDB engines together. Also, mtr creates environment variables for every plugin with the corresponding plugin name. For example, if the InnoDB engine was built, `$HA_INNODB_SO` will be set to `ha_innodb.so` (or `ha_innodb.dll` on Windows). And the test can safely use the corresponding environment variable on all platforms to refer to a plugin file; it will always have the correct platform-dependent extension.

Second, when combining server command-line options (which may come from many different sources) into one long list before starting `mariadbd`, mtr treats `--plugin-load` specially. Normal server semantics is to use the latest value of any particular option on the command line. If one starts the server with, for example, `--port=2000 --port=3000`, the server will use the last value for the port, that is 3000. To allow different `.opt` files to require different plugins, mtr goes through the assembled server command line, and joins all `--plugin-load` options into one. Additionally it removes all empty `--plugin-load` options. For example, suppose a test is affected by three `.opt` files which contain, respectively:

```
--plugin-load=$HA_INNODB_SO
```

```
--plugin-load=$AUTH_PAM_SO
```

```
--plugin-load=$HA_EXAMPLE_SO
```

...and, let's assume the Example engine was not built (`$HA_EXAMPLE_SO` is empty). Then the server will get:

```
--plugin-load=ha_innodb.so:auth_pam.so
```

instead of

```
--plugin-load=ha_innodb.so --plugin-load=auth_pam.so --plugin-load=
```

Third, to allow plugin sources to be simply copied into the `plugin/` or `storage/` directories, and still not affect existing tests (even if new plugins are statically linked into the server), mtr automatically disables all optional plugins on server startup. A plugin is optional if it can be disabled with the corresponding `--skip-XXX` server command-line option. Mandatory plugins, like MyISAM or MEMORY, do not have `--skip-XXX` options (e.g. there is no `--skip-myisam` option). This mtr behavior means that no plugin, statically or dynamically built, has any effect on the server unless it was explicitly enabled. A convenient way to enable a given plugin *XXX* for specific tests is to create a `have_XXX.opt` file which contains the necessary command-line options, and a `have_XXX.inc` file which checks whether a plugin was loaded. Then any test that needs this plugin can source the `have_XXX.inc` file and have the plugin loaded automatically.

mtr communication procedure

`mtr` is first creating the server socket (`master`).

After that, `workers` are created using `fork()` .

For each worker `run_worker()` function is called, which is executing the following:

- creates a new socket to connect to `server_port` obtained from the `master`
- initiate communication with the `master` using `START` command
- `master` sends first test from list of tests supplied by the user and immediately sends command `TESTCASE` to the `worker`
- `worker` gets command `TESTCASE` and processes test case, by calling `run_testcase()` function which

starts(/restarts if needed) the server and sends `TESTRESULT` (in case of restart `WARNINGS` command is issued to the `master` in case some warnings/error logs are found)
- `master` accepts `TESTRESULT` command and run `mtr_report_test()` function which check does the test fail and also generates the new command `TESTCASE` if some new test case exist
- If there is no other test case `master` sends `BYE` command which gets accepted by the `worker` which is properly closing the connection.

1.3.18.2 mariadb-test Auxiliary Files

Contents

The mariadb-test framework utilizes many other files that affect the testing process, in addition to test and result files.

`disabled.def` file

This file can be used to disable certain tests temporarily. For example, if one test fails and you are working on that, you may want to push the changeset that disables the test into the test suite so that other tests won't be disturbed by this failure.

The file contains test names and a comment (that should explain why the test was disabled), separated by a colon. Lines that start with a hash sign (`#`) are ignored. A typical `disabled.def` may look like this (note that a hash sign in the middle of a line does not start a comment):

```
# List of disabled tests
# test name : comment
rpl_redirect : Fails due to bug#49978
events_time_zone  : need to fix the timing
```

During testing, mtr will print disabled tests like this:

```
...
rpl.rpl_redirect              [ disabled ]  Fails due to bug#49978
rpl.events_time_zone          [ disabled ]  need to fix the timing
...
```

This file should be located in the suite directory.

`suite.opt` file

This file lists server options that will be added to the `mariadbd` command line for every test of this suite. It can refer to environment variables with the `$NAME` syntax. Shell meta-characters should be quoted. For example

```
--plugin-load=$AUTH_PAM_SO
--max-connections=40 --net_read_timeout=5
"--replicate-rewrite-db=test->rewrite"
```

Note that options may be put either on one line or on separate lines. It is a good idea to start an option name with the `--loose-` prefix if the server may or may not recognize the option depending on the configuration. An unknown option in the `.opt` file will stop the server from starting, and the test will be aborted.

This file should be located in the suite directory.

other `*.opt` files

For every test or include file `somefile.test` or `somefile.inc`, mtr will look for `somefile.opt`, `somefile-master.opt` and `somefile-slave.opt`. These files have exactly the same syntax as the `suite.opt` above. Options from these files will also be added to the server command line (all servers started for this test, only master, or only slave respectively) for all affected tests, for example, for all tests that include `somefile.inc` directly or indirectly.

A typical usage example is `include/have_blackhole.inc` and `include/have_blackhole.opt`. The latter contains the necessary command-line options to load the Blackhole storage engine, while the former verifies that the engine was really loaded. Any test that needs the Blackhole engine needs only to start from `source include/have_blackhole.inc;` and the engine will be automatically loaded for the test.

`my.cnf` file

This is not the `my.cnf` file that tests from this suite will use, but rather a *template* of it. It will be converted later to an actual `my.cnf`. If a suite contains no `my.cnf` template, a default template, — `include/default_my.cnf` — will be used. Or `suite/rpl/my.cnf` if the test includes `master-slave.inc` (it's one of the few bits of the old MySQL `mysql-test-run` magic that we have not removed yet). Typically a suite template will not contain a complete server configuration, but rather start from

```
!include include/default_my.cnf
```

and then add the necessary modifications.

The syntax of `my.cnf` template is the same of a normal `my.cnf` file, with a few extensions and assumptions. They are:

- For any group with the name `[mysqld.N]`, where **N** is a number, mtr will start one `mysqld` process. Usually one needs to have only `[mysqld.1]` group, and `[mysqld.2]` group for replication tests.
- There can be groups with non-standard names (`[foo]`, `[bar]`, whatever), not used by `mysqld`. The `suite.pm` files (see below) may use them somehow.
- Values can refer to each other using the syntax `@groupname.optionname` — these references be expanded as needed. For example

  ```
  [mysqld.2]
  master-port= @mysqld.1.port
  ```

  it sets the value of the `master-port` in the `[mysqld.2]` group to the value of `port` in the `[mysqld.1]` group.
- An option name may start with a hash sign `#`. In the resulting `my.cnf` it will look like a comment, but it still can be referred to. For example:

```
[example]
#location = localhost:@mysqld.1.port
bar = server:@example.#location/data
```

- There is the `[ENV]` group. It sets values for the environment variables. For example

  ```
  [ENV]
  MASTER_MYPORT = @mysqld.1.port
  ```

  Also, one can refer to values of environment variables via this group:

  ```
  [mysqld.1]
  user = @ENV.LOGNAME
  ```

- There is the `[OPT]` group. It allows to invoke functions and generate values. Currently it contains only one option — `@OPT.port`. Every time this option is referred to in some other group in the `my.cnf` template, a new unique port number is generated. It will not match any other port number used by this test run. For example

  ```
  [ENV]
  SPHINXSEARCH_PORT = @OPT.port
  ```

This file should be located in the suite directory.

other `*.cnf` files

For every test file `somefile.test` (but for not included files) mtr will look for `somefile.cnf` file. If such a file exists, it will be used as a template instead of suite `my.cnf` or a default `include/default_my.cnf` templates.

`combinations` file

The `combinations` file defines few sets of alternative configurations, and every test in this suite will be run many times - once for every configuration. This can be used, for example, to run all replication tests in the *rpl* suite for all three binlog format modes (row, statement, and mixed). A corresponding `combinations` file would look as following:

```
[row]
binlog-format=row

[stmt]
binlog-format=statement

[mix]
binlog-format=mixed
```

It uses `my.cnf` file syntax, with groups (where group names define combination names) and options. But, despite the similarity, it is not a `my.cnf` template, and it cannot use the templating extentions. Instead, options from the `combinations` file are added to the server command line. In this regard, combination file is closer to `suite.opt` file. And just like it, combination file can use environment variables using the `$NAME` syntax.

Not all tests will necessarily run for all combinations. A particular test may require to be run only in one specific combination. For example, in replication, if a test can only be run with the row binlog format, it will have `--binlog-format=row` in one of the `.opt` files. In this case, mtr will notice that server command line already has an option that matches one of the combinations, and will skip all other combinations for this particular test.

The `combinations` file should be located in the suite directory.

other `*.combinations` files

Just like with the `*.opt` files, mtr will use `somefile.combinations` file for any `somefile.test` and `somefile.inc` that is used in testing. These files have exactly the same format as a suite `combinations` file.

This can cause many combination files affecting one test file (if a test includes two `.inc` files, and both of them have corresponding `.combinations` files). In this case, mtr will run the test for all combinations of combinations from both files. In MariaDB 5.5, for example, `rpl_init.inc` adds combinations for row/statement/mixed, and `have_innodb.inc` adds combinations for innodb/xtradb. Thus any replication test that uses innodb will be run six times.

`suite.pm` file

This (optional) file is a perl module. It must declare a package that inherits from `My::Suite`.

This file must normally end with `bless {}` — that is it must return an object of that class. It can also return a string — in this case all tests in the suite will be skipped, with this string being printed as a reason (for example "PBXT engine was not compiled").

A suite class can define the following methods:

- `config_files()`
- `is_default()`
- `list_cases()`
- `servers()`
- `skip_combinations()`
- `start_test()`

A `config_files()` method returns a list of additional config files (besides `my.cnf`), that this suite needs to be created.

For every file it specifies a function that will create it, when given a `My::Config` object. For example:

```
sub config_files {(
    'config.ini' => \&write_ini,
    'new.conf'   => \&do_new
)}
```

A `servers()` method returns a list of processes that needs to be started for this suite. A process is specified as a [regex, hash] pair. The regular expression must match a section in the `my.cnf` template (for example, `qr/mysqld\./` corresponds to all `mysqld` processes), the hash contains these options:

`SORT`	a number. Processes are started in the order of increasing `SORT` values (and stopped in the reverse order). `mysqld` has number 300.
`START`	a function to start a process. It takes two arguments, `My::Config::Group` and `My::Test`. If `START` is undefined a process will not be started.
`WAIT`	a function to wait for the process to be started. It takes `My::Config::Group` as an argument. Internally mtr first invokes `START` for all processes, then `WAIT` for all started processes.

```
sub servers {(
    qr/^foo$/ => { SORT => 200,  # start foo before mysqld
                   START => \&start_foo,
                   WAIT => \&wait_foo }
)}
```

See the *sphinx* suite for a working example.

A `list_cases()` method returns a complete list of tests for this suite. By default it will be the list of files that have `.test` extension, but without the extension. This list will be filtered by mtr, subject to different mtr options (`--big-test`, `--start-from`, etc), the suite object does not have to do it.

A `start_test()` method starts one test process, by default it will be `mariadb-test`. See the *unit* suite for a working example of `list_cases()` and `start_test()` methods.

A `skip_combinations()` method returns a hash that maps file names (where combinations are defined) to a list of combinations that should be skipped. As a special case, it can disable a complete file by using a string instead of a hash. For example

```
sub skip_combinations {(
    'combinations' => [ 'mix', 'rpl' ],
    'inc/many.combinations' => [ 'a', 'bb', 'c' ],
    'windows.inc' => "Not on windows",
)}
```

The last line will cause all tests of this suite that include `windows.inc` to be skipped with the reason being "Not on windows".

An `is_default()` method returns 1 if this particular suite should be run by default, when the `mariadb-test-run.pl` script is run without explicitly specified test suites or test cases.

`*.sh` files

For every test file `sometest.test` mtr looks for `sometest-master.sh` and `sometest-slave.sh`. If either of these files is found, it will be run before the test itself.

`*.require` files

These files are obsolete. Do not use them anymore. If you need to skip a test use the `skip` command instead.

`*.rdiff` files

These files also define what the test result should be. But unlike `*.result` files, they contain a patch that should be applied to one result file to create a new result file. This is very useful when a result of some test in one combination differs slightly from the result of the same test, but in another combination. Or when a result of a test in an overlay differs from the test result in the overlayed suite.

It is quite difficult to edit `.rdiff` files to update them after the test file has changed. But luckily, it is never needed. When a test fails, mtr creates a `.reject` file. Having it, one can create `.rdiff` file as easy as (for example)

```
diff -u main/foo.result main/foo.reject > main/foo,comb.rdiff
or
diff -u main/foo.result main/foo,comb.reject > main/foo,comb.rdiff
```

Some example:

```
diff -u main/innodb_ext_key.result main/innodb_ext_key,off.reject >
main/innodb_ext_key,off.rdiff

diff -u suite/sys_vars/r/sysvars_server_notembedded.result
suite/sys_vars/r/sysvars_server_notembedded,32bit.reject >
suite/sys_vars/r/sysvars_server_notembedded,32bit.rdiff
```

Note: This will also add a timestamp in the .rdiff file, so if you are submitting a patch you could remove it manually. If the same .rdiff file is used for multiple combinations, then it would be good to omit in the header that would identify the combination, to allow git to pack the repository better. Example:

```
--- testname.result
+++ testname.reject
```

Because a combination can be part of the `.result` or `.rdiff` file name, mtr has to look in many different places for a test result. For example, consider a test `foo.test` in the combination pair `aa,bb`, that is run in the overlay *rty* of the suite *qwe*, in other words, for the test that mtr prints as

```
qwe-rty.foo 'aa,bb'                      [ pass ]
```

For this test a result can be in

- either `.rdiff` or `.result` file
- either in the overlay "`rty/`" or in the overlayed suite "`qwe/`"
- with or without combinations in the file name ("`,a`", "`,b`", "`,a,b`", or nothing)

which means any of the following 15 file names can be used:

1. `rty/r/foo,aa,bb.result`
2. `rty/r/foo,aa,bb.rdiff`
3. `qwe/r/foo,aa,bb.result`
4. `qwe/r/foo,aa,bb.rdiff`
5. `rty/r/foo,aa.result`
6. `rty/r/foo,aa.rdiff`
7. `qwe/r/foo,aa.result`
8. `qwe/r/foo,aa.rdiff`
9. `rty/r/foo,bb.result`
10. `rty/r/foo,bb.rdiff`
11. `qwe/r/foo,bb.result`
12. `qwe/r/foo,bb.rdiff`
13. `rty/r/foo.result`
14. `rty/r/foo.rdiff`
15. `qwe/r/foo.result`

They are listed, precisely, in the order of preference, and mtr will walk that list from top to bottom and the first file that is found will be used.

If this found file is a `.rdiff`, mtr continues walking down the list until the first `.result` file is found. A `.rdiff` is applied to that `.result`.

`valgrind.supp` file

This file defines valgrind suppressions, and it is used when mtr is started with a `--valgrind` option.

1.3.18.3 mariadb-test-run.pl Options

Contents

Syntax

```
./mariadb-test-run.pl [ OPTIONS ] [ TESTCASE ]
```

Where the test case can be specified as: `testcase[.test]` Runs the test case named 'testcase' from all suits

```
path-to-testcase
[suite.]testcase[,combination]
```

Examples

`alias main.alias` 'main' is the name of the suite for the 't' directory.

```
rpl.rpl_invoked_features,mix,xtradb_plugin
suite/rpl/t/rpl.rpl_invoked_features
```

Options

Options to Control What Engine/Variation to Run

Option	Description
`--embedded-server`	Use the embedded server, i.e. no mysqld daemons.
`--ps-protocol`	Use the binary protocol between client and server.
`--cursor-protocol`	Use the cursor protocol between client and server (implies `--ps-protocol`).
`--view-protocol`	Create a view to execute all non updating queries.
`--sp-protocol`	Create a stored procedure to execute all queries.
`--compress`	Use the compressed protocol between client and server if both support it.
`--ssl`	If mariadb-test-run.pl is started with the `--ssl` option, it sets up a secure connection for all test cases. In this case, if mysqld does not support TLS, mariadb-test-run.pl exits with an error message: `Couldn´t find support for SSL`.
`--skip-ssl`	Dont start server with support for TLS connections.
`--vs-config`	Visual Studio configuration used to create executables (default: MTR_VS_CONFIG environment variable).
`--parallel=num`	How many parallel tests should be run. Default is `1`, use `--parallel=auto` for auto-setting of *num*.

`--defaults-file=<config template>`	Use fixed config template for all tests.
`--defaults_extra_file=<config template>`	Extra config template to add to all generated configs.
`--combination=<opt>`	Extra options to pass to mysqld. The value should consist of one or more comma-separated mysqld options. This option is similar to `--mysqld` but should be given two or more times. mariadb-test-run.pl executes multiple test runs, using the options for each instance of `--combination` in successive runs. If `--combination` is given only once, it has no effect. For test runs specific to a given test suite, an alternative to the use of `--combination` is to create a combinations file in the suite directory. The file should contain a section of options for each test run.
`--dry-run`	Don't run any tests, print the list of tests that were selected for execution.

Options to Control Directories to Use

Option	Description
`--tmpdir=DIR`	The directory where temporary files are stored (default: ./var/tmp). The environment variable MYSQL_TMP_DIR will be set to the path for this directory, whether it has the default value or has been set explicitly. This may be referred to in tests.
`--vardir=DIR`	The directory where files generated from the test run is stored (default: ./var). Specifying a ramdisk or tmpfs will speed up tests. The environment variable MYSQLTEST_VARDIR will be set to the path for this directory, whether it has the default value or has been set explicitly. This may be referred to in tests.
`--mem`	Run testsuite in "memory" using tmpfs or ramdisk. This can decrease test times significantly, in particular if you would otherwise be running over a remote file system. Attempts to find a suitable location using a builtin list of standard locations for tmpfs (/dev/shm). The option can also be set using environment variable MTR_MEM=[DIR]. If DIR is given, it is added to the beginning of the list of locations to search, so it takes precedence over any built-in locations. Once you have run tests with --mem within a mariadb-testdirectory, a soflink var will have been set up to the temporary directory, and this will be re-used the next time, until the soflink is deleted. Thus, you do not have to repeat the `--mem` option next time.
`--client-bindir=PATH`	Path to the directory where client binaries are located.
`--client-libdir=PATH`	Path to the directory where client libraries are located.

Options to Control What Test Suites or Cases to Run

Option	Description
`--force`	Normally, mariadb-test-run.pl exits if a test case fails. `--force` causes execution to continue regardless of test case failure.
`--with-ndbcluster-only`	Run only tests that include "ndb" in the filename.
`--skip-ndb[cluster]`	Skip all tests that need cluster. Default.
`--do-test=PREFIX or REGEX`	Run test cases with names prefixed with PREFIX or which fulfil the REGEX. For example, `--do-test=testa` matches tests that begin with *testa*, `--do-test=main.testa` matches tests in the main test suite that begin with *testa*, and `--do-test=main.*testa` matches test names that contain *main* followed by *testa* with anything in between. In the latter case, the pattern match is not anchored to the beginning of the test name, so it also matches names such as *xmainytestz*.
`--skip-test=PREFIX or REGEX`	Skip test cases with names prefixed with PREFIX or which fulfil the REGEX. See `-do-test` for examples.
`--start-from=PREFIX`	Sorts the list of names of the test cases to be run, and then starts with the test prefixed with PREFIX, where the prefix may be *suite.testname* or just *testname*.

`--suite[s]=NAME1,..,NAMEN`	Comma separated list of suite names to run. The default, as of MariaDB 10.4.5, is: "main-, archive-, binlog-, binlog_encryption-, csv-, compat/oracle-, encryption-, federated-, funcs_1-, funcs_2-, gcol-, handler-, heap-, innodb-, innodb_fts-, innodb_gis-, innodb_zip-, json-, maria-, mariabackup-, multi_source-, optimizer_unfixed_bugs-, parts-, perfschema-, plugins-, roles-, rpl-, sys_vars-, sql_sequence-, unit-, vcol-, versioning-,period-".
`--skip-rpl`	Skip the replication test cases.
`--big-test`	Allow tests marked as "big" to run. Tests can be thus marked by including the line `--source include/big_test.inc`, and they will only be run if this option is given, or if the environment variable BIG_TEST is set to 1. Repeat this option twice to run only "big" tests. This is typically used for tests that take a very long to run, or that use many resources, so that they are not suitable for running as part of a normal test suite run
`--staging-run`	Run a limited number of tests (no slow tests). Used for running staging trees with valgrind.
`--enable-disabled`	Ignore any *disabled.def* file, and also run tests marked as disabled. Success or failure of those tests will be reported the same way as other tests.
`--print-testcases`	Don't run the tests but print details about all the selected tests, in the order they would be run.
`--skip-test-list=FILE`	Skip the tests listed in FILE. Each line in the file is an entry and should be formatted as: <TESTNAME> : <COMMENT>

Options That Specify Ports

Option	Description
`--[mtr-]port-base=num`	Base for port numbers. Ports from this number to number+9 are reserved. Should be divisible by 10; if not it will be rounded down. May be set with environment variable MTR_PORT_BASE. If this value is set and is not "auto", it overrides build-thread.
`--[mtr-]build-thread=num`	Specify unique number to calculate port number(s) from. Can be set in environment variable MTR_BUILD_THREAD. Set MTR_BUILD_THREAD="auto" to automatically acquire a build thread id that is unique to current host. The more logical `--port-base` is supported as an alternative.

Options For Test Case Authoring

Option	Description
`--record TESTNAME`	(Re)generate the result file for TESTNAME.
`--check-testcases`	Check testcases for side-effects. This is done by checking system state before and after each test case; if there is any difference, a warning to that effect will be written, but the test case will not be marked as failed because of it. This check is enabled by default. Use `--nocheck-testcases` to disable.
`mark-progress`	Log line number and elapsed time to <testname>.progress

Options That Pass On Options

Option	Description
`--mysqld=ARGS`	Specify additional arguments to "mysqld"
`--mysqltest=ARGS`	Specify additional arguments to "mariadb-test". Use additional `--mysqld-env` options to set more than one variable.

Options to Run Test On Running Server

Option	Description

`extern option=value`	Use an already running server. The option/value pair is what is needed by the mariadb client to connect to the server. Each `--extern` option can only take one option/value pair as an argument, so you need to repeat `--extern` for each pair needed. Example: `./mariadb-test-run.pl --extern socket=var/tmp/mysqld.1.sock alias` . Note: If a test case has an .opt file that requires the server to be restarted with specific options, the file will not be used. The test case likely will fail as a result.

Options For Debugging the Product

In `mariadb-test-run.pl` there is a concept of a "debugger". A "debugger" is a tool that `mariadb-test-run.pl` will execute instead of `mariadbd` . This tool will then start `mariadbd` and can control its execution as it wants. The following "debuggers" are supported:

name	Description
`gdb`	GNU debugger
`ddd`	GUI frontend for gdb
`dbx`	https://en.wikipedia.org/wiki/Dbx_(debugger)
`devenv`	Visual Studio debugger
`windbg`	https://en.wikipedia.org/wiki/WinDbg
`lldb`	Debugger from LLVM project
`valgrind`	Detects memory management problems and more
`strace`	syscall tracer
`rr`	"record and replay" — record the program execution and then replay it forward, backward, or in any other direction

Every "debugger" from the list above supports the following set of options (replace XXX below with a debugger name)

Option	Description
`--XXX`	Start `mariadbd` process under a debugger
`--client-XXX`	Start `mariadb-test` process under a debugger
`--boot-XXX`	Before running tests `mariadb-test-run` executes `mariadbd` to bootstrap, prepare the datadir. This options causes this bootstrapping `mariadbd` process to be run under a debugger.
`--manual-XXX`	Don't start anything, instead print the command that the user needs to run to start `mariadbd` under a debugger. Then wait.

Every option from the above accepts an optional argument. It can be used to specify additional command line options to pass to the tool. Or additional commands that the tool will run on startup. Or both. Commands are separated from each other and from options with a semicolon. For example:

```
./mtr 1st --strace
./mtr 1st --client-rr=--chaos
./mtr 1st --manual-gdb='b mysql_parse;r'
./mtr 1st --boot-gdb='--quiet --tui;b mysql_parse;r'
```

Misc Debugging Related Options

Option	Description
`--debug`	Dump trace output for all servers and client programs.
`--debug-common`	Same as `--debug` , but sets the 'd' debug flags to "query,info,error,enter,exit"

`--debug-server`	Use debug version of server, but without turning on tracing.
`--max-save-core`	Limit the number of core files saved (to avoid filling up disks for heavily crashing server). Defaults to 5, set to 0 for no limit. Set its default with MTR_MAX_SAVE_CORE.
`--max-save-datadir`	Limit the number of datadir saved (to avoid filling up disks for heavily crashing server). Defaults to 20, set to 0 for no limit. Set its default with MTR_MAX_SAVE_DATDIR.
`--max-test-fail`	Limit the number of test failurs before aborting the current test run. Defaults to 10, set to 0 for no limit. Set its default with MTR_MAX_TEST_FAIL.

Misc Options

Option	Description
`--user=USER`	User for connecting to mysqld (default: root)
`--comment=STR`	Write STR to the output within lines filled with #, as a form of banner.
`--timer`	Show test case execution time. Use `no-timer` to disable.
`--verbose`	More verbose output(use multiple times for even more)
`--verbose-restart`	Write when and why servers are restarted between test cases.
`--start`	Only initialize and start the servers, using the startup settings for the first specified test case Example: `./mariadb-test-run.pl --start alias & start-dirty` Only start the servers (without initialization) for the first specified test case
`--start-and-exit`	Same as `--start` , but *mariadb-test-run* terminates and leaves just the server running.
`--start-dirty`	This is similar to `--start` , but will skip the database initialization phase and assume that database files are already available. Usually this means you must have run another test first.
`--user-args`	In combination with start* and no test name, drops arguments to mysqld except those specified with `--mysqld` (if any).
`--wait-all`	If `--start` or `--start-dirty` option is used, wait for all servers to exit before finishing the process. Otherwise, it will terminate if one (of several) servers is restarted.
`--fast`	Do not perform controlled shutdown when servers need to be restarted or at the end of the test run. This is equivalent to using `--shutdown-timeout=0` .
`--force-restart`	Always restart servers between tests.
`--parallel=N`	Run tests in N parallel threads (default 1) Use parallel=auto for auto-setting of N.
`--repeat=N`	Run each test N number of times.
`--retry=N`	If a test fails, it is retried up to a maximum of N runs (default 1). Retries are also limited by the maximum number of failures before stopping, set with the `--retry-failure` option. This option has no effect unless `--force` is also used; without it, test execution will terminate after the first failure. The `--retry` and `--retry-failure` options do not affect how many times a test repeated with `--repeat` may fail in total, as each repetition is considered a new test case, which may in turn be retried if it fails.
`--retry-failure=N`	When using the --retry option to retry failed tests, stop when N failures have occured (default 2). Setting it to 0 or 1 effectively turns off retries.
`--reorder`	Reorder tests to get fewer server restarts. This is the default behavior. There is no guarantee that a particular set of tests will always end up in the same order. Use `-no-reorder` to disable.
`--help`	Display help text.
`--testcase-timeout=MINUTES`	Max test case run time in minutes (default 15).

`--suite-timeout=MINUTES`	Max test suite run time in minutes (default 360).
`--shutdown-timeout=SECONDS`	Max number of seconds to wait for server shutdown before killing servers (default 10).
`--warnings`	Scan the log files for warnings and report any suspicious ones; if any are found, the test will be marked as failed. Use `--nowarnings` to turn off.
`--stop-file=file`	If this file is detected, mariadb-test will not start new tests until the file is removed (also MTR_STOP_FILE environment variable).
`--stop-keep-alive=sec`	Works with `--stop-file`, print messages every *sec* seconds when mariadb-test is waiting to remove the file (for buildbot) (also MTR_STOP_KEEP_ALIVE environment variable).
`--sleep=SECONDS`	Passed to mariadb-test; will be used as fixed sleep time.
`--debug-sync-timeout=NUM`	Set default timeout for WAIT_FOR debug sync actions. Disable facility with NUM=0.
`--gcov`	Collect coverage information after the test. The result is a dgcov file per source and header file and a `last_changes.dgcov` file in the vardir with the coverage for the uncommitted changes if any (or the last commit).
`--gprof`	Collect profiling information using the gprof profiling tool.
`--experimental=<file>`	Specify a file that contains a list of test cases that should be displayed with the [exp-fail] code rather than [fail] if they fail. For an example of a file that might be specified via this option, see *mariadb-test/collections/default.experimental*.
`--report-features`	First run a "test" that reports MariaDB features, displaying the output of SHOW ENGINES and SHOW VARIABLES. This can be used to verify that binaries are built with all required features.
`--timestamp`	Print timestamp before each test report line, showing when the test ended.
`--timediff`	Used with `--timestamp`, also print time passed since the previous test started.
`--max-connections=N`	Maximum number of simultaneous server connections that may be used per test. Default is 128. Minimum is 8, maximum is 5120. Corresponds to the same option for mariadb-test.
`--default-myisam`	Set default storage engine to MyISAM for non-innodb tests. This is needed after switching default storage engine to InnoDB.
`--report-times`	Report how much time has been spent on different phases of test execution.
`--stress=ARGS`	Run stress test, providing options to mysql-stress-test.pl. Options are separated by comma.
`xml-report=<file>`	Output jUnit xml file of the results. From MariaDB 10.1.45, MariaDB 10.2.32, MariaDB 10.3.23, MariaDB 10.4.13, MariaDB 10.5.3
`tail-lines=N`	Number of lines of the result to include in a failure report. From MariaDB 10.3.4.

1.3.18.4 Pausing mariadb-test-run.pl

Contents

Sometimes you need to work when your computer is busy running mariadb-test-run.pl. The mariadb-test-run.pl script allows you to stop it temporarily so you can use your computer and then restart the tests when you're ready.

There are two ways to enable this:

1. **Command-line:** The `--stop-file` and `--stop-keep-alive` options.
2. **Environment Variables:** If you are calling mariadb-test-run.pl indirectly (i.e from a script or program such as buildbot) you can set `MTR_STOP_FILE` and `MTR_STOP_KEEP_ALIVE`.

Keep Alive

If you plan on using this feature with other programs, such as buildbot, you should set the

<code>MTR_STOP_KEEP_ALIVE</code> environment variable or the <code>--stop-keep-alive</code> command-line option with a value in seconds. This will make the script print messages to whatever program is calling mariadb-test-run.pl at the interval you set to prevent timeouts.

If you are calling mariadb-test-run.pl directly, you do not need to specify a timeout.

The mariadb-test-run Stop File

The stop file is a temporary file that you create on your system when you want to pause the execution of mariadb-test-run. When enabled via the command-line or environment variable options, mariadb-test-run will periodically check for the existence of the file and if it exists it will stop until the file is no longer present.

Examples

Command-line:

```
mariadb-test-run.pl --stop-file="/path/to/stop/file" --stop-keep-alive=120
```

Environment Variables:

```
export MTR_STOP_FILE="/path/to/stop/file"
export MTR_STOP_KEEP_ALIVE=120
mariadb-test-run.pl
```

1.3.18.5 mariadb-test and mariadb-test-embedded

Contents

The `mariadb-test` program runs a test case against a MariaDB or MySQL server and optionally compares the output with a result file. This program reads input written in a special test language. Typically, you invoke *mariadb-test* via `mariadb-test-run.pl` rather than invoking it directly.

`mariadb-test_embedded` is similar but is built with support for the `libmariadbd` embedded server.

Features of mariadb-test:

- Can send SQL statements to the server for execution
- Can execute external shell commands
- Can test whether the result from an SQL statement or shell command is as expected
- Can connect to one or more standalone mariadbd servers and switch between connections
- Can connect to an embedded server (libmariadbd), if MariaDB is compiled with support for libmariadbd. (In this case, the executable is named mariadb-test_embedded rather than mariadb-test.)

By default, mariadb-test reads the test case on the standard input. To run mariadb-test this way, you normally invoke it like this:

```
shell> mariadb-test **[options] [db_name]** < //test_file//
```

You can also name the test case file with a `--test-file=file_name` option.

The exit value from mariadb-test is 0 for success, 1 for failure, and 62 if it skips the test case (for example, if after checking some preconditions it decides not to run the test).

Options

mariadb-test supports the following options:

Option	Description
`--help , -?`	Display a help message and exit.
`--basedir=dir , -b dir`	The base directory for tests.

`--character-sets-dir=path`	The directory where character sets are installed.
`--compress`, `-C`	Compress all information sent between the client and the server if both support compression.
`--connect-timeout=N`	This can be used to set the MYSQL_OPT_CONNECT_TIMEOUT parameter of mysql_options to change the number of seconds before an unsuccessful connection attempt times out.
`--continue-on-error`	Continue test even if we got an error. This is mostly useful when testing a storage engine to see what from a test file it can execute, or to find all syntax errors in a newly created big test file
`--cursor-protocol`	Use cursors for prepared statements.
`--database=db_name`, `-D db_name`	The default database to use.
`--debug[=debug_options]`, `-#[debug_options]`	Write a debugging log if MariaDB is built with debugging support. The default debug_options value is `d:t:S:i:O,/tmp/mysqltest.trace` on Unix and `d:t:i:O,\mysqld.trace` on Windows.
`--debug-check`	Print some debugging information when the program exits.
`--debug-info`	Print debugging information and memory and CPU usage statistics when the program exits.
`--host=host_name`, `-h host_name`	Connect to the server on the given host.
`--logdir=dir_name`	The directory to use for log files.
`--mark-progress`	Write the line number and elapsed time to *test_file.progress*.
`--max-connect-retries=num`	The maximum number of connection attempts when connecting to server.
`--max-connections=num`	The maximum number of simultaneous server connections per client (that is, per test). If not set, the maximum is 128. Minimum allowed limit is 8, maximum is 5120.
`--no-defaults`	Do not read default options from any option files. If used, this must be the first option.
`--non-blocking-api`	Use the non-blocking client API for communication.
`--overlay-dir=name`	Overlay directory.
`--password[=password]`, `-p[password]`	The password to use when connecting to the server. If you use the short option form (-p), you cannot have a space between the option and the password. If you omit the password value following the `--password` or `-p` option on the command line, you are prompted for one.
`plugin-dir`	Directory for client-side plugins.
`--port=port_num`, `-P port_num`	The TCP/IP port number to use for the connection, or 0 for default to, in order of preference, my.cnf, $MYSQL_TCP_PORT, /etc/services, built-in default (3306).
`--prologue=name`	Include the contents of the given file before processing the contents of the test file. The included file should have the same format as other *mariadb-test* test files. This option has the same effect as putting a --source file_name command as the first line of the test file.
`--protocol=name`	The protocol to use for connection (tcp, socket, pipe, memory).
`--ps-protocol`	Use the prepared-statement protocol for communication.
`--quiet`	Suppress all normal output. This is a synonym for `--silent`.
`--record`, `-r`	Record the output that results from running the test file into the file named by the `--result-file` option, if that option is given. It is an error to use this option without also using `--result-file`.

`--result-file=file_name`, `-R file_name`	This option specifies the file for test case expected results. `--result-file`, together with `--record`, determines how mariadb-test treats the test actual and expected results for a test case: If the test produces no results, mariadb-test exits with an error message to that effect, unless `--result-file` is given and the named file is an empty file. Otherwise, if `--result-file` is not given, mariadb-test sends test results to the standard output. With `--result-file` but not `--record`, mariadb-test reads the expected results from the given file and compares them with the actual results. If the results do not match, mariadb-test writes a reject file in the same directory as the result file, outputs a diff of the two files, and exits with an error. With both `--result-file` and `--record`, mariadb-test updates the given file by writing the actual test results to it.
`--result-format-version=#`	Version of the result file format to use.
`--server-arg=value`, `-A value`	Pass the argument as an argument to the embedded server. For example, `--server-arg=--tmpdir=/tmp` or `--server-arg=--core`. Up to 64 arguments can be given.
`--server-file=file_name`, `-F file_name`	Read arguments for the embedded server from the given file. The file should contain one argument per line.
`--shared-memory-base-name`	Shared-memory name to use for Windows connections using shared memory to a local server (started with the `--shared-memory` option). Case-sensitive.
`--silent`, `-s`	Suppress all normal output.
`--sleep=num`, `-T num`	Cause all sleep commands in the test case file to sleep num seconds. This option does not affect real_sleep commands. An option value of 0 can be used, which effectively disables sleep commands in the test case.
`--socket=path`, `-S path`	The socket file to use when connecting to localhost (which is the default host).
`--sp-protocol`	Execute DML statements within a stored procedure. For every DML statement, mariadb-test creates and invokes a stored procedure that executes the statement rather than executing the statement directly.
`--ssl`	Enable TLS for secure connection (automatically enabled with other flags). Disable with `--skip-ssl`.
`--ssl-ca=name`	CA file in PEM format (check OpenSSL docs, implies `--ssl`).
`--ssl-capath=name`	CA directory (check OpenSSL docs, implies `--ssl`).
`--ssl-cert=name`	X509 cert in PEM format (implies `--ssl`).
`--ssl-cipher=name`	SSL cipher to use (implies `--ssl`).
`--ssl-key=name`	X509 key in PEM format (implies `--ssl`).
`--ssl-crl=name`	Certificate revocation list (implies `--ssl`).
`--ssl-crlpath=name`	Certificate revocation list path (implies `--ssl`).
`--ssl-verify-server-cert`	Verify server's "Common Name" in its cert against hostname used when connecting. This option is disabled by default.
`--suite-dir=name`	Suite directory.
`--tail-lines=nn`	Specify how many lines of the result to include in the output if the test fails because an SQL statement fails. The default is 0, meaning no lines of result printed.
`--test-file=file_name`, `-x file_name`	Read test input from this file. The default is to read from the standard input.

`--timer-file=file_name`, `-m file_name`	If given, the number of microseconds spent running the test will be written to this file. This is used by mariadb-test-run.pl for its reporting.
`--tmpdir=dir_name`, `-t dir_name`	The temporary directory where socket files are created.
`--user=user_name`, `-u user_name`	The user name to use when connecting to the server.
`--verbose`, `-v`	Verbose mode. Print out more information about what the program does.
`--version`, `-V`	Display version information and exit.
`--view-protocol`	Every SELECT statement is wrapped inside a view.
`--wait-longer-for-timeouts`	Wait longer for timeouts. Useful when running under valgrind.

1.3.18.6 New Features for mysqltest in MariaDB

Note that not all MariaDB-enhancements are listed on this page. See mariadb-test and mariadb-test-embedded for a full set of options.

Startup Option --connect-timeout

```
--connect-timeout=N
```

This can be used to set the MYSQL_OPT_CONNECT_TIMEOUT parameter of mysql_options, to change the number of seconds before an unsuccessful connection attempt times out.

Test Commands for Handling Warnings During Prepare Statements

- `enable_prepare_warnings;`
- `disable_prepare_warnings;`

Normally, when running with the prepared statement protocol with warnings enabled and executing a statement that returns a result set (like SELECT), warnings that occur during the execute phase are shown, but warnings that occur during the prepare phase are "not" shown. The reason for this is that some warnings are returned both during prepare and execute; if both copies of warnings were shown, then test cases would show different number of warnings between prepared statement execution and normal execution (where there is no prepare phase).

The `enable_prepare_warnings` command changes this so that warnings from both the prepare and execute phase are shown, regardless of whether the statement produces a result set in the execute phase. The `disable_prepare_warnings` command reverts to the default behaviour.

These commands only have effect when running with the prepared statement protocol (--ps-protocol) *and* with warnings enabled (enable_warnings). Furthermore, they only have effects for statements that return a result set (as for statements without result sets, warnings from are always shown when warnings are enabled).

The `replace_regex` command supports paired delimiters (like in perl, etc). If the first non-space character in the `replace_regex` argument is one of `(`, `[`, `{`, `<`, then the pattern should end with `)`, `]`, `}`, `>` accordingly. The replacement string can use its own pair of delimiters, not necessarily the same as the pattern. If the first non-space character in the `replace_regex` argument is not one of the above, then it should also separate the pattern and the replacement string and it should end the replacement string. Backslash can be used to escape the current terminating character as usual. The examples below demonstrate valid usage of `replace_regex`:

```
--replace_regex (/some/path)</another/path>
--replace_regex !/foo/bar!foobar!
--replace_regex {pat\}tern}/replace\/ment/i
```

1.3.18.7 Debugging MariaDB With a Debugger

Contents

If you have MariaDB compiled for debugging you can both use it in a debugger, like ddd or gdb, and get comprehensive trace files of the execution of MariaDB. The trace files allow you to both see the flow of the code and to see the differences in execution by by comparing two trace files.

Core dumps are also much easier to investigate if they come from a debug binary.

Note that a binary compiled for debugging and tracing is about 10-20% slower than a normal binary. If you just compile a binary for debugging (option `-g` with gcc) the speed difference compared to a normal binary is negligible.

Checking That MariaDB is Compiled For Debugging

Execute:

```
mariadbd --debug --help
```

If you are using MariaDB before 10.5, then you should use `mysqld` instead of `mariadbd`!

If you get an error `unknown option '--debug`, then MariaDB is not compiled for debugging and tracing.

Building MariaDB for Debugging Starting from 5.5

On Unix you need to pass `-DCMAKE_BUILD_TYPE=Debug` to cmake to compile with debug information.

Building MariaDB 5.3 and Older

Here is how you compile with debug on older versions:

Use the scripts in the BUILD directory that will compile MariaDB with most common debug options and plugins, for example:

```
./BUILD/compile-pentium64-debug-max
```

For the most common configurations there exists a fine-tuned script in the BUILD directory.

If you want to use valgrind, a very good memory instrumentation tool and memory overrun checker, you should use

```
./BUILD/compile-pentium64-valgrind-max
```

Some recommended debugging scripts for Intel/AMD are:

```
BUILD/compile-pentium64-debug-max
BUILD/compile-pentium64-valgrind-max
```

This is an example of how to compile MariaDB for debugging in your home directory with MariaDB 5.2.9 as an example:

```
cd ~
mkdir mariadb
cd mariadb
tar xvf mariadb-5.2.9.tar.gz
ln -s mariadb-5.2.9 current
cd current
./BUILD/compile-pentium64-debug-max
```

The last command will produce a debug version of `sql/mysqld`.

Debugging MariaDB From the Source Directory

Creating the MariaDB Database Directory

The following example creates the MariaDB databases in `/data`.

```
./scripts/mariadb-install-db --srcdir=. --datadir=/data
```

Running MariaDB in a Debugger

The following example is using `ddd`, an excellent graphical debugger in Linux. If you don't have `ddd` installed, you can use `gdb` instead.

```
cd sql
ddd ./mariadbd &
```

In `ddd` or `gdb`

```
run --datadir=/data --language=./share/english --gdb
```

You can set the options in your /.my.cnf file so as not to have to repeat them on the `run` line.

If you run `mysqld` with `--debug`, you will get a trace file in /tmp/mysqld.trace that shows what is happening.

Note that you can have different options in the configuration file for each MariaDB version (like having a specific language directory).

Debugging MariaDB Server with mariadb-test-run

If you get a crash while running `mariadb-test-run` you can debug this in a debugger by using one of the following options:

```
mariadb-test-run --gdb failing-test-name
```

or if you prefer the `ddd` debugger:

```
mariadb-test-run --ddd failing-test-name
```

Sample .my.cnf file to Make Debugging Easier

```
[client-server]
socket=/tmp/mysql-dbug.sock
port=3307

[mariadb]
datadir=/my/data
loose-innodb_file_per_table
server_id= 1
log-basename=master
loose-debug-mutex-deadlock-detector
max-connections=20
lc-messages=en_us

[mariadb-10.0]
lc-messages-dir=/my/maria-10.0/sql/share

[mariadb-10.1]
lc-messages-dir=/my/maria-10.1/sql/share

[mariadb-10.2]
lc-messages-dir=/my/maria-10.2/sql/share

[mariadb-10.3]
lc-messages-dir=/my/maria-10.3/sql/share
```

The above `.my.cnf` file:

- Uses an explicit socket for both client and server.
- Assumes the server source is in /my/maria-xxx. You should change this to point to where your sources are located.
- Has a unique patch for each MariaDB version so that one doesn't have to specify --lc-messages-dir or --language even if one switches between debugging different MariaDB versions.

1.3.18.8 The Debug Sync Facility

Contents

The Debug Sync Facility allows placement of synchronization points in the server code by using the DEBUG_SYNC macro:

```
open_tables(...)

DEBUG_SYNC(thd, "after_open_tables");

lock_tables(...)
```

When activated, a sync point can

- Emit a signal and/or
- Wait for a signal

Nomenclature	Description
signal	A value of a global variable that persists until overwritten by a new signal. The global variable can also be seen as a "signal post" or "flag mast". Then the signal is what is attached to the "signal post" or "flag mast".
emit a signal	Assign the value (the signal) to the global variable ("set a flag") and broadcast a global condition to wake those waiting for a signal.
wait for a signal	Loop over waiting for the global condition until the global value matches the wait-for signal.

By default, all sync points are inactive. They do nothing (except to burn a couple of CPU cycles for checking if they are active).

A sync point becomes active when an action is requested for it. To do so, put a line like this in the test case file:

```
SET DEBUG_SYNC= 'after_open_tables SIGNAL opened WAIT_FOR flushed';
```

This activates the sync point `'after_open_tables'`. It requests it to emit the signal `'opened'` and wait for another thread to emit the signal `'flushed'` when the thread's execution runs through the sync point.

For every sync point there can be one action per thread only. Every thread can request multiple actions, but only one per sync point. In other words, a thread can activate multiple sync points.

Here is an example how to activate and use the sync points:

```
--connection conn1
SET DEBUG_SYNC= 'after_open_tables SIGNAL opened WAIT_FOR flushed';
send INSERT INTO t1 VALUES(1);
    --connection conn2
    SET DEBUG_SYNC= 'now WAIT_FOR opened';
    SET DEBUG_SYNC= 'after_abort_locks SIGNAL flushed';
    FLUSH TABLE t1;
```

When `conn1` runs through the `INSERT` statement, it hits the sync point `'after_open_tables'`. It notices that it is active and executes its action. It emits the signal `'opened'` and waits for another thread to emit the signal `'flushed'`.

`conn2` waits immediately at the special sync point `'now'` for another thread to emit the `'opened'` signal.

A signal remains in effect until it is overwritten. If `conn1` signals `'opened'` before `conn2` reaches `'now'`, `conn2` will still find the `'opened'` signal. It does not wait in this case.

When `conn2` reaches `'after_abort_locks'`, it signals `'flushed'`, which lets `conn1` awake.

Normally the activation of a sync point is cleared when it has been executed. Sometimes it is necessary to keep the sync point active for another execution. You can add an execute count to the action:

```
SET DEBUG_SYNC= 'name SIGNAL sig EXECUTE 3';
```

This sets the signal point's activation counter to 3. Each execution decrements the counter. After the third execution the sync point becomes inactive.

One of the primary goals of this facility is to eliminate sleeps from the test suite. In most cases it should be possible to rewrite test cases so that they do not need to sleep. (But this facility cannot synchronize multiple processes.) However, to support test development, and as a last resort, sync point waiting times out. There is a default timeout, but it can be overridden:

```
SET DEBUG_SYNC= 'name WAIT_FOR sig TIMEOUT 10 EXECUTE 2';
```

`TIMEOUT 0` is special: If the signal is not present, the wait times out immediately.

When a wait timed out (even on `TIMEOUT 0`), a warning is generated so that it shows up in the test result.

You can throw an error message and kill the query when a synchronization point is hit a certain number of times:

```
SET DEBUG_SYNC= 'name HIT_LIMIT 3';
```

Or combine it with signal and/or wait:

```
SET DEBUG_SYNC= 'name SIGNAL sig EXECUTE 2 HIT_LIMIT 3';
```

Here the first two hits emit the signal, the third hit returns the error message and kills the query.

For cases where you are not sure that an action is taken and thus cleared in any case, you can force to clear (deactivate) a sync point:

```
SET DEBUG_SYNC= 'name CLEAR';
```

If you want to clear all actions and clear the global signal, use:

```
SET DEBUG_SYNC= 'RESET';
```

This is the only way to reset the global signal to an empty string.

For testing of the facility itself you can execute a sync point just as if it had been hit:

```
SET DEBUG_SYNC= 'name TEST';
```

Formal Syntax

The string to "assign" to the DEBUG_SYNC variable can contain:

```
RESET |
<sync point name> TEST |
<sync point name> CLEAR |
<sync point name> {{SIGNAL <signal name> |
                   WAIT_FOR <signal name> [TIMEOUT <seconds>]}
                   [EXECUTE <count>] &| HIT_LIMIT <count>}
```

Here '&|' means 'and/or'. This means that one of the sections separated by '&|' must be present or both of them.

Activation/Deactivation

With a MariaDB for debug build, it can be enabled by a mysqld command line option:

```
 --debug-sync-timeout[=default_wait_timeout_value_in_seconds]
```

`'default_wait_timeout_value_in_seconds'` is the default timeout for the `WAIT_FOR` action. If set to zero, the facility

stays disabled.

The facility is enabled by default in the test suite, but can be disabled with:

```
mariadb-test-run.pl ... --debug-sync-timeout=0 ...
```

Likewise the default wait timeout can be set:

```
mariadb-test-run.pl ... --debug-sync-timeout=10 ...
```

The command line option influences the readable value of the debug_sync system variable.

- If the facility is not compiled in, the system variable does not exist.
- If `--debug-sync-timeout=0` the value of the variable reads as `"OFF"`.
- Otherwise the value reads as `"ON - current signal: "` followed by the current signal string, which can be empty.

The readable variable value is the same, regardless if read as a global or session value.

Setting the debug_sync system variable requires the `'SUPER'` privilege. You can never read back the string that you assigned to the variable, unless you assign the value that the variable already has. But that would give a parse error. A syntactically correct string is parsed into a debug sync action and stored apart from the variable value.

Implementation

Pseudo code for a sync point:

```
#define DEBUG_SYNC(thd, sync_point_name)
        if (unlikely(opt_debug_sync_timeout))
          debug_sync(thd, STRING_WITH_LEN(sync_point_name))
```

The sync point performs a binary search in a sorted array of actions for this thread.

The `SET DEBUG_SYNC` statement adds a requested action to the array or overwrites an existing action for the same sync point. When it adds a new action, the array is sorted again.

A typical synchronization pattern

There are quite a few places in MariaDB and MySQL where we use a synchronization pattern like this:

```
mysql_mutex_lock(&mutex);
thd->enter_cond(&condition_variable, &mutex, new_message);
#if defined(ENABLE_DEBUG_SYNC)
if (!thd->killed && !end_of_wait_condition)
   DEBUG_SYNC(thd, "sync_point_name");
#endif
while (!thd->killed && !end_of_wait_condition)
  mysql_cond_wait(&condition_variable, &mutex);
thd->exit_cond(old_message);
```

Here are some explanations:

`thd->enter_cond()` is used to register the condition variable and the mutex in `thd->mysys_var`. This is done to allow the thread to be interrupted (killed) from its sleep. Another thread can find the condition variable to signal and mutex to use for synchronization in this thread's `THD::mysys_var`.

`thd->enter_cond()` requires the mutex to be acquired in advance.

`thd->exit_cond()` unregisters the condition variable and mutex and releases the mutex.

If you want to have a Debug Sync point with the wait, please place it behind `enter_cond()`. Only then you can safely decide, if the wait will be taken. Also you will have `THD::proc_info` correct when the sync point emits a signal. `DEBUG_SYNC` sets its own proc_info, but restores the previous one before releasing its internal mutex. As soon as another thread sees the signal, it does also see the proc_info from before entering the sync point. In this case it will be "new_message", which is associated with the wait that is to be synchronized.

In the example above, the wait condition is repeated before the sync point. This is done to skip the sync point, if no wait takes place. The sync point is before the loop (not inside the loop) to have it hit once only. It is possible that the condition variable is signaled multiple times without the wait condition to be true.

A bit off-topic: At some places, the loop is taken around the whole synchronization pattern:

```
while (!thd->killed && !end_of_wait_condition)
{
  mysql_mutex_lock(&mutex);
  thd->enter_cond(&condition_variable, &mutex, new_message);
  if (!thd->killed [&& !end_of_wait_condition])
  {
    [DEBUG_SYNC(thd, "sync_point_name");]
    mysql_cond_wait(&condition_variable, &mutex);
  }
  thd->exit_cond(old_message);
}
```

Note that it is important to repeat the test for thd->killed after `enter_cond()` . Otherwise the killing thread may kill this thread after it tested `thd->killed` in the loop condition and before it registered the condition variable and mutex in `enter_cond()` . In this case, the killing thread does not know that this thread is going to wait on a condition variable. It would just set `THD::killed` . But if we would not test it again, we would go asleep though we are killed. If the killing thread would kill us when we are after the second test, but still before sleeping, we hold the mutex, which is registered in mysys_var. The killing thread would try to acquire the mutex before signaling the condition variable. Since the mutex is only released implicitly in `mysql_cond_wait()` , the signaling happens at the right place. We have a safe synchronization.

Co-work with the DBUG facility

When running the MariaDB test suite with the `--debug-dbug` command line option, the Debug Sync Facility writes trace messages to the DBUG trace. The following shell commands proved very useful in extracting relevant information:

```
egrep 'query:|debug_sync_exec:' mysql-test/var/log/mysqld.1.trace
```

It shows all executed SQL statements and all actions executed by synchronization points.

Sometimes it is also useful to see, which synchronization points have been run through (hit) with or without executing actions. Then add `"|debug_sync_point:"` to the egrep pattern.

Synchronizing DEBUG_SYNC Actions

Tests may need additional synchronization mechanisms between `DEBUG_SYNC` actions, because certain combinations of actions can result in lost signals. More specifically, once a `SIGNAL` action is issued, it is stored in a global variable for any waiting threads to determine if they are depending on that signal for continuing. However, if a subsequent action overwrites that variable before a waiting thread is able to check against it, the original signal is lost. Examples of actions which would change the variable state are another `SIGNAL` or a `RESET` . Therefore, before issuing these commands, the test writer should verify the previous signal has been acknowledged. The following code snippets show an example of a problematic pattern and a potential solution.

```
SET DEBUG_SYNC='now SIGNAL sig';
SET DEBUG_SYNC='RESET'; # Problematic because sig can be cleared before a waiting thread can
acknowledge it
```

```
SET DEBUG_SYNC='now SIGNAL sig';

# Don't issue the RESET until we have proven the waiting thread has received the signal
let $wait_condition= select count(*)=0 from information_schema.processlist where state like
"debug sync point%";
source include/wait_condition.inc;

SET DEBUG_SYNC='RESET'; # Now this is safe
```

1.3.18.9 Code Coverage with dgcov

Contents

The dgcov tool helps you check the coverage for new code. The dgcov.pl script is part of the mariadb-test framework (and any packages that include mariadb-test).

Overview

The dgcov program runs gcov for code coverage analysis, aggregates the coverage data, and (optionally) reports coverage only for those lines that are changed by the commit(s). Commits are specified in the `git diff` format.

If no commits are specified, the default is to work on all uncommitted changes, if any, otherwise on the last commit (in other words, on `git diff HEAD` or `git diff HEAD^`).

It's recommended that a developer runs dgcov on their new code before pushing it into a MariaDB repository.

Usage

```
./dgcov.pl --help
./dgcov.pl [options] [<commit> [<commit>]]
```

Options and Variables

Short Option	Long Option	Description
`-h`	`--help`	Print help and exit
`-v`	`--verbose`	Show commands run.
`-p`	`--purge`	Delete all test coverage information, to prepare for a new coverage test.
`-o`	`--only-gcov`	Stop after running gcov, don't run git
`-s`	`--skip-gcov`	Do not run gcov, assume .gcov files are already in place
`-g`	`--generate`	Create .dgcov files for all source files

How to Prepare the Code for dgcov

Prior to running this tool, MariaDB should be built with

```
  cmake -DENABLE_GCOV=ON
```

and the testsuite should be run. dgcov will report the coverage for all lines modified in the specified commits.

Output

Output .dgcov files have a conventional gcov format: lines not covered are prefixed with `#####` , lines without generated code are prefixed with `-` , and other lines are prefixed with the number of times they were executed. See `info gcov` for more information.

The patch-like coverage for commits uses gcov format (as above) for lines, changed in these commits, and no prefix at all for lines that were not changed.

Examples

Checking the coverage for all unpushed commits:

```
dgcov.pl @{u} HEAD
```

Checking the coverate for all uncommitted changes:

```
dgcov.pl HEAD
```

Checking the coverage for a specific commit 1234567:

```
dgcov.pl 1234567^ 1234567
```

mariadb-test-run can invoke dgcov automatically:

```
./mtr --gcov
```

in the latter case the coverate for the uncommitted changes (or the last commit) will be not printed to the stdout, but will be put into `var/last_changes.dgcov` file.

Caveats

Note that to be able to run gcov with the mariadb-test framework you need to have gcc version 4.8 or newer.

References

- dgcov was created by Kristian Nielsen and was first announced here.
- dgcov was re-implemented to aggregate the data and to work for git and cmake by Sergei Golubchik.

1.3.18.10 Installing MinIO for Usage With mariadb-test-run

When testing the S3 storage engine with the s3 test suite, mariadb-test-run needs access to Amazon S3 compatible storage.

The easiest way to achieve this is to install MinIO, an open source S3 compatible storage.

Here is a shell script that you can use to install MinIO with the right credentials for mariadb-test-run. This should work on most Linux systems as the binaries are statically linked. You can alternatively download MinIO binaries directly from here.

```
# Where to install the MinIO binaries and where to store the data
install=/my/local/minio
data=/tmp/shared

# Get the MinIO binaries. You can skip this test if you already have MinIO installed.
mkdir -p $install
wget https://dl.min.io/server/minio/release/linux-amd64/minio -O $install/minio
wget https://dl.min.io/client/mc/release/linux-amd64/mc -O $install/mc
chmod a+x $install/minio $install/mc

# Setup MinIO for usage with mariadb-test-run
MINIO_ACCESS_KEY=minio MINIO_SECRET_KEY=minioadmin $install/minio server $data 2>&1 &
$install/mc config host add local http://127.0.0.1:9000 minio minioadmin
$install/mc mb --ignore-existing local/storage-engine
```

Now you can run the S3 test suite:

```
cd "mysql-source-dir"/mariadb-test
./mariadb-test-run --suite=s3
```

If there is an issue while running the test suite, you can see the files created by MinIO with:

```
$install/mc ls -r local/storage-engine
```

or

```
ls $data/storage-engine
```

If you want to use MinIO with different credentials or you want to run the test against another S3 storage you ave to update the update the following files:

```
mariadb-test/suite/s3/my.cnf
mariadb-test/suite/s3/slave.cnf
```

1.3.19 perror

Contents

perror is a utility that displays descriptions for system or storage engine error codes.

See MariaDB Error Codes for a full list of MariaDB error codes, and Operating System Error Codes for a list of Linux and Windows error codes.

Usage

```
perror [OPTIONS] [ERRORCODE [ERRORCODE...]]
```

If you need to describe a negative error code, use `--` before the first error code to end the options.

Options

Option	Description
`-?`, `--help`	Display help and exit.
`-I`, `--info`	Synonym for `--help`.
`-s`, `--silent`	Only print the error message.
`-v`, `--verbose`	Print error code and message (default). (Defaults to on; use `--skip-verbose` to disable.)
`-V`, `--version`	Displays version information and exits.

Examples

System error code:

```
shell> perror 96
OS error code  96:  Protocol family not supported
```

MariaDB/MySQL error code:

```
shell> perror 1005 1006
MySQL error code 1005 (ER_CANT_CREATE_TABLE): Can't create table %`s.%`s (errno: %M)
MySQL error code 1006 (ER_CANT_CREATE_DB): Can't create database '%-.192s' (errno: %M)
```

```
shell> perror --silent 1979
You are not owner of query %lu
```

1.3.20 replace Utility

Description

The replace utility program changes strings in place in files or on the standard input. Invoke replace in one of the following ways:

```
shell> replace from to [from to] ... -- file_name [file_name] ...
shell> replace from to [from to] ... < file_name
```

" `from` " represents a string to look for and " `to` " represents its replacement. There can be one or more pairs of strings.

A from-string can contain these special characters:

Character	Description
`\^`	Match start of line.
`\$`	Match end of line.
`\b`	Match space-character, start of line or end of line. For an end `\b` the next replace starts looking at the end space-character. A `\b` alone in a string matches only a space-character

Use the `--` option to indicate where the string-replacement list ends and the file names begin. Any file named on the command line is modified in place, so you may want to make a copy of the original before converting it. `replace` prints a message indicating which of the input files it actually modifies.

If the `--` option is not given, replace reads standard input and writes to standard output.

replace uses a finite state machine to match longer strings first. It can be used to swap strings. For example, the following command swaps "a" and "b" in the given files, /file1 *and* file2*:*

```
shell> replace a b b a -- file1 file2 ...
```

The replace program is used by msql2mysql.

Options

`replace` supports the following options.

Option	Description
`-?, -I`	Display a help message and exit.
`-#debug_options`	Enable debugging.
`-s`	Silent mode. Print less information about what the program does.
`-v`	Verbose mode. Print more information about what the program does.
`-V`	Display version information and exit.

1.3.21 resolveip

Contents

resolveip is a utility for resolving IP addresses to host names and vice versa.

Usage

```
resolveip [OPTIONS] hostname or IP-address
```

Options

Option	Description
`-?, --help`	Display help and exit.

`-I`, `--info`	Synonym for `--help`.
`-s`, `--silent#`	Be more silent.
`-V`, `--version`	Display version information and exit.

Examples

```
shell> resolveip mariadb.org
IP address of mariadb.org is 166.78.144.191
```

```
resolveip 166.78.144.191
Host name of 166.78.144.191 is mariadb.org
```

1.3.22 resolve_stack_dump

resolve_stack_dump is a tool that resolves numeric stack strace dumps into symbols.

Usage

```
resolve_stack_dump [OPTIONS] symbols-file [numeric-dump-file]
```

The symbols-file should include the output from: `nm --numeric-sort mysqld`. The numeric-dump-file should contain a numeric stack trace from mysqld. If the numeric-dump-file is not given, the stack trace is read from stdin.

Options

Option	Description
`-h`, `--help`	Display this help and exit.
`-V`, `--version`	Output version information and exit.
`-s`, `--symbols-file=name`	Use specified symbols file.
`-n`, `--numeric-dump-file=name`	Read the dump from specified file.

1.3.23 xtstat

Contents

`xtstat` can be used to monitor all internal activity of PBXT.

`xtstat` polls the `INFORMATION_SCHEMA.PBXT_STATISTICS` table. The poll interval can be set using the `--delay` option, and is 1 second by default.

For most statistics, `xtstat` will display the difference in values between the current and previous polls. For example, if bytes written current value is 1000, and on the previous call it was 800, then `xtstat` will display 200. This means that 200 bytes were written to disk in the intervening period.

Using `xtstat`

Invoke xtstat as follows:

```
$ xtstat [ options ]
```

For example, to poll every 10 seconds:

```
xtstat -D10
```

Note that statistic counters are never reset, even if a rollback occurs. For example, if an `UPDATE` statement is rolled back, `xtstat` will still indicate that one write statement (see stat-write below) was executed.

If MariaDB shuts down or crashes, `xtstat` will attempt to reconnect. `xtstat` can be terminated any time using the `CTRL-C` key cimbination.

Before PBXT has recovered, not all statistics are available. In particular, the statistics relating to PBXT background threads are not available (including the `sweep` and `chkpnt` statistics).

Command line options

`xtstat` options are as follows:

Option	Description
`-?, --help`	Prints help text.
`-h, --host=value`	Connect to host.
`-u, --user=value`	User for login if not current user.
`-p, --password[=value]`	Password to use when connecting to server. If password is not given it's asked from the tty.
`-d, --database=value`	Database to be used (`pbxt` or `information_schema` required), default is `information_schema`
`-P, --port=value`	Port number to use for connection.
`-S, --socket=value`	Socket file to use for connection.
`-D, --delay=value`	Delay in seconds between polls of the database.
`--protocol=value`	Connection protocol to use: `default/tcp/socket/pipe/memory`
`--display=value`	Columns to display: use short names separated by `|` (the pipe character), partial match allowed. Use `--display=all` to display all columns available.

Connection options will also be taken from the MySQL config file if available.

Size indicators

Values displayed by `xtstat` are either a time in milliseconds, a value in bytes, or a counter. If these values are too large to be displayed then the value is rounded and a size indicator is added.

The following size indicators are used:

- `K` : Kilobytes *(1,024 bytes)*
- `M` : Megabytes *(1,048,576 bytes)*
- `G` : Gigabytes *(1,073,741,024 bytes)*
- `T` : Terabytes *(1,099,511,627,776 bytes)*
- `t` : thousands *(1,000s)*
- `m` : millions *(1,000,000s)*
- `b` : billions *(1,000,000,000s)*

Statistics

The following is a list of the statistics displayed by `xtstat` . Each statistic as a two-part display name. The first part is the

category and the second part is the type.

You can select categories and types for display, as you require. For example `--display=read` will display all read activity, `--display=xact|stat` will display transaction and statement activity.

Note, for diagnostics it is best to capture all statistics. The reason is because you never now where a problem might turn up, so without certain statistics you may not be able to identify the problem.

Display name	Name	Description
`time-curr`	Current Time	The current time in seconds
`time-msec`	Time Since Last Call	Time passed in milliseconds since last statistics call
`xact-commt`	Commit Count	Number of transactions committed
`xact-rollb`	Rollback Count	Number of transactions rolled back
`xact-waits`	Wait for Xact Count	Number of times waited for another transaction
`xact-dirty`	Dirty Xact Count	Number of transactions still to be cleaned up. This also includes all the currently running transactions. Cleanup means that the Sweeper thread must still scan the transcation and collect/mark any "garbage" left by the transaction. Garbage is, for example, versions of rows that are no longer visiable by any transaction.
`stat-read`	Read Statements	Number of SELECT statements
`stat-write`	Write Statements	Number of UPDATE/INSERT/DELETE statements
`rec-in`	Record Bytes Read	Bytes read from the record/row files
`rec-out`	Record Bytes Written	Bytes written to the record/row files. This data is transfered from the transaction logs to the handle data (xtd) and the row index files (xtr).
`rec-syncs/ms`	Record File Flushes	2 values separated by a '/': the number of flushes to data handle (.xtd) and row index (.xtr) files and the time taken in milliseconds to perform the flush operations.
`rec-hits`	Record Cache Hits	Hits when accessing the record cache. The record cache caches the data handle (.xtd) and row index (.xtr) files.
`rec-miss`	Record Cache Misses	Misses when accessing the record cache
`rec-frees`	Record Cache Frees	Number of record cache pages freed
`rec-%use`	Record Cache Usage	Percentage of record cache in use. This value is displayed by xtstat as a percentage of the total cache available, but the value returned by PBXT_STATISTICS table is in bytes used.
`ind-in`	Index Bytes Read	Bytes read from the index files
`ind-out`	Index Bytes Written	Bytes written to the index files. This data is transfered from the index log files (ilog) to the index files (xti), during a consistent flush of the index.
`ind-syncs/ms`	Index File Flushes	2 values separated by a '/': the number of flushes to index files and the time taken for the flush operations in milliseconds.
`ind-hits`	Index Cache Hits	Hits when accessing the index cache
`ind-miss`	Index Cache Misses	Misses when accessing the index cache
`ind-%use`	Index Cache Usage	Percentage of index cache used. This value is displayed by xtstat as a percentage of the total cache available, but the value returned by PBXT_STATISTICS table is in bytes used.
`ilog-in`	Index Log Bytes In	Bytes read from the index log files
`ilog-out`	Index Log Bytes Out	Bytes written to the index log files. This data is transfered from the index cache in main memory to the index log files (ilog) during a consistent flush of the index.
`ilog-syncs/ms`	Index Log File Syncs	2 values separated by a '/': the number of flushes to index log files and the time taken for the flush operations in milliseconds
`xlog-in`	Xact Log Bytes In	Bytes read from the transaction log files
`xlog-out`	Xact Log Bytes Out	Bytes written to the transaction log files. This is data transfered from the transaction log buffer (pbxt_transaction_buffer_size) to the transaction log files (.xlog). This transfer occurs on commit or when the transaction log buffer is full.
`xlog-syncs`	Xact Log File Syncs	Number of flushes to transaction log files

`xlog-msec`	Xact Log Sync Time	The time in milliseconds to flush transaction log files
`xlog-hits`	Xact Log Cache Hits	Hits when accessing the transaction log cache
`xlog-miss`	Xact Log Cache Misses	Misses when accessing the transaction log cache
`xlog-%use`	Xact Log Cache Usage	Percentage of transaction log cache used. This value is displayed by xtstat as a percentage of the total cache available, but the value returned by PBXT_STATISTICS table is in bytes used.
`data-in`	Data Log Bytes In	Bytes read from the data log files
`data-out`	Data Log Bytes Out	Bytes written to the data log files. This data is transfered from the data log buffer (pbxt_log_buffer_size) to the data log files (.dlog), when the buffer is full, or on commit.
`data-syncs`	Data Log File Syncs	Number of flushes to data log files
`data-msec`	Data Log Sync Time	The time in milliseconds spent flushing data log files
`to-chkpt`	Bytes to Checkpoint	Bytes written to the transaction log since the last checkpoint
`to-write`	Log Bytes to Write	Bytes written to the transaction log, still to be written to the database
`to-sweep`	Log Bytes to Sweep	Bytes written to the transaction log, still to be read by the Sweeper thread
`sweep-waits`	Sweeper Wait on Xact	Attempts to cleanup a transaction
`scan-index`	Index Scan Count	Number of index scans
`scan-table`	Table Scan Count	Number of table scans
`row-sel`	Select Row Count	Number of rows selected
`row-ins`	Insert Row Count	Number of rows inserted
`row-upd`	Update Row Count	Number of rows updated
`row-del`	Delete Row Count	Number of rows deleted

More Information

Documentation on this page is based on the xtstat documentation on the PrimeBase website.

Paul McCullagh's presentation from the 2010 User's Conference has some usage examples:
http://www.primebase.org/download/pbxt-uc-2010.pdf

1.3.24 mariadb-access

`mariadb-access` is a tool for checking access privileges, developed by Yves Carlier.

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadb-access` is a symlink to `mysqlaccess`.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mariadb-access` is the name of the tool, with `mysqlaccess` a symlink.

Contents

It checks the access privileges for a host name, user name, and database combination. Note that mariadb-access checks access using only the user, db, and host tables. It does not check table, column, or routine privileges specified in the tables_priv, columns_priv, or procs_priv tables.

Usage

```
mariadb-access [host [user [db]]] OPTIONS
```

If your MariaDB distribution is installed in some non-standard location, you must change the location where *mariadb-access* expects to find the mariadb client. Edit the *mariadb-access* script at approximately line 18. Search for a line that looks like this: <<code> $MYSQL = ´/usr/local/bin/mariadb; # path to mariadb executable <</code>> Change the path to reflect the location where *mariadb* actually is stored on your system. If you do not do this, a *Broken pipe error* will occur when you run *mariadb-access*.

Options

Option	Description
`-?, --help`	Displayhelp and exit.
`-v, --version`	Display version.
`-u username, --user=username`	Username for logging in to the db.
`-p[password], --password[=password]`	Password to use for user. If ommitted, `mariadb-access` prompts for one.
`-h hostname, --host=hostname`	Name or IP of the host.
`-d dbname, --db=dbname`	Name of the database.
`-U username, --superuser=username`	Connect as superuser.
`-P password, --spassword=password`	Password for superuser.
`-H server, --rhost=server`	Remote server to connect to.
`--old_server`	Connect to a very old MySQL servers (before version 3.21) that does not know how to handle full WHERE clauses.
`-b, --brief`	Single-line tabular report.
`-t, --table`	Report in table-format.
`--relnotes`	Print release-notes.
`--plan`	Print suggestions/ideas for future releases.
`--howto`	Some examples of how to run `mariadb-access'.
`--debug=N`	Enter debug level N (0..3).
`--copy`	Reload temporary grant-tables from original ones.
`--preview`	Show differences in privileges after making changes in (temporary) grant-tables.
`--commit`	Copy grant-rules from temporary tables to grant-tables (the grant tables must be flushed after, for example with mariadb-admin reload).
`--rollback`	Undo the last changes to the grant-tables.

Note

At least the user (`-u`) and the database (`-d`) must be given, even with wildcards. If no host is provided, `localhost' is assumed. Wildcards (*,?,%,_) are allowed for *host*, *user* and *db*, but be sure to escape them from your shell!! (ie type * or '*')

1.3.25 mariadb-admin

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadb-admin` is a symlink to `mysqladmin`, the administration program for the mysqld daemon.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mariadb-admin` is the name of the administration program for the mysqld daemon, with

`mysqladmin` a symlink .

Contents

`mariadb-admin` is an administration program for the mysqld daemon. It can be used to:

- Monitor what the MariaDB clients are doing (processlist)
- Get usage statistics and variables from the MariaDB server
- Create/drop databases
- Flush (reset) logs, statistics and tables
- Kill running queries.
- Stop the server (shutdown)
- Start/stop replicas
- Check if the server is alive (ping)

Using mariadb-admin

The command to use `mariadb-admin` and the general syntax is:

```
mariadb-admin [options] command [command-arg] [command [command-arg]] ...
```

Options

`mariadb-admin` supports the following options:

Option	Description
`--character-sets-dir=name`	Directory where the character set files are located.
`-C , --compress`	Compress all information sent between the client and the server if both support compression.
`--connect_timeout=val`	Maximum time in seconds before connection timeout. The default value is 43200 (12 hours).
`-c val , --count=val`	Number of iterations to make. This works with `-i` (`--sleep`) only.
`--debug[=debug_options] , -# [debug_options]`	Write a debugging log. A typical debug_options string is `d:t:o,file_name` . The default is `d:t:o,/tmp/mysqladmin.trace` .
`--debug-check`	Check memory and open file usage at exit.
`--debug-info`	Print debugging information and memory and CPU usage statistics when the program exits.
`--default-auth=plugin`	Default authentication client-side plugin to use.
`--default-character-set=name`	Set the default character set.
`-f , --force`	Don't ask for confirmation on drop database; with multiple commands, continue even if an error occurs.
`-? , --help`	Display this help and exit.
`-h name , --host=name`	Hostname to connect to.
`-l , --local`	Suppress the SQL command(s) from being written to the binary log by enabling sql_log_bin=0 for the session, or, from MariaDB 10.2.7 and MariaDB 10.1.24, for flush commands only, using `FLUSH LOCAL` rather than `SET sql_log_bin=0` , so the privilege requirement is RELOAD rather than SUPER.

`-b , --no-beep`	Turn off beep on error.
`-p[password] , --password[=password]`	Password to use when connecting to server. If password is not given it's asked from the terminal.
`--pipe , -W`	On Windows, connect to the server via a named pipe. This option applies only if the server supports named-pipe connections.
`-P portnum , --port=portnum`	Port number to use for connection, or 0 for default to, in order of preference, my.cnf, $MYSQL_TCP_PORT, /etc/services, built-in default (3306).
`--protocol=name`	The protocol to use for connection (tcp, socket, pipe, memory).
`-r , --relative`	Show difference between current and previous values when used with `-i` . Currently only works with extended-status.
`-O value , --set-variable=vaue`	Change the value of a variable. Please note that this option is deprecated; you can set variables directly with `--variable-name=value` .
`--shutdown_timeout=val`	Maximum number of seconds to wait for server shutdown. The default value is 3600 (1 hour).
`-s , --silent`	Silently exit if one can't connect to server.
`-i delay , --sleep=delay`	Execute commands repeatedly, sleeping for *delay* seconds in between. The `--count` option determines the number of iterations. If `--count` is not given, *mariadb-admin* executes commands indefinitely until interrupted.
`-S name , --socket=name`	For connections to localhost, the Unix socket file to use, or, on Windows, the name of the named pipe to use.
`--ssl`	Enables TLS. TLS is also enabled even without setting this option when certain other TLS options are set. The `--ssl` option will not enable verifying the server certificate by default. In order to verify the server certificate, the user must specify the `--ssl-verify-server-cert` option.
`--ssl-ca=name`	Defines a path to a PEM file that should contain one or more X509 certificates for trusted Certificate Authorities (CAs) to use for TLS. This option requires that you use the absolute path, not a relative path. See Secure Connections Overview: Certificate Authorities (CAs) for more information. This option implies the `--ssl` option.
`--ssl-capath=name`	Defines a path to a directory that contains one or more PEM files that should each contain one X509 certificate for a trusted Certificate Authority (CA) to use for TLS. This option requires that you use the absolute path, not a relative path. The directory specified by this option needs to be run through the `openssl rehash` command. See Secure Connections Overview: Certificate Authorities (CAs) for more information. This option is only supported if the client was built with OpenSSL or yaSSL. If the client was built with GnuTLS or Schannel, then this option is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms. This option implies the `--ssl` option.
`--ssl-cert=name`	Defines a path to the X509 certificate file to use for TLS. This option requires that you use the absolute path, not a relative path. This option implies the `--ssl` option.
`--ssl-cipher=name`	List of permitted ciphers or cipher suites to use for TLS. This option implies the `--ssl` option.
`--ssl-crl=name`	Defines a path to a PEM file that should contain one or more revoked X509 certificates to use for TLS. This option requires that you use the absolute path, not a relative path. See Secure Connections Overview: Certificate Revocation Lists (CRLs) for more information. This option is only supported if the client was built with OpenSSL or Schannel. If the client was built with yaSSL or GnuTLS, then this option is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.

`--ssl-crlpath=name`	Defines a path to a directory that contains one or more PEM files that should each contain one revoked X509 certificate to use for TLS. This option requires that you use the absolute path, not a relative path. The directory specified by this option needs to be run through the `openssl rehash` command. See Secure Connections Overview: Certificate Revocation Lists (CRLs) for more information. This option is only supported if the client was built with OpenSSL. If the client was built with yaSSL, GnuTLS, or Schannel, then this option is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.
`--ssl-key=name`	Defines a path to a private key file to use for TLS. This option requires that you use the absolute path, not a relative path. This option implies the `--ssl` option.
`--ssl-verify-server-cert`	Enables server certificate verification. This option is disabled by default.
`--tls-version=name`	This option accepts a comma-separated list of TLS protocol versions. A TLS protocol version will only be enabled if it is present in this list. All other TLS protocol versions will not be permitted. See Secure Connections Overview: TLS Protocol Versions for more information.
`-u`, `--user=name`	User for login if not current user.
`-v`, `--verbose`	Write more information.
`-V`, `--version`	Output version information and exit.
`-E`, `--vertical`	Print output vertically. Is similar to ' `--relative` ', but prints output vertically.
`-w[count]`, `--wait[=count]`	If the connection cannot be established, wait and retry instead of aborting. If a *count* value is given, it indicates the number of times to retry. The default is one time.
`--wait-for-all-slaves`	Wait for the last binlog event to be sent to all connected replicas before shutting down. This option is off by default.

Option Files

In addition to reading options from the command-line, `mariadb-admin` can also read options from option files. If an unknown option is provided to `mariadb-admin` in an option file, then it is ignored.

The following options relate to how MariaDB command-line tools handles option files. They must be given as the first argument on the command-line:

Option	Description
`--print-defaults`	Print the program argument list and exit.
`--no-defaults`	Don't read default options from any option file.
`--defaults-file=#`	Only read default options from the given file #.
`--defaults-extra-file=#`	Read this file after the global files are read.
`--defaults-group-suffix=#`	In addition to the default option groups, also read option groups with this suffix.

`mariadb-admin` is linked with MariaDB Connector/C. However, MariaDB Connector/C does not yet handle the parsing of option files for this client. That is still performed by the server option file parsing code. See MDEV-19035 for more information.

Option Groups

`mariadb-admin` reads options from the following option groups from option files:

Group	Description
`[mysqladmin]`	Options read by `mysqladmin`, which includes both MariaDB Server and MySQL Server.
`[mariadb-admin]`	Options read by `mariadb-admin`. Available starting with MariaDB 10.4.6.
`[client]`	Options read by all MariaDB and MySQL client programs, which includes both MariaDB and MySQL clients. For example, `mysqldump`.

`[client-server]`	Options read by all MariaDB client programs and the MariaDB Server. This is useful for options like socket and port, which is common between the server and the clients.
`[client-mariadb]`	Options read by all MariaDB client programs.

mariadb-admin Variables

Variables can be set with `--variable-name=value`.

Variables and boolean options	Value
`count`	`0`
`debug-check`	`FALSE`
`debug-info`	`FALSE`
`force`	`FALSE`
`compress`	`FALSE`
`character-sets-dir`	*(No default value)*
`default-character-set`	*(No default value)*
`host`	*(No default value)*
`no-beep`	`FALSE`
`port`	`3306`
`relative`	`FALSE`
`socket`	`/var/run/mysqld/mysqld.sock`
`sleep`	`0`
`ssl`	`FALSE`
`ssl-ca`	*(No default value)*
`ssl-capath`	*(No default value)*
`ssl-cert`	*(No default value)*
`ssl-cipher`	*(No default value)*
`ssl-key`	*(No default value)*
`ssl-verify-server-cert`	`FALSE`
`user`	*(No default value)*
`verbose`	`FALSE`
`vertical`	`FALSE`
`connect_timeout`	`43200`
`shutdown_timeout`	`3600`

mariadb-admin Commands

```
mariadb-admin [options] command [command-arg] [command [command-arg]] ...
```

Command is one or more of the following. Commands may be shortened to a unique prefix.

Command	Description
`create databasename`	Create a new database.
`debug`	Instruct server to write debug information to log.

`drop databasename`	Delete a database and all its tables.
`extended-status`	Return all status variables and their values.
`flush-all-statistics`	Flush all statistics tables
`flush-all-status`	Flush status and statistics.
`flush-binary-log`	Flush binary log.
`flush-client-statistics`	Flush client statistics.
`flush-engine-log`	Flush engine log.
`flush-error-log`	Flush error log.
`flush-general-log`	Flush general query log.
`flush-hosts`	Flush all cached hosts.
`flush-index-statistics`	Flush index statistics.
`flush-logs`	Flush all logs.
`flush-privileges`	Reload grant tables (same as reload).
`flush-relay-log`	Flush relay log.
`flush-slow-log`	Flush slow query log.
`flush-ssl`	Flush SSL certificates. Added in MariaDB 10.6.0.
`flush-status`	Clear status variables.
`flush-table-statistics`	Clear table statistics.
`flush-tables`	Flush all tables.
`flush-threads`	Flush the thread cache.
`flush-user-resources`	Flush user resources.
`flush-user-statistics`	Flush user statistics.
`kill id,id,...`	Kill mysql threads.
`password new-password`	Change old password to new-password. The new password can be passed on the commandline as the next argument (for example, `mariadb-admin password "new_password"`, or can be omitted (as long as no other command follows), in which case the user will be prompted for a password. If the password contains special characters, it needs to be enclosed in quotation marks. In Windows, the quotes can only be double quotes, as single quotes are assumed to be part of the password. If the server was started with the --skip-grant-tables option, changing the password in this way will have no effect.

`old-password new-password`	Change old password to new-password using the old pre-MySQL 4.1 format.
`ping`	Check if mysqld is alive. Return status is 0 if the server is running (even in the case of an error such as access denied), 1 if it is not.
`processlist`	Show list of active threads in server, equivalent to SHOW PROCESSLIST. With `--verbose`, equivalent to SHOW FULL PROCESSLIST.
`reload`	Reload grant tables.
`refresh`	Flush all tables and close and open log files.
`shutdown`	Take server down by executing the SHUTDOWN command on the server. If connected to a local server using a Unix socket file, mariadb-admin waits until the server's process ID file has been removed to ensure that the server has stopped properly. See also the `--wait-for-all-slaves` option.
`status`	Gives a short status message from the server.
`start-all-slaves`	Start all replicas.
`start-slave`	Start replication on a replica server.
`stop-all-slaves`	Stop all replicas.
`stop-slave`	Stop replication on a replica server.
`variables`	Prints variables available.
`version`	Returns version as well as status info from the server.

The shutdown Command and the --wait-for-all-slaves Option

The `--wait-for-all-slaves` option was first added in MariaDB 10.4.4. When a primary server is shutdown and it goes through the normal shutdown process, the primary kills client threads in random order. By default, the primary also considers its binary log dump threads to be regular client threads. As a consequence, the binary log dump threads can be killed while client threads still exist, and this means that data can be written on the primary during a normal shutdown that won't be replicated. This is true even if semi-synchronous replication is being used.

In MariaDB 10.4 and later, this problem can be solved by shutting down the server with the mariadb-admin utility and by providing the `--wait-for-all-slaves` option to the utility and by executing the `shutdown` command with the utility. For example:

```
mariadb-admin --wait-for-all-slaves shutdown
```

When the `--wait-for-all-slaves` option is provided, the server only kills its binary log dump threads after all client threads have been killed, and it only completes the shutdown after the last binary log has been sent to all connected replicas.

See Replication Threads: Binary Log Dump Threads and the Shutdown Process for more information.

Examples

Quick check of what the server is doing:

```
shell> mariadb-admin status
Uptime: 8023 Threads: 1 Questions: 14 Slow queries: 0 Opens: 15 Flush tables: 1 Open tables: 8
Queries per second avg: 0.1
shell> mariadb-admin processlist
+----+-------+-----------+----+---------+------+-------+------------------+
| Id | User | Host | db | Command | Time | State | Info |
+----+-------+-----------+----+---------+------+-------+------------------+
....
+----+-------+-----------+----+---------+------+-------+------------------+
```

More extensive information of what is happening 'just now' changing (great for troubleshooting a slow server):

```
shell> mariadb-admin --relative --sleep=1 extended-status | grep -v " 0 "
```

Check the variables for a running server:

```
shell> mariadb-admin variables | grep datadir
| datadir | /my/data/ |
```

Using a shortened prefix for the `version` command:

```
shell> mariadb-admin ver
mariadb-admin from 11.1.0-preview-MariaDB, client 9.1 for linux-systemd (x86_64)
Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Server version  11.1.0-preview-MariaDB
Protocol version 10
Connection  localhost via TCP/IP
TCP port  11100
Uptime:   3 min 21 sec

Threads: 1  Questions: 1  Slow queries: 0  Opens: 17  Open tables: 10  Queries per second avg:
0.004
```

Other Ways To Stop mysqld (Unix)

If you get the error:

```
mariadb-admin: shutdown failed; error: 'Access denied; you need (at least one of) the SHUTDOWN
privilege(s) for this operation'
```

It means that you didn't use `mariadb-admin` with a user that has the SUPER or SHUTDOWN privilege.

If you don't know the user password, you can still take the mysqld process down with a system `kill` command:

```
kill -SIGTERM pid-of-mysqld-process
```

The above is identical to `mariadb-admin shutdown` .

On windows you should use:

```
NET STOP MySQL
```

You can use the SHUTDOWN command from any client.

1.3.26 mariadb-check

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadb-check` is a symlink to `mysqlcheck` , the tool for checking, repairing, analyzing and optimizing tables.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mariadb-check` is the name of the tool, with `mysqlcheck` a symlink .

Contents

`mariadb-check` is a maintenance tool that allows you to check, repair, analyze and optimize multiple tables from the command line.

It is essentially a commandline interface to the CHECK TABLE, REPAIR TABLE, ANALYZE TABLE and OPTIMIZE TABLE commands, and so, unlike myisamchk and aria_chk, requires the server to be running.

This tool does not work with partitioned tables.

Using mariadb-check

```
./client/mariadb-check [OPTIONS] database [tables]
```

OR

```
./client/mariadb-check [OPTIONS] --databases DB1 [DB2 DB3...]
```

OR

```
./client/mariadb-check [OPTIONS] --all-databases
```

`mariadb-check` can be used to CHECK (-c, -m, -C), REPAIR (-r), ANALYZE (-a), or OPTIMIZE (-o) tables. Some of the options (like -e or -q) can be used at the same time. Not all options are supported by all storage engines.

The -c, -r, -a and -o options are exclusive to each other.

The option `--check` will be used by default, if no other options were specified. You can change the default behavior by making a symbolic link to the binary, or copying it somewhere with another name, the alternatives are:

`mysqlrepair`	The default option will be `-r` (`--repair`)
`mysqlanalyze`	The default option will be `-a` (`--analyze`)
`mysqloptimize`	The default option will be `-o` (`--optimize`)

Options

`mariadb-check` supports the following options:

Option	Description
`-A` , `--all-databases`	Check all the databases. This is the same as `--databases` with all databases selected.
`-1` , `--all-in-1`	Instead of issuing one query for each table, use one query per database, naming all tables in the database in a comma-separated list.
`-a` , `--analyze`	Analyze given tables.
`--auto-repair`	If a checked table is corrupted, automatically fix it. Repairing will be done after all tables have been checked.
`--character-sets-dir=name`	Directory where character set files are installed.
`-c` , `--check`	Check table for errors.
`-C` , `--check-only-changed`	Check only tables that have changed since last check or haven't been closed properly.
`-g` , `--check-upgrade`	Check tables for version-dependent changes. May be used with `--auto-repair` to correct tables requiring version-dependent updates. Automatically enables the `--fix-db-names` and `--fix-table-names` options. Used when upgrading
`--compress`	Compress all information sent between the client and server if both support compression.
`-B` , `--databases`	Check several databases. Note that normally *mariadb-check* treats the first argument as a database name, and following arguments as table names. With this option, no tables are given, and all name arguments are regarded as database names.

`-# , --debug[=name]`	Output debug log. Often this is 'd:t:o,filename'.
`--debug-check`	Check memory and open file usage at exit.
`--debug-info`	Print some debug info at exit.
`--default-auth=plugin`	Default authentication client-side plugin to use.
`--default-character-set=name`	Set the default character set.
`-e , --extended`	If you are using this option with `--check` , it will ensure that the table is 100 percent consistent, but will take a long time. If you are using this option with `--repair` , it will force using the old, slow, repair with keycache method, instead of the much faster repair by sorting.
`-F , --fast`	Check only tables that haven't been closed properly.
`--fix-db-names`	Convert database names to the format used since MySQL 5.1. Only database names that contain special characters are affected. Used when upgrading from an old MySQL version.
`--fix-table-names`	Convert table names (including views) to the format used since MySQL 5.1. Only table names that contain special characters are affected. Used when upgrading from an old MySQL version.
`--flush`	Flush each table after check. This is useful if you don't want to have the checked tables take up space in the caches after the check.
`-f , --force`	Continue even if we get an SQL error.
`-? , --help`	Display this help message and exit.
`-h name , --host=name`	Connect to the given host.
`-m , --medium-check`	Faster than extended-check, but only finds 99.99 percent of all errors. Should be good enough for most cases.
`-o , --optimize`	Optimize tables.
`-p , --password[=name]`	Password to use when connecting to the server. If you use the short option form (`-p`), you cannot have a space between the option and the password. If you omit the password value following the `--password` or `-p` option on the command line, mariadb-check prompts for one. Specifying a password on the command line should be considered insecure. You can use an option file to avoid giving the password on the command line.
`-Z , --persistent`	When using ANALYZE TABLE (`--analyze`), uses the PERSISTENT FOR ALL option, which forces Engine-independent Statistics for this table to be updated. Added in MariaDB 10.1.10
`-W , --pipe`	On Windows, connect to the server via a named pipe. This option applies only if the server supports named-pipe connections.
`--plugin-dir`	Directory for client-side plugins.
`-P num , --port=num`	Port number to use for connection or 0 for default to, in order of preference, my.cnf, $MYSQL_TCP_PORT, /etc/services, built-in default (3306).
`--process-tables`	Perform the requested operation (check, repair, analyze, optimize) on tables. Enabled by default. Use `--skip-process-tables` to disable.
`--process-views[=val]`	Perform the requested operation (only CHECK VIEW or REPAIR VIEW). Possible values are NO, YES (correct the checksum, if necessary, add the mariadb-version field), UPGRADE_FROM_MYSQL (same as YES and toggle the algorithm MERGE<->TEMPTABLE.
`--protocol=name`	The connection protocol (tcp, socket, pipe, memory) to use for connecting to the server. Useful when other connection parameters would cause a protocol to be used other than the one you want.
`-q , --quick`	If you are using this option with CHECK TABLE, it prevents the check from scanning the rows to check for wrong links. This is the fastest check. If you are using this option with REPAIR TABLE, it will try to repair only the index tree. This is the fastest repair method for a table.
`-r , --repair`	Can fix almost anything except unique keys that aren't unique.

`--shared-memory-base-name`	Shared-memory name to use for Windows connections using shared memory to a local server (started with the `--shared-memory` option). Case-sensitive.
`-s`, `--silent`	Print only error messages.
`--skip-database`	Don't process the database (case-sensitive) specified as argument.
`-S name`, `--socket=name`	For connections to localhost, the Unix socket file to use, or, on Windows, the name of the named pipe to use.
`--ssl`	Enables TLS. TLS is also enabled even without setting this option when certain other TLS options are set. Starting with MariaDB 10.2, the `--ssl` option will not enable verifying the server certificate by default. In order to verify the server certificate, the user must specify the `--ssl-verify-server-cert` option.
`--ssl-ca=name`	Defines a path to a PEM file that should contain one or more X509 certificates for trusted Certificate Authorities (CAs) to use for TLS. This option requires that you use the absolute path, not a relative path. See Secure Connections Overview: Certificate Authorities (CAs) for more information. This option implies the `--ssl` option.
`--ssl-capath=name`	Defines a path to a directory that contains one or more PEM files that should each contain one X509 certificate for a trusted Certificate Authority (CA) to use for TLS. This option requires that you use the absolute path, not a relative path. The directory specified by this option needs to be run through the openssl rehash command. See Secure Connections Overview: Certificate Authorities (CAs) for more information. This option is only supported if the client was built with OpenSSL or yaSSL. If the client was built with GnuTLS or Schannel, then this option is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms. This option implies the `--ssl` option.
`--ssl-cert=name`	Defines a path to the X509 certificate file to use for TLS. This option requires that you use the absolute path, not a relative path. This option implies the `--ssl` option.
`--ssl-cipher=name`	List of permitted ciphers or cipher suites to use for TLS. This option implies the `--ssl` option.
`--ssl-crl=name`	Defines a path to a PEM file that should contain one or more revoked X509 certificates to use for TLS. This option requires that you use the absolute path, not a relative path. See Secure Connections Overview: Certificate Revocation Lists (CRLs) for more information. This option is only supported if the client was built with OpenSSL or Schannel. If the client was built with yaSSL or GnuTLS, then this option is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.
`--ssl-crlpath=name`	Defines a path to a directory that contains one or more PEM files that should each contain one revoked X509 certificate to use for TLS. This option requires that you use the absolute path, not a relative path. The directory specified by this option needs to be run through the openssl rehash command. See Secure Connections Overview: Certificate Revocation Lists (CRLs) for more information. This option is only supported if the client was built with OpenSSL. If the client was built with yaSSL, GnuTLS, or Schannel, then this option is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.
`--ssl-key=name`	Defines a path to a private key file to use for TLS. This option requires that you use the absolute path, not a relative path. This option implies the `--ssl` option.
`--ssl-verify-server-cert`	Enables server certificate verification. This option is disabled by default.
`--tables`	Overrides the `--databases` or `-B` option such that all name arguments following the option are regarded as table names.
`--use-frm`	For repair operations on MyISAM tables, get table structure from .frm file, so the table can be repaired even if the .MYI header is corrupted.
`-u`, `--user=name`	User for login if not current user.
`-v`, `--verbose`	Print info about the various stages. You can give this option several times to get even more information. See mariadb-check and verbose, below.
`-V`, `--version`	Output version information and exit.

`--write-binlog`	Write ANALYZE, OPTIMIZE and REPAIR TABLE commands to the binary log. Enabled by default; use `--skip-write-binlog` when commands should not be sent to replication slaves.

Option Files

In addition to reading options from the command-line, `mariadb-check` can also read options from option files. If an unknown option is provided to `mariadb-check` in an option file, then it is ignored.

The following options relate to how MariaDB command-line tools handles option files. They must be given as the first argument on the command-line:

Option	Description
`--print-defaults`	Print the program argument list and exit.
`--no-defaults`	Don't read default options from any option file.
`--defaults-file=#`	Only read default options from the given file #.
`--defaults-extra-file=#`	Read this file after the global files are read.
`--defaults-group-suffix=#`	In addition to the default option groups, also read option groups with this suffix.

`mariadb-check` is linked with MariaDB Connector/C. However, MariaDB Connector/C does not yet handle the parsing of option files for this client. That is still performed by the server option file parsing code. See MDEV-19035 for more information.

Option Groups

`mariadb-check` reads options from the following option groups from option files:

Group	Description
`[mysqlcheck]`	Options read by `mysqlcheck`, which includes both MariaDB Server and MySQL Server.
`[mariadb-check]`	Options read by `mariadb-check`. Available starting with MariaDB 10.4.6.
`[client]`	Options read by all MariaDB and MySQL client programs, which includes both MariaDB and MySQL clients. For example, `mysqldump`.
`[client-server]`	Options read by all MariaDB client programs and the MariaDB Server. This is useful for options like socket and port, which is common between the server and the clients.
`[client-mariadb]`	Options read by all MariaDB client programs.

Notes

Default Values

To see the default values for the options and also to see the arguments you get from configuration files you can do:

```
./client/mariadb-check --print-defaults
./client/mariadb-check --help
```

mariadb-check and auto-repair

When running `mariadb-check` with `--auto-repair` (as done by mariadb-upgrade), `mariadb-check` will first check all tables and then in a separate phase repair those that failed the check.

mariadb-check and all-databases

`mariadb-check --all-databases` will ignore the internal log tables general_log and slow_log as these can't be checked, repaired or optimized.

mariadb-check and verbose

Using one `--verbose` option will give you more information about what mariadb-check is doing.

Using two `--verbose` options will also give you connection information.

If you use three `--verbose` options you will also get, on stdout, all ALTER, RENAME, and CHECK commands that mariadb-check executes.

1.3.27 mariadb-conv

MariaDB starting with 10.5.1

`mariadb-conv` is a character set conversion utility for MariaDB and was added in MariaDB 10.5.1.

Contents

Usage

```
mariadb-conv [OPTION...] [FILE...]
```

Options

`mariadb-conv` supports the following options:

Option	Description
-f, --from=name	Specifies the encoding of the input.
-t, --to=name	Specifies the encoding of the output.
-c, --continue	Silently ignore conversion errors.
--delimiter=name	Treat the specified characters as delimiters.

By default, `mariadb-conv` exits whenever it encounters any conversion problems, for example:

- the input byte sequence is not valid in the source character set
- the character cannot be converted to the target character set

The `-c` option makes `mariadb-conv` ignore such errors and use the question mark '?' to replace bytes in bad input sequences, or unconvertable characters.

The `--delimiter=...` option makes `mariadb-conv` treat the specified characters as delimiters rather than data to convert, so the input is treated as a combination of:

- data chunks, which are converted according to the `-f` and `-t` options.
- delimiters, which are not converted and are copied from the input to the output as is.

Examples

Converts the file `file.latin1.txt` from `latin1` to `utf8`.

```
mariadb-conv -f latin1 -t utf8 file.latin1.txt
```

Convert the file `file.latin1.txt` from `latin1` to `utf8`, reading the input data from stdin.

```
mariadb-conv -f latin1 -t utf8 < file.latin1.txt
```

Using mariadb-conv in a pipe:

```
echo test | ./mariadb-conv -f utf8 -t ucs2 >file.ucs2.txt
```

As a side effect, mariadb-conv can be used to list MariaDB data directories in a human readable form. Suppose you create the following tables:

```
SET NAMES utf8;
CREATE OR REPLACE TABLE t1 (a INT);
CREATE OR REPLACE TABLE ß (a INT);
CREATE OR REPLACE TABLE абв (a INT);
CREATE OR REPLACE TABLE 桌子 (a INT);
```

The above makes the server create the following files in the MariaDB data directory:

```
@1j.frm
@1j.ibd
@684c@5b50.frm
@684c@5b50.ibd
@g0@h0@i0.frm
@g0@h0@i0.ibd
t1.frm
t1.ibd
```

It's not precisely clear which file stores which table, because MariaDB uses a special table-name-to-file-name encoding.

This command on Linux (assuming an utf-8 console) can print the table list in a readable way::

```
ls | mariadb-conv -f filename -t utf8 --delimiter=".\n"

ß.frm
ß.ibd
桌子.frm
桌子.ibd
абв.frm
абв.ibd
t1.frm
t1.ibd
```

Note, the `--delimiter=".\n"` option is needed to make `mariadb-conv` treat the dot character (delimiting the encoded table name from the file extension) and the new line character (delimiting separate lines) as delimiters rather than as the data to convert (otherwise the conversion would fail).

Windows users can use the following command to list the data directory in the ANSI text console:

```
dir /b | mariadb-conv -c -f filename -t cp850 --delimiter=".\r\n"
```

Note:

- The `-t` options assume a Western machine.
- The `-c` option is needed to ignore conversion errors for Cyrillic and CJK characters.
- `--delimiter=` additionally needs the carriage return character \r

1.3.28 mariadb-convert-table-format

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadb-convert-table-format` is a symlink to `mysql_convert_table_format`, the tool for converting the tables in a database to use a particular storage engine.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mariadb-convert-table-format` is the name of the tool, with `mysql_convert_table_format` a symlink .

Usage

```
mariadb-convert-table-format [options] db_name
```

Contents
1. Usage
2. Description
3. Options

Description

`mariadb-convert-table-format` converts the tables in a database to use a particular storage engine (MyISAM by default).

`mariadb-convert-table-format` is written in Perl and requires that the DBI and DBD::mysql Perl modules be installed

Invoke `mariadb-convert-table-format` like this:

```
shell> mariadb-convert-table-format [options]db_name
```

The `db_name` argument indicates the database containing the tables to be converted.

Options

`mariadb-convert-table-format` supports the options described in the following list:

Option	Description
-?, --help	Display help and exit.
-e, --engine=ENGINE	Specify the storage engine that the tables should be converted to use. The default is MyISAM if this option is not given.
-f, --force	Continue even if errors occur.
-h, --host=host_name	Connect to the MariaDB server on the given host. Default localhost.
-p, --password=password	The password to use when connecting to the server. Note that the password value is not optional for this option, unlike for other client programs. Specifying the password on the command-line is generally considered insecure.
-P, --port=port_num	The TCP/IP port number to use for the connection.
-S, --socket=path	For connections to localhost, the Unix socket file to use.
-u, --user=user_name	The MariaDB user name to use when connecting to the server.
-v, --verbose	Verbose mode. Print more information about what the program does.
-V, --version	Display version information and exit.

1.3.29 mariadb-dumpslow

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadb-dumpslow` is a symlink to `mysqldumpslow` , the tool for examining the slow query log.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mariadb-dumpslow` is the name of the tool, with `mysqldumpslow` a symlink .

`mariadb-dumpslow` is a tool to examine the slow query log.

It parses the slow query log files, printing a summary result. Normally, mariadb-dumpslow groups queries that are similar except for the particular values of number and string data values. It “abstracts” these values to N and ´S´ when displaying summary output. The `-a` and `-n` options can be used to modify value abstracting behavior.

Usage

```
mariadb-dumpslow [ options... ] [ logs... ]
```

Options

Option	Description
`-a`	Don't abstract all numbers to N and strings to 'S'
`-d`, `--debug`	Debug
`-g PATTERN`	Grep: only consider statements that include this string
`--help`	Display help
`-h HOSTNAME`	Hostname of db server for *-slow.log filename (can be wildcard), default is '*', i.e. match all
`-i NAME`	Name of server instance (if using mysql.server startup script)
`-l`	Don't subtract lock time from total time
`-n NUM`	Abstract numbers with at least *NUM* digits within names
`-r`	Reverse the sort order (largest last instead of first)
`-s ORDER`	What to sort by (aa, ae, al, ar, at, a, c, e, l, r, t). `at` is default. `aa` average rows affected `ae` aggregated number of rows examined `al` average lock time `ar` average rows sent `at` average query time `a` rows affected `c` count `e` rows examined `l` lock time `r` rows sent `t` query time
`-t NUM`	Just show the top *NUM* queries.
`-v`, `--verbose`	Verbose mode.

1.3.30 mariadb-embedded

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadb-embedded` is a symlink to `mysql_embedded`, the embedded server.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mariadb-embedded` is the name of the tool, with `mysql_embedded` a symlink.

`mariadb-embedded` is a mariadb client statically linked to libmariadbd, the embedded server. Upon execution, an embedded MariaDB server is instantiated and you can execute statements just as you would using the normal mariadb client, using the same options.

Do not run *mariadb-embedded* using the same database as a running MariaDB server!

Examples

```
sudo mariadb-embedded -e 'select user, host, password from mysql.user where user="root"'
+------+-----------+-------------------------------------------+
| user | host      | password                                  |
+------+-----------+-------------------------------------------+
| root | localhost | *196BDEDE2AE4F84CA44C47D54D78478C7E2BD7B7 |
| root | db1       | *196BDEDE2AE4F84CA44C47D54D78478C7E2BD7B7 |
| root | 127.0.0.1 | *196BDEDE2AE4F84CA44C47D54D78478C7E2BD7B7 |
| root | ::1       | *196BDEDE2AE4F84CA44C47D54D78478C7E2BD7B7 |
+------+-----------+-------------------------------------------+
```

Sending options with `--server-arg`:

```
sudo mariadb-embedded --server-arg='--skip-innodb'
  --server-arg='--default-storage-engine=myisam'
  --server-arg='--log-error=/tmp/mysql.err'
  -e 'select user, host, password from mysql.user where user="root"'
+------+-----------+-------------------------------------------+
| user | host      | password                                  |
+------+-----------+-------------------------------------------+
| root | localhost | *196BDEDE2AE4F84CA44C47D54D78478C7E2BD7B7 |
| root | db1       | *196BDEDE2AE4F84CA44C47D54D78478C7E2BD7B7 |
| root | 127.0.0.1 | *196BDEDE2AE4F84CA44C47D54D78478C7E2BD7B7 |
| root | ::1       | *196BDEDE2AE4F84CA44C47D54D78478C7E2BD7B7 |
+------+-----------+-------------------------------------------+
```

1.3.31 mariadb-find-rows

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadb-find-rows` is a symlink to `mysql_find_rows`, the tool for reading files containing SQL statements and extracting statements that match a given regular expression or that contain USE db_name or SET statements.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mariadb-find-rows` is the name of the tool, with `mysql_find_rows` a symlink.

Contents

`mariadb-find-rows` reads files containing SQL statements and extracts statements that match a given regular expression or that contain USE db_name or SET statements. The utility was written for use with update log files (as used prior to MySQL 5.0) and as such expects statements to be terminated with semicolon (;) characters. It may be useful with other files that contain SQL statements as long as statements are terminated with semicolons.

Usage

```
mariadb-find-rows [options] [file_name ...]
```

Each file_name argument should be the name of file containing SQL statements. If no file names are given, *mariadb-find-rows* reads the standard input.

Options

mariadb-find-rows supports the following options:

Option	Description
`--help`, `--Information`	Display help and exit.
`--regexp=pattern`	Display queries that match the pattern.
`--rows=N`	Quit after displaying N queries.

`--skip-use-db`	Do not include USE db_name statements in the output.
`--start_row=N`	Start output from this row (first row is 1).

Examples

```
mariadb-find-rows --regexp=problem_table --rows=20 < update.log
mariadb-find-rows --regexp=problem_table  update-log.1 update-log.2
```

1.3.32 mariadb-fix-extensions

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadb-fix-extensions` is a symlink to `mysql_fix_extensions`, the tool for converting the extensions for MyISAM (or ISAM) table files to their canonical forms.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mysql_fix_extensions` is the symlink, and `mariadb-fix-extensions` the binary name.

`mariadb-fix-extensions` converts the extensions for MyISAM (or ISAM) table files to their canonical forms.

It looks for files with extensions matching any lettercase variant of `.frm`, `.myd`, `.myi`, `.isd`, and `.ism` and renames them to have extensions of `.frm`, `.MYD`, `.MYI`, `.ISD`, and `.ISM`, respectively. This can be useful after transferring the files from a system with case-insensitive file names (such as Windows) to a system with case-sensitive file names.

Invoke mariadb-fix-extensions as follows, where data_dir is the path name to the MariaDB data directory.

```
mariadb-fix-extensions data_dir
```

1.3.33 mariadb-install-db

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadb-install-db` is a symlink to `mysql_install_db`. the tool for initializing the MariaDB data directory and creating the system tables

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mysql_install_db` is the symlink, and `mariadb-install-db` the binary name.

This page is for the `mariadb-install-db` script for Linux/Unix only

For the Windows specific tool of similar name and purpose see mysql_install_db.exe.

The Windows version shares the common theme (creating system tables), yet has a lot of functionality specific to Windows systems, for example creating a Windows service. The Windows version does *not* share command line parameters with the Unix shell script.

Contents

`mariadb-install-db` initializes the MariaDB data directory and creates the system tables in the mysql database, if they do not exist. MariaDB uses these tables to manage privileges, roles, and plugins. It also uses them to provide the data for the help command in the mariadb client.

`mariadb-install-db` works by starting MariaDB Server's `mysqld` process in --bootstrap mode and sending commands to create the system tables and their content.

Using mariadb-install-db

To invoke `mariadb-install-db`, use the following syntax:

```
$ mariadb-install-db [options]
```

Because the MariaDB server, `mysqld`, needs to access the data directory when it runs later, you should either run `mariadb-install-db` from the same account that will be used for running `mysqld` or run it as root and use the `--user` option to indicate the user name that `mysqld` will run as. It might be necessary to specify other options such as `--basedir` or `--datadir` if `mariadb-install-db` does not use the correct locations for the installation directory or data directory. For example:

```
$ scripts/mariadb-install-db --user=mysql \
   --basedir=/opt/mysql/mysql \
   --datadir=/opt/mysql/mysql/data
```

Options

`mariadb-install-db` supports the following options:

Option	Description
`--auth-root-authentication-method={normal | socket}`	If set to `normal`, it creates a `root@localhost` account that authenticates with the `mysql_native_password` authentication plugin and that has no initial password set, which can be insecure. If set to `socket`, it creates a `root@localhost` account that authenticates with the `unix_socket` authentication plugin. Set to `socket` by default from MariaDB 10.4 (see Authentication from MariaDB 10.4), or `normal` by default in earlier versions. Available since MariaDB 10.1.
`--auth-root-socket-user=USER`	Used with `--auth-root-authentication-method=socket`. It specifies the name of the second account to create with `SUPER` privileges in addition to `root`, as well as of the system account allowed to access it. Defaults to the value of `--user`.
`--basedir=path`	The path to the MariaDB installation directory.
`--builddir=path`	If using `--srcdir` with out-of-directory builds, you will need to set this to the location of the build directory where built files reside.
`--catalogs=["list"]`	Initialize MariaDB for catalogs. Argument is a list, separated with space, of the catalogs to create. The def catalog is created automatically. Added in MariaDB 11.3.
`--catalog-user=user`	User when adding catalogs to running server. Added in MariaDB 11.3.

`--catalog-password[=password]`	Password for catalog-user. Added in MariaDB 11.3.
`--catalog-client-arg=arg`	Other arguments to 'mariadb' when adding new catalogs. Added in MariaDB 11.3.
`--cross-bootstrap`	For internal use. Used when building the MariaDB system tables on a different host than the target.
`--datadir=path`, `--ldata=path`	The path to the MariaDB data directory.
`--debug=path`	Write commands to-be executed in 'path'. Added in MariaDB 11.3.
`--defaults-extra-file=name`	Read this file after the global files are read. Must be given as the first option.
`--defaults-file=name`	Only read default options from the given file *name* Must be given as the first option.
`--defaults-group-suffix=name`	In addition to the given groups, read also groups with this suffix. From MariaDB 10.1.31, MariaDB 10.2.13 and MariaDB 10.3.5.
`--force`	Causes `mariadb-install-db` to run even if DNS does not work. In that case, grant table entries that normally use host names will use IP addresses.
`--no-defaults`	Don't read default options from any option file. Must be given as the first option.
`--print-defaults`	Print the program argument list and exit. Must be given as the first option.
`--rpm`	For internal use. This option is used by RPM files during the MariaDB installation process.
`--skip-name-resolve`	Uses IP addresses rather than host names when creating grant table entries. This option can be useful if your DNS does not work.
`--skip-test-db`	Don't install the test database
`--srcdir=path`	For internal use. The path to the MariaDB source directory. This option uses the compiled binaries and support files within the source tree, useful for if you don't want to install MariaDB yet and just want to create the system tables. The directory under which `mariadb-install-db` looks for support files such as the error message file and the file for populating the help tables.
`--user=user_name`	The login user name to use for running `mysqld`. Files and directories created by `mysqld` will be owned by this user. You must be `root` to use this option. By default, `mysqld` runs using your current login name and files and directories that it creates will be owned by you.
`--verbose`	Verbose mode. Print more information about what the program does.
`--windows`	For internal use. This option is used for creating Windows distributions.

Option Files

In addition to reading options from the command-line, `mariadb-install-db` can also read options from option files. If an unknown option is provided to `mariadb-install-db` in an option file, then it is ignored.

The following options relate to how MariaDB command-line tools handles option files. They must be given as the first argument on the command-line:

Option	Description
`--print-defaults`	Print the program argument list and exit.
`--no-defaults`	Don't read default options from any option file.
`--defaults-file=#`	Only read default options from the given file #.
`--defaults-extra-file=#`	Read this file after the global files are read.
`--defaults-group-suffix=#`	In addition to the default option groups, also read option groups with this suffix.

Option Groups

`mariadb-install-db` reads options from the following option groups from option files:

Group	Description
`[mysql_install_db]`	Options read by `mysqld_safe`, which includes both MariaDB Server and MySQL Server.

`mariadb-install-db` also reads options from the following server option groups from option files:

Group	Description
`[mysqld]`	Options read by `mysqld`, which includes both MariaDB Server and MySQL Server.
`[server]`	Options read by MariaDB Server.
`[mysqld-X.Y]`	Options read by a specific version of `mysqld`, which includes both MariaDB Server and MySQL Server. For example, `[mysqld-5.5]`.
`[mariadb]`	Options read by MariaDB Server.
`[mariadb-X.Y]`	Options read by a specific version of MariaDB Server.
`[client-server]`	Options read by all MariaDB client programs and the MariaDB Server. This is useful for options like socket and port, which is common between the server and the clients.
`[galera]`	Options read by a galera-capable MariaDB Server. Available on systems compiled with Galera support.

Installing System Tables

Installing System Tables From a Source Tree

If you have just compiled MariaDB from source, and if you want to use `mariadb-install-db` from your source tree, then that can be done without having to actually install MariaDB. This is very useful if you want to test your changes to MariaDB without disturbing any existing installations of MariaDB.

To do so, you would have to provide the `--srcdir` option. For example:

```
./scripts/mariadb-install-db --srcdir=. --datadir=path-to-temporary-data-dir
```

Installing System Tables From a Binary Tarball

If you install a binary tarball package in a non standard path, like your home directory, and if you already have a MariaDB / MySQL package installed, then you may get conflicts with the default `/etc/my.cnf`. This often results in permissions errors.

One possible solution is to use the `--no-defaults` option, so that it does not read any option files. For example:

```
./scripts/mariadb-install-db --no-defaults --basedir=. --datadir=data
```

Another possible solution is to use the `defaults-file` option, so that you can specify your own option file. For example:

```
./scripts/mariadb-install-db --defaults-file=~/.my.cnf
```

User Accounts Created by Default

MariaDB starting with 10.4

In MariaDB 10.4 and later, `mariadb-install-db` sets `--auth-root-authentication-method=socket` by default. When this is set, the default `root@localhost` user account is created with the ability to use two authentication plugins:

- First, it is configured to try to use the unix_socket authentication plugin. This allows the the `root@localhost` user to login without a password via the local Unix socket file defined by the socket system variable, as long as the login is attempted from a process owned by the operating system `root` user account.
- Second, if authentication fails with the unix_socket authentication plugin, then it is configured to try to use the mysql_native_password authentication plugin.

The definition of the default `root@localhost` user account is:

```
CREATE USER 'root'@'localhost' IDENTIFIED VIA unix_socket
  OR mysql_native_password USING 'invalid';
GRANT ALL PRIVILEGES ON *.* TO 'root'@'localhost' WITH GRANT OPTION;
GRANT PROXY ON ''@'%' TO 'root'@'localhost' WITH GRANT OPTION;
```

Since `mariadb-install-db` sets `--auth-root-authentication-method=socket` by default, the following additional user accounts are **not** created by default:

- `root@127.0.0.1`
- `root@::1`
- `root@${current_hostname}`

However, an additional user account that is defined by the `--auth-root-socket-user` option is created. If this option is not set, then the value defaults to the value of the `--user` option. On most systems, the `--user` option will use the value of `mysql` by default, so this additional user account would be called `mysql@localhost`.

The definition of this `mysql@localhost` user account is similar to the `root@localhost` user account:

```
CREATE USER 'mysql'@'localhost' IDENTIFIED VIA unix_socket
  OR mysql_native_password USING 'invalid';
GRANT ALL PRIVILEGES ON *.* TO 'mysql'@'localhost' WITH GRANT OPTION;
```

An invalid password is initially set for both of these user accounts. This means that before a password can be used to authenticate as either of these user accounts, the accounts must first be given a valid password by executing the `SET PASSWORD` statement.

For example, here is an example of setting the password for the `root@localhost` user account immediately after installation:

```
$ sudo yum install MariaDB-server
$ sudo systemctl start mariadb
$ sudo mariadb
...
MariaDB> SET PASSWORD = PASSWORD('XH4VmT3_jt');
```

You may notice in the above example that the mariadb command-line client is executed via sudo. This allows the `root@localhost` user account to successfully authenticate via the unix_socket authentication plugin.

MariaDB until 10.3

In MariaDB 10.3 and before, `mariadb-install-db` sets `--auth-root-authentication-method=normal` by default. When this is set, the following default accounts are created with no password:

- `root@localhost`
- `root@127.0.0.1`
- `root@::1`
- `root@${current_hostname}`

The definition of the default `root@localhost` user account is:

```
CREATE USER 'root'@'localhost';
GRANT ALL PRIVILEGES ON *.* TO 'root'@'localhost' WITH GRANT OPTION;
GRANT PROXY ON ''@'%' TO 'root'@'localhost' WITH GRANT OPTION;
```

The definition of the other default `root` accounts is similar.

A password should be set for these user accounts immediately after installation. This can be done either by executing the SET PASSWORD statement or by running mysql_secure_installation.

For example, here is an example of setting the password for the `root@localhost` user account immediately after installation:

```
$ sudo yum install MariaDB-server
$ sudo systemctl start mariadb
$ mysql -u root
...
MariaDB> SET PASSWORD = PASSWORD('XH4VmT3_jt');
```

Since `mariadb-install-db` sets `--auth-root-authentication-method=normal` by default, the `--auth-root-socket-user` option is ignored by default.

Troubleshooting Issues

Checking the Error Log

If `mariadb-install-db` fails, you should examine the error log in the data directory, which is the directory specified with `--datadir` option. This should provide a clue about what went wrong.

Testing With mysqld

You can also test that this is not a general fault of MariaDB Server by trying to start the `mysqld` process. The -skip-grant-tables option will tell it to ignore the system tables. Enabling the general query log can help you determine what queries are being run on the server. For example:

```
mysqld --skip-grant-tables --general-log
```

At this point, you can use the mariadb client to connect to the mysql database and look at the system tables. For example:

```
$ /usr/local/mysql/bin/mysql -u root mysql
MariaDB [mysql]> show tables
```

Using a Server Compiled With --disable-grant-options

The following only apply in the exceptional case that you are using a mysqld server which is configured with the `--disable-grant-options` option:

`mariadb-install-db` needs to invoke `mysqld` with the `--bootstrap` and `--skip-grant-tables` options. A MariaDB configured with the `--disable-grant-options` option has `--bootstrap` and `--skip-grant-tables` disabled. To handle this case, set the `MYSQLD_BOOTSTRAP` environment variable to the full path name of a mysqld server that is configured without `--disable-grant-options`. `mariadb-install-db` will use that server.

The test and test_% Databases

When calling the `mariadb-install-db` script, a new folder called `test` is created in the data directory. It only has the single `db.opt` file, which sets the client options `default-character-set` and `default-collation` only.

If you run `mysql` as an anonymous user, `mysql -u''@localhost`, and look for the grants and databases you are able to work with, you will get the following:

```
SELECT current_user;
+--------------+
| current_user |
+--------------+
| @localhost   |
+--------------+

SHOW GRANTS FOR current_user;
+---------------------------------------+
| Grants for @localhost                 |
+---------------------------------------+
| GRANT USAGE ON *.* TO ``@`localhost` |
+---------------------------------------+

SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| test               |
+--------------------+
```

Shown are the `information_schema` as well as `test` databases that are built in databases. But looking from SHOW GRANTS appears to be a paradox; how can the current user see something if they don't have privileges for that?

Let's go a step further.
Now, use the `root` / `unix` user, which has all rights, in order to create a new database with the prefix `test_`, something

like:

```
CREATE DATABASE test_electricity;
```

With the above change, a new directory will be created in the data directory.
Now login again with the anonymous user and run SHOW DATABASES:

```
SHOW DATABASES
+--------------------+
| Database           |
+--------------------+
| information_schema |
| test               |
| test_electricity   |
+--------------------+
```

Again we are able to see the newly created database, without any rights? We have an anonymous user that has no privileges, but still can see the `test` and `test_electricity` databases.
Where does this come from?

Login with the `root` / `unix` user to find out all privileges that the anonymous user has:

```
SELECT * FROM mysql.user WHERE user='' AND host='localhost'\G
*************************** 1. row ***************************
                  Host: localhost
                  User:
              Password:
           Select_priv: N
           Insert_priv: N
           Update_priv: N
           Delete_priv: N
           Create_priv: N
             Drop_priv: N
           Reload_priv: N
         Shutdown_priv: N
          Process_priv: N
             File_priv: N
            Grant_priv: N
       References_priv: N
            Index_priv: N
            Alter_priv: N
          Show_db_priv: N
            Super_priv: N
 Create_tmp_table_priv: N
      Lock_tables_priv: N
          Execute_priv: N
       Repl_slave_priv: N
      Repl_client_priv: N
      Create_view_priv: N
        Show_view_priv: N
   Create_routine_priv: N
    Alter_routine_priv: N
      Create_user_priv: N
            Event_priv: N
          Trigger_priv: N
Create_tablespace_priv: N
   Delete_history_priv: N
              ssl_type:
            ssl_cipher:
           x509_issuer:
          x509_subject:
         max_questions: 0
           max_updates: 0
       max_connections: 0
  max_user_connections: 0
                plugin:
 authentication_string:
      password_expired: N
               is_role: N
          default_role:
    max_statement_time: 0.000000
```

As seen above from the mysql.user table, the anonymous user doesn't have any global privileges. Still, the anonymous user can see databases, so there must be a way so that anonymous user can see the `test` and `test_electricity` databases.

Let's check for grants on the database level. That information can be found in the mysql.db table. Looking at the `mysql.db` table, it already contains 2 rows created when the `mariadb-install-db` script was invoked.

The anonymous user has database privileges (without `grant`, `alter_routine` and `execute`) on `test` and `test_%` databases:

```
SELECT * FROM mysql.db\G
*************************** 1. row ***************************
                 Host: %
                   Db: test
                 User:
          Select_priv: Y
          Insert_priv: Y
          Update_priv: Y
          Delete_priv: Y
          Create_priv: Y
            Drop_priv: Y
           Grant_priv: N
      References_priv: Y
           Index_priv: Y
           Alter_priv: Y
Create_tmp_table_priv: Y
     Lock_tables_priv: Y
     Create_view_priv: Y
       Show_view_priv: Y
  Create_routine_priv: Y
   Alter_routine_priv: N
         Execute_priv: N
           Event_priv: Y
         Trigger_priv: Y
  Delete_history_priv: Y
*************************** 2. row ***************************
                 Host: %
                   Db: test\_%
                 User:
          Select_priv: Y
          Insert_priv: Y
          Update_priv: Y
          Delete_priv: Y
          Create_priv: Y
            Drop_priv: Y
           Grant_priv: N
      References_priv: Y
           Index_priv: Y
           Alter_priv: Y
Create_tmp_table_priv: Y
     Lock_tables_priv: Y
     Create_view_priv: Y
       Show_view_priv: Y
  Create_routine_priv: Y
   Alter_routine_priv: N
         Execute_priv: N
           Event_priv: Y
         Trigger_priv: Y
  Delete_history_priv: Y
```

The first row is reserved for explicit usage for the `test` database, which is automatically created with `mariadb-install-db` .

Since database `test_electricity` satisfies the `test_%` pattern where `test_` is a prefix, we can understand why the user has the right to work with the newly-created database.

As long as records in `mysql.db` for the anonymous user exists, each new user created will have the privileges for the `test` and `test_%` databases.

Other databases privileges **are not automatically granted** for the newly created user. We have to grant privileges, which will be visible in `mysql.db` table.

Not Creating the test Database and Anonymous User

If you run `mariadb-install-db` with the `--skip-test-db` option, no `test` database will be created, which we can see as follows:

```
SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
+--------------------+

SELECT * FROM mysql.db;
Empty set (0.001 sec)
```

Also, no anonymous user is created (only `unix` / `mariadb.sys` / `root` users):

```
SELECT user,host FROM mysql.user;
+-------------+-----------+
| User        | Host      |
+-------------+-----------+
| anel        | localhost |
| mariadb.sys | localhost |
| root        | localhost |
+-------------+-----------+
```

1.3.34 mariadb-plugin

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadb-plugin` is a symlink to `mysql_plugin`, the tool for enabling or disabling plugins.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mysql_plugin` is the symlink, and `mariadb-plugin` the binary name.

Contents

`mariadb-plugin` is a tool for enabling or disabling plugins.

It is a commandline alternative to the INSTALL PLUGIN and UNINSTALL PLUGIN statements, and the `--plugin-load` option to mysqld.

`mariadb-plugin` must be run while the server is offline, and works by adding or removing rows from the mysql.plugin table.

`mariadb-plugin` basically has two use cases:

- adding a plugin even before the first real server startup
- removing a plugin that crashes the server on startup

For the install use case, adding a plugin-load-add entry to `my.cnf` or in a separate include option file, is probably a better alternative. In case of a plugin loaded via a `mysql.plugin` crashing the server, uninstalling the plugin with the help of `mariadb-plugin` can be the only viable action though.

Usage

```
mariadb-plugin [options] <plugin> ENABLE|DISABLE
```

`mariadb-plugin` expects to find a configuration file that indicates how to configure the plugins. The configuration file is by default the same name as the plugin, with a `.ini` extension. For example:

```
mariadb-plugin crazyplugins ENABLE
```

Here, `mariadb-plugin` will look for a file called `crazyplugins.ini`

```
crazyplugins
crazyplugin1
crazyplugin2
crazyplugin3
```

The first line should contain the name of the library object file, with no extension. The other lines list the names of the components. Each value should be on a separate line, and the `#` character at the start of the line indicates a comment.

Options

The following options can be specified on the command line, while some can be specified in the `[mysqld]` group of any option file. For options specified in a `[mysqld]` group, only the `--basedir`, `--datadir`, and `--plugin-dir` options can be used - the rest are ignored.

Option	Description
`-b`, `--basedir=name`	The base directory for the server.
`-d`, `--datadir=name`	The data directory for the server.
`-?`, `--help`	Display help and exit.
`-f`, `--my-print-defaults=name`	Path to `my_print_defaults` executable. Example: `/source/temp11/extra`
`-m`, `--mysqld=name`	Path to `mysqld` executable. Example: `/sbin/temp1/mysql/bin`
`-n`, `--no-defaults`	Do not read values from configuration file.
`-p`, `--plugin-dir=name`	The plugin directory for the server.
`-i`, `--plugin-ini=name`	Read plugin information from configuration file specified instead of from `<plugin-dir>/<plugin_name>.ini`.
`-P`, `--print-defaults`	Show default values from configuration file.
`-v`, `--verbose`	More verbose output; you can use this multiple times to get even more verbose output.
`-V`, `--version`	Output version information and exit.

1.3.35 mariadb-report

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadb-report` is a symlink to `mysqlreport`, the binary for showing the value of important status variables.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mariadb-report` is the name of the binary, with `mysqlreport` a symlink.

Contents

`mariadb-report` makes a friendly report of important MariaDB status values. Actually, it makes a friendly report of nearly every status value from SHOW STATUS. Unlike SHOW STATUS which simply dumps over 100 values to the screen in one long list, mariadb-report interprets and formats the values and presents the basic values and many more inferred values in a human-readable format. Numerous example reports are available in the archive of the old hackmysql.com/mysqlreport, archived here.

The benefit of mariadb-report is that it allows you to very quickly see a wide array of performance indicators for your MariaDB server which would otherwise need to be calculated by hand from all the various SHOW STATUS values. For example, the Index Read Ratio is an important value but it's not present in SHOW STATUS; it's an inferred value (the ratio of Key_reads to Key_read_requests).

This documentation outlines all the command line options in mariadb-report, most of which control which reports are printed. This document does not address how to interpret these reports; that topic is covered in the document Guide To Understanding mysqlreport, archived here.

Usage

```
mariadb-report [options]
```

mariadb-report options

Technically, command line options are in the form `--option`, but `-option` works too. All options can be abbreviated if the abbreviation is unique. For example, option `--host` can be abbreviated to `--ho` but not `--h` because `--h` is ambiguous: it could mean `--host` or `--help`.

Option	Description
`--all`	Equivalent to `--dtq --dms --com 3 --sas --qcache`. (Notice `--tab` is not invoked by `--all`.)
`--com N`	Print top N number of non-DMS Com_ status values in descending order (after DMS in Questions report). If N is not given, default is 3. Such non-DMS Com_ values include Com_change_db, Com_show_tables, Com_rollback, etc.
`--dms`	Print Data Manipulation Statements (DMS) report (under DMS in Questions report). DMS are those from the Data Manipulation section. mariadb-report considers only SELECT, INSERT, REPLACE, UPDATE, and DELETE. Each DMS is listed in descending order by count.
`--dtq`	Print Distribution of Total Queries (DTQ) report (under Total in Questions report). Queries (or Questions) can be divided into four main areas: DMS (see `--dms`), Com_ (see `--com`), COM_QUIT (see COM_QUIT and Questions, archived here), and Unknown. `--dtq` lists the number of queries in each of these areas in descending order.
`--email ADDRESS`	After printing the report to screen, email the report to ADDRESS. This option requires sendmail in /usr/sbin/, therefore it does not work on Windows. /usr/sbin/sendmail can be a sym link to qmail, for example, or any MTA that emulates sendmail's -t command line option and operation. The FROM: field is "mariadb-report", SUBJECT: is "MySQL status report".
`--flush-status`	Execute a FLUSH STATUS after generating the reports. If you do not have permissions in MariaDB to do this an error from DBD::mysql::st will be printed after the reports.
`--help`	Output help information and exit.
`--host ADDRESS`	Host address.
`--infile FILE`	Instead of getting SHOW STATUS values from MariaDB, read values from FILE. FILE is often a copy of the output of SHOW STATUS including formatting characters (+, -). *mariadb-report* expects FILE to have the format " value number " where value is only alpha and underscore characters (A-Z and _) and number is a positive integer. Anything before, between, or after value and number is ignored. *mariadb-report* also needs the following MariaDB server variables: version, table_cache, max_connections, key_buffer_size, query_cache_size. These values can be specified in INFILE in the format "name = value" where name is one of the aforementioned server variables and value is a positive integer with or without a trailing M and possible periods (for version). For example, to specify an 18M key_buffer_size: key_buffer_size = 18M. Or, a 256 table_cache: table_cache = 256. The M implies Megabytes not million, so 18M means 18,874,368 not 18,000,000. If these server variables are not specified the following defaults are used (respectively) which may cause strange values to be reported: 0.0.0, 64, 100, 8M, 0.
`--no-mycnf`	Makes mariadb-report not read /.my.cnf which it does by default otherwise. `--user` and `--password` always override values from /.my.cnf.
`--outfile FILE`	After printing the report to screen, print the report to FILE too. Internally, mariadb-report always writes the report to a temp file first: `/tmp/mysqlreport.PID` on *nix, `c:sqlreport.PID` on Windows (PID is the script's process ID). Then it prints the temp file to screen. Then if `--outfile` is specified, the temp file is copied to OUTFILE. After `--email` (above), the temp file is deleted.
`--password`	As of version 2.3 `--password` can take the password on the command line like `--password FOO`. Using `--password` alone without giving a password on the command line causes mariadb-report to prompt for a password.
`--port PORT`	Port number.
`--qcache`	Print Query Cache report.

`--sas`	Print report for Select_ and Sort_ status values (after Questions report). See MySQL Select and Sort Status Variables, archived here.
`--socket SOCKET`	For connections to localhost, the Unix socket file to use, or, on Windows, the name of the named pipe to use.
`--tab`	Print Threads, Aborted, and Bytes status reports (after Created temp report). The Threads report reports on all Threads_ status values.
`--user USERNAME`	Username.

1.3.36 mariadb-secure-installation

Note that many of the reasons for the existence of this script no longer apply (and therefore the guidelines in many online tutorials. In particular, from MariaDB 10.4, Unix socket authentication is applied by default, and there is usually no need to create a root password. See Authentication from MariaDB 10.4.

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadb-secure-installation` is a symlink to mysql_secure_installation, the script for enabling you to improve the security of your MariaDB installation.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mysql_secure_installation` is the symlink, and `mariadb-secure-installation` the binary name.

Contents

Description

`mariadb-secure-installation` is a shell script available on Unix systems, and enables you to improve the security of your MariaDB installation in the following ways:

- You can set a password for root accounts.
- You can remove root accounts that are accessible from outside the local host.
- You can remove anonymous-user accounts.
- You can remove the test database, which by default can be accessed by anonymous users.

`mariadb-secure-installation` can be invoked without arguments:

```
shell> mariadb-secure-installation
```

The script will prompt you to determine which actions to perform.

```
Example:
localhost:# mariadb-secure-installation

NOTE: RUNNING ALL PARTS OF THIS SCRIPT IS RECOMMENDED FOR ALL MariaDB
      SERVERS IN PRODUCTION USE!  PLEASE READ EACH STEP CAREFULLY!

In order to log into MariaDB to secure it, we'll need the current
password for the root user.  If you've just installed MariaDB, and
you haven't set the root password yet, the password will be blank,
so you should just press enter here.

Enter current password for root (enter for none):
OK, successfully used password, moving on...

Setting the root password ensures that nobody can log into the MariaDB
root user without the proper authorisation.

You already have a root password set, so you can safely answer 'n'.

Change the root password? [Y/n] n
 ... skipping.

By default, a MariaDB installation has an anonymous user, allowing anyone
to log into MariaDB without having to have a user account created for
them.  This is intended only for testing, and to make the installation
go a bit smoother.  You should remove them before moving into a
production environment.

Remove anonymous users? [Y/n] y
 ... Success!

Normally, root should only be allowed to connect from 'localhost'.  This
ensures that someone cannot guess at the root password from the network.

Disallow root login remotely? [Y/n] y
 ... Success!

By default, MariaDB comes with a database named 'test' that anyone can
access.  This is also intended only for testing, and should be removed
before moving into a production environment.

Remove test database and access to it? [Y/n] y
 - Dropping test database...
 ... Success!
 - Removing privileges on test database...
 ... Success!

Reloading the privilege tables will ensure that all changes made so far
will take effect immediately.

Reload privilege tables now? [Y/n] y
 ... Success!

Cleaning up...

All done!  If you've completed all of the above steps, your MariaDB
installation should now be secure.

Thanks for using MariaDB!
```

Options

`mariadb-secure-installation` accepts some options:

Option	Description
`--basedir=dir`	Base directory.

`--no-defaults`	Don't read default options from any option file.
`--defaults-file=#`	Only read default options from the given file #.
`--defaults-extra-file=#`	Read this file after the global files are read.

Other unrecognized options will be passed on to the server.

Option Files

In addition to reading options from the command-line, `mariadb-secure-installation` can also read options from option files. If an unknown option is provided to `mariadb-secure-installation` in an option file, then it is ignored.

The following options relate to how MariaDB command-line tools handles option files. They must be given as the first argument on the command-line:

Option	Description
`--no-defaults`	Don't read default options from any option file.
`--defaults-file=#`	Only read default options from the given file #.
`--defaults-extra-file=#`	Read this file after the global files are read.
`--defaults-group-suffix=#`	In addition to the default option groups, also read option groups with this suffix.

Option Groups

`mariadb-secure-installation` reads options from the following option groups from option files:

Group	Description
`[client]`	Options read by all MariaDB and MySQL client programs, which includes both MariaDB and MySQL clients. For example, `mysqldump`.
`[client-server]`	Options read by all MariaDB client programs and the MariaDB Server. This is useful for options like socket and port, which is common between the server and the clients.
`[client-mariadb]`	Options read by all MariaDB client programs.

Use With Galera Cluster

This script is not 100% safe for use with Galera Cluster as it directly manipulates the mysql.user/mysql.global_priv table, which is not transported by Galera to the other nodes.

You should run this script on the first node in the cluster before adding more nodes.

If you want to run this after the cluster is up and running you should find alternative ways.

Anyone can vote for this to be fixed at https://jira.mariadb.org/browse/MDEV-10112.

1.3.37 mariadb-setpermission

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadb-setpermission` is a symlink to `mysql_setpermission`, the script for assisting with adding users or databases or changing passwords in MariaDB.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mysql_setpermission` is the symlink, and `mariadb-setpermission` the binary name.

Contents

Syntax

```
mariadb-setpermission [options]
```

Description

mariadb-setpermission is a Perl script that was originally written and contributed by Luuk de Boer. It requires the DBI and DBD::mysql Perl modules to be installed. *mariadb-setpermission* can help you add users or databases or change passwords in MariaDB.

It interactively sets permissions in the MariaDB grant tables, but does not check permissions which have already been set in MariaDB. So if you can't connect to MariaDB using the permission you just added, take a look at the permissions which have already been set in MariaDB.

The account used when you connect determines which permissions you have when attempting to modify existing permissions in the grant tables.

mariadb-setpermission also reads options from the [client] and [perl] groups in the .my.cnf file in your home directory, if the file exists.

The following options are available:

Options

Option	Description
`--help`	Display a help message and exit.
`--host=host_name`	Connect to the MariaDB server on the given host.
`--password=password`	The password to use when connecting to the server. Note that the password value is not optional for this option, unlike for other MariaDB programs *Specifying a password on the command line should be considered insecure. You can use an option file to avoid giving the password on the command line.*
`--port=port_num`	The TCP/IP port number to use for the connection.
`--socket=path`	For connections to localhost, the Unix socket file to use.
`--user=user_name`	The MariaDB user name to use when connecting to the server.

Example

```
./mariadb-setpermission --user=msandbox --password=msandbox --host=127.0.0.1 --port=11200
##########################################################################
## Welcome to the permission setter 1.4 for MariaDB.
## made by Luuk de Boer
##########################################################################
What would you like to do:
  1. Set password for an existing user.
  2. Create a database + user privilege for that database
     and host combination (user can only do SELECT)
  3. Create/append user privilege for an existing database
     and host combination (user can only do SELECT)
  4. Create/append broader user privileges for an existing
     database and host combination
     (user can do SELECT,INSERT,UPDATE,DELETE)
  5. Create/append quite extended user privileges for an
     existing database and host combination (user can do
     SELECT,INSERT,UPDATE,DELETE,CREATE,DROP,INDEX,
     LOCK TABLES,CREATE TEMPORARY TABLES)
  6. Create/append full privileges for an existing database
     and host combination (user has FULL privilege)
  7. Remove all privileges for for an existing database and
     host combination.
     (user will have all permission fields set to N)
  0. exit this program
```

1.3.38 mariadb-show

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadb-show` is a symlink to `mysqlshow`, the script showing the structure of a MariaDB database.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mysqlshow` is the symlink, and `mariadb-show` the binary name.

Contents

Shows the structure of a MariaDB database (databases, tables, columns and indexes). You can also use SHOW DATABASES, SHOW TABLES, SHOW COLUMNS, SHOW INDEX and SHOW TABLE STATUS, as well as the Information Schema tables (TABLES, COLUMNS, STATISTICS), to get similar functionality.

Using mariadb-show

```
mariadb-show [OPTIONS] [database [table [column]]]
```

The output displays only the names of those databases, tables, or columns for which you have some privileges.

If no database is given then all matching databases are shown. If no table is given, then all matching tables in database are shown. If no column is given, then all matching columns and column types in table are shown.

If the last argument contains a shell or SQL wildcard (*,?,% or _) then only what's matched by the wildcard is shown. If a database name contains any underscores, those should be escaped with a backslash (some Unix shells require two) to get a list of the proper tables or columns. “*” and “?” characters are converted into SQL “%” and “_” wildcard characters. This might cause some confusion when you try to display the columns for a table with a “_” in the name, because in this case, mariadb-show shows you only the table names that match the pattern. This is easily fixed by adding an extra “%” last on the command line as a separate argument.

Options

`mariadb-show` supports the following options:

Option	Description
`-c name, --character-sets-dir=name`	Directory for character set files.
`-C, --compress`	Use compression in server/client protocol if both support it.
`--count`	Show number of rows per table (may be slow for non-MyISAM tables).
`-# [name], --debug[=name]`	Output debug log. Typical is `d:t:o,filename`, the default is `d:t:o`.
`--debug-check`	Check memory and open file usage at exit.
`--debug-info`	Print some debug info at exit.
`--default-auth=name`	Default authentication client-side plugin to use.
`--default-character-set=name`	Set the default character set.
`--defaults-extra-file=name`	Read the file *name* after the global files are read. Must be given as the first option.

`--defaults-file=name`	Only read default options from the given file *name*. Must be given as the first option.
`--defaults-group-suffix=suffix`	In addition to the given groups, also read groups with this suffix.
`-?, --help`	Display help and exit.
`-h name, --host=name`	Connect to the MariaDB server on the given host.
`-k, --keys`	Show indexes for table.
`--no-defaults`	Don't read default options from any option file. Must be given as the first option.
`-p[password], --password[=password]`	Password to use when connecting to server. If password is not given, it's solicited on the command line. Specifying a password on the command line should be considered insecure. You can use an option file to avoid giving the password on the command line.
`-W, --pipe`	On Windows, connect to the server via a named pipe. This option applies only if the server supports named-pipe connections.
`--plugin-dir=name`	Directory for client-side plugins.
`-P num, --port=num`	Port number to use for connection or 0 for default to, in order of preference, my.cnf, $MYSQL_TCP_PORT, /etc/services, built-in default (3306).
`--print-defaults`	Print the program argument list and exit. Must be given as the first option.
`--protocol=name`	The protocol to use for connection (tcp, socket, pipe, memory).
`--shared-memory-base-name=name`	On Windows, the shared-memory name to use, for connections made using shared memory to a local server. The default value is `MYSQL`. The shared-memory name is case sensitive. The server must be started with the `--shared-memory` option to enable shared-memory connections.
`-t, --show-table-type`	Show table type column, as in SHOW FULL TABLES. The type is BASE TABLE or VIEW.
`-S name, --socket=name`	For connections to localhost, the Unix socket file to use, or, on Windows, the name of the named pipe to use.
`--ssl`	Enables TLS. TLS is also enabled even without setting this option when certain other TLS options are set. Starting with MariaDB 10.2, the `--ssl` option will not enable verifying the server certificate by default. In order to verify the server certificate, the user must specify the `--ssl-verify-server-cert` option.
`--ssl-ca=name`	Defines a path to a PEM file that should contain one or more X509 certificates for trusted Certificate Authorities (CAs) to use for TLS. This option requires that you use the absolute path, not a relative path. See Secure Connections Overview: Certificate Authorities (CAs) for more information. This option implies the `--ssl` option.
`--ssl-capath=name`	Defines a path to a directory that contains one or more PEM files that should each contain one X509 certificate for a trusted Certificate Authority (CA) to use for TLS. This option requires that you use the absolute path, not a relative path. The directory specified by this option needs to be run through the `openssl rehash` command. See Secure Connections Overview: Certificate Authorities (CAs) for more information. This option is only supported if the client was built with OpenSSL. If the client was built with yaSSL, GnuTLS, or Schannel, then this option is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms. This option implies the `--ssl` option.
`--ssl-cert=name`	Defines a path to the X509 certificate file to use for TLS. This option requires that you use the absolute path, not a relative path. This option implies the `--ssl` option.
`--ssl-cipher=name`	List of permitted ciphers or cipher suites to use for TLS. This option implies the `--ssl` option.
`--ssl-key=name`	Defines a path to a private key file to use for TLS. This option requires that you use the absolute path, not a relative path. This option implies the `--ssl` option.

`--ssl-crl=name`	Defines a path to a PEM file that should contain one or more revoked X509 certificates to use for TLS. This option requires that you use the absolute path, not a relative path. See Secure Connections Overview: Certificate Revocation Lists (CRLs) for more information. This option is only supported if the client was built with OpenSSL or Schannel. If the client was built with yaSSL or GnuTLS, then this option is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms. This option implies the `--ssl` option.
`--ssl-crlpath=name`	Defines a path to a directory that contains one or more PEM files that should each contain one revoked X509 certificate to use for TLS. This option requires that you use the absolute path, not a relative path. The directory specified by this option needs to be run through the `openssl rehash` command. See Secure Connections Overview: Certificate Revocation Lists (CRLs) for more information. This option is only supported if the client was built with OpenSSL. If the client was built with yaSSL, GnuTLS, or Schannel, then this option is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms. This option implies the `--ssl` option.
`--ssl-verify-server-cert`	Enables (or disables) server certificate verification. This option is disabled by default.
`-i , --status`	Shows a lot of extra information about each table. See the INFORMATION_SCHEMA.TABLES table for more details on the returned information.
`--tls-version=name`	This option accepts a comma-separated list of TLS protocol versions. A TLS protocol version will only be enabled if it is present in this list. All other TLS protocol versions will not be permitted. See Secure Connections Overview: TLS Protocol Versions for more information. This option was added in MariaDB 10.4.6.
`-u , --user=name`	User for login if not current user.
`-v , --verbose`	More verbose output; you can use this multiple times to get even more verbose output.
`-V , --version`	Output version information and exit.

Option Files

In addition to reading options from the command-line, `mariadb-show` can also read options from option files. If an unknown option is provided to `mariadb-show` in an option file, then it is ignored.

The following options relate to how MariaDB command-line tools handles option files. They must be given as the first argument on the command-line:

Option	Description
`--print-defaults`	Print the program argument list and exit.
`--no-defaults`	Don't read default options from any option file.
`--defaults-file=#`	Only read default options from the given file #.
`--defaults-extra-file=#`	Read this file after the global files are read.
`--defaults-group-suffix=#`	In addition to the default option groups, also read option groups with this suffix.

In MariaDB 10.2 and later, `mariadb-show` is linked with MariaDB Connector/C. However, MariaDB Connector/C does not yet handle the parsing of option files for this client. That is still performed by the server option file parsing code. See MDEV-19035 for more information.

Option Groups

`mariadb-show` reads options from the following option groups from option files:

Group	Description
`[mysqlshow]`	Options read by `mysqlshow`, which includes both MariaDB Server and MySQL Server.
`[mariadb-show]`	Options read by `mariadb-show`. Available starting with MariaDB 10.4.6.
`[client]`	Options read by all MariaDB and MySQL client programs, which includes both MariaDB and MySQL clients. For example, `mysqldump`.

`[client-server]`	Options read by all MariaDB client programs and the MariaDB Server. This is useful for options like socket and port, which is common between the server and the clients.
`[client-mariadb]`	Options read by all MariaDB client programs.

Examples

Getting a list of databases:

```
bin/mariadb-show
+--------------------+
|     Databases      |
+--------------------+
| information_schema |
| test               |
+--------------------+
```

Getting a list of tables in the `test` database:

```
bin/mariadb-show test
Database: test
+---------+
| Tables  |
+---------+
| author  |
| book    |
| city    |
| country |
+---------+
```

Getting a list of columns in the `test.book` table:

```
bin/mariadb-show test book
Database: test  Table: book
+-----------+-----------------------+--------------------+------+-----+---------+-------------
---+---------------------------------+---------+
| Field     | Type                  | Collation          | Null | Key | Default | Extra
| Privileges                      | Comment |
+-----------+-----------------------+--------------------+------+-----+---------+-------------
---+---------------------------------+---------+
| id        | mediumint(8) unsigned |                    | NO   | PRI |         |
auto_increment | select,insert,update,references |         |
| title     | varchar(200)          | latin1_swedish_ci | NO   |     |         |
| select,insert,update,references |         |
| author_id | smallint(5) unsigned  |                    | NO   | MUL |         |
| select,insert,update,references |         |
+-----------+-----------------------+--------------------+------+-----+---------+-------------
---+---------------------------------+---------+
```

1.3.39 mariadb-slap

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadb-slap` is a symlink to `mysqlslap`, the tool for load-testing MariaDB.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mysqlslap` is the symlink, and `mariadb-slap` the binary name.

Contents

`mariadb-slap` is a tool for load-testing MariaDB. It allows you to emulate multiple concurrent connections, and run a set of queries multiple times.

It returns a benchmark including the following information:

- Average number of seconds to run all queries
- Minimum number of seconds to run all queries
- Maximum number of seconds to run all queries
- Number of clients running queries
- Average number of queries per client

Using mariadb-slap

The command to use `mariadb-slap` and the general syntax is:

```
mariadb-slap [options]
```

Options

`mariadb-slap` supports the following options:

Option	Description
`-a`, `--auto-generate-sql`	Generate SQL statements automatically when they are not supplied in files or via command options.
`--auto-generate-sql-add-autoincrement`	Add an AUTO_INCREMENT column to auto-generated tables.
`--auto-generate-sql-execute-number=num`	Specify how many queries to generate automatically.
`--auto-generate-sql-guid-primary`	Add GUID based primary keys to auto-generated tables.
`--auto-generate-sql-load-type=name`	Specify the test load type. The allowable values are `read` (scan tables), `write` (insert into tables), `key` (read primary keys), `update` (update primary keys), or `mixed` (half inserts, half scanning selects). The default is mixed.
`--auto-generate-sql-secondary-indexes=num`	Number of secondary indexes to add to auto-generated tables. By default, none are added.
`--auto-generate-sql-unique-query-number=num`	Number of unique queries to generate for automatic tests. For example, if you run a key test that performs 1000 selects, you can use this option with a value of 1000 to run 1000 unique queries, or with a value of 50 to perform 50 different selects. The default is 10.
`--auto-generate-sql-unique-write-number=num`	Number of unique queries to generate for `auto-generate-sql-write-number`.
`--auto-generate-sql-write-number=num`	Number of row inserts to perform for each thread. The default is 100.
`--commit=num`	Number of statements to execute before committing. The default is 0.
`-C`, `--compress`	Use compression in server/client protocol if both support it.
`-c name`, `--concurrency=name`	Number of clients to simulate for query to run.

`--create=name`	File or string containing the statement to use for creating the table.
`--create-schema=name`	Schema to run tests in.
`--csv[=name]`	Generate comma-delimited output to named file or to standard output if no file is named.
`-# , --debug[=options]`	For debug builds, write a debugging log. A typical debug_options string is `d:t:o,file_name`. The default is `d:t:o,/tmp/mariadb-slap.trace`.
`--debug-check`	Check memory and open file usage at exit.
`-T, --debug-info`	Print some debug info at exit.
`--default-auth=name`	Default authentication client-side plugin to use.
`--defaults-extra-file=name`	Read this file after the global files are read. Must be given as the first option.
`--defaults-file=name`	Only read default options from the given file *name* Must be given as the first option.
`-F name , --delimiter=name`	Delimiter to use in SQL statements supplied in file or command line.
`--detach=num`	Detach (close and reopen) connections after the specified number of requests. The default is 0 (connections are not detached).
`-e name , --engine=name`	Comma separated list of storage engines to use for creating the table. The test is run for each engine. You can also specify an option for an engine after a #:#, for example `memory:max_row=2300`.
`-? , --help`	Display help and exit.
`-h name , --host=name`	Connect to the MariaDB server on the given host.
`--init-command=name`	SQL Command to execute when connecting to the MariaDB server. Will automatically be re-executed when reconnecting. Added in MariaDB 5.5.34.
`-i num , --iterations=num`	Number of times to run the tests.
`--no-defaults`	Don't read default options from any option file. Must be given as the first option.
`--no-drop`	Do not drop any schema created during the test after the test is complete.
`-x name , --number-char-cols=name`	Number of VARCHAR columns to create in table if specifying `--auto-generate-sql`.
`-y name , --number-int-cols=name`	Number of INT columns to create in table if specifying `--auto-generate-sql`.
`--number-of-queries=num`	Limit each client to approximately this number of queries. Query counting takes into account the statement delimiter. For example, if you invoke as follows, `mariadb-slap --delimiter=";" --number-of-queries=10 --query="use test;insert into t values(null)"`, the #;# delimiter is recognized so that each instance of the query string counts as two queries. As a result, 5 rows (not 10) are inserted.
`--only-print`	Do not connect to the databases, but instead print out what would have been done.
`-p[password] , --password[=password]`	Password to use when connecting to server. If password is not given it's asked from the command line. Specifying a password on the command line should be considered insecure. You can use an option file to avoid giving the password on the command line.
`-W , --pipe`	On Windows, connect to the server via a named pipe. This option applies only if the server supports named-pipe connections.
`--plugin-dir=name`	Directory for client-side plugins.
`-P num , --port=num`	Port number to use for connection.

`--post-query=name`	Query to run or file containing query to execute after tests have completed. This execution is not counted for timing purposes.
`--post-system=name`	system() string to execute after tests have completed. This execution is not counted for timing purposes.
`--pre-query=name`	Query to run or file containing query to execute before running tests. This execution is not counted for timing purposes.
`--pre-system=name`	system() string to execute before running tests. This execution is not counted for timing purposes.
`--print-defaults`	Print the program argument list and exit. Must be given as the first option.
`--protocol=name`	The protocol to use for connection (tcp, socket, pipe, memory).
`-q name , --query=name`	Query to run or file containing query to run.
`--shared-memory-base-name`	Shared-memory name to use for Windows connections using shared memory to a local server (started with the `--shared-memory` option). Case-sensitive.
`-s , --silent`	Run program in silent mode - no output.
`-S , --socket=name`	For connections to localhost, the Unix socket file to use, or, on Windows, the name of the named pipe to use.
`--ssl`	Enables TLS. TLS is also enabled even without setting this option when certain other TLS options are set. The `--ssl` option will not enable verifying the server certificate by default. In order to verify the server certificate, the user must specify the `--ssl-verify-server-cert` option.
`--ssl-ca=name`	Defines a path to a PEM file that should contain one or more X509 certificates for trusted Certificate Authorities (CAs) to use for TLS. This option requires that you use the absolute path, not a relative path. See Secure Connections Overview: Certificate Authorities (CAs) for more information. This option implies the `--ssl` option.
`--ssl-capath=name`	Defines a path to a directory that contains one or more PEM files that should each contain one X509 certificate for a trusted Certificate Authority (CA) to use for TLS. This option requires that you use the absolute path, not a relative path. The directory specified by this option needs to be run through the `openssl rehash` command. See Secure Connections Overview: Certificate Authorities (CAs) for more information. This option is only supported if the client was built with OpenSSL or yaSSL. If the client was built with GnuTLS or Schannel, then this option is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms. This option implies the `--ssl` option.
`--ssl-cert=name`	Defines a path to the X509 certificate file to use for TLS. This option requires that you use the absolute path, not a relative path. This option implies the `--ssl` option.
`--ssl-cipher=name`	List of permitted ciphers or cipher suites to use for TLS. This option implies the `--ssl` option.
`--ssl-crl=name`	Defines a path to a PEM file that should contain one or more revoked X509 certificates to use for TLS. This option requires that you use the absolute path, not a relative path. See Secure Connections Overview: Certificate Revocation Lists (CRLs) for more information. This option is only supported if the client was built with OpenSSL or Schannel. If the client was built with yaSSL or GnuTLS, then this option is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.
`--ssl-crlpath=name`	Defines a path to a directory that contains one or more PEM files that should each contain one revoked X509 certificate to use for TLS. This option requires that you use the absolute path, not a relative path. The directory specified by this option needs to be run through the `openssl rehash` command. See Secure Connections Overview: Certificate Revocation Lists (CRLs) for more information. This option is only supported if the client was built with OpenSSL. If the client was built with yaSSL, GnuTLS, or Schannel, then this option is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.
`--ssl-key=name`	Defines a path to a private key file to use for TLS. This option requires that you use the absolute path, not a relative path. This option implies the `--ssl` option.
`--ssl-verify-server-cert`	Enables server certificate verification. This option is disabled by default.
`-u , --user=name`	User for login if not current user.

`-v , --verbose`	More verbose output; you can use this multiple times to get even more verbose output.
`-V , --version`	Output version information and exit.

Option Files

In addition to reading options from the command-line, `mariadb-slap` can also read options from option files. If an unknown option is provided to `mariadb-slap` in an option file, then it is ignored.

The following options relate to how MariaDB command-line tools handles option files. They must be given as the first argument on the command-line:

Option	Description
`--print-defaults`	Print the program argument list and exit.
`--no-defaults`	Don't read default options from any option file.
`--defaults-file=#`	Only read default options from the given file #.
`--defaults-extra-file=#`	Read this file after the global files are read.
`--defaults-group-suffix=#`	In addition to the default option groups, also read option groups with this suffix.

`mariadb-slap` is linked with MariaDB Connector/C. However, MariaDB Connector/C does not yet handle the parsing of option files for this client. That is still performed by the server option file parsing code. See MDEV-19035 for more information.

Option Groups

`mariadb-slap` reads options from the following option groups from option files:

Group	Description
`[mysqlslap]`	Options read by `mariadb-slap`, which includes both MariaDB Server and MySQL Server.
`[mariadb-slap]`	Options read by `mariadb-slap`. Available starting with MariaDB 10.4.6.
`[client]`	Options read by all MariaDB and MySQL client programs, which includes both MariaDB and MySQL clients. For example, `mysqldump`.
`[client-server]`	Options read by all MariaDB client programs and the MariaDB Server. This is useful for options like socket and port, which is common between the server and the clients.
`[client-mariadb]`	Options read by all MariaDB client programs.

Examples

Create a table with data, and then query it with 40 simultaneous connections 100 times each.

```
mariadb-slap
 --delimiter=";"
 --create="CREATE TABLE t (a int);INSERT INTO t VALUES (5)"
 --query="SELECT * FROM t"
 --concurrency=40
 --iterations=100

Benchmark
 Average number of seconds to run all queries: 0.010 seconds
 Minimum number of seconds to run all queries: 0.009 seconds
 Maximum number of seconds to run all queries: 0.020 seconds
 Number of clients running queries: 40
 Average number of queries per client: 1
```

Using files to store the create and query SQL. Each file can contain multiple statements separated by the specified delimiter.

```
mariadb-slap
 --create=define.sql
 --query=query.sql
 --concurrency=10
 --iterations=20
 --delimiter=";"

Benchmark
 Average number of seconds to run all queries: 0.002 seconds
 Minimum number of seconds to run all queries: 0.002 seconds
 Maximum number of seconds to run all queries: 0.004 seconds
 Number of clients running queries: 10
 Average number of queries per client: 1
```

1.3.40 mariadb-tzinfo-to-sql

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadb-tzinfo-to-sql` is a symlink to `mysql_tzinfo_to_sql`, the tool for loading time zones on systems that have a zoneinfo database.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mysql_tzinfo_to_sql` is the symlink, and `mariadb-tzinfo-to-sql` the binary name.

`mariadb-tzinfo-to-sql` is a utility used to load time zones on systems that have a zoneinfo database to load the time zone tables (time_zone, time_zone_leap_second, time_zone_name, time_zone_transition and time_zone_transition_type) into the mysql database.

Most Linux, Mac OS X, FreeBSD and Solaris systems will have a zoneinfo database - Windows does not. The database is commonly found in the /usr/share/zoneinfo directory, or, on Solaris, the /usr/share/lib/zoneinfo directory.

Usage

`mariadb-tzinfo-to-sql` can be called in several ways. The output is usually passed straight to the mariadb client for direct loading in the mysql database.

```
shell> mariadb-tzinfo-to-sql timezone_dir
shell> mariadb-tzinfo-to-sql timezone_file timezone_name
shell> mariadb-tzinfo-to-sql --leap timezone_file
```

Examples

Most commonly, the whole directory is passed:

```
shell>mariadb-tzinfo-to-sql /usr/share/zoneinfo | mariadb -u root mysql
```

Load a single time zone file, `timezone_file`, corresponding to the time zone called `timezone_name`.

```
shell> mariadb-tzinfo-to-sql timezone_file timezone_name | mariadb -u root mysql
```

A separate command for each time zone and time zone file the server needs is required.

To account for leap seconds, use:

```
shell> mariadb-tzinfo-to-sql --leap timezone_file | mariadb -u root mysql
```

After populating the time zone tables, you should usually restart the server so that the new time zone data is correctly loaded.

1.3.41 mariadb-upgrade

`mariadb-upgrade` is a tool that checks and updates your tables to the latest version.

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadb-upgrade` is a symlink to `mysql_upgrade`, the tool that checks and updates your tables to the latest version.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mysql_upgrade` is the symlink, and `mariadb-upgrade` the binary name.

MariaDB starting with 10.4.23

Starting from `mysql_upgrade` / `mariadb-upgrade` 2.0, the user running the upgrade tool must have write access to `datadir/mysql_upgrade_info`, so that the tool can write the current MariaDB version into the file. `mysql-upgrade` was updated in MariaDB 10.2.42, MariaDB 10.3.33, MariaDB 10.4.23, MariaDB 10.5.14, MariaDB 10.6.6, MariaDB 10.7.2 and newer.

Contents

You should run `mariadb-upgrade` after upgrading from one major MySQL/MariaDB release to another, such as from MySQL 5.0 to MariaDB 10.4 or MariaDB 10.4 to MariaDB 10.5. You also have to use `mariadb-upgrade` after a direct "horizontal" migration, for example from MySQL 5.5.40 to MariaDB 5.5.40. It's also safe to run `mariadb-upgrade` for minor upgrades, as if there are no incompatibilities nothing is changed.

It needs to be run as a user with write access to the data directory.

`mariadb-upgrade` is run after starting the new MariaDB server. Running it before you shut down the old version will not hurt anything and will allow you to make sure it works and figure out authentication for it ahead of time.

It is recommended to make a backup of all the databases before running `mariadb-upgrade`.

In most cases, `mariadb-upgrade` should just take a few seconds. The main work of `mariadb-upgrade` is to:

- Update the system tables in the `mysql` database to the latest version (normally just add new fields to a few tables).
- Check that all tables are up to date (runs CHECK TABLE table_name FOR UPGRADE). For tables that are not up to date, runs ALTER TABLE table_name FORCE on the table to update it. A table is not up to date if:
  - The table uses an index for which there has been a collation change (rare)
  - A format change in the storage engine requires an update (very rare)

Using mariadb-upgrade

```
mariadb-upgrade [--force] [--user=# --password=#
  --host=hostname --port=# --socket=#
  --protocol=tcp|socket|pipe|memory
  --verbose] [OTHER_OPTIONS]
```

`mariadb-upgrade` is mainly a framework to call mariadb-check. `mariadb-upgrade` works by doing the following operations:

```
# Find out path to datadir
echo "show show variables like 'datadir'" | mysql
mariadb-check --no-defaults --check-upgrade --auto-repair --databases mysql
mysql_fix_privilege_tables
mariadb-check --no-defaults --all-databases --fix-db-names --fix-table-names
mariadb-check --no-defaults --check-upgrade --all-databases --auto-repair
```

The connect options given to `mariadb-upgrade` are passed along to mariadb-check and mysql.

The `mysql_fix_privilege_tables` script is not actually called; it's included as part of `mariadb-upgrade`

If you have a problem with `mariadb-upgrade` try run it in very verbose mode:

```
mariadb-upgrade --verbose --verbose other-options
```

`mariadb-upgrade` also saves the MariaDB version number in a file named `mysql_upgrade_info` in the data directory. This is used to quickly check whether all tables have been checked for this release so that table-checking can be skipped. For this reason, `mariadb-upgrade` needs to be run as a user with write access to the data directory. To ignore this file and perform the check regardless, use the `--force` option.

Options

`mariadb-upgrade` supports the following options:

Option	Description	Version
`-?, --help`	Display this help message and exit.	
`--basedir=path`	Old option accepted for backward compatibility but ignored.	
`--character-sets-dir=path`	Old option accepted for backward compatibility but ignored.	
`check-if-upgrade-is-needed`	Do a quick check if upgrade is needed. Returns 0 if yes, 1 if no.	2.0
`--compress=name`	Old option accepted for backward compatibility but ignored.	
`--datadir=name`	Old option accepted for backward compatibility but ignored.	
`-# [name], --debug[=name]`	For debug builds, output debug log.	
`--debug-check`	Check memory and open file usage at exit.	
`-T, --debug-info`	Print some debug info at exit.	
`--default-character-set=name`	Old option accepted for backward compatibility but ignored.	
`-f, --force`	Force execution of mariadb-check even if `mariadb-upgrade` has already been executed for the current version of MariaDB. Ignores `mysql_upgrade_info`.	
`-h, --host=name`	Connect to MariaDB on the given host.	
`-p, --password[=name]`	Password to use when connecting to server. If password is not given, it's solicited on the command line (which should be considered insecure). You can use an option file to avoid giving the password on the command line.	
`-P, --port=name`	Port number to use for connection or 0 for default to, in order of preference, my.cnf, the MYSQL_TCP_PORT environment variable, /etc/services, built-in default (3306).	
`--protocol=name`	The protocol to use for connection (tcp, socket, pipe, memory).	
`--silent`	Print less information.	
`-S, --socket=name`	For connections to localhost, the Unix socket file to use, or, on Windows, the name of the named pipe to use.	
`--ssl`	Enables TLS. TLS is also enabled even without setting this option when certain other TLS options are set. Starting with MariaDB 10.2, the `--ssl` option will not enable verifying the server certificate by default. In order to verify the server certificate, the user must specify the `--ssl-verify-server-cert` option.	
`--ssl-ca=name`	Defines a path to a PEM file that should contain one or more X509 certificates for trusted Certificate Authorities (CAs) to use for TLS. This option requires that you use the absolute path, not a relative path. See Secure Connections Overview: Certificate Authorities (CAs) for more information. This option implies the `--ssl` option.	

`--ssl-capath=name`	Defines a path to a directory that contains one or more PEM files that should each contain one X509 certificate for a trusted Certificate Authority (CA) to use for TLS. This option requires that you use the absolute path, not a relative path. The directory specified by this option needs to be run through the openssl rehash command. See Secure Connections Overview: Certificate Authorities (CAs) for more information. This option is only supported if the client was built with OpenSSL or yaSSL. If the client was built with GnuTLS or Schannel, then this option is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms. This option implies the `--ssl` option.	
`--ssl-cert=name`	Defines a path to the X509 certificate file to use for TLS. This option requires that you use the absolute path, not a relative path. This option implies the `--ssl` option.	
`--ssl-cipher=name`	List of permitted ciphers or cipher suites to use for TLS. This option implies the `--ssl` option.	
`--ssl-crl=name`	Defines a path to a PEM file that should contain one or more revoked X509 certificates to use for TLS. This option requires that you use the absolute path, not a relative path. See Secure Connections Overview: Certificate Revocation Lists (CRLs) for more information. This option is only supported if the client was built with OpenSSL or Schannel. If the client was built with yaSSL or GnuTLS, then this option is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.	
`--ssl-crlpath=name`	Defines a path to a directory that contains one or more PEM files that should each contain one revoked X509 certificate to use for TLS. This option requires that you use the absolute path, not a relative path. The directory specified by this option needs to be run through the openssl rehash command. See Secure Connections Overview: Certificate Revocation Lists (CRLs) for more information. This option is only supported if the client was built with OpenSSL. If the client was built with yaSSL, GnuTLS, or Schannel, then this option is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.	
`--ssl-key=name`	Defines a path to a private key file to use for TLS. This option requires that you use the absolute path, not a relative path. This option implies the `--ssl` option.	
`--ssl-verify-server-cert`	Enables server certificate verification. This option is disabled by default.	
`-t, --tmpdir=name`	Directory for temporary files.	
`-s, --upgrade-system-tables`	Only upgrade the system tables in the mysql database. Tables in other databases are not checked or touched.	
`-u, --user=name`	User for login if not current user.	
`-v, --verbose`	Display more output about the process, using it twice will print connection arguments; using it 3 times will print out all CHECK, RENAME and ALTER TABLE commands used during the check phase; using it 4 times will also write out all mariadb-check commands used.	
`-V, --version`	Output version information and exit.	
`-k, --version-check`	Run this program only if its 'server version' matches the version of the server to which it's connecting check. Note: the 'server version' of the program is the version of the MariaDB server with which it was built/distributed. (Defaults to on; use `--skip-version-check` to disable.)	
`--write-binlog`	All commands including those run by mariadb-check are written to the binary log. Disabled by default. Before MariaDB 10.0.6 and MariaDB 5.5.34, this was enabled by default, and `--skip-write-binlog` should be used when commands should not be sent to replication slaves.	

mariadb-upgrade 2.0

`mariadb-upgrate/mysql_upgrade 2.0` was introduced in MariaDB 10.2.42, MariaDB 10.3.33, MariaDB 10.4.23, MariaDB 10.5.14, MariaDB 10.6.6, MariaDB 10.7.2.

Previously the tool first ran the upgrade process and then created the `datadir/mysql_upgrade_info` file. If the file could

not be created because of permissions (`mariadb-upgrade` did not have rights to create the file), `mariadb-upgrad` gave an error, but this was often ignored. One effect of not being able to create the `mysql_upgrade_info` file was that every new `mariadb-upgrade` run would have to do a full upgrade check, which can take a while if there are a lot of tables.

`mariadb-upgrade` 2.0 fixes the following issues:

- The `datadir/mysql_upgrade_info` is now created at the start of the upgrade process and locked. This ensures that two `mariadb-upgrade` processes cannot be run in parallel, which can cause deadlocks (MDEV-27068). One side-effect of this is that `mariadb-upgrade` has to have write access to `datadir` , which means it has to be run as as the user that installed MariaDB, normally 'mysql' or 'root' .
- One can use `mariadb-upgrade --force --force` to force the upgrade to be run, even if there was no version change or if one doesn't have write access to `datadir` . Note that if this option is used, the next `mariadb-upgrade` run will assume that there is a major version change and the upgrade must be done (again).
- The upgrade will only be done if there is a major server version change (10.4.X -> 10.5.X). This will avoid unnecessary upgrades.
- New option added: `--check-if-upgrade-is-needed` . If this is used, `mariadb-upgrade` will return 0 if there has been a major version change and one should run `mariadb-upgrade` . If not upgrade is need, 1 will be returned.
- `--verbose` writes more information, including from which version to which version the upgrade will be done.
- Better messages when there is no need to run `mariadb-upgrade` .

Option Files

In addition to reading options from the command-line, `mariadb-upgrade` can also read options from option files. If an unknown option is provided to `mariadb-upgrade` in an option file, then it is ignored.

The following options relate to how MariaDB command-line tools handles option files. They must be given as the first argument on the command-line:

Option	Description
`--print-defaults`	Print the program argument list and exit.
`--no-defaults`	Don't read default options from any option file.
`--defaults-file=#`	Only read default options from the given file #.
`--defaults-extra-file=#`	Read this file after the global files are read.
`--defaults-group-suffix=#`	In addition to the default option groups, also read option groups with this suffix.

In MariaDB 10.2 and later, `mariadb-upgrade` is linked with MariaDB Connector/C . However, MariaDB Connector/C does not yet handle the parsing of option files for this client. That is still performed by the server option file parsing code. See MDEV-19035 for more information.

Option Groups

`mariadb-upgrade` reads options from the following option groups from option files:

Group	Description
`[mysql_upgrade]`	Options read by `mariadb-upgrade` , which includes both MariaDB Server and MySQL Server.
`[mariadb-upgrade]`	Options read by `mariadb-upgrade` . Available starting with MariaDB 10.4.6.
`[client]`	Options read by all MariaDB and MySQL client programs, which includes both MariaDB and MySQL clients. For example, `mysqldump` .
`[client-server]`	Options read by all MariaDB client programs and the MariaDB Server. This is useful for options like socket and port, which is common between the server and the clients.
`[client-mariadb]`	Options read by all MariaDB client programs.

Differences Between mysql_upgrade in MariaDB and MySQL

This is as of MariaDB 5.1.50 :

- MariaDB will convert long table names properly.

- MariaDB will convert InnoDB tables (no need to do a dump/restore or `ALTER TABLE`).
- MariaDB will convert old archive tables to the new 5.1 format.
- "mysql_upgrade --verbose" will run "mariadb-check --verbose" so that you get more information of what is happening. Running with 3 times --verbose will in MariaDB 10.0 print out all CHECK, RENAME and ALTER TABLE commands executed.
- The mysql.event table is upgraded live; no need to restart the server to use events if the event table has changed (MariaDB 10.0.22 and MariaDB 10.1.9).
- More descriptive output.

Speeding Up mariadb-upgrade

- If you are sure that all your tables are up to date with the current version, then you can run `mariadb-upgrade --upgrade-system-tables`, which will only fix your system tables in the mysql database to be compatible with the latest version.

The main reason to run `mariadb-upgrade` on all your tables is to allow it to check that:

- There has not been any change in table formats between versions.
  - This has not happened since MariaDB 5.1.
- If some of the tables are using an index for which we have changed sort order.
  - This has not happened since MariaDB 5.5.

If you are 100% sure this applies to you, you can just run `mariadb-upgrade` with the `--upgrade-system-tables` option.

Symptoms of Not Having Run mariadb-upgrade When It Was Needed

- Errors in the error log that some system tables don't have all needed columns.
- Updates or searches may not find the record they are attempting to update or search for.
- CHECKSUM TABLE may report the wrong checksum for MyISAM or Aria tables.
- The error message "Cannot load from mysql.proc. The table is probably corrupted."

To fix issues like this, run `mariadb-upgrade`, mariadb-check, CHECK TABLE and if needed REPAIR TABLE on the wrong table.

Other Uses

- `mariadb-upgrade` will re-create any missing tables in the mysql database. It will not touch any data in existing tables.

1.3.42 mariadb-waitpid

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadb-waitpid` is a symlink to `mysql_waitpid`, the utility for terminating processes.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mysql_waitpid` is the symlink, and `mariadb-waitpid` the binary name.From MariaDB 10.5.2, `mysql_waitpid` is the symlink, and `mariadb-waitpid` the binary name.

`mysql_pid` is a utility for terminating processes. It runs on Unix-like systems, making use of the `kill()` system call.

Usage

```
mariadb-waitpid [options] pid time
```

Description

`mariadb-waitpid` sends signal 0 to the process *pid* and waits up to *time* seconds for the process to terminate. *pid* and *time* must be positive integers.

Returns 0 if the process terminates in time, or does not exist, and 1 otherwise.

Signal 1 is used if the kill() system call cannot handle signal 0

Options

Option	Description
`-?`, `--help`	Display help and exit
`-I`, `--help`	Synonym for -?
`-v`, `--verbose`	Be more verbose. Give a warning, if kill can't handle signal 0
`-V`, `--version`	Print version information and exit

1.3.43 Legacy Clients and Utilities

Category for removed, deprecated or unmaintained MariaDB clients and utilities.

Percona XtraBackup

Open source tool for performing hot backups of MariaDB, MySQL and Percona Server databases.

MySQL Sandbox

Installing multiple MariaDB versions in isolation.

mysqlaccess

Symlink or old name for mariadb-access.

mysqladmin

Old name or symlink for mariadb-admin.

mysqlcheck

Symlink or old name for mariadb-check.

mysqldump

Symlink or old name for mariadb-dump.

mysqldumpslow

Symlink or old name for mariadb-dumpslow.

mysqlhotcopy

Symlink or old name for mariadb-hotcopy.

mysqlimport

Symlink or old name for mariadb-import.

mysqlreport

Symlink or old name for mariadb-report.

mysqlshow

Symlink or old name for mariadb-show.

mysqlslap

Symlink or old name for mariadb-slap.

mysql_convert_table_format

Symlink or old name for mariadb-convert-table-format.

mysql_embedded

Symlink or old name for mariadb-embedded.

mysql_find_rows

Symlink or old name for mariadb-find-rows.

mysql_fix_extensions

Symlink or old name for mariadb-fix-extensions.

mysql_install_db

Symlink or old name for mariadb-install-db.

mysql_plugin

Symlink or old name for mariadb-plugin.

mysql_secure_installation

Symlink or old name for mariadb-secure-installation.

mysql_setpermission

Symlink or old name for mariadb_setpermission.

mysql_tzinfo_to_sql

Symlink or old name for mariadb-tzinfo-to-sql.

mysql_upgrade

Symlink or old name for mariadb-upgrade.

mysql_waitpid

Symlink or old name for mariadb-waitpid.

mysql_zap

Kill processes that match a pattern.

1.3.43.1 mysqlaccess

`mysqlaccess` is a tool for checking access privileges, developed by Yves Carlier.

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadb-access` is a symlink to `mysqlaccess`.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mariadb-access` is the name of the tool, with `mysqlaccess` a symlink.

Contents

See mariadb-access for details.

1.3.43.2 mysqldump

mysqldump is used to dump a database or a collection of databases for backup or transfer to another database server.

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadb-dump` is a symlink to `mysqldump`.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mariadb-dump` is the name of the command-line client, with `mysqldump` a symlink.

MariaDB starting with 11.0.1

From MariaDB 11.0.1, `mysqldump` (the symlink) is deprecated and removed from the `mariadb` Docker Official Image. Use `mariadb-dump` instead.

See mariadb-dump for details.

1.3.43.3 mysqldumpslow

`mysqldumpslow` is a tool to examine the slow query log.

MariaDB starting with 10.4.6
From MariaDB 10.4.6, mariadb-dumpslow is a symlink to `mysqldumpslow`.

MariaDB starting with 10.5.2
From MariaDB 10.5.2, mariadb-dumpslow is the name of the tool, with `mysqldumpslow` a symlink.

Contents

See mariadb-dumpslow for details.

1.1.1.2.1.10

1.3.43.5 mysql_convert_table_format

`mysql-convert-table-format` converts the tables in a database to use a particular storage engine (MyISAM by default).

MariaDB starting with 10.4.6
From MariaDB 10.4.6, mariadb-convert-table-format is a symlink to `mysql_convert_table_format`.

MariaDB starting with 10.5.2
From MariaDB 10.5.2, mariadb-convert-table-format is the name of the tool, with `mysql_convert_table_format` a symlink.

See mariadb-convert-table-format for details.

1.3.43.6 mysql_embedded

MariaDB starting with 10.4.6
From MariaDB 10.4.6, mariadb-embedded is a symlink to `mysql_embedded`.

MariaDB starting with 10.5.2
From MariaDB 10.5.2, mariadb-embedded is the name of the tool, with `mysql_embedded` a symlink.

See mariadb-embedded for details.

1.3.43.7 mysql_find_rows

`mysql_find_rows` reads files containing SQL statements and extracts statements that match a given regular expression or that contain USE db_name or SET statements.

MariaDB starting with 10.4.6
From MariaDB 10.4.6, mariadb-find-rows is a symlink to `mysql_find_rows`.

MariaDB starting with 10.5.2
From MariaDB 10.5.2, mariadb-find-rows is the name of the binary, with `mysql_find_rows` a symlink.

See mariadb-find-rows for details.

1.3.43.8 mysql_fix_extensions

MariaDB starting with 10.4.6
From MariaDB 10.4.6, mariadb-fix-extensions is a symlink to `mysql_fix_extensions`.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mysql_fix_extensions` is the symlink, and mariadb-fix-extensions the binary name.

See mariadb-fix-extensions for details.

1.3.43.9 mysqlhotcopy

mysqlhotcopy is deprecated and may be removed in a future release.

`mysqlhotcopy` uses FLUSH TABLES, LOCK TABLES, and cp or scp to make a database backup.

MariaDB starting with 10.4.6

From MariaDB 10.4.6, mariadb-hotcopy is a symlink to `mysqlhotcopy` .

MariaDB starting with 10.5.2

From MariaDB 10.5.2, mariadb-hotcopy is the name of the script, with `mysqlhotcopy` a symlink .

See mariadb-hotcopy for details.

1.3.43.10 mysqlimport

mysqlimport is used to load tables from text files in various formats

MariaDB starting with 10.4.6

From MariaDB 10.4.6, mariadb-import is a symlink to `mysqlimport` .

MariaDB starting with 10.5.2

From MariaDB 10.5.2, mariadb-import is the name of the script, with `mysqlimport` a symlink .

See mariadb-import for details.

1.3.43.11 mysql_install_db

`mariadb-install-db` initializes the MariaDB data directory and creates the system tables in the mysql database.

MariaDB starting with 10.4.6

From MariaDB 10.4.6, mariadb-install-db is a symlink to `mysql_install_db` .

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mysql_install_db` is the symlink, and mariadb-install-db the binary name.

See mariadb-install-db for details.

1.3.43.12 mysql_plugin

`mysql_plugin` is a tool for enabling or disabling plugins.

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadb-plugin` is a symlink to `mysql_plugin` .

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mysql_plugin` is the symlink, and `mariadb-plugin` the binary name.

See mariadb-plugin for details.

1.3.43.13 mysqlreport

`mysqlreport` makes a friendly report of important MariaDB status values.

MariaDB starting with 10.4.6
From MariaDB 10.4.6, mariadb-report is a symlink to `mysqlreport`.

MariaDB starting with 10.5.2
From MariaDB 10.5.2, `mysqlreport` is the symlink, and mariadb-report the binary name.

See mariadb-report for details.

1.3.43.14 mysql_secure_installation

Note that many of the reasons for the existence of this script no longer apply. In particular, from MariaDB 10.4, Unix socket authentication is applied by default, and there is usually no need to create a root password. See Authentication from MariaDB 10.4.

MariaDB starting with 10.4.6
From MariaDB 10.4.6, mariadb-secure-installation is a symlink to `mysql_secure_installation`.

MariaDB starting with 10.5.2
From MariaDB 10.5.2, `mysql_secure_installation` is the symlink, and mariadb-secure-installation the binary name.

See mariadb-secure-installation for details.

1.3.43.15 mysql_setpermission

MariaDB starting with 10.4.6
From MariaDB 10.4.6, mariadb-setpermission is a symlink to `mysql_setpermission`.

MariaDB starting with 10.5.2
From MariaDB 10.5.2, `mysql_setpermission` is the symlink, and mariadb-setpermission the binary name.

See mariadb-setpermission for details.

1.3.43.16 mysqlshow

Shows the structure of a MariaDB database (databases, tables, columns and indexes).

MariaDB starting with 10.4.6
From MariaDB 10.4.6, mariadb-show is a symlink to `mysqlshow`.

MariaDB starting with 10.5.2
From MariaDB 10.5.2, `mysqlshow` is the symlink, and mariadb-show the binary name.

See mariadb-show for details.

1.3.43.17 mysqlslap

`mysqlslap` is a tool for load-testing MariaDB. It allows you to emulate multiple concurrent connections, and run a set of queries multiple times.

MariaDB starting with 10.4.6
From MariaDB 10.4.6, mariadb-slap is a symlink to `mysqlslap`.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mysqlslap` is the symlink, and mariadb-slap the binary name.

See mariadb-slap for details.

1.3.43.18 mysql_tzinfo_to_sql

MariaDB starting with 10.4.6

From MariaDB 10.4.6, mariadb-tzinfo-to-sql is a symlink to `mysql_tzinfo_to_sql`.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mysql_tzinfo_to_sql` is the symlink, and mariadb-tzinfo-to-sql the binary name.

See mariadb-tzinfo-to-sql for details.

1.3.43.19 mysql_upgrade

`mariadb-upgrade/mysql_upgrade` is a tool that checks and updates your tables to the latest version.

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadb-upgrade` is a symlink to `mysql_upgrade`.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mysql_upgrade` is the symlink, and `mariadb-upgrade` the binary name.

See mariadb-upgrade for details.

1.3.43.20 mysql_waitpid

`mysql_pid` is a utility for terminating processes. It runs on Unix-like systems, making use of the `kill()` system call.

MariaDB starting with 10.4.6

From MariaDB 10.4.6, mariadb-waitpid is a symlink to `mysql_waitpid`.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mysql_waitpid` is the symlink, and mariadb-waitpid the binary name.

Contents

See mariadb-waitpid for details.

2 MariaDB Administration

There are many tasks that database administrators (DBAs) have to perform. This section of the MariaDB Documentation provides information on how to do these tasks.

Getting, Installing, and Upgrading MariaDB

Getting, installing, and upgrading MariaDB Server and related software.

User & Server Security

Creating users, granting privileges, and encryption.

Backing Up and Restoring Databases

Tools and methods for backing up and restoring databases.

Server Monitoring & Logs

Monitoring MariaDB Server and enabling and using logs.

Partitioning Tables
Splitting huge tables into multiple table files.

MariaDB Audit Plugin
Logging user activity with the MariaDB Audit Plugin.

Variables and Modes
Server variables and SQL modes.

Copying Tables Between Different MariaDB Databases and MariaDB Servers
Copy table files.

2.1 Getting, Installing, and Upgrading MariaDB

Where to Download MariaDB
Downloading tarballs, binaries, packages, and the source code for MariaDB.

MariaDB Binary Packages
Instructions on installing MariaDB binaries and packages.

Upgrading MariaDB
Upgrading from an earlier version, or from MySQL

Downgrading between Major Versions of MariaDB
Downgrading MariaDB is not officially supported between major versions.

Compiling MariaDB From Source
Articles on compiling MariaDB from source

Starting and Stopping MariaDB
Articles related to starting and stopping MariaDB Server.

MariaDB Performance & Advanced Configurations
Articles of how to setup your MariaDB optimally on different systems

Troubleshooting Installation Issues
Articles relating to installation issues users might run into

Installing System Tables (mariadb-install-db)
Using mariadb-install-db to create the system tables in the 'mysql' database directory.

mysql_install_db.exe
Windows equivalent of mysql_install_db for creating the system tables etc.

Configuring MariaDB with Option Files
Configuring MariaDB with my.cnf and other option files.

MariaDB Environment Variables
List of environment variables used by MariaDB.

Puppet and MariaDB
Puppet modules that allow you to use MariaDB.

MariaDB on Amazon RDS
Getting started with MariaDB on Amazon RDS

Obsolete Installation Information
Installation-related items that are obsolete

Migrating to MariaDB
Migrating to MariaDB from another DBMS.

Installing MariaDB on IBM Cloud

Get MariaDB on IBM Cloud You should have an IBM Cloud account, otherwise ...

mysqld Configuration Files and Groups

Which configuration files and groups mysqld reads.

There are 64 related questions.

2.1.1 Where to Download MariaDB

Contents

The Latest Packages

Tarballs, binaries (Linux, Solaris, and Windows), and packages for some Linux distributions are available at mariadb.com/downloads/ or mariadb.org/download/ (which also contains a PDF version of the MariaDB Server documentation).

We hope that interested community package maintainers will step forward, as others already have, to build packages for their distributions. We ask for strict adherence to your packaging system's best practices and invite you to create a bug report if our project impedes this in any way.

Instructions how to install the packages can be found here.

Distributions Which Include MariaDB

- Most distributions already include MariaDB. See Distributions Which Include MariaDB.

Pre-Release Binaries

Binaries from our Buildbot system (see also the Buildbot page), are available at http://hasky.askmonty.org/archive. They are not suitable for use in production systems but may be of use for debugging.

Once at the above URL you will need to click on the MariaDB tree you are interested in, and then the build. The build number corresponds to the `tarbuildnum` variable in Buildbot.

For example, if you were interested in the bsd9-64 build of the MariaDB 5.5 tree, revision 3497, the `tarbuildnum` is listed in the "Build Properties" table of the Buildbot build report. In this case, the value is "2434".

Getting the Source

You can find all the source code at https://github.com/MariaDB/server

To retrieve the code, the Git source control software offers the path of least resistance. If you are unfamiliar with git, please refer to the git documentation for an understanding of version control with git.

For instructions on creating a local branch of MariaDB, see the Getting the MariaDB Source Code page.

See the Generic Build Instructions page for general instructions on compiling MariaDB from the source. The source page has links to platform and distribution-specific information, including information on how we build the release packages.

2.1.2 MariaDB Binary Packages

This section contains information on and installation instructions for MariaDB binaries and packages.

Installing MariaDB RPM Files

Information and instructions on using the RPM packages and the related repositories.

Installing MariaDB .deb Files
Installing MariaDB .deb Files.

Installing MariaDB MSI Packages on Windows
MSI packages are available for both x86 (32 bit) and x64 (64 bit) processor architectures

Installing MariaDB Server PKG packages on macOS
MariaDB Server does not currently provide a .pkg installer for macOS

Installing MariaDB Binary Tarballs
Installing MariaDB binary tarballs, systemd, and glibc-2.14.

Installing MariaDB Server on macOS Using Homebrew
Installing MariaDB on macOS via the Homebrew package manager, the "missing ...

Installing MariaDB Windows ZIP Packages
Getting started with ZIP packages on Windows.

Compiling MariaDB From Source
Articles on compiling MariaDB from source

Distributions Which Include MariaDB
Distributions including MariaDB.

Running Multiple MariaDB Server Processes
Running multiple MariaDB Server processes on the same server.

Installing MariaDB Alongside MySQL
MariaDB was designed as a drop in place replacement for MySQL, but you can ...

GPG
The MariaDB project signs their MariaDB packages for Debian, Ubuntu, Fedora, CentOS, and Red Hat

MariaDB Platform Deprecation Policy
Information on MariaDB's Software Deprecation Policy and Schedule.

Automated MariaDB Deployment and Administration
Tools for automating deployment and management of MariaDB servers.

MariaDB Package Repository Setup and Usage
Executing and using a convenient shell script to set up the MariaDB Package Repository.

There are 7 related questions.

2.1.2.1 Installing MariaDB RPM Files

MariaDB provides RPM packages for several RPM-based Linux distributions. MariaDB also provides YUM/DNF and ZYpp repositories for these Linux distributions. The articles here provide information and instructions on using the RPM packages and the related repositories.

About the MariaDB RPM Files
Describes the contents of the RPM packages that come with each MariaDB release.

Installing MariaDB with yum/dnf
Installing MariaDB with yum or dnf on RHEL, CentOS, Fedora, and similar distros.

Installing MariaDB with zypper
How to install MariaDB with zypper on SLES, OpenSUSE, and other similar Linux distributions.

Installing MariaDB With the rpm Tool
Downloading and installing RPM files using the rpm command

Checking MariaDB RPM Package Signatures

Steps to check MariaDB RPM package signatures

Troubleshooting MariaDB Installs on Red Hat/CentOS

Issues people have encountered when installing MariaDB on Red Hat / CentOS

MariaDB for DirectAdmin Using RPMs

Using DirectAdmin when installing MariaDB with YUM

MariaDB Installation (Version 10.1.21) via RPMs on CentOS 7

Detailed steps for installing MariaDB (version 10.1.21) via RPMs on CentOS 7

Why Source RPMs (SRPMs) Aren't Packaged For Some Platforms

Explanation for why source RPM (SRPMs) aren't packaged for some platforms

Building MariaDB from a Source RPM

How to build MariaDB from a source RPM (SRPM).

There are 4 related questions.

2.1.2.1.1 About the MariaDB RPM Files

Contents

Available RPM Packages

The available RPM packages depend on the specific MariaDB release series.

Available RPM Packages in MariaDB 10.4

MariaDB starting with 10.4

From MariaDB 10.4, the following RPMs are available:

Package Name	Description
`galera-4`	The WSREP provider for Galera 4.
`MariaDB-backup`	Mariabackup
`MariaDB-backup-debuginfo`	Debuginfo for Mariabackup
`MariaDB-client`	Client tools like mariadb CLI, mariadb-dump, and others.
`MariaDB-client-debuginfo`	Debuginfo for client tools like `mariadb` CLI, `mariadb-dump` , and others.
`MariaDB-common`	Character set files and `/etc/my.cnf`
`MariaDB-common-debuginfo`	Debuginfo for character set files and `/etc/my.cnf`
`MariaDB-compat`	Old shared client libraries, may be needed by old MariaDB or MySQL clients
`MariaDB-connect-engine`	The `CONNECT` storage engine.
`MariaDB-connect-engine-debuginfo`	Debuginfo for the `CONNECT` storage engine.

`MariaDB-cracklib-password-check`	The `cracklib_password_check` password validation plugin.
`MariaDB-cracklib-password-check`	Debuginfo for the `cracklib_password_check` password validation plugin.
`MariaDB-devel`	Development headers and static libraries.
`MariaDB-devel-debuginfo`	Debuginfo for development headers and static libraries.
`MariaDB-gssapi-server`	The `gssapi` authentication plugin.
`MariaDB-gssapi-server-debuginfo`	Debuginfo for the `gssapi` authentication plugin.
`MariaDB-rocksdb-engine`	The MyRocks storage engine.
`MariaDB-rocksdb-engine-debuginfo`	Debuginfo for the MyRocks storage engine.
`MariaDB-server`	The server and server tools, like myisamchk and mariadb-hotcopy are here.
`MariaDB-server-compat`	Symbolic links from old MySQL tool names to MariaDB, like `mysqladmin` -> `mariadb-admin` or `mysql` -> `mariadb` . Good to have if you are using MySQL tool names in your scripts.
`MariaDB-server-debuginfo`	Debuginfo for the server and server tools, like myisamchk and mariadb-hotcopy are here.
`MariaDB-shared`	Dynamic client libraries.
`MariaDB-shared-debuginfo`	Debuginfo for dynamic client libraries.
`MariaDB-test`	`mysql-client-test` executable, and **mysql-test** framework with the tests.
`MariaDB-test-debuginfo`	Debuginfo for `mysql-client-test` executable, and **mysql-test** framework with the tests.
`MariaDB-tokudb-engine`	The TokuDB storage engine.
`MariaDB-tokudb-engine-debuginfo`	Debuginfo for the TokuDB storage engine.

Available RPM Packages in MariaDB 10.2 and MariaDB 10.3

MariaDB starting with 10.2

In MariaDB 10.2 and MariaDB 10.3, the following RPMs are available:

Package Name	Description
`galera`	The WSREP provider for Galera 3.
`MariaDB-backup`	Mariabackup
`MariaDB-backup-debuginfo`	Debuginfo for Mariabackup
`MariaDB-client`	Client tools like `mysql` CLI, `mysqldump` , and others.
`MariaDB-client-debuginfo`	Debuginfo for client tools like `mysql` CLI, `mariadb-dump` , and others.
`MariaDB-common`	Character set files and `/etc/my.cnf`
`MariaDB-common-debuginfo`	Debuginfo for character set files and `/etc/my.cnf`

`MariaDB-compat`	Old shared client libraries, may be needed by old MariaDB or MySQL clients
`MariaDB-connect-engine`	The `CONNECT` storage engine.
`MariaDB-connect-engine-debuginfo`	Debuginfo for the `CONNECT` storage engine.
`MariaDB-cracklib-password-check`	The `cracklib_password_check` password validation plugin.
`MariaDB-cracklib-password-check`	Debuginfo for the `cracklib_password_check` password validation plugin.
`MariaDB-devel`	Development headers and static libraries.
`MariaDB-devel-debuginfo`	Debuginfo for development headers and static libraries.
`MariaDB-gssapi-server`	The `gssapi` authentication plugin.
`MariaDB-gssapi-server-debuginfo`	Debuginfo for the `gssapi` authentication plugin.
`MariaDB-rocksdb-engine`	The MyRocks storage engine.
`MariaDB-rocksdb-engine-debuginfo`	Debuginfo for the MyRocks storage engine.
`MariaDB-server`	The server and server tools, like myisamchk and mariadb-hotcopy are here.
`MariaDB-server-debuginfo`	Debuginfo for the server and server tools, like myisamchk and mariadb-hotcopy are here.
`MariaDB-shared`	Dynamic client libraries.
`MariaDB-shared-debuginfo`	Debuginfo for dynamic client libraries.
`MariaDB-test`	`mysql-client-test` executable, and **mysql-test** framework with the tests.
`MariaDB-test-debuginfo`	Debuginfo for `mysql-client-test` executable, and **mysql-test** framework with the tests.
`MariaDB-tokudb-engine`	The TokuDB storage engine.
`MariaDB-tokudb-engine-debuginfo`	Debuginfo for the TokuDB storage engine.

Installing RPM Packages

Preferably, you should install MariaDB RPM packages using the package manager of your Linux distribution, for example **`yum`** or **`zypper`**. But you can also use the lower-level **`rpm`** tool.

Actions Performed by RPM Packages

Users and Groups Created

When the `MariaDB-server` RPM package is installed, it will create a user and group named `mysql`, if it does not already exist.

2.1.2.1.2 Installing MariaDB with yum/dnf

On RHEL, CentOS, Fedora, and other similar Linux RPM based distributions, these provide MariaDB packages. These are supported by those distributions. If you have a particular need for a later version than what is in the distribution, then MariaDB provides repositories for them.

Using repositories rather than installing RPM allows for an ease of update when a new release is made. It is highly recommended to install the relevant RPM packages from MariaDB's repository using yum or dnf. Centos 7 still uses `yum`, most others use `dnf`, and SUSE/openSUSE use `zypper`.

This page walks you through the simple installation steps using `dnf` and `yum`.

Contents

Adding the MariaDB YUM repository

We currently have YUM/DNF repositories for the following Linux distributions, and for the versions that are in standard (not extended) support:

- Red Hat Enterprise Linux (RHEL)
- CentOS
- Fedora
- openSUSE
- SUSE

Using the MariaDB Package Repository Setup Script

If you want to install MariaDB with `yum`, then you can configure `yum` to install from MariaDB Corporation's MariaDB Package Repository by using the MariaDB Package Repository setup script.

MariaDB Corporation provides a MariaDB Package Repository for several Linux distributions that use `yum` to manage packages. This repository contains software packages related to MariaDB Server, including the server itself, clients and utilities, client libraries, plugins, and Mariabackup. The MariaDB Package Repository setup script automatically configures your system to install packages from the MariaDB Package Repository.

To use the script, execute the following command:

```
curl -sS https://downloads.mariadb.com/MariaDB/mariadb_repo_setup | sudo bash
```

Note that this script also configures a repository for MariaDB MaxScale and a repository for MariaDB Tools, which currently only contains Percona XtraBackup and its dependencies.

See MariaDB Package Repository Setup and Usage for more information.

Using the MariaDB Repository Configuration Tool

If you want to install MariaDB with `yum`, then you can configure `yum` to install from MariaDB Foundation's MariaDB Repository by using the MariaDB Repository Configuration Tool.

The MariaDB Foundation provides a MariaDB repository for several Linux distributions that use `yum` to manage packages. This repository contains software packages related to MariaDB Server, including the server itself, clients and utilities, client libraries, plugins, and Mariabackup. The MariaDB Repository Configuration Tool can easily generate the appropriate configuration file to add the repository for your distribution.

Once you have the appropriate repository configuration section for your distribution, add it to a file named `MariaDB.repo` under `/etc/yum.repos.d/`.

For example, if you wanted to use the repository to install MariaDB 10.6 on RHEL (any version), then you could use the following `yum` repository configuration in `/etc/yum.repos.d/MariaDB.repo`:

```
[mariadb]
name = MariaDB
baseurl = https://rpm.mariadb.org/10.6/rhel/$releasever/$basearch
gpgkey= https://rpm.mariadb.org/RPM-GPG-KEY-MariaDB
gpgcheck=1
```

The example file above includes a `gpgkey` line to automatically fetch the GPG public key that is used to verify the digital signatures of the packages in our repositories. This allows the the `yum`, `dnf`, and `rpm` utilities to verify the integrity of the packages that they install.

Pinning the MariaDB Repository to a Specific Minor Release

If you wish to pin the `yum` repository to a specific minor release, or if you would like to do a `yum downgrade` to a specific minor release, then you can create a `yum` repository configuration with a `baseurl` option set to that specific minor release.

The MariaDB Foundation archives repositories all releases is at the following URL:

- http://archive.mariadb.org/

Note this isn't configured as a highly available server. For that purpose please use the main mirrors.

For example, if you wanted to pin your repository to MariaDB 10.3.34 on CentOS 7, then you could use the following `yum` repository configuration in `/etc/yum.repos.d/MariaDB.repo`:

```
[mariadb]
name = MariaDB-10.3.34
baseurl= http://archive.mariadb.org/mariadb-10.3.34/yum/centos/$releasever/$basearch
gpgkey= https://archive.mariadb.org/PublicKey
gpgcheck=1
```

Note that if you change an existing repository configuration, then you may need to execute the following:

```
sudo yum clean all
```

Updating the MariaDB YUM repository to a New Major Release

MariaDB's `yum` repository can be updated to a new major release. How this is done depends on how you originally configured the repository.

Updating the Major Release with the MariaDB Package Repository Setup Script

If you configured `yum` to install from MariaDB Corporation's MariaDB Package Repository by using the MariaDB Package Repository setup script, then you can update the major release that the repository uses by running the script again.

Updating the Major Release with the MariaDB Repository Configuration Tool

If you configured `yum` to install from MariaDB Foundation's MariaDB Repository by using the MariaDB Repository Configuration Tool, then you can update the major release that the repository uses by updating the `yum` repository configuration file in-place. For example, if you wanted to change the repository from MariaDB 10.6 to MariaDB 10.11, and if the repository configuration file was at `/etc/yum.repos.d/MariaDB.repo`, then you could execute the following:

```
sudo sed -i 's/10.6/10.11/' /etc/yum.repos.d/MariaDB.repo
```

After that, the repository should refer to MariaDB 10.11.

If the `yum` repository is pinned to a specific minor release, then the above `sed` command can result in an invalid repository configuration. In that case, the recommended options are:

- Edit the `MariaDB.repo` repository file manually.
- Or delete the `MariaDB.repo` repository file, and then install the repository of the new version with the more robust

MariaDB Package Repository setup script.

Importing the MariaDB GPG Public Key

Before MariaDB can be installed, you also have to import the GPG public key that is used to verify the digital signatures of the packages in our repositories. This allows the `yum`, `dnf` and `rpm` utilities to verify the integrity of the packages that they install.

The id of our GPG public key is:

- short form: `0xC74CD1D8`
- long form: `0xF1656F24C74CD1D8`
- full fingerprint: `177F 4010 FE56 CA33 3630 0305 F165 6F24 C74C D1D8`

`yum` should prompt you to import the GPG public key the first time that you install a package from MariaDB's repository. However, if you like, the rpm utility can be used to manually import this key instead. For example:

```
sudo rpm --import https://supplychain.mariadb.com/MariaDB-Server-GPG-KEY
```

Once the GPG public key is imported, you are ready to install packages from the repository.

Old Key

For releases before 2023 an older SHA1 based GPG key was used.

The id of this older GPG public key was `0xcbcb082a1bb943db`. The short form was `0x1BB943DB`. The full key fingerprint was:

```
1993 69E5 404B D5FC 7D2F E43B CBCB 082A 1BB9 43DB
```

Installing MariaDB Packages with YUM/DNF

After the `dnf` / `yum` repository is configured, you can install MariaDB by executing the dnf or yum command. The specific command that you would use would depend on which specific packages that you want to install.

Installing the Most Common Packages

MariaDB starting with 10.4

In MariaDB 10.4 and later, to Install the most common packages, execute the following command:

```
sudo dnf install MariaDB-server galera-4 MariaDB-client MariaDB-shared MariaDB-backup MariaDB
```

MariaDB until 10.3

In MariaDB 10.3 and before, to Install the most common packages, execute the following command:

```
sudo yum install MariaDB-server galera MariaDB-client MariaDB-shared MariaDB-backup MariaDB-c
```

Installing MariaDB Server

To Install MariaDB Server, execute the following command:

```
sudo dnf install MariaDB-server
```

Installing MariaDB Galera Cluster with YUM

The process to install MariaDB Galera Cluster with the MariaDB `yum` repository is practically the same as installing standard MariaDB Server.

In MariaDB 10.4 and later, you also need to install the `galera-4` package to obtain the Galera 4 wsrep provider library.

In MariaDB 10.3 and before, you also need to install the `galera` package to obtain the Galera 3 wsrep provider library.

MariaDB starting with 10.4

In MariaDB 10.4 and later, to install MariaDB Galera Cluster, you could execute the following command:

```
sudo yum install MariaDB-server MariaDB-client galera-4
```

MariaDB until 10.3

In MariaDB 10.3 and before, to install MariaDB Galera Cluster, you could execute the following command:

```
sudo yum install MariaDB-server MariaDB-client galera
```

If you haven't yet imported the MariaDB GPG public key, then `yum` will prompt you to import it after it downloads the packages, but before it prompts you to install them.

See MariaDB Galera Cluster for more information on MariaDB Galera Cluster.

Installing MariaDB Clients and Client Libraries with YUM

In MariaDB 10.2 and later, MariaDB Connector/C has been included as the client library (statically linked). However, the package name for the client library has not been changed.

To Install the clients and client libraries, execute the following command:

```
sudo yum install MariaDB-client MariaDB-shared
```

If you want compile your own programs against MariaDB Connector/C, execute the following command:

```
sudo yum install MariaDB-devel
```

Installing Mariabackup with YUM

To install Mariabackup, execute the following command:

```
sudo yum install MariaDB-backup
```

Installing Plugins with YUM

Some plugins may also need to be installed.

For example, to install the cracklib_password_check password validation plugin, execute the following command:

```
sudo yum install MariaDB-cracklib-password-check
```

Installing Debug Info Packages with YUM

MariaDB starting with 5.5.64

The MariaDB `yum` repository first added debuginfo packages in MariaDB 5.5.64, MariaDB 10.1.39, MariaDB 10.2.23, MariaDB 10.3.14, and MariaDB 10.4.4.

The MariaDB `yum` repository also contains debuginfo packages. These package may be needed when debugging a problem.

Installing Debug Info for the Most Common Packages with YUM

To install debuginfo for the most common packages, execute the following command:

```
sudo yum install MariaDB-server-debuginfo MariaDB-client-debuginfo MariaDB-shared-debuginfo
MariaDB-backup-debuginfo MariaDB-common-debuginfo
```

All packages have their debuginfo by appending `-debuginfo` to the package name.

Installing Debug Info for MariaDB Server with YUM

To install debuginfo for MariaDB Server, execute the following command:

```
sudo yum install MariaDB-server-debuginfo
```

Installing Older Versions from the Repository

The MariaDB `yum` repository contains the last few versions of MariaDB. To show what versions are available, use the following command:

```
yum list --showduplicates MariaDB-server
```

In the output you will see the available versions. For example:

```
$ yum list --showduplicates MariaDB-server
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: centos.mirrors.ovh.net
 * extras: centos.mirrors.ovh.net
 * updates: centos.mirrors.ovh.net
Available Packages
MariaDB-server.x86_64   10.3.10-1.el7.centos    mariadb
MariaDB-server.x86_64   10.3.11-1.el7.centos    mariadb
MariaDB-server.x86_64   10.3.12-1.el7.centos    mariadb
mariadb-server.x86_64   1:5.5.60-1.el7_5         base
```

The MariaDB `yum` repository in this example contains MariaDB 10.3.10, MariaDB 10.3.11, and MariaDB 10.3.12. The CentOS base `yum` repository also contains MariaDB 5.5.60.

To install an older version of a package instead of the latest version we just need to specify the package name, a dash, and then the version number. And we only need to specify enough of the version number for it to be unique from the other available versions.

However, when installing an older version of a package, if `yum` has to install dependencies, then it will automatically choose to install the latest versions of those packages. To ensure that all MariaDB packages are on the same version in this scenario, it is necessary to specify them all.

The packages that the MariaDB-server package depend on are: MariaDB-client, MariaDB-shared, and MariaDB-common. Therefore, to install MariaDB 10.3.11 from this `yum` repository, we would do the following:

```
sudo yum install MariaDB-server-10.3.11 MariaDB-client-10.3.11 MariaDB-shared-10.3.11 MariaDB-
backup-10.3.11 MariaDB-common-10.3.11
```

The rest of the install and setup process is as normal.

After Installation

After the installation is complete, you can start MariaDB.

If you are using MariaDB Galera Cluster, then keep in mind that the first node will have to be bootstrapped.

2.1.2.1.3 Installing MariaDB with zypper

On SLES, OpenSUSE, and other similar Linux distributions, it is highly recommended to install the relevant RPM packages from MariaDB's repository using `zypper` .

This page walks you through the simple installation steps using `zypper` .

Contents

Adding the MariaDB ZYpp repository

We currently have ZYpp repositories for the following Linux distributions:

- SUSE Linux Enterprise Server (SLES) 12
- SUSE Linux Enterprise Server (SLES) 15
- OpenSUSE 15
- OpenSUSE 42

Using the MariaDB Package Repository Setup Script

If you want to install MariaDB with `zypper`, then you can configure `zypper` to install from MariaDB Corporation's MariaDB Package Repository by using the MariaDB Package Repository setup script.

MariaDB Corporation provides a MariaDB Package Repository for several Linux distributions that use `zypper` to manage packages. This repository contains software packages related to MariaDB Server, including the server itself, clients and utilities, client libraries, plugins, and Mariabackup. The MariaDB Package Repository setup script automatically configures your system to install packages from the MariaDB Package Repository.

To use the script, execute the following command:

```
curl -sS https://downloads.mariadb.com/MariaDB/mariadb_repo_setup | sudo bash
```

Note that this script also configures a repository for MariaDB MaxScale and a repository for MariaDB Tools, which currently only contains Percona XtraBackup and its dependencies.

See MariaDB Package Repository Setup and Usage for more information.

Using the MariaDB Repository Configuration Tool

If you want to install MariaDB with `zypper`, then you can configure `zypper` to install from MariaDB Foundation's MariaDB Repository by using the MariaDB Repository Configuration Tool.

The MariaDB Foundation provides a MariaDB repository for several Linux distributions that use `zypper` to manage packages. This repository contains software packages related to MariaDB Server, including the server itself, clients and utilities, client libraries, plugins, and Mariabackup. The MariaDB Repository Configuration Tool can easily generate the appropriate commands to add the repository for your distribution.

For example, if you wanted to use the repository to install MariaDB 10.3 on SLES 15, then you could use the following commands to add the MariaDB `zypper` repository:

```
sudo zypper addrepo --gpgcheck --refresh https://yum.mariadb.org/10.3/sles/15/x86_64 mariadb
sudo zypper --gpg-auto-import-keys refresh
```

Pinning the MariaDB Repository to a Specific Minor Release

If you wish to pin the `zypper` repository to a specific minor release, or if you would like to downgrade to a specific minor release, then you can create a `zypper` repository with the URL hard-coded to that specific minor release.

The MariaDB Foundation archives repositories of old minor releases at the following URL:

- http://archive.mariadb.org/

So if you can't find the repository of a specific minor release at `yum.mariadb.org`, then it would be a good idea to check the archive.

For example, if you wanted to pin your repository to MariaDB 10.3.14 on SLES 15, then you could use the following commands to add the MariaDB `zypper` repository:

```
sudo zypper removerepo mariadb
sudo zypper addrepo --gpgcheck --refresh https://yum.mariadb.org/10.3.14/sles/15/x86_64 mariadb
```

Updating the MariaDB ZYpp repository to a New Major Release

MariaDB's `zypper` repository can be updated to a new major release. How this is done depends on how you originally configured the repository.

Updating the Major Release with the MariaDB Package Repository Setup Script

If you configured `zypper` to install from MariaDB Corporation's MariaDB Package Repository by using the MariaDB Package Repository setup script, then you can update the major release that the repository uses by running the script again.

Updating the Major Release with the MariaDB Repository Configuration Tool

If you configured `zypper` to install from MariaDB Foundation's MariaDB Repository by using the MariaDB Repository Configuration Tool, then you can update the major release that the repository uses by removing the repository for the old version and adding the repository for the new version.

First, you can remove the repository for the old version by executing the following command:

```
sudo zypper removerepo mariadb
```

After that, you can add the repository for the new version. For example, if you wanted to use the repository to install MariaDB 10.3 on SLES 15, then you could use the following commands to add the MariaDB `zypper` repository:

```
sudo zypper addrepo --gpgcheck --refresh https://yum.mariadb.org/10.3/sles/15/x86_64 mariadb
sudo zypper --gpg-auto-import-keys refresh
```

After that, the repository should refer to MariaDB 10.3.

Importing the MariaDB GPG Public Key

Before MariaDB can be installed, you also have to import the GPG public key that is used to verify the digital signatures of the packages in our repositories. This allows the the `zypper` and `rpm` utilities to verify the integrity of the packages that they install.

The id of our GPG public key is `0xcbcb082a1bb943db`. The short form of the id is `0x1BB943DB`. The full key fingerprint is:

```
1993 69E5 404B D5FC 7D2F E43B CBCB 082A 1BB9 43DB
```

The `rpm` utility can be used to import this key. For example:

```
sudo rpm --import https://yum.mariadb.org/RPM-GPG-KEY-MariaDB
```

Once the GPG public key is imported, you are ready to install packages from the repository.

Installing MariaDB Packages with ZYpp

After the `zypper` repository is configured, you can install MariaDB by executing the `zypper` command. The specific command that you would use would depend on which specific packages that you want to install.

Installing the Most Common Packages with ZYpp

MariaDB starting with 10.4

In MariaDB 10.4 and later, to Install the most common packages, execute the following command:

```
sudo zypper install MariaDB-server galera-4 MariaDB-client MariaDB-shared MariaDB-backup
MariaDB-common
```

MariaDB until 10.3

In MariaDB 10.3 and before, to Install the most common packages, execute the following command:

```
sudo zypper install MariaDB-server galera MariaDB-client MariaDB-shared MariaDB-backup
MariaDB-common
```

Installing MariaDB Server with ZYpp

To Install MariaDB Server, execute the following command:

```
sudo zypper install MariaDB-server
```

Installing MariaDB Galera Cluster with ZYpp

The process to install MariaDB Galera Cluster with the MariaDB `zypper` repository is practically the same as installing standard MariaDB Server.

In MariaDB 10.1 and later, Galera Cluster support has been included in the standard MariaDB Server packages, so you will need to install the `MariaDB-server` package, as you normally would.

In MariaDB 10.4 and later, you also need to install the `galera-4` package to obtain the Galera 4 wsrep provider library.

In MariaDB 10.3 and before, you also need to install the `galera` package to obtain the Galera 3 wsrep provider library.

MariaDB starting with 10.4

In MariaDB 10.4 and later, to install MariaDB Galera Cluster, you could execute the following command:

```
sudo zypper install MariaDB-server MariaDB-client galera-4
```

MariaDB until 10.3

In MariaDB 10.3 and before, to install MariaDB Galera Cluster, you could execute the following command:

```
sudo zypper install MariaDB-server MariaDB-client galera
```

If you haven't yet imported the MariaDB GPG public key, then `zypper` will prompt you to import it after it downloads the packages, but before it prompts you to install them.

See MariaDB Galera Cluster for more information on MariaDB Galera Cluster.

Installing MariaDB Clients and Client Libraries with ZYpp

In MariaDB 10.2 and later, MariaDB Connector/C has been included as the client library. However, the package name for

the client library has not been changed.

To Install the clients and client libraries, execute the following command:

```
sudo zypper install MariaDB-client MariaDB-shared
```

Installing Mariabackup with ZYpp

To install Mariabackup, execute the following command:

```
sudo zypper install MariaDB-backup
```

Installing Plugins with ZYpp

Some plugins may also need to be installed.

For example, to install the `cracklib_password_check` password validation plugin, execute the following command:

```
sudo zypper install MariaDB-cracklib-password-check
```

Installing Debug Info Packages with ZYpp

MariaDB starting with 5.5.64

The MariaDB `zypper` repository first added `debuginfo` packages in MariaDB 5.5.64, MariaDB 10.1.39, MariaDB 10.2.23, MariaDB 10.3.14, and MariaDB 10.4.4.

The MariaDB `zypper` repository also contains `debuginfo` packages. These package may be needed when debugging a problem.

Installing Debug Info for the Most Common Packages with ZYpp

To install debuginfo for the most common packages, execute the following command:

```
sudo zypper install MariaDB-server-debuginfo MariaDB-client-debuginfo MariaDB-shared-debuginfo
MariaDB-backup-debuginfo MariaDB-common-debuginfo
```

Installing Debug Info for MariaDB Server with ZYpp

To install `debuginfo` for MariaDB Server, execute the following command:

```
sudo zypper install MariaDB-server-debuginfo
```

Installing Debug Info for MariaDB Clients and Client Libraries with ZYpp

In MariaDB 10.2 and later, MariaDB Connector/C has been included as the client library. However, the package name for the client library has not been changed.

To install `debuginfo` for the clients and client libraries, execute the following command:

```
sudo zypper install MariaDB-client-debuginfo MariaDB-shared-debuginfo
```

Installing Debug Info for Mariabackup with ZYpp

To install `debuginfo` for Mariabackup, execute the following command:

```
sudo zypper install MariaDB-backup-debuginfo
```

Installing Debug Info for Plugins with ZYpp

For some plugins, `debuginfo` may also need to be installed.

For example, to install `debuginfo` for the `cracklib_password_check` password validation plugin, execute the following command:

```
sudo zypper install MariaDB-cracklib-password-check-debuginfo
```

Installing Older Versions from the Repository

The MariaDB `zypper` repository contains the last few versions of MariaDB. To show what versions are available, use the following command:

```
zypper search --details MariaDB-server
```

In the output you will see the available versions.

To install an older version of a package instead of the latest version we just need to specify the package name, a dash, and then the version number. And we only need to specify enough of the version number for it to be unique from the other available versions.

However, when installing an older version of a package, if `zypper` has to install dependencies, then it will automatically choose to install the latest versions of those packages. To ensure that all MariaDB packages are on the same version in this scenario, it is necessary to specify them all.

The packages that the MariaDB-server package depend on are: MariaDB-client, MariaDB-shared, and MariaDB-common. Therefore, to install MariaDB 10.3.14 from this `zypper` repository, we would do the following:

```
sudo zypper install MariaDB-server-10.3.14 MariaDB-client-10.3.14 MariaDB-shared-10.3.14
MariaDB-backup-10.3.14 MariaDB-common-10.3.14
```

The rest of the install and setup process is as normal.

After Installation

After the installation is complete, you can start MariaDB.

If you are using MariaDB Galera Cluster, then keep in mind that the first node will have to be bootstrapped.

2.1.2.1.4 Installing MariaDB With the rpm Tool

This article describes how to download the RPM files and install them using the `rpm` command.

It is highly recommended to Install MariaDB with yum where possible.

Navigate to https://mariadb.org/download/ and choose the desired database version and then select the RPMs that match your Linux distribution and architecture.

Clicking those links takes you to a local mirror. Choose the rpms link and download the desired packages. The packages will be similar to the following:

```
MariaDB-client-5.2.5-99.el5.x86_64.rpm
MariaDB-debuginfo-5.2.5-99.el5.x86_64.rpm
MariaDB-devel-5.2.5-99.el5.x86_64.rpm
MariaDB-server-5.2.5-99.el5.x86_64.rpm
MariaDB-shared-5.2.5-99.el5.x86_64.rpm
MariaDB-test-5.2.5-99.el5.x86_64.rpm
```

For a standard server installation you will need to download at least the *client*, *shared*, and *server* RPM files. See About the MariaDB RPM Files for more information about what is included in each RPM package.

After downloading the MariaDB RPM files, you might want to check their signatures. See Checking MariaDB RPM Package Signatures for more information about checking signatures.

```
rpm --checksig $(find . -name '*.rpm')
```

Prior to installing MariaDB, be aware that it will conflict with an existing installation of MySQL. To check whether MySQL is already installed, issue the command:

```
rpm -qa 'mysql*'
```

If necessary, you can remove found MySQL packages before installing MariaDB.

To install MariaDB, use the command:

```
rpm -ivh MariaDB-*
```

You should see output such as the following:

```
Preparing...                ########################################### [100%]
   1:MariaDB-shared         ########################################### [ 14%]
   2:MariaDB-client         ########################################### [ 29%]
   3:MariaDB-client         ########################################### [ 43%]
   4:MariaDB-debuginfo      ########################################### [ 57%]
   5:MariaDB-devel          ########################################### [ 71%]
   6:MariaDB-server         ########################################### [ 86%]

PLEASE REMEMBER TO SET A PASSWORD FOR THE MariaDB root USER !
To do so, start the server, then issue the following commands:

/usr/bin/mariadb-admin -u root password 'new-password'
/usr/bin/mariadb-admin -u root -h hostname password 'new-password'

Alternatively you can run:
/usr/bin/mysql_secure_installation

which will also give you the option of removing the test
databases and anonymous user created by default.  This is
strongly recommended for production servers.

See the MySQL manual for more instructions.

Please report any problems with the /usr/bin/mysqlbug script!

The latest information about MariaDB is available at http://www.askmonty.org/.
You can find additional information about the MySQL part at:
http://dev.mysql.com
Support MariaDB development by buying support/new features from
Monty Program Ab. You can contact us about this at sales@askmonty.org.
Alternatively consider joining our community based development effort:
http://askmonty.org/wiki/index.php/MariaDB#How_can_I_participate_in_the_development_of_MariaDB

Starting MySQL....[  OK  ]
Giving mysqld 2 seconds to start
   7:MariaDB-test           ########################################### [100%]
```

Be sure to follow the instructions given in the preceding output and create a password for the root user either by using mariadb-admin or by running the /usr/bin/mysql_secure_installation script.

Installing the MariaDB RPM files installs the MySQL tools in the `/usr/bin` directory. You can confirm that MariaDB has been installed by using the mariadb client program. Issuing the command `mariadb` should give you the MariaDB cursor.

2.1.2.1.5 Checking MariaDB RPM Package Signatures

MariaDB RPM packages since MariaDB 5.1.55 are signed.

The key we use has an id of `1BB943DB` and the key fingerprint is:

```
1993 69E5 404B D5FC 7D2F E43B CBCB 082A 1BB9 43DB
```

To check the signature you first need to import the public part of the key like so:

```
gpg --keyserver hkp://pgp.mit.edu --recv-keys 1BB943DB
```

Next you need to let pgp know about the key like so:

```
gpg --export --armour 1BB943DB > mariadb-signing-key.asc
sudo rpm --import mariadb-signing-key.asc
```

You can check to see if the key was imported with:

```
rpm -qa gpg-pubkey*
```

Once the key is imported, you can check the signature of the MariaDB RPM files by running the something like the following in your download directory:

```
rpm --checksig $(find . -name '*.rpm')
```

The output of the above will look something like this (make sure gpg shows up on each OK line):

```
me@desktop:~$ rpm --checksig $(find . -name '*.rpm')
./kvm-rpm-centos5-amd64/rpms/MariaDB-test-5.1.55-98.el5.x86_64.rpm: (sha1) dsa sha1 md5 gpg OK
./kvm-rpm-centos5-amd64/rpms/MariaDB-server-5.1.55-98.el5.x86_64.rpm: (sha1) dsa sha1 md5 gpg OK
./kvm-rpm-centos5-amd64/rpms/MariaDB-client-5.1.55-98.el5.x86_64.rpm: (sha1) dsa sha1 md5 gpg OK
./kvm-rpm-centos5-amd64/rpms/MariaDB-shared-5.1.55-98.el5.x86_64.rpm: (sha1) dsa sha1 md5 gpg OK
./kvm-rpm-centos5-amd64/rpms/MariaDB-devel-5.1.55-98.el5.x86_64.rpm: (sha1) dsa sha1 md5 gpg OK
./kvm-rpm-centos5-amd64/rpms/MariaDB-debuginfo-5.1.55-98.el5.x86_64.rpm: (sha1) dsa sha1 md5 gpg
./kvm-rpm-centos5-amd64/srpms/MariaDB-5.1.55-98.el5.src.rpm: (sha1) dsa sha1 md5 gpg OK
```

2.1.2.1.6 Troubleshooting MariaDB Installs on Red Hat/CentOS

The following article is about different issues people have encountered when installing MariaDB on Red Hat / CentOS.

It is highly recommended to install with yum where possible.

In Red Hat / CentOS it is also possible to install a RPM or a tar ball. The RPM is the preferred version, except if you want to install many versions of MariaDB or install MariaDB in a non standard location.

Replacing MySQL

If you removed an MySQL RPM to install MariaDB, note that the MySQL RPM on uninstall renames /etc/my.cnf to /etc/my.cnf.rpmsave.

After installing MariaDB you should do the following to restore your configuration options:

```
mv /etc/my.cnf.rpmsave /etc/my.cnf
```

Unsupported configuration options

If you are using any of the following options in your /etc/my.cnf or other my.cnf file you should remove them. This is also true for MySQL 5.1 or newer:

```
skip-bdb
```

2.1.2.1.7 MariaDB for DirectAdmin Using RPMs

If you are using DirectAdmin and you encounter any issues with Installing MariaDB with YUM, then the directions below may help. The process is very straightforward.

Note: Installing with YUM is preferable to installing the MariaDB RPM packages manually, so only do this if you are having issues such as:

```
Starting httpd:
  httpd:
    Syntax error on line 18 of /etc/httpd/conf/httpd.conf:
    Syntax error on line 1 of /etc/httpd/conf/extra/httpd-phpmodules.conf:
      Cannot load /usr/lib/apache/libphp5.so into server:
        libmysqlclient.so.18: cannot open shared object file: No such file or directory
```

Or:

```
Starting httpd:
  httpd:
    Syntax error on line 18 of /etc/httpd/conf/httpd.conf:
    Syntax error on line 1 of /etc/httpd/conf/extra/httpd-phpmodules.conf:
      Cannot load /usr/lib/apache/libphp5.so into server:
        /usr/lib/apache/libphp5.so: undefined symbol: client_errors
```

RPM Installation

To install the RPMs, there is a quick and easy guide to Installing MariaDB with the RPM Tool. Follow the instructions there.

Necessary Edits

We do not want DirectAdmin's custombuild to remove/overwrite our MariaDB installation whenever an update is performed. To rectify this, disable automatic MySQL installation.

Edit `/usr/local/directadmin/custombuild/options.conf`

Change:

```
mysql_inst=yes
```

To:

```
mysql_inst=no
```

Note: When MariaDB is installed manually (i.e. not using YUM), updates are not automatic. You will need to update the RPMs yourself.

2.1.2.1.8 MariaDB Installation (Version 10.1.21) via RPMs on CentOS 7

Here are the detailed steps for installing MariaDB (version 10.1.21) via RPMs on CentOS 7.

The RPM's needed for the installation are all available on the MariaDB website and are given below:

- jemalloc-3.6.0-1.el7.x86_64.rpm
- MariaDB-10.1.21-centos7-x86_64-client.rpm
- MariaDB-10.1.21-centos7-x86_64-compat.rpm
- galera-25.3.19-1.rhel7.el7.centos.x86_64.rpm
- jemalloc-devel-3.6.0-1.el7.x86_64.rpm
- MariaDB-10.1.21-centos7-x86_64-common.rpm
- MariaDB-10.1.21-centos7-x86_64-server.rpm

Step by step installation:

- 1) First install all of the dependencies needed. Its easy to do this via YUM packages: yum install rsync nmap lsof perl-DBI nc
- 2) rpm -ivh jemalloc-3.6.0-1.el7.x86_64.rpm
- 3) rpm -ivh jemalloc-devel-3.6.0-1.el7.x86_64.rpm
- 4) rpm -ivh MariaDB-10.1.21-centos7-x86_64-common.rpm MariaDB-10.1.21-centos7-x86_64-compat.rpm MariaDB-10.1.21-centos7-x86_64-client.rpm galera-25.3.19-1.rhel7.el7.centos.x86_64.rpm MariaDB-10.1.21-centos7-x86_64-server.rpm

While installing MariaDB-10.1.21-centos7-x86_64-common.rpm there might be a conflict with older MariaDB packages. we need to remove them and install the original rpm again.

Here is the error message for dependencies:

```
# rpm -ivh MariaDB-10.1.21-centos7-x86_64-common.rpm
warning: MariaDB-10.1.21-centos7-x86_64-common.rpm: Header V4 DSA/SHA1 Signature, key ID
1bb943db: NOKEY
error: Failed dependencies:
 mariadb-libs < 1:10.1.21-1.el7.centos conflicts with MariaDB-common-10.1.21-
1.el7.centos.x86_64
```

Solution: search for this package:

```
# rpm -qa | grep mariadb-libs
mariadb-libs-5.5.52-1.el7.x86_64
```

Remove this package:

```
# rpm -ev --nodeps mariadb-libs-5.5.52-1.el7.x86_64
Preparing packages...
mariadb-libs-1:5.5.52-1.el7.x86_64
```

While installing the Galera package there might be a conflict in installation for a dependency package. Here is the error message:

```
[root@centos-2 /]# rpm -ivh galera-25.3.19-1.rhel7.el7.centos.x86_64.rpm
error: Failed dependencies:
 libboost_program_options.so.1.53.0()(64bit) is needed by galera-25.3.19-
1.rhel7.el7.centos.x86_64

The dependencies for Galera package is: libboost_program_options.so.1.53.0
```

Solution:

```
yum install boost-devel.x86_64
```

Another warning message while installing Galera package is as shown below:

```
warning: galera-25.3.19-1.rhel7.el7.centos.x86_64.rpm: Header V4 DSA/SHA1 Signature, key ID
1bb943db: NOKEY
```

The solution for this is to import the key:

```
#rpm --import http://yum.mariadb.org/RPM-GPG-KEY-MariaDB
```

After step 4, the installation will be completed. The last step will be to run mysql_secure_installation to secure the production server by dis allowing remote login for root, creating root password and removing the test database.

- 5) mysql_secure_installation

2.1.2.1.9 Why Source RPMs (SRPMs) Aren't Packaged For Some Platforms

MariaDB source RPMs (SRPMs) are not packaged on all platforms for which MariaDB RPMs are packaged.

The reason is that MariaDB's build process relies heavily on `cmake` for a lot of things. In this specific case, MariaDB's build process relies on CMake CPack Package Generators to build RPMs. The specific package generator that it uses to build RPMs is called `CPackRPM`.

Support for source RPMs in `CPackRPM` became usable with MariaDB's build system starting from around cmake 3.10. This means that we do not produce source RPMs on platforms where the installed `cmake` version is older than that.

See also Building MariaDB from a Source RPM.

2.1.2.1.10 Building MariaDB from a Source RPM

For some distributions you can build MariaDB from a source RPM. (See also Why Source RPMs (SRPMs) Aren't Packaged For Some Platforms).

You can build it as follows:

using dnf

On RHEL8 you might need to start with:

```
sudo dnf config-manager --set-enabled codeready-builder-beta-for-rhel-8-x86_64-rpms
```

Then, on all dnf distributions:

```
sudo dnf install rpm-build perl-generators
dnf download --source MariaDB
sudo dnf builddep MariaDB-*.src.rpm
rpmbuild --rebuild MariaDB-*.src.rpm
```

using yum

```
sudo yum install rpm-build yum-utils
yumdownloader --source MariaDB
sudo yum-builddep MariaDB-*.src.rpm
rpmbuild --rebuild MariaDB-*.src.rpm
```

using zypper

```
sudo zypper in rpm-build
sudo zypper si MariaDB
sudo rpmbuild -bb /usr/src/packages/SPECS/MariaDB.spec
```

Or (to avoid building as root):

```
sudo zypper in rpm-build
sudo zypper si -d MariaDB
zypper --pkg-cache-dir=`pwd` si --download-only MariaDB
rpmbuild --rebuild mariadb/srpms/MariaDB-*.src.rpm
```

2.1.2.2 Installing MariaDB .deb Files

Contents

Installing MariaDB with APT

On Debian, Ubuntu, and other similar Linux distributions, it is highly recommended to install the relevant `.deb` packages from MariaDB's repository using `apt`, `aptitude`, Ubuntu Software Center, Synaptic Package Manager, or another package manager.

This page walks you through the simple installation steps using `apt`.

Adding the MariaDB APT repository

We currently have APT repositories for the following Linux distributions:

- Debian 10 (Buster)
- Debian 11 (Bullseye)
- Debian 12 (Bookworm)
- Debian Unstable (Sid)
- Ubuntu 18.04 LTS (Bionic)
- Ubuntu 20.04 LTS (Focal)
- Ubuntu 22.04 (Jammy)
- Ubuntu 22.10 (Kinetic)
- Ubuntu 23.04 (Lunar)

Using the MariaDB Package Repository Setup Script

If you want to install MariaDB with `apt`, then you can configure `apt` to install from MariaDB Corporation's MariaDB Package Repository by using the MariaDB Package Repository setup script.

MariaDB Corporation provides a MariaDB Package Repository for several Linux distributions that use `apt` to manage packages. This repository contains software packages related to MariaDB Server, including the server itself, clients and utilities, client libraries, plugins, and Mariabackup. The MariaDB Package Repository setup script automatically configures your system to install packages from the MariaDB Package Repository.

To use the script, execute the following command:

```
curl -sS https://downloads.mariadb.com/MariaDB/mariadb_repo_setup | sudo bash
```

Note that this script also configures a repository for MariaDB MaxScale and a repository for MariaDB Tools, which currently only contains Percona XtraBackup and its dependencies.

See MariaDB Package Repository Setup and Usage for more information.

Using the MariaDB Repository Configuration Tool

If you want to install MariaDB with `apt`, then you can configure `apt` to install from MariaDB Foundation's MariaDB Repository by using the MariaDB Repository Configuration Tool.

The MariaDB Foundation provides a MariaDB repository for several Linux distributions that use `apt-get` to manage packages. This repository contains software packages related to MariaDB Server, including the server itself, clients and utilities, client libraries, plugins, and Mariabackup. The MariaDB Repository Configuration Tool can easily generate the appropriate commands to add the repository for your distribution.

There are several ways to add the repository.

Executing add-apt-repository

One way to add an `apt` repository is by using the `add-apt-repository` command. This command will add the repository configuration to `/etc/apt/sources.list`.

For example, if you wanted to use the repository to install MariaDB 10.3 on Ubuntu 18.04 LTS (Bionic), then you could use the following commands to add the MariaDB `apt` repository:

```
sudo apt-get install software-properties-common
sudo add-apt-repository 'deb [arch=amd64,arm64,ppc64el]
http://sfo1.mirrors.digitalocean.com/mariadb/repo/10.3/ubuntu bionic main'
```

And then you would have to update the package cache by executing the following command:

```
sudo apt update
```

Creating a Source List File

Another way to add an `apt` repository is by creating a source list file in `/etc/apt/sources.list.d/`.

For example, if you wanted to use the repository to install MariaDB 10.3 on Ubuntu 18.04 LTS (Bionic), then you could create the `MariaDB.list` file in `/etc/apt/sources.list.d/` with the following contents to add the MariaDB `apt` repository:

```
# MariaDB 10.3 repository list - created 2019-01-27 09:50 UTC
# http://downloads.mariadb.org/mariadb/repositories/
deb [arch=amd64,arm64,ppc64el] http://sfo1.mirrors.digitalocean.com/mariadb/repo/10.3/ubuntu
bionic main
deb-src http://sfo1.mirrors.digitalocean.com/mariadb/repo/10.3/ubuntu bionic main
```

And then you would have to update the package cache by executing the following command:

```
sudo apt update
```

Using Ubuntu Software Center

Another way to add an `apt` repository is by using Ubuntu Software Center.

You can do this by going to the **Software Sources** window. This window can be opened either by navigating to **Edit** > **Software Sources** or by navigating to **System** > **Administration** > **Software Sources**.

Once the **Software Sources** window is open, go to the **Other Software** tab, and click the **Add** button. At that point, you can input the repository information provided by the MariaDB Repository Configuration Tool.

See here for more information.

Using Synaptic Package Manager

Another way to add an `apt` repository is by using Synaptic Package Manager.

You can do this by going to the **Software Sources** window. This window can be opened either by navigating to **System** > **Administrator** > **Software Sources** or by navigating to **Settings** > **Repositories**.

Once the **Software Sources** window is open, go to the **Other Software** tab, and click the **Add** button. At that point, you can

input the repository information provided by the MariaDB Repository Configuration Tool.

See here for more information.

Pinning the MariaDB Repository to a Specific Minor Release

If you wish to pin the `apt` repository to a specific minor release, or if you would like to downgrade to a specific minor release, then you can create a `apt` repository with the URL hard-coded to that specific minor release.

The MariaDB Foundation archives repositories of old minor releases at the following URL:

- http://archive.mariadb.org/

Archives are only of the distros and architectures supported at the time of release. For example MariaDB 10.0.38 is exists for Ubuntu `precise`, `trusty`, `xenial`, `wily`, and `yakkety` obtained looking in https://archive.mariadb.org/mariadb-10.0.38/repo/ubuntu/dists.

For example, if you wanted to pin your repository to MariaDB 10.5.9 on Ubuntu 20.04 LTS (Focal), then you would have to first remove any existing MariaDB repository source list file from `/etc/apt/sources.list.d/`. And then you could use the following commands to add the MariaDB `apt-get` repository:

```
sudo add-apt-repository 'deb [arch=amd64,arm64,ppc64el,s390x]
http://archive.mariadb.org/mariadb-10.5.9/repo/ubuntu/ focal main main/debug'
```

Ensure you have the signing key installed.

Ubuntu Xenial and older will need:

```
sudo apt-get install -y apt-transport-https
```

And then you would have to update the package cache by executing the following command:

```
sudo apt update
```

Updating the MariaDB APT repository to a New Major Release

MariaDB's `apt` repository can be updated to a new major release. How this is done depends on how you originally configured the repository.

Updating the Major Release with the MariaDB Package Repository Setup Script

If you configured `apt` to install from MariaDB Corporation's MariaDB Package Repository by using the MariaDB Package Repository setup script, then you can update the major release that the repository uses by running the script again.

Updating the Major Release with the MariaDB Repository Configuration Tool

If you configured `apt` to install from MariaDB Foundation's MariaDB Repository by using the MariaDB Repository Configuration Tool, then you can update the major release in various ways, depending on how you originally added the repository.

Updating a Repository with add-apt-repository

If you added the `apt` repository by using the `add-apt-repository` command, then you can update the major release that the repository uses by using the the `add-apt-repository` command again.

First, look for the repository string for the old version in `/etc/apt/sources.list`.

And then, you can remove the repository for the old version by executing the `add-apt-repository` command and providing the `--remove` option. For example, if you wanted to remove a MariaDB 10.2 repository, then you could do so by executing something like the following:

```
sudo add-apt-repository --remove 'deb [arch=amd64,arm64,ppc64el]
http://sfo1.mirrors.digitalocean.com/mariadb/repo/10.2/ubuntu bionic main'
```

After that, you can add the repository for the new version with the `add-apt-repository` command. For example, if you wanted to use the repository to install MariaDB 10.3 on Ubuntu 18.04 LTS (Bionic), then you could use the following commands to add the MariaDB `apt` repository:

```
sudo apt-get install software-properties-common
sudo add-apt-repository 'deb [arch=amd64,arm64,ppc64el]
http://sfo1.mirrors.digitalocean.com/mariadb/repo/10.3/ubuntu bionic main'
```

And then you would have to update the package cache by executing the following command:

```
sudo apt update
```

After that, the repository should refer to MariaDB 10.3.

Updating a Source List File

If you added the `apt` repository by creating a source list file in `/etc/apt/sources.list.d/`, then you can update the major release that the repository uses by updating the source list file in-place. For example, if you wanted to change the repository from MariaDB 10.2 to MariaDB 10.3, and if the source list file was at `/etc/apt/sources.list.d/MariaDB.list`, then you could execute the following:

```
sudo sed -i 's/10.2/10.3/' /etc/apt/sources.list.d/MariaDB.list
```

And then you would have to update the package cache by executing the following command:

```
sudo apt update
```

After that, the repository should refer to MariaDB 10.3.

Importing the MariaDB GPG Public Key

Before MariaDB can be installed, you also have to import the GPG public key that is used to verify the digital signatures of the packages in our repositories. This allows the `apt` utility to verify the integrity of the packages that it installs.

- Prior to Debian 9 (Stretch), and Debian Unstable (Sid), and Ubuntu 16.04 LTS (Xenial), the id of our GPG public key is `0xcbcb082a1bb943db`. The full key fingerprint is:

```
1993 69E5 404B D5FC 7D2F E43B CBCB 082A 1BB9 43DB
```

The `apt-key` utility can be used to import this key. For example:

```
sudo apt-key adv --recv-keys --keyserver hkp://keyserver.ubuntu.com:80 0xcbcb082a1bb943db
```

- Starting with Debian 9 (Stretch) and Ubuntu 16.04 LTS (Xenial), the id of our GPG public key is `0xF1656F24C74CD1D8`. The full key fingerprint is:

```
177F 4010 FE56 CA33 3630  0305 F165 6F24 C74C D1D8
```

The `apt-key` utility can be used to import this key. For example:

```
sudo apt-key adv --recv-keys --keyserver hkp://keyserver.ubuntu.com:80 0xF1656F24C74CD1D8
```

Starting with Debian 9 (Stretch), the `dirmngr` package needs to be installed before the GPG public key can be imported. To install it, execute: `sudo apt install dirmngr`

If you are unsure which GPG public key you need, then it is perfectly safe to import both keys.

The command used to import the GPG public key is the same on both Debian and Ubuntu. For example:

```
$ sudo apt-key adv --recv-keys --keyserver hkp://keyserver.ubuntu.com:80 0xcbcb082a1bb943db
Executing: gpg --ignore-time-conflict --no-options --no-default-keyring --secret-keyring
/tmp/tmp.ASyOPV87XC --trustdb-name /etc/apt/trustdb.gpg --keyring /etc/apt/trusted.gpg --
primary-keyring /etc/apt/trusted.gpg --recv-keys --keyserver hkp://keyserver.ubuntu.com:80
0xcbcb082a1bb943db
gpg: requesting key 1BB943DB from hkp server keyserver.ubuntu.com
gpg: key 1BB943DB: "MariaDB Package Signing Key <package-signing-key@mariadb.org>" imported
gpg: no ultimately trusted keys found
gpg: Total number processed: 1
gpg:               imported: 1
```

Once the GPG public key is imported, you are ready to install packages from the repository.

Installing MariaDB Packages with APT

After the `apt` repository is configured, you can install MariaDB by executing the `apt-get` command. The specific command that you would use would depend on which specific packages that you want to install.

Installing the Most Common Packages with APT

To Install the most common packages, first you would have to update the package cache by executing the following command:

```
sudo apt update
```

MariaDB starting with 10.4

In MariaDB 10.4 and later, to Install the most common packages, execute the following command:

```
sudo apt-get install mariadb-server galera-4 mariadb-client libmariadb3 mariadb-backup mariadb
```

MariaDB 10.2 - 10.3

In MariaDB 10.2 and MariaDB 10.3, to Install the most common packages, execute the following command:

```
sudo apt-get install mariadb-server galera mariadb-client libmariadb3 mariadb-backup mariadb-
```

Installing MariaDB Server with APT

To Install MariaDB Server, first you would have to update the package cache by executing the following command:

```
sudo apt update
```

Then, execute the following command:

```
sudo apt-get install mariadb-server
```

Installing MariaDB Galera Cluster with APT

The process to install MariaDB Galera Cluster with the MariaDB `apt-get` repository is practically the same as installing standard MariaDB Server.

In MariaDB 10.1 and later, Galera Cluster support has been included in the standard MariaDB Server packages, so you will need to install the `mariadb-server` package, as you normally would.

In MariaDB 10.4 and later, you also need to install the `galera-4` package to obtain the Galera 4 wsrep provider library.

In MariaDB 10.3 and before, you also need to install the `galera-3` package to obtain the Galera 3 wsrep provider library.

To install MariaDB Galera Cluster, first you would have to update the package cache by executing the following command:

```
sudo apt update
```

MariaDB starting with 10.4

In MariaDB 10.4 and later, to install MariaDB Galera Cluster, you could execute the following command:

```
sudo apt-get install mariadb-server mariadb-client galera-4
```

MariaDB until 10.3

In MariaDB 10.3 and before, to install MariaDB Galera Cluster, you could execute the following command:

```
sudo apt-get install mariadb-server mariadb-client galera-3
```

MariaDB Galera Cluster also has a separate package that can be installed on arbitrator nodes. In MariaDB 10.4 and later, the package is called `galera-arbitrator-4`. In MariaDB 10.3 and before, the package is called `galera-arbitrator-3`. This package should be installed on whatever node you want to serve as the arbitrator. It can either run on a separate server that is not acting as a cluster node, which is the recommended configuration, or it can run on a server that is also acting as an existing cluster node.

MariaDB starting with 10.4

In MariaDB 10.4 and later, to install the arbitrator package, you could execute the following command:

```
sudo apt-get install galera-arbitrator-4
```

MariaDB until 10.3

In MariaDB 10.3 and before, to install the arbitrator package, you could execute the following command:

```
sudo apt-get install galera-arbitrator-3
```

See MariaDB Galera Cluster for more information on MariaDB Galera Cluster.

Installing MariaDB Clients and Client Libraries with APT

In MariaDB 10.2 and later, MariaDB Connector/C has been included as the client library.

To Install the clients and client libraries, first you would have to update the package cache by executing the following command:

```
sudo apt update
```

Then, in MariaDB 10.2 and later, execute the following command:

```
sudo apt-get install mariadb-client libmariadb3
```

Installing Mariabackup with APT

To install Mariabackup, first you would have to update the package cache by executing the following command:

```
sudo apt update
```

Then, execute the following command:

```
sudo apt-get install mariadb-backup
```

Installing Plugins with APT

Some plugins may also need to be installed.

For example, to install the `cracklib_password_check` password validation plugin, first you would have to update the package cache by executing the following command:

```
sudo apt update
```

Then, execute the following command:

```
sudo apt-get install mariadb-cracklib-password-check
```

Installing Older Versions from the Repository

The MariaDB `apt` repository contains the last few versions of MariaDB. To show what versions are available, use the `apt-cache` command:

```
sudo apt-cache showpkg mariadb-server
```

In the output you will see the available versions.

To install an older version of a package instead of the latest version we just need to specify the package name, an equal sign, and then the version number.

However, when installing an older version of a package, if `apt-get` has to install dependencies, then it will automatically choose to install the latest versions of those packages. To ensure that all MariaDB packages are on the same version in this scenario, it is necessary to specify them all. Therefore, to install MariaDB 10.3.14 from this `apt` repository, we would do the following:

```
sudo apt-get install mariadb-server=10.3.14-1 mariadb-client=10.3.14-1 libmariadb3=10.3.14-1
mariadb-backup=10.3.14-1 mariadb-common=10.3.14-1
```

The rest of the install and setup process is as normal.

Installing MariaDB with dpkg

While it is not recommended, it is possible to download and install the `.deb` packages manually. However, it is generally recommended to use a package manager like `apt-get`.

A tarball that contains the `.deb` packages can be downloaded from the following URL:

- https://downloads.mariadb.com

For example, to install the MariaDB 10.4.8 `.deb` packages on Ubuntu 18.04 LTS (Bionic), you could execute the following:

```
sudo apt-get update
sudo apt-get install libdbi-perl libdbd-mysql-perl psmisc libaio1 socat
wget https://downloads.mariadb.com/MariaDB/mariadb-10.4.8/repo/ubuntu/mariadb-10.4.8-ubuntu-
bionic-amd64-debs.tar
tar -xvf mariadb-10.4.8-ubuntu-bionic-amd64-debs.tar
cd mariadb-10.4.8-ubuntu-bionic-amd64-debs/
sudo dpkg --install ./mariadb-common*.deb \
   ./mysql-common*.deb \
   ./mariadb-client*.deb \
   ./libmariadb3*.deb \
   ./libmysqlclient18*.deb
sudo dpkg --install ./mariadb-server*.deb \
   ./mariadb-backup*.deb \
   ./galera-4*.deb
```

After Installation

After the installation is complete, you can start MariaDB.

If you are using MariaDB Galera Cluster, then keep in mind that the first node will have to be bootstrapped.

Available DEB Packages

The available DEB packages depend on the specific MariaDB release series.

Available DEB Packages in MariaDB 10.4

MariaDB starting with 10.4

In MariaDB 10.4, the following DEBs are available:

Package Name	Description

galera-4	The WSREP provider for Galera 4.
libmariadb3	Dynamic client libraries.
libmariadb-dev	Development headers and static libraries.
libmariadbclient18	Virtual package to satisfy external depends
libmysqlclient18	Virtual package to satisfy external depends
mariadb-backup	Mariabackup
mariadb-client	Client tools like mariadb CLI, mariadb-dump, and others.
mariadb-client-core	Core client tools
mariadb-common	Character set files and /etc/my.cnf
mariadb-plugin-connect	The CONNECT storage engine.
mariadb-plugin-cracklib-password-check	The cracklib_password_check password validation plugin.
mariadb-plugin-gssapi-client	The client-side component of the gssapi authentication plugin.
mariadb-plugin-gssapi-server	The server-side component of the gssapi authentication plugin.
mariadb-plugin-rocksdb	The MyRocks storage engine.
mariadb-plugin-spider	The SPIDER storage engine.
mariadb-plugin-tokudb	The TokuDB storage engine.
mariadb-server	The server and server tools, like myisamchk and mariadb-hotcopy are here.
mariadb-server-core	The core server.
mariadb-test	mysql-client-test executable, and **mysql-test** framework with the tests.
mariadb-test-data	MariaDB database regression test suite - data files

Available DEB Packages in MariaDB 10.2 and MariaDB 10.3

MariaDB starting with 10.2

In MariaDB 10.2 and MariaDB 10.3, the following DEBs are available:

Package Name	Description
galera	The WSREP provider for Galera 3.
libmariadb3	Dynamic client libraries.
libmariadb-dev	Development headers and static libraries.
libmariadbclient18	Virtual package to satisfy external depends
libmysqlclient18	Virtual package to satisfy external depends
mariadb-backup	Mariabackup
mariadb-client	Client tools like mariadb CLI, mariadb-dump, and others.
mariadb-client-core	Core client tools
mariadb-common	Character set files and /etc/my.cnf
mariadb-plugin-connect	The CONNECT storage engine.
mariadb-plugin-cracklib-password-check	The cracklib_password_check password validation plugin.
mariadb-plugin-gssapi-client	The client-side component of the gssapi authentication plugin.
mariadb-plugin-gssapi-server	The server-side component of the gssapi authentication plugin.
mariadb-plugin-rocksdb	The MyRocks storage engine.

`mariadb-plugin-spider`	The `SPIDER` storage engine.
`mariadb-plugin-tokudb`	The TokuDB storage engine.
`mariadb-server`	The server and server tools, like myisamchk and mariadb-hotcopy are here.
`mariadb-server-core`	The core server.
`mariadb-test`	`mysql-client-test` executable, and **mysql-test** framework with the tests.
`mariadb-test-data`	MariaDB database regression test suite - data files

Actions Performed by DEB Packages

Users and Groups Created

When the `mariadb-server` DEB package is installed, it will create a user and group named `mysql`, if they do not already exist.

2.1.2.3 Installing MariaDB MSI Packages on Windows

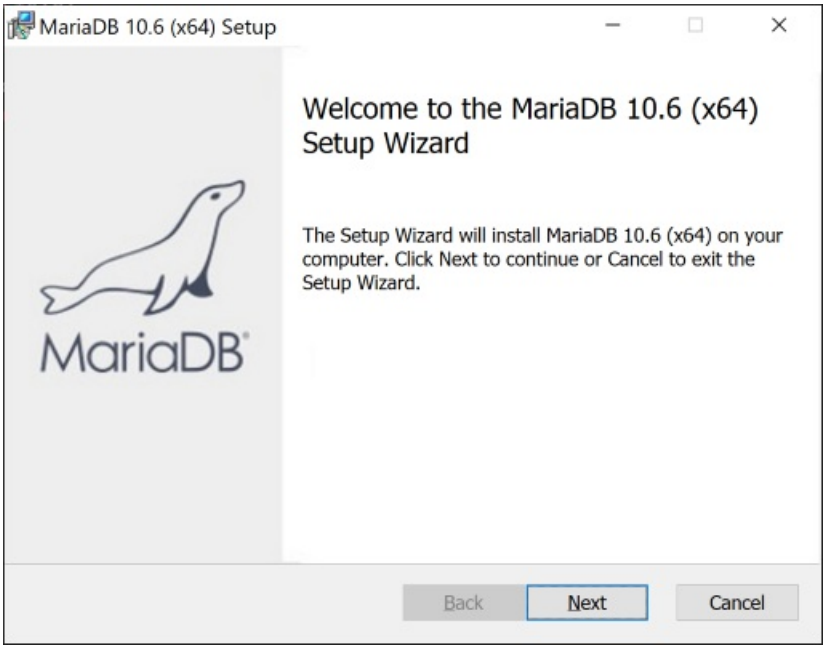
MSI packages is available for x64 (64 bit) processor architectures and, in some older releases only, for x86 (32 bit). We'll use screenshots from an x64 installation below (the 32 bit installer is very similar).

Contents

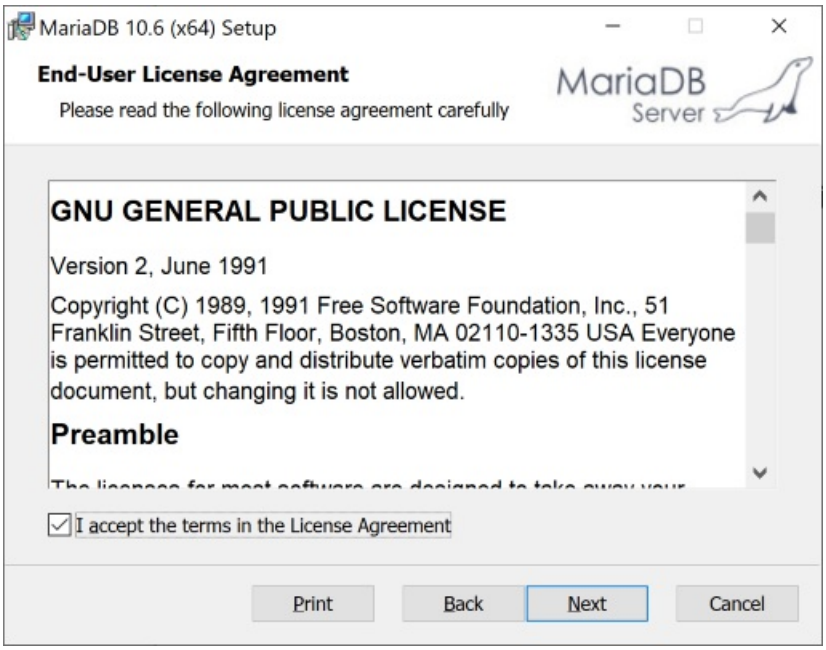
Installation UI

This is the typical mode of installation. To start the installer, just click on the mariadb-<major>.<minor>.<patch>.msi

Welcome

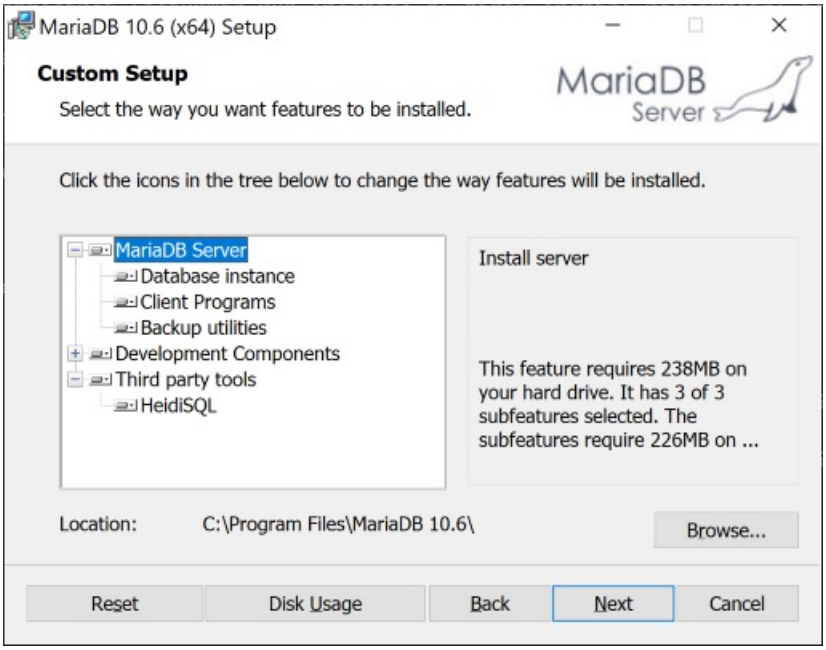


License Agreement



Click on "I accept the terms"

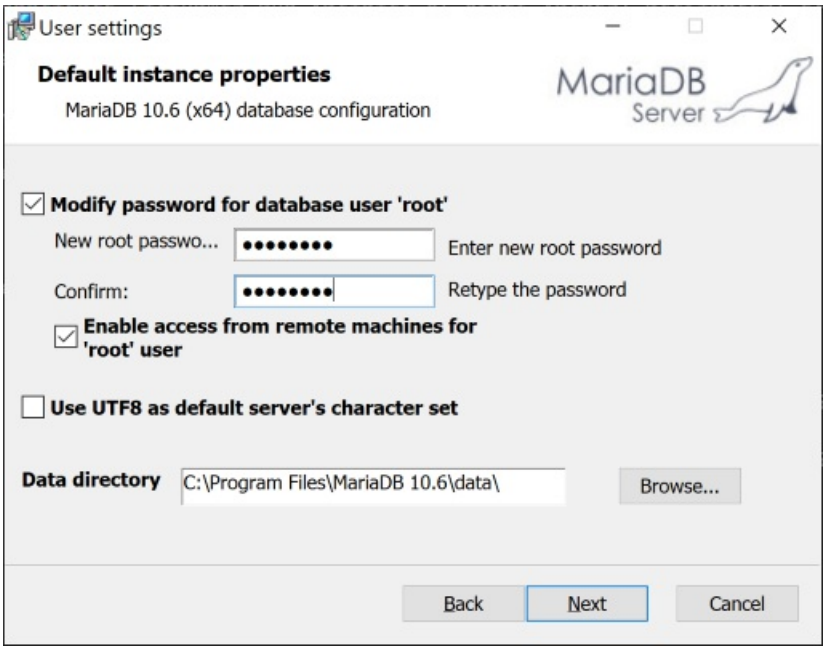
Custom Setup



Here, you can choose what features to install. By default, all features are installed with the exception of the debug symbols. If the "Database instance" feature is selected, the installer will create a database instance, by default running as a service. In this case the installer will present additional dialogs to control various database properties. Note that you do not necessarily have to create an instance at this stage. For example, if you already have MySQL or MariaDB databases running as services, you can just upgrade them during the installation. Also, you can create additional database instances after the installation, with the `mysql_install_db.exe` utility.

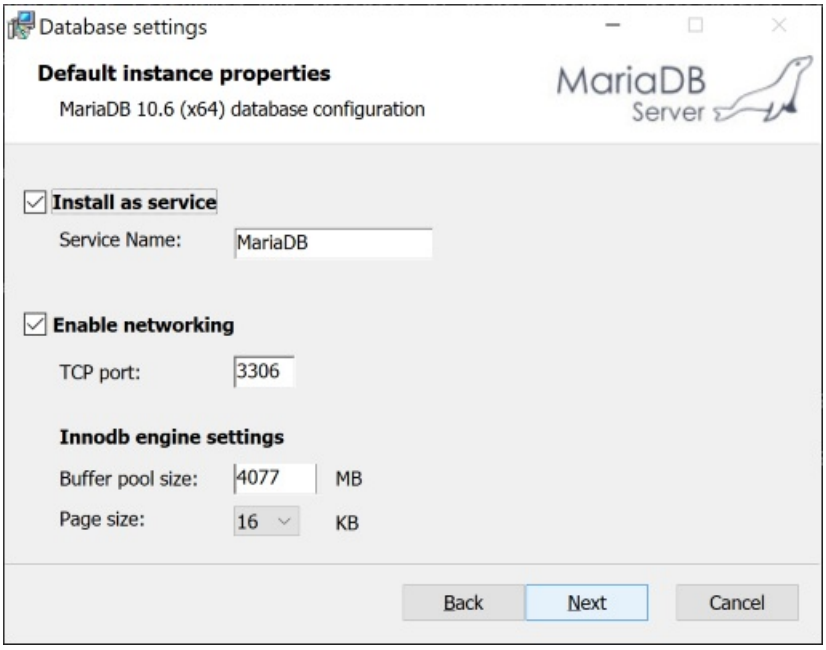
NOTE: By default, if you install a database instance, the data directory will be in the "data" folder under the installation root. To change the data directory location, select "Database instance" in the feature tree, and use the "Browse" button to point to another place.

Database Authentication/Security Related Properties



This dialog is shown if you selected the "Database instance" feature. Here, you can set the password for the "root" database user and specify whether root can access database from remote machines. The "Create anonymous account" setting allows for anonymous (non-authenticated) users. It is off by default and it is not recommended to change this setting.

Other Database Properties



- Install as service

  Defines whether the database should be run as a service. If it should be run as a service, then it also defines the service name. It is recommended to run your database instance as a service as it greatly simplifies database management. In MariaDB 10.4 and later, the default service name used by the MSI installer is "MariaDB". In 10.3 and before, the default service name used by the MSI installer is "MySQL". Note that the default service name for the `--install` and `--install-manual` options for `mysqld.exe` is "MySQL" in all versions of MariaDB.

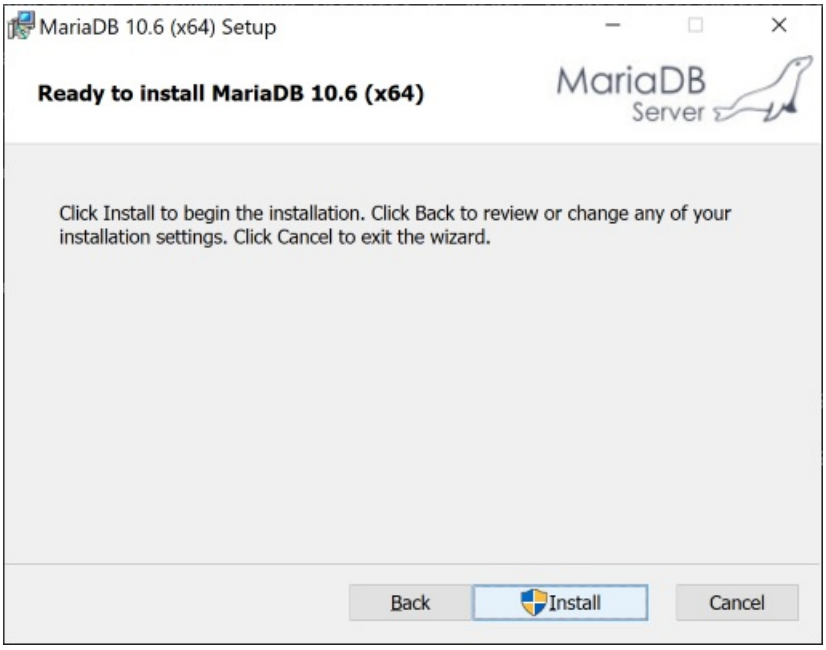
- Enable Networking

  Whether to enable TCP/IP (recommended) and which port MariaDB should listen to. If security is a concern, you can change the bind-address parameter post-installation to bind to only local addresses. If the "Enable networking" checkbox is deselected, the database will use named pipes for communication.

- InnoDB engine settings

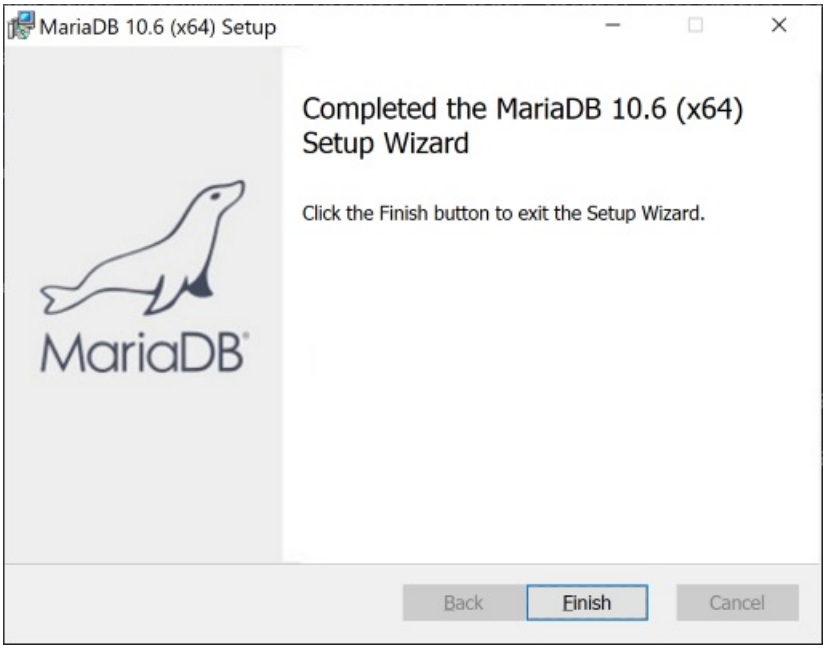
  Defines the InnoDB buffer pool size, and the InnoDB page size. The default buffer pool size is 12.5% of RAM, and depending on your requirements you can give InnoDB more (up to 70-80% RAM). 32 bit versions of MariaDB have restrictions on maximum buffer pool size, which is approximately 1GB, due to virtual address space limitations for 32bit processes. A 16k page size is suitable for most situations. See the innodb_page_size system variable for details on other settings.

Ready to Install



At this point, all installation settings are collected. Click on the "Install" button.

End

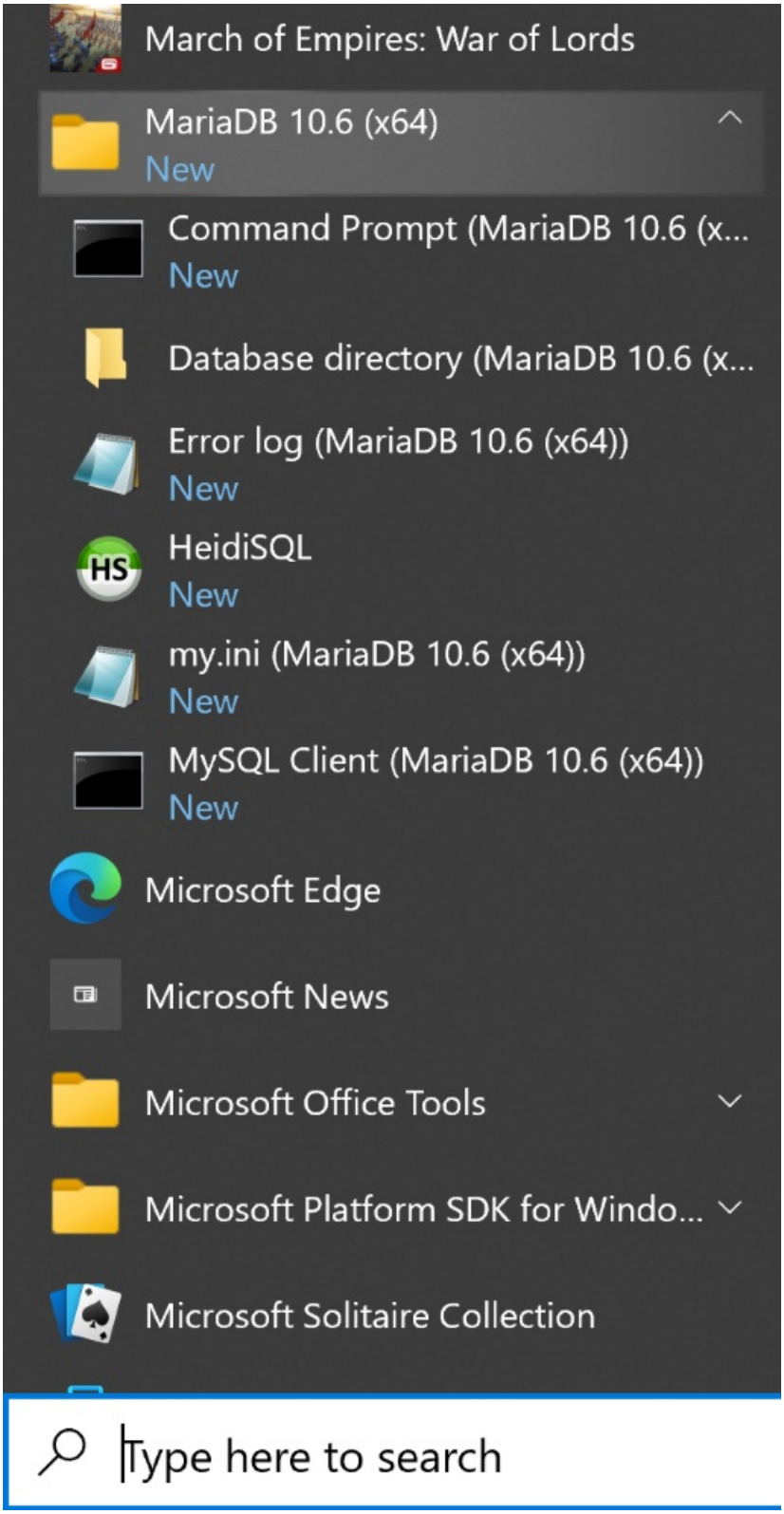


Installation is finished now. If you have upgradable instances of MariaDB/MySQL, running as services, this dialog will present a "Do you want to upgrade existing instances" checkbox (if selected, it launches the Upgrade Wizard post-installation).

If you installed a database instance as service, the service will be running already.

New Entries in Start Menu

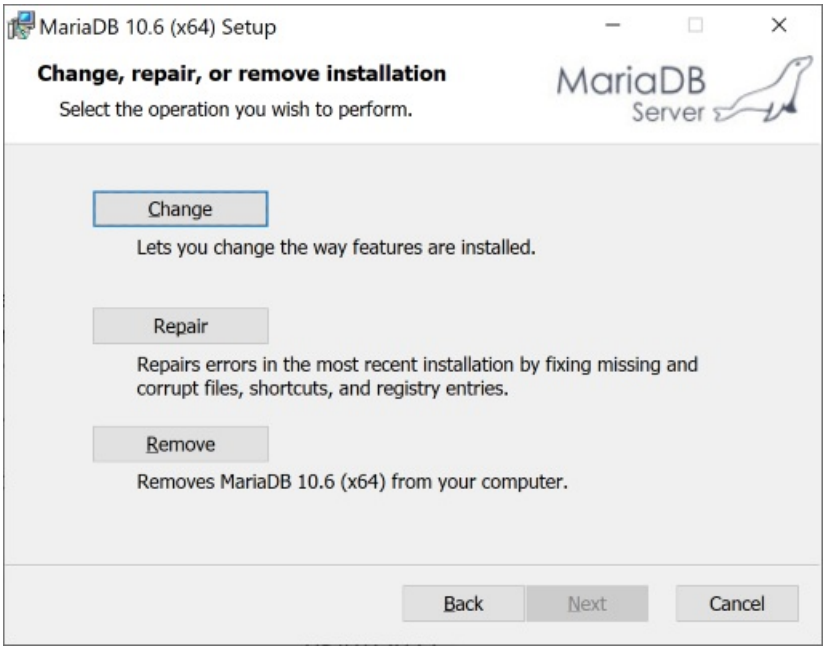
Installation will add some entries in the Start Menu:



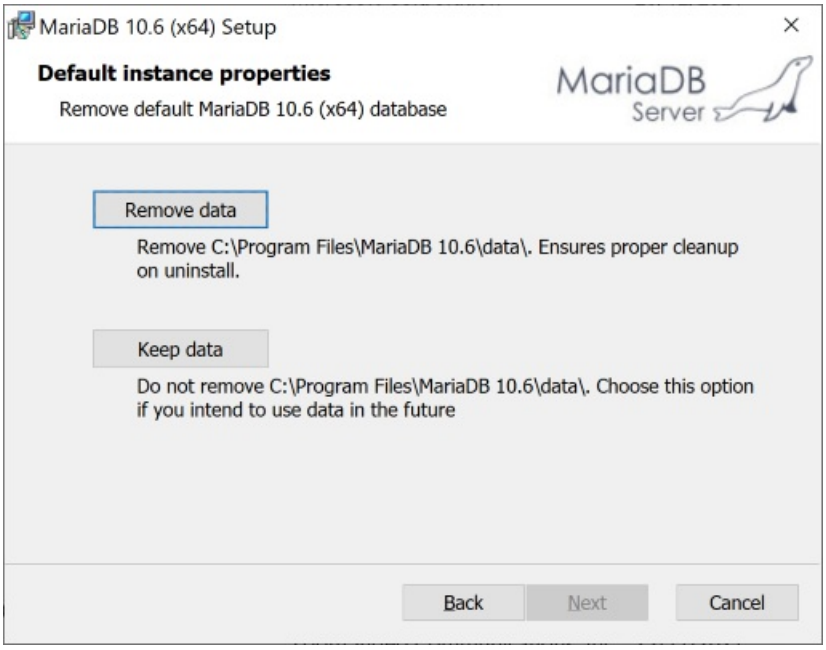
- MariaDB Client - Starts command line client mysql.exe
- Command Prompt - Starts a command prompt. Environment is set such that "bin" directory of the installation is included into PATH environment variable, i.e you can use this command prompt to issue MariaDB commands (mysqldadmin, mysql etc...)
- Database directory - Opens the data directory in Explorer.
- Error log - Opens the database error log in Notepad.
- my.ini - Opens the database configuration file my.ini in Notepad.
- Upgrade Wizard - Starts the Wizard to upgrade an existing MariaDB/MySQL database instance to this MariaDB version.

Uninstall UI

In the Explorer applet "Programs and Features" (or "Add/Remove programs" on older Windows), find the entry for MariaDB, choose Uninstall/Change and click on the "Remove" button in the dialog below.



If you installed a database instance, you will need to decide if you want to remove or keep the data in the database directory.



Silent Installation

The MSI installer supports silent installations as well. In its simplest form silent installation with all defaults can be performed from an elevated command prompt like this:

```
msiexec /i path-to-package.msi /qn
```

Note: the installation is silent due to msiexe.exe's /qn switch (no user interface), if you omit the switch, the installation will have the full UI.

Properties

Silent installations also support installation properties (a property would correspond for example to checked/unchecked state of a checkbox in the UI, user password, etc). With properties the command line to install the MSI package would look like this:

```
msiexec /i path-to-package.msi [PROPERTY_1=VALUE_1 ... PROPERTY_N=VALUE_N] /qn
```

The MSI installer package requires property names to be all capitals and contain only English letters. By convention, for a boolean property, an empty value means "false" and a non-empty is "true".

MariaDB installation supports the following properties:

Property name	Default value	Description
INSTALLDIR	%ProgramFiles%\MariaDB <version>\	Installation root
PORT	3306	--port parameter for the server
ALLOWREMOTEROOTACCESS		Allow remote access for root user
BUFFERPOOLSIZE	RAM/8	Bufferpoolsize for innodb
CLEANUPDATA	1	Remove the data directory (uninstall only)
DATADIR	INSTALLDIR\data	Location of the data directory
DEFAULTUSER		Allow anonymous users
PASSWORD		Password of the root user
SERVICENAME		Name of the Windows service. A service is not created if this value is empty.
SKIPNETWORKING		Skip networking
STDCONFIG	1	Corresponds to "optimize for transactions" in the GUI, default engine innodb, strict sql mode
UTF8		if set, adds character-set-server=utf8 to my.ini file
PAGESIZE	16K	page size for innodb

Features

Feature is a Windows installer term for a unit of installation. Features can be selected and deselected in the UI in the feature tree in the "Custom Setup" dialog.

Silent installation supports adding features with the special property `ADDLOCAL=Feature_1,..,Feature_N` and removing features with `REMOVE=Feature_1,..., Feature_N`

Features in the MariaDB installer:

Feature id	Installed by default?	Description
DBInstance	yes	Install database instance
Client	yes	Command line client programs
MYSQLSERVER	yes	Install server
SharedLibraries	yes	Install client shared library
DEVEL	yes	install C/C++ header files and client libraries
HeidiSQL	yes	Installs HeidiSQL

Silent Installation Examples

All examples here require running as administrator (and elevated command line in Vista and later)

- Install default features, database instance as service, non-default datadir and port

```
msiexec /i path-to-package.msi SERVICENAME=MySQL DATADIR=C:\mariadb5.2\data PORT=3307 /qn
```

- Install service, add debug symbols, do not add development components (client libraries and headers)

```
msiexec /i path-to-package.msi SERVICENAME=MySQL ADDLOCAL=DEBUGSYMBOLS REMOVE=DEVEL /qn
```

Silent Uninstall

To uninstall silently, use the `REMOVE=ALL` property with msiexec:

```
msiexec /i path-to-package.msi REMOVE=ALL /qn
```

To keep the data directory during an uninstall, you will need to pass an additional parameter:

```
msiexec /i path-to-package.msi REMOVE=ALL CLEANUPDATA="" /qn
```

Installation Logs

If you encounter a bug in the installer, the installer logs should be used for diagnosis. Please attach verbose logs to the bug reports you create. To create a verbose installer log, start the installer from the command line with the `/l*v` switch, like so:

```
  msiexec.exe /i path-to-package.msi  /l*v path-to-logfile.txt
```

Running 32 and 64 Bit Distributions on the Same Machine

It is possible to install 32 and 64 bit packages on the same Windows x64.

Apart from testing, an example where this feature can be useful is a development scenario, where users want to run a 64 bit server and develop both 32 and 64 bit client components. In this case the full 64 bit package can be installed, including a database instance plus development-related features (headers and libraries) from the 32 bit package.

2.1.2.4 Installing MariaDB Server PKG packages on macOS

MariaDB Server does not currently provide a `.pkg` installer for macOS. For information about how to install MariaDB Server on macOS using Homebrew, see Installing MariaDB Server on macOS Using Homebrew.

2.1.2.5 Installing MariaDB Binary Tarballs

Contents

MariaDB Binary tarballs are named following the pattern: mariadb-VERSION-OS.tar.gz. Be sure to download the correct version for your machine.

Note: Some older binary tarballs are marked *'(GLIBC_2.14)'* or *'(requires GLIBC_2.14+)'*. These binaries are built the same as the others, but on a newer build host, and they require GLIBC 2.14 or higher. Use the other binaries for machines with older versions of GLIBC installed. Run `ldd --version` to see which version is running on your distribution.

Others are marked *'systemd'*, which are for systems with `systemd` and GLIBC 2.19 or higher.

To install the binaries, unpack the distribution into the directory of your choice and run the `mariadb-install-db` script.

In the example below we install MariaDB in the `/usr/local/mysql` directory (this is the default location for MariaDB for many platforms). However any other directory should work too.

We install the binary with a symlink to the original name. This is done so that you can easily change MariaDB versions just by moving the symlink to point to another directory.

Ensure You Use the Correct my.cnf Files

MariaDB searches for the configuration files ' `/etc/my.cnf` ' (on some systems ' `/etc/mysql/my.cnf` ') and ' `~/.my.cnf` '. If you have an old `my.cnf` file (maybe from a system installation of MariaDB or MySQL) you need to take care that you don't accidentally use the old one with your new binary .tar installation.

The normal solution for this is to ignore the `my.cnf` file in `/etc` when you use the programs in the tar file.

This is done by creating your own .my.cnf file in your home directory and telling mariadb-install-db, mysqld_safe and possibly mysql (the command-line client utility) to **only** use this one with the option '`--defaults-file=~/.my.cnf`'. Note that this has to be first option for the above commands!

Installing MariaDB as root in /usr/local/mysql

If you have root access to the system, you probably want to install MariaDB under the user and group 'mysql' (to keep compatibility with MySQL installations):

```
groupadd mysql
useradd -g mysql mysql
cd /usr/local
tar -zxvpf /path-to/mariadb-VERSION-OS.tar.gz
ln -s mariadb-VERSION-OS mysql
cd mysql
./scripts/mariadb-install-db --user=mysql
chown -R root .
chown -R mysql data
```

The symlinking with `ln -s` is recommended as it makes it easy to install many MariaDB version at the same time (for easy testing, upgrading, downgrading etc).

If you are installing MariaDB to replace MySQL, then you can leave out the call to `mariadb-install-db`. Instead shut down MySQL. MariaDB should find the path to the data directory from your old `/etc/my.cnf` file (path may vary depending on your system).

To start mysqld you should now do:

```
./bin/mysqld_safe --user=mysql &
or
./bin/mysqld_safe --defaults-file=~/.my.cnf --user=mysql &
```

To test connection, modify your $PATH so you can invoke client such as mysql, mysqldump, etc.

```
export PATH=$PATH:/usr/local/mysql/bin/
```

You may want to modify your .bashrc or .bash_profile to make it permanent.

Installing MariaDB as Not root in Any Directory

Below, change /usr/local to the directory of your choice.

```
cd /usr/local
gunzip < /path-to/mariadb-VERSION-OS.tar.gz | tar xf -
ln -s mariadb-VERSION-OS mysql
cd mysql
./scripts/mariadb-install-db --defaults-file=~/.my.cnf
```

If you have problems with the above gunzip command line, you can instead, if you have gnu tar, do:

```
tar xfz /path-to/mariadb-VERSION-OS.tar.gz
```

To start mysqld you should now do:

```
./bin/mysqld_safe --defaults-file=~/.my.cnf &
```

Auto Start of mysqld

You can get mysqld (the MariaDB server) to autostart by copying the file `mysql.server` file to the right place.

```
cp support-files/mysql.server /etc/init.d/mysql.server
```

The exact place depends on your system. The `mysql.server` file contains instructions of how to use and fine tune it.

For systemd installation the mariadb.service file will need to be copied from the support-files/systemd folder to the /usr/lib/systemd/system/ folder.

```
cp support-files/systemd/mariadb.service /usr/lib/systemd/system/mariadb.service
```

Note that by default the /usr/ directory is write protected by systemd though, so when having the data directory in /usr/local/mysql/data as per the instructions above you also need to make that directory writable. You can do so by adding an extra service include file:

```
mkdir /etc/systemd/system/mariadb.service.d/

cat > /etc/systemd/system/mariadb.service.d/datadir.conf <<EOF
[Service]
ReadWritePaths=/usr/local/mysql/data
EOF

systemctl daemon-reload
```

After this you can start and stop the service using

```
systemctl start mariadb.service
```

and

```
systemctl stop mariadb.service
```

respectively.

Please refer to the systemd page for further information.

Post Installation

After this, remember to set proper passwords for all accounts accessible from untrusted sources, to avoid exposing the host to security risks!

Also consider using the mysql.server to start MariaDB automatically when your system boots.

On systems using systemd you can instead enable automatic startup during system boot with

```
systemctl enable mariadb.service
```

instead.

Our MariaDB binaries are similar to the Generic binaries available for the MySQL binary distribution. So for more options on using these binaries, the MySQL 5.5 manual entry on installing generic binaries can be consulted.

For details on the exact steps used to build the binaries, see the compiling MariaDB section of the KB.

2.1.2.6 Installing MariaDB Server on macOS Using Homebrew

Contents

MariaDB Server is available for installation on macOS (formerly Mac OS X) via the Homebrew package manager.

MariaDB Server is available as a Homebrew "bottle", a pre-compiled package. This means you can install it without having to build from source yourself. This saves time.

After installing Homebrew, MariaDB Server can be installed with this command:

```
brew install mariadb
```

After installation, start MariaDB Server:

```
mysql.server start
```

To auto-start MariaDB Server, use Homebrew's services functionality, which configures auto-start with the launchctl utility from launchd:

```
brew services start mariadb
```

After MariaDB Server is started, you can log in as your user:

```
mysql
```

Or log in as root:

```
sudo mysql -u root
```

Upgrading MariaDB

First you may need to update your brew installation:

```
brew update
```

Then, to upgrade MariaDB Server:

```
brew upgrade mariadb
```

Building MariaDB Server from source

In addition to the "bottled" MariaDB Server package available from Homebrew, you can use Homebrew to build MariaDB from source. This is useful if you want to use a different version of the server or enable some different capabilities that are not included in the bottle package.

Two components not included in the bottle package are the CONNECT and OQGRAPH engines, because they have non-standard dependencies. To build MariaDB Server with these engines, you must first install `boost` and `judy`. Follow these steps to install the dependencies and build the server:

```
brew install boost judy
brew install mariadb --build-from-source
```

You can also use Homebrew to build and install a pre-release version of MariaDB Server. Use this command to build and install a "development" version of MariaDB Server:

```
brew install mariadb --devel
```

Other resources

- mariadb.rb on github
- MariaDB Server on macOS: Does it even make sense to try? (video)

2.1.2.7 Installing MariaDB Windows ZIP Packages

Users need to run mysql_install_db.exe, without parameters to create a data directory, e.g

```
C:\zip_unpack\directory> bin\mysqld_install_db.exe
```

Then you can start server like this

```
C:\zip_unpack\directory> bin\mysqld.exe --console
```

For very old distributions (10.3 and earlier), a prebuilt data directory is already provided.

If you like to customize the server instance (data directory, install as service etc), please refer to mysql_install_db.exe documentation

2.1.2.8 Compiling MariaDB From Source

Get, Build and Test Latest MariaDB the Lazy Way

Instructions for people who don't have time to read the whole manual.

MariaDB Source Code

How to get the source code for MariaDB from GitHub.

Build Environment Setup for Linux

Requirements and build environment setup for Linux.

Generic Build Instructions

Instructions to help compile MariaDB from source.

Compiling MariaDB with Extra Modules/Options

Articles on compiling MariaDB with extra modules and options

Creating the MariaDB Source Tarball

How to create a source tar.gz file

Creating the MariaDB Binary Tarball

How to generate binary tar.gz files.

Build Environment Setup for Mac

Setting up the build environment for Mac

Building MariaDB from a Source RPM

How to build MariaDB from a source RPM (SRPM).

Building MariaDB on CentOS

CentOS build requirements and steps.

Building MariaDB on Fedora

Guide to building MariaDB from source code on Fedora Linux.

Building MariaDB on Debian

Steps to compiling MariaDB on Debian Linux.

Building MariaDB on FreeBSD

How to build MariaDB on FreeBSD.

Building MariaDB on Gentoo

Steps to build MariaDB on Gentoo

Building MariaDB on Solaris and OpenSolaris

Links and notes for building MariaDB on Solaris and OpenSolaris

Building MariaDB on Ubuntu

Requirements and steps for building MariaDB on Ubuntu.

Building MariaDB on Windows

Instructions for building MariaDB on Windows.

Installing MariaDB Server on macOS Using Homebrew

Installing MariaDB on macOS via the Homebrew package manager, the "missing ...

Compiling with the InnoDB Plugin from Oracle

Compiling MariaDB with the InnoDB plugin from Oracle.

Creating a Debian Repository
Instructions for creating your own Debian repository

Building MariaDB From Source Using musl-based GNU/Linux
Instructions on compiling MariaDB on musl-based operating systems (Alpine)

Compiling MariaDB for Debugging
Passing -DCMAKE_BUILD_TYPE=Debug to cmake to compile with debug information.

Cross-compiling MariaDB
To cross-compile with cmake you will need a toolchain file

MariaDB Source Configuration Options
Options for configuring a MariaDB source distribution.

Building RPM Packages From Source
Building MariaDB RPM packages with CMake and CPackRPM.

Compile and Using MariaDB with Sanitizers (ASAN, UBSAN, TSAN, MSAN)
How to compile and use MariaDB with AddressSanitizer (ASAN).

There are 16 related questions.

2.1.2.8.1 Get, Build and Test Latest MariaDB the Lazy Way

The intention of this documentation is show all the steps of getting, building and testing the latest MariaDB server (10.5 at time of writing) from GitHub. Each stage links to the full documentation for that step if you need to find out more.

Install all tools needed to build MariaDB

OpenSuse

```
sudo zypper install git gcc gcc-c++ make bison ncurses ncurses-devel zlib-devel libevent-devel
cmake openssl
```

Debian

```
apt install -y build-essential bison
apt build-dep mariadb-server
```

Set Up git

Fetch and checkout the MariaDB source to a subdirectory of the current directory

```
git clone https://github.com/MariaDB/server.git mariadb
cd mariadb
git checkout 10.5
```

Build It

The following command builds a server the same way that is used for building releases. Use `cmake . -DCMAKE_BUILD_TYPE=Debug` to build for debugging.

```
cmake . -DBUILD_CONFIG=mysql_release && make -j8
```

Check the Server (If You Want To)

```
cd mysql-test
./mtr --parallel=8 --force
```

Install the Default Databases

```
./scripts/mariadb-install-db --srcdir=.
```

(Older MariaDB version use mysql_install_db)

Install the Server (If needed)

You can also run and test mariadb directly from the build directory, in which case you can skip the rest of the steps below.

```
make install
```

Start the Server

Start the server in it's own terminal window for testing. Note that the directory depends on your system!

```
/usr/sbin/mysqld
```

2.1.2.8.2 MariaDB Source Code

Checking out the Source with Git

The MariaDB source is hosted on GitHub: https://github.com/MariaDB/server

If you want a source tarball for a specific released MariaDB version, you can find it at http://downloads.mariadb.org.

At any given time, developers will be working on their own branches locally or on GitHub, with the main MariaDB branches receiving pushes less often.

The main MariaDB branches are:

- 11.3
- 11.2
- 11.1
- 11.0
- 10.11
- 10.10
- 10.6
- 10.5
- 10.4

Source repositories for the MariaDB Connectors are:

- MariaDB Connector/C
- MariaDB Connector/J
- MariaDB Connector/Node.js
- MariaDB Connector/ODBC
- MariaDB Connector/Python

See the Using git page for instructions on how to use git to check out the source code and switch between the various branches.

2.1.2.8.3 Build Environment Setup for Linux

Contents

Required Tools

The following is a list of tools that are required for building MariaDB on Linux and Mac OS X. Most, if not all, of these will exist as packages in your distribution's package repositories, so check there first. See Building MariaDB on Ubuntu, Building MariaDB on CentOS, and Building MariaDB on Gentoo pages for specific requirements for those platforms.

- git
- gzip
- GNU tar
- gcc/g++ 4.8.5 or later, recommend above 9 or clang/clang++
- GNU make 3.75 or later or Ninja
- bison (3.0)
- libncurses
- zlib-dev
- libevent-dev
- cmake above 2.8.7 though preferably above 3.3
- gnutls or openssl
- jemalloc (optional)
- snappy (compression library, optional)
- valgrind (only needed if running mysql-test-run --valgrind)
- libcurl (only needed if you want to use the S3 storage engine)
- libxml2-devel
- boost
- libaio-devel
- systemd-devel
- pcre2-devel (optiona, will be automatically downloaded and installed if not on the system)
- ccache ; Will speed up builds if you are going to use the scripts in the BUILD directory.

You can install these programs individually through your package manager.

In addition, some package managers support the use a build dependency command. When using this command, the package manager retrieves a list of build dependencies and install them for you, making it much easier to get started on the compile. The actual option varies, depending on the distribution you use.

On Ubuntu and Debian you can use the `build-dep` command.

```
# apt build-dep mariadb-server
```

Fedora uses the `builddep` command with DNF.

```
# dnf builddep mariadb-server
```

If building on Centos 7, use the building MariaDB on Centos instructions.

With openSUSE and SUSE, you can use the source-install command.

```
# zypper source-install -d mariadb
```

Each of these commands works off of the release of MariaDB provided in the official software repositories of the given distribution. In some instances and especially in older versions of Linux, MariaDB may not be available in the official repositories. In these cases you can use the MariaDB repositories as an alternative.

Bear in mind, the release of MariaDB provided by your distribution may not be the same as the version you are trying to install. Additionally, the package managers don't always retrieve all of the packages you need to compile MariaDB. There may be some missed or unlisted in the process. When this is the case, CMake fails during checks with an error message telling you what's missing.

Note: On Debian-based distributions, you may receive a *"You must put some 'source' URIs in your sources.list"* error. To avoid this, ensure that /etc/apt/sources.list contains the source repositories.

For example, for Debian buster:

```
deb http://ftp.debian.org/debian buster main contrib
deb http://security.debian.org buster/updates main contrib
deb-src http://ftp.debian.org/debian buster main contrib
deb-src  http://security.debian.org buster/updates main contrib
```

Refer to the documentation for your Linux distribution for how to do this on your system.

After editing the sources.list, do:

```
sudo apt update
```

...and then the above mentioned `build-dep` command.

Note: On openSUSE the source package repository may be disabled. The following command will enable it:

```
sudo zypper mr -er repo-source
```

After enabling it, you will be able to run the zypper command to install the build dependencies.

You should now have your build environment set up and can proceed to Getting the MariaDB Source Code and then using the Generic Build Instructions to build MariadB (or following the steps for your Linux distribution or Creating a MariaDB Binary Tarball).

2.1.2.8.4 Generic Build Instructions

Contents

The instructions on this page will help you compile MariaDB from source. Links to more complete instructions for specific platforms can be found on the source page.

First, get a copy of the MariaDB source.

Next, prepare your system to be able to compile the source.

If you don't want to run MariaDB as yourself, then you should create a `mysql` user. The example below uses this user.

Using cmake

MariaDB 5.5 and above is compiled using *cmake*.

It is recommended to create a build directory **beside** your source directory

```
mkdir build-mariadb
cd build-mariadb
```

NOTE If you have built MariaDB in the past and have recently updated the repository, you should perform a complete cleanup of old artifacts (such as cmake configured files). In the base repository run:

```
git clean -xffd && git submodule foreach --recursive git clean -xffd
```

You can configure your build simply by running *cmake* without any special options, like

```
cmake ../server
```

where `server` is where you installed MariaDB. If you are building in the source directory, just omit `../server`.

If you want it to be configured exactly as a normal MariaDB server release is built, use

```
cmake ../server -DBUILD_CONFIG=mysql_release
```

This will configure the build to generate binary tarballs similar to release tarballs from downloads.mariadb.org. Unfortunately this doesn't work on old platforms, like OpenSuse Leap 15.0, because MariaDB binary tarballs are built to minimize external dependencies, and that needs static libraries that might not be provided by the platform by default, and would need to be installed manually.

To do a build suitable for debugging use:

```
cmake ../server -DCMAKE_BUILD_TYPE=Debug
```

By default, MariaDB is compiled with the `-Werror` flag, which causes compiling to abort if there is a compiler warning. You can disable that by configuring with `-DMYSQL_MAINTAINER_MODE=OFF`.

```
cmake ../server -DCMAKE_BUILD_TYPE=Debug -DMYSQL_MAINTAINER_MODE=OFF.
```

All *cmake* configuration options for MariaDB can be displayed with:

```
cmake ../server -LH
```

To build and install MariaDB after running *cmake* use

```
cmake --build .
sudo cmake --install .
```

If the commands above fail, you can enable more compilation information by doing:

```
cmake --build . --verbose
```

If you want to generate a binary tarball, run

```
cpack
```

Using BUILD Scripts

There are also `BUILD` scripts for the most common systems for those that doesn't want to dig into cmake options. These are optimized for in source builds.

The scripts are of type 'compile-#cpu#-how_to_build'. Some common scripts-are

Script	Description
compile-pentium64	Compile an optimized binary optimized for 64 bit pentium (works also for amd64)
compile-pentium-debug	Compile a debug binary optimized for 64 bit pentium
compile-pentium-valgrind-max	Compile a debug binary that can be used with valgrind to find wrong memory accesses and memory leaks. Should be used if one want's to run the `mysql-test-run` test suite with the `--valgrind` option

Some common suffixes used for the scripts:

Suffix	Description
`32`	Compile for 32 bit cpu's
`64`	Compile for 64 bit cpu's
`-max`	Enable (almost) all features and plugins that MariaDB supports
`-gprof`	binary is compiled with profiling (gcc --pg)
`-gcov`	binary is compiled with code coverage (gcc -fprofile-arcs -ftest-coverage)

`-valgrind`	The binary is compiled for debugging and optimized to be used with valgrind.
`-debug`	The binary is compiled with all symbols (gcc -g) and the `DBUG` log system is enabled.

All `BUILD` scripts support the following options:

Suffix	Description
-h, --help	Show this help message.
-n, --just-print	Don't actually run any commands; just print them.
-c, --just-configure	Stop after running configure. Combined with --just-print shows configure options.
--extra-configs=xxx	Add this to configure options
--extra-flags=xxx	Add this C and CXX flags
--extra-cflags=xxx	Add this to C flags
--extra-cxxflags=xxx	Add this to CXX flags
--verbose	Print out full compile lines
--with-debug=full	Build with full debug(no optimizations, keep call stack).

A typical compilation used by a developer would be:

```
shell> ./BUILD/compile-pentium64-debug
```

This configures the source for debugging and runs make. The server binary will be `sql/mariadbd` or `sql/mysqld`.

Starting MariaDB for the First Time

After installing MariaDB (using `sudo make install`), but prior to starting MariaDB for the first time, one should:

1. ensure the directory where you installed MariaDB is owned by the mysql user (if the user doesn't exist, you'll need to create it)
2. run the mariadb-install-db script to generate the needed system tables

Here is an example:

```
# The following assumes that the 'mysql' user exists and that we installed MariaDB
# in /usr/local/mysql
chown -R mysql /usr/local/mysql/
cd /usr/local/mysql/
scripts/mariadb-install-db --user=mysql
/usr/local/mysql/bin/mysqld_safe --user=mysql &
```

Testing MariaDB

If you want to test your compiled MariaDB, you can do either of:

Run unit tests:

```
cmake --build . --target test
```

Or run mtr tests:

```
mysql-test/mysql-test-run --force
```

Each of the above are run from the build directory. There is no need to ' `make install` / `cmake --install .` ' MariaDB prior to running them.

NOTE: If you are doing more extensive testing or debugging of MariaDB (like with real application data and workloads) you may want to start and run MariaDB directly from the source directory instead of installing it with ' `sudo make install` '. If so, see Running MariaDB from the Source Directory.

Increasing Version Number or Tagging a Version

If you have made code changes and want to increase the version number or tag our version with a specific tag you can do this by editing the `VERSION` file. Tags are shown when running the ' `mysqld --version` ' command.

Non-ascii Symbols

MariaDB builds with `readline` ; using an alternative such as `Editline` may result in problems with non-ascii symbols.

Post-install Tasks

- Installing system tables (mariadb-install-db)
- Starting and Stopping MariaDB Automatically

2.1.2.8.5 Compiling MariaDB with Extra Modules/Options

Compiling MariaDB with Vanilla XtraDB

Sometimes, one needs to have MariaDB compiled with Vanilla XtraDB. This pag...

Specifying Which Plugins to Build

Specifying which plugins to build.

Using MariaDB with TCMalloc or jemalloc

TCMalloc is a malloc replacement library optimized for multi-threaded usage.

2.1.2.8.5.1 Using MariaDB with TCMalloc or jemalloc

Using tcmalloc

TCMalloc is a malloc replacement library optimized for multi-threaded usage. It also features a built-in heap debugger and profiler. You can build MariaDB with it or LD_PRELOAD it.

To build MariaDB 5.5 with `TCMalloc` , you need to use the following command

```
cmake -DCMAKE_EXE_LINKER_FLAGS='-ltcmalloc'  -DWITH_SAFEMALLOC=OFF
```

Many other malloc replacement libraries (as well as heap debuggers and profilers) can be used with MariaDB in a similar fashion.

You can also start a standard MariaDB server with `TCmalloc` with:

```
/usr/sbin/mysqld_safe --malloc-lib=tcmalloc
```

or add it to `my.cnf` . Alternatively just set LD_PRELOAD to point to the to be used malloc library:

```
export LD_PRELOAD=/path/to/library
```

Using jemalloc

The above procedure also works with jemalloc. Just replace tcmalloc with jemalloc in the above commands.

Finding memory leaks with jemalloc

jemalloc can provide a report of memory leaks at program exit:

```
MALLOC_CONF=prof_leak:true,lg_prof_sample:0,prof_final:true \
LD_PRELOAD=${JEMALLOC_PATH}/lib/libjemalloc.so.2  path-to-mariadbd
```

This will produce something like:

```
<jemalloc>: Leak summary: 267184 bytes, 473 objects, 20 contexts
<jemalloc>: Run jeprof on "jeprof.19678.0.f.heap" for leak detail
```

You can learn more about the memory leaks with jeprof, that is included with jemalloc:

```
jeprof --show_bytes path-to-mariadbd jeprof.19678.0.f.heap
```

You can also generate a PDF call graph of the leak:

```
jeprof --show_bytes --pdf path-to-mariadbd  jeprof.19678.0.f.heap > /tmp/mariadbd.pdf
```

2.1.2.8.5.2 Specifying Which Plugins to Build

By default all plugins are enabled and built as dynamic `.so` (or `.dll`) modules. If a plugin does not support dynamic builds, it is not built at all.

Use `PLUGIN_xxx` cmake variables. They can be set on the command line with `-DPLUGIN_xxx=value` or in the cmake gui. Supported values are

Value	Effect
NO	the plugin will be not compiled at all
STATIC	the plugin will be compiled statically, if supported. Otherwise it will be not compiled.
DYNAMIC	the plugin will be compiled dynamically, if supported. Otherwise it will be not compiled. This is the default behavior.
AUTO	the plugin will be compiled statically, if supported. Otherwise it will be compiled dynamically.
YES	same as **AUTO**, but if plugin prerequisites (for example, specific libraries) are missing, it will not be skipped, it will abort cmake with an error.

Note that unlike autotools, cmake tries to configure and build incrementally. You can modify one configuration option and cmake will only rebuild the part of the tree affected by it. For example, when you do `cmake -DWITH_EMBEDDED_SERVER=1` in the already-built tree, it will make libmysqld to be built, but no other configuration options will be changed or reset to their default values.

In particular this means that if you have run, for example `cmake -DPLUGIN_OQGRAPH=NO` and later you want to restore the default behavior (with OQGraph being built) in the same build tree, you would need to run `cmake -DPLUGIN_OQGRAPH=DYNAMIC`

Alternatively, you might simply delete the `CMakeCache.txt` file — this is the file where cmake stores current build configuration — and rebuild everything from scratch.

2.1.2.8.6 Creating the MariaDB Source Tarball

How to create a source tar.gz file.

First Setup your build environment.

Then use automake/configure/make to generate the tar file:

```
BUILD/autorun.sh
./configure --with-plugin-xtradb
make dist
```

This creates a source file with a name like `mysql-5.3.2-MariaDB-beta.tar.gz`

See also the generic build instructions.

2.1.2.8.7 Creating the MariaDB Binary Tarball

How to generate binary tar.gz files.

- Setup your build environment.
- Build binaries with your preferred options/plugins.

If the binaries are already built, you can generate a binary tarball with

```
make package
```

Prior to MariaDB 5.5, the following steps were required:

- Use `make_binary_distribution` to generate a binary tar file:

```
cd mariadb-source
./scripts/make_binary_distribution
```

This creates a source file with a name like `mariadb-5.3.2-MariaDB-beta-linux-x86_64.tar.gz` in your current directory.

The other option is to use the bakery scripts. In this case you don't have to compile MariaDB source first.

```
cd $PACKAGING_WORK
bakery/preheat.sh
cd bakery_{number}
bakery/tarbake51.sh last:1 $MARIA_WORK
bakery/autobake51-bintar.sh mariadb-{version_num}-maria-beta-ourdelta{number}.tar.gz
```

2.1.2.8.8 Build Environment Setup for Mac

XCode

- Install Xcode from Apple (free registration required): https://developer.apple.com/xcode/ or from your Mac OS X installation disk (macports needs XCode >= 3.1, so if you do not have that version or greater you will need to download the latest version, which is 900+ MB)

You can install the necessary dependencies using either MacPorts or Homebrew.

Using MacPorts

- Download and install the MacPorts dmg image from http://www.macports.org
- After installing, update it from the terminal: `sudo port -v selfupdate`

```
sudo port install cmake jemalloc judy openssl boost gnutls
```

Using Homebrew

- Download and install Homebrew from https:*brew.sh*

```
brew install cmake jemalloc traildb/judy/judy openssl boost gnutls
```

Your Mac should now have everything it needs to get, compile, and otherwise work with the MariaDB source code. The next step is to actually get a copy of the code. For help with this see the Getting the MariaDB Source Code page.

When building with Mac, you'll need `-DOPENSSL_ROOT_DIR=/usr/local/openssl` passed as a `cmake` argument to build against openssl correctly.

2.1.2.1.10 Building MariaDB From a Source RPM

2.1.2.8.10 Building MariaDB on CentOS

Contents

In the event that you are using the Linux-based operating system CentOS or any of its derivatives, you can optionally compile MariaDB from source code. This is useful in cases where you want use a more advanced release than the one that's available in the official repositories, or when you want to enable certain feature that are not otherwise accessible.

Installing Build Dependencies for MariaDB 5.5

Before you start building MariaDB, you first need to install the build dependencies required to run the compile. CentOS provides a tool for installing build dependencies. The `yum-builddep` utility reads a package and generates a list of the packages required to build from source, then calls YUM to install them for you. In the event that this utility is not available on your system, you can install it through the `yum-utils` package. Once you have it, install the MariaDB build dependencies.

```
yum install yum-utils
yum-builddep mariadb-server
```

Running the above command installs many of the build dependencies, but it **doesn't install all of them**. It will only install dependencies for MariaDB 5.5, which is not enough if you want to compile a newer MariaDB version!

Installing Build Dependencies for newer MariaDB versions

The following commands installs all packages needed to get and compile MariaDB 10.11:

```
yum install git
yum install gcc
yum install gcc-c++
yum install tar make cmake
yum install bison
yum install ncurses-devel
yum install openssl openssl-devel
yum install snappy snappy-devel
yum install valgrind
yum install libcurl-devel
yum install gzip
yum install zlib-devel
yum install lz4-devel
yum install lzo-devel
yum install bzip2-devel
yum install libxml2-devel
yum install libevent-devel
yum install libaio-devel
yum install boost
yum install pcre2-devel
yum install systemd-devel
yum install rpm-build
yum install libaio-devel
yum install zstd
yum install pam-devel
yum install checkpolicy
yum install policycoreutils-python
yum install galera.x86_64
```

You can replace `openssl` with `gnutls` above, depending on the TLS implementation you want to use.

If you plan to use the BUILD scripts to make it easier to build different configurations of MariaDB, then you should also install ccache to speed up your compilations:

```
yum install ccache
```

If you want to test the MariaDB installation, with the include mysql-test-run (mtr) system, you also need to install and configure cpan:

```
yum install cpan
# Complete Perl installing with the next command. Use the default answer to all questions
cpan App::cpanminus
```

For more information on dependencies, see Linux Build Environment.

Building MariaDB

Once you have the base dependencies installed, you can retrieve the source code and start building MariaDB. The source code is available on GitHub. Use the `--branch` option to specify the particular version of MariaDB you want to build.

```
$ git clone --branch 10.11 https://github.com/MariaDB/server.git
```

With the source repository cloned onto your system, you can start building MariaDB. Run CMake to ready MariaDB for the build,

```
$ cmake -DRPM=centos7 server/
```

Once CMake readies the relevant Makefile for your system, use Make to build MariaDB.

```
$ umask 0022
$ make package
```

This generates an RPM file, which you can then install on your system or copy over to install on other CentOS hosts. The umask is needed because of a bug in cmake / cmake scripts.

Alternative, use one of the build scripts in the `BUILD` directory that allows you to compile different versions of MariaDB (debug, optimized, profiling etc).

A good option for developers is:

```
./BUILD/compile-pentium64-debug
```

Creating MariaDB-compat package

MariaDB-compat package contains libraries from older MariaDB releases. They cannot be built from the current source tree, so cpack creates them by repackaging old MariaDB-shared packages. If you want to have -compat package created, you need to download MariaDB-shared-5.3 and MariaDB-shared-10.1 rpm packages for your architecture (any minor version will do) and put them *one level above* the source tree you're building. CMake will pick them up and create a MariaDB-compat package. CMake reports it as

```
$ ls ../*.rpm
../MariaDB-shared-10.1.17-centos7-x86_64.rpm
../MariaDB-shared-5.3.12-122.el5.x86_64.rpm
$ cmake -DRPM=centos7 .
...
Using ../MariaDB-shared-5.3.12-122.el5.x86_64.rpm to build MariaDB-compat
Using ../MariaDB-shared-10.1.17-centos7-x86_64.rpm to build MariaDB-compat
```

Additional Dependencies

In the event that you miss a package while installing build dependencies, CMake may continue to fail after you install the necessary packages. If this happens to you, delete the CMake cache then run the above the command again:

```
$ rm CMakeCache.txt
```

When CMake runs through the tests again, it should now find the packages it needs, instead of the cache telling it they're unavailable.

More about CMake and CPackRPM

See also building RPM packages from source

2.1.2.8.11 Building MariaDB on Fedora

In the event that you are using the Linux-based operating system Fedora or any of its derivatives and would like to compile MariaDB from source code, you can do so using the MariaDB build in the official repositories.

Installing Build Dependencies

MariaDB requires a number of packages to compile from source. Fortunately, you can use the package in the Fedora repository to retrieve most of the relevant build dependencies through DNF.

```
# dnf builddep mariadb-server
```

Running DNF in this way pulls the build dependencies for the release of MariaDB compiled by your version of Fedora. These may not be all the dependencies you need to build the particular version of MariaDB you want to use, but it will retrieve most of them.

You'll also need to install Git to retrieve the source repository:

```
# dnf install git
```

Building MariaDB

Once you have the base dependencies installed, you can retrieve the source code and start building MariaDB. The source code is available on GitHub. Use the `--branch` option to specify the particular version of MariaDB you want to build.

```
$ git clone --branch 10.3 https://github.com/MariaDB/server.git mariadb-server
```

With the source repository cloned onto your system, you can start building MariaDB. Change into the new `mariadb-server/` directory and run CMake to prepare the build.

```
$ mkdir mariadb-build
$ cd mariadb-build
$ cmake -DRPM=fedora ../mariadb-server
```

Once CMake readies the relevant Makefile for your system, use Make to build MariaDB.

```
$ make package
```

2.1.2.8.12 Building MariaDB on Debian

Contents

In the event that you are using the Linux-based operating system Debian or any of its direct derivatives and would like to compile MariaDB from source code, you can do so using the MariaDB source repository for the release that interests you. For Ubuntu and its derivatives, see Building on Ubuntu.

Before you begin, install the `software-properties-common` and `devscripts` packages:

```
$ sudo apt-get install -y software-properties-common \
      devscripts
```

Installing Build Dependencies

MariaDB requires a number of packages to compile from source. Fortunately, you can use the MariaDB repositories to retrieve the necessary code for the version you want. Use the Repository Configuration tool to determine how to set up the MariaDB repository for your release of Debian, the version of MariaDB that you want to install, and the mirror that you

want to use.

First add the authentication key for the repository, then add the repository.

```
$ sudo apt-key adv --recv-keys \
      --keyserver hkp://keyserver.ubuntu.com:80 \
      0xF1656F24C74CD1D8
$ sudo add-apt-repository 'deb [arch=amd64]
http://nyc2.mirrors.digitalocean.com/mariadb/repo/10.3/debian stretch main'
```

The second command added text to the `/etc/apt/sources.list` file. One of these lines is the repository containing binary packages for MariaDB, the other contains the source packages. The line for the source packages is commented out by default. This can be scripted:

```
sed -e '/^# deb-src.*mariadb/s/^# //' -i /etc/apt/sources.list
```

Alternately, open the file using your preferred text editor and uncomment the source repository.

```
$ sudo vim /etc/apt/sources.list

...
deb [arch=amd64] http://nyc2.mirrors.digitalocean.com/mariadb/repo/10.3/debian stretch main
deb-src [arch=amd64] http://nyc2.mirrors.digitalocean.com/mariadb/repo/10.3/debian stretch main
```

Once the repository is set up, you can use `apt-get` to retrieve the build dependencies. MariaDB packages supplied by Ubuntu and packages supplied by the MariaDB repository have the same base name of `mariadb-server`. You need to specify the specific version you want to retrieve.

```
$ sudo apt-get update
$ sudo apt-get build-dep -y mariadb-server-10.3
```

Building MariaDB

Once you have the base dependencies installed, you can retrieve the source code and start building MariaDB. The source code is available on GitHub. Use the `--branch` option to specify the particular version of MariaDB you want to build.

```
$ git clone --branch 10.3 https://github.com/MariaDB/server.git
```

The source code includes scripts to install the remaining build dependencies. For Ubuntu, they're located in the `debian/` directory. Navigate into the repository and run the `autobake-deb.sh` script. Then use

```
$ export DEB_BUILD_OPTIONS=parallel=$(nproc)
$ cd server/
$ ./debian/autobake-deb.sh
```

After Building

After building the packages, it is a good idea to put them in a repository. See the Creating a Debian Repository page for instructions.

2.1.2.8.13 Building MariaDB on FreeBSD

Contents

It is relatively straightforward to build MariaDB from source on FreeBSD. When working with an individual host, you can use Ports to compile particular or multiple versions of MariaDB. When working with multiple hosts, you can use Poudriere to build MariaDB once, then serve it as a package to multiple FreeBSD hosts.

Using Ports

The FreeBSD Ports Collection provides a series of Makefiles that you can use to retrieve source code, configure builds, install dependencies and compile software. This allows you to use more advanced releases than what is normally available through the package managers as well as enable any additional features that interest you.

In the event that you have not used Ports before on your system, you need to first fetch and extract the Ports tree. This downloads the Ports tree from FreeBSD and extracts it onto your system, placing the various Makefiles, patches and so on in the `/usr/ports/` directory.

```
# portsnap fetch extract
```

In the event that you have used Ports before on this system, run Portsnap again to download and install any updates to the Ports tree.

```
# portsnap fetch update
```

This ensures that you are using the most up to date release of the Ports tree that is available on your system.

Building MariaDB from Ports

Once Portsnap has installed or updated your Ports tree, you can change into the relevant directory and install MariaDB. Ports for MariaDB are located in the `/usr/ports/databases/` directory.

```
$ ls /usr/ports/databases | grep mariadb

mariadb-connector-c
mariadb-connector-odbc
mariadb100-client
mariadb100-server
mariadb101-client
mariadb101-server
mariadb102-client
mariadb102-server
mariadb103-client
mariadb103-server
mariadb55-client
mariadb55-server
```

Note that FreeBSD treats the Server and Client as separate packages. The Client is a dependency of the Server, so you only need to build the Server to get both. It also provides a number of different versions. You can search the available ports from Fresh Ports. Decide what version of MariaDB you want to install, the change into the relevant directory. Once in the directory, run Make to build MariaDB.

```
# cd /usr/ports/databases/mariadb103-server
# make
```

In addition to downloading and building MariaDB, Ports also downloads and build any libraries on which MariaDB depends. Each port it builds will take you to a GUI window where you can select various build options. In the case of MariaDB, this includes various storage engines and specific features you need in your build.

Once you finish building the ports, install MariaDB on your system and clean the directory to free up disk space.

```
# make install clean
```

This installs FreeBSD on your server. You can now enable, configure and start the service as you normally would after installing MariaDB from a package.

Using Poudriere

Poudriere is a utility for building FreeBSD packages. It allows you to build MariaDB from a FreeBSD Jail, then serve it as a binary package to other FreeBSD hosts. You may find this is particularly useful when building to deploy multiple MariaDB servers on FreeBSD, such as with Galera Cluster or similar deployments.

Building MariaDB

Once you've configured your host to use Jails and Poudriere, initialize a jail to use in building packages and a jail for managing ports.

```
# poudriere jail -c -j package-builder -v 11.2-RELEASE
# poudriere ports -c -p local-ports
```

This creates two jails, `package-builder` and `local-ports`, which you can then use to build MariaDB. Create a text file to define the packages you want to build. Poudriere will build these packages as well as their dependencies. MariaDB is located at `databases/mariadb103-server`. Adjust the path to match the version you want to install.

```
$ vi maraidb-package-builder.txt

databases/mariadb103-server
```

Use the `options` command to initialize the build options for the packages you want Poudriere to compile.

```
# poudriere options -j package-builder -p local-ports -z mariadb-builder -f mariadb-package-build
```

Lastly, use the `bulk` command to compile the packages.

```
# poudriere bulk -j package-builder -p local-ports -z mariadb-builder -f mariadb-package-builder.
```

Using Poudriere Repositories

In order to use Poudriere, you need to set up and configure a web server, such as Nginx or Apache to serve the directory that Poudriere built. For instance, in the case of the above example, you would map to the `package-builder` jail: `/usr/local/poudriere/data/packages/package-builder/`. You may find it useful to map this directory to a sub-domain, for instance `https`*`pkg.example.com`* or something similar.

Lastly, you need to configure the FreeBSD hosts to use the Poudriere repository you just created. On each host, disable the FreeBSD official repositories and enable your Poudriere repository as an alternative.

```
# vi /usr/local/etc/pkg/repos/FreeBSD.conf

FreeBSD: {
      enabled: no
}
```

Then add the URL for your Poudriere repository to configuration file:

```
# vi /usr/local/etc/pkg/repos/mariadb.conf

custom: {
      url: "https://pkg.example.com",
      enabled: yes
}
```

You can then install MariaDB from Poudriere using the package manager.

```
# pkg install mariadb103-server
```

2.1.2.8.14 Building MariaDB on Gentoo

MariaDB is in Gentoo, so the steps to build it are:

1. Synchronize your tree with
   ```
   emerge --sync
   ```
2. Build MariaDB using
   ```
   emerge mariadb
   ```

2.1.2.8.15 Building MariaDB on Solaris and OpenSolaris

The following two articles should help you get your Solaris machine prepared to build MariaDB (just ignore the parts about installing buildbot):

- Buildbot Setup for Solaris Sparc
- Buildbot Setup for Solaris x86

Notes

- The BUILD dir contains various scripts for compiling MariaDB on Solaris. The `BUILD/compile-solaris-amd64` and `BUILD/compile-solaris-amd64-debug` are probably the most useful.
- The scripts do not play nice with non-bash shells such as the Korn Shell (ksh). So if your /bin/sh is pointing at ksh or ksh93, you'll want to change it so that it points at bash.

2.1.2.8.16 Building MariaDB on Ubuntu

Contents

In the event that you are using the Linux-based operating system Ubuntu or any of its derivatives and would like to compile MariaDB from source code, you can do so using the MariaDB source repository for the release that interests you.

Before you begin, install the `software-properties-common`, `devscripts` and `equivs` packages.

```
$ sudo apt-get install software-properties-common \
      devscripts \
      equivs
```

Installing Build Dependencies

MariaDB requires a number of packages to compile from source. Fortunately, you can use the MariaDB repositories to retrieve the necessary code for the version you want. Use the Repository Configuration tool to determine how to set up the MariaDB repository for your release of Ubuntu, the version of MariaDB that you want to install, and the mirror that you want to use.

First add the authentication key for the repository, then add the repository.

```
$ sudo apt-key adv --recv-keys \
      --keyserver hkp://keyserver.ubuntu.com:80 \
      0xF1656F24C74CD1D8
$ sudo add-apt-repository --update --yes --enable-source \
       'deb [arch=amd64] http://nyc2.mirrors.digitalocean.com/mariadb/repo/10.3/ubuntu
'$(lsb_release -sc)' main'
```

Once the repository is set up, you can use `apt-get` to retrieve the build dependencies. MariaDB packages supplied by Ubuntu and packages supplied by the MariaDB repository have the same base name of `mariadb-server`. You need to specify the specific version you want to retrieve.

```
$ sudo apt-get build-dep mariadb-10.3
```

Building MariaDB

Once you have the base dependencies installed, you can retrieve the source code and start building MariaDB. The source code is available on GitHub. Use the `--branch` option to specify the particular version of MariaDB you want to build.

```
$ git clone --branch 10.3 https://github.com/MariaDB/server.git
```

The source code includes scripts to install the remaining build dependencies. For Ubuntu, they're located in the `debian/` directory. Navigate into the repository and run the `autobake-deb.sh` script. Then use

```
$ cd server/
$ ./debian/autobake-deb.sh
```

After Building

After building the packages, it is a good idea to put them in a repository. See the Creating a Debian Repository page for instructions.

2.1.2.8.17 Building MariaDB on Windows

Contents

Build Requirements

To build MariaDB you need the following:

- Visual C++: We currently support Visual Studio 2019 and 2022. Generally we try to support the two most recent VS versions, but build ourselves using the last one. Community editions will work fine; we only use them in our builds. While installing Visual Studio, make sure to add "Desktop Development with C++".
- CMake: We recommend the latest release. Older releases might not support your version of Visual Studio. Visual Studio 2019 requires cmake 3.14 at least.
- Git: Required to build newer versions from the source tree.
  - NOTE: run

```
git config --global core.autocrlf input
```

after the installation, otherwise some mtr tests will fail

In the "Adjusting your PATH" dialog, choose "Use Git from Windows command prompt", otherwise wrong (mingw64) git and perl will be in your PATH

- Bison from GnuWin32: Bison creates parts of the SQL parser. Choose "Complete package except sources" when downloading.
  - NOTE: **Do not** install this into your default path with spaces (e.g. under `C:\Program Files\GnuWin32`); the build will break due to this bison bug. Instead, install into `C:\GnuWin32`.
  - Add `C:\GnuWin32\bin` to your system `PATH` after installation.
- Strawberry perl: Used to run the test suite. ActiveState Perl is another Win32 Perl distribution and should work as well (but it is not as well tested). NOTE: Cygwin or mingw Perl versions **will not work** for testing. Use Windows native Perl, please.
- Optional: If you intend to build the MSI packages, install Windows Installer XML. If you build MSI with 10.4, also modify your Visual Studio installation, add "Redistributable MSMs" (see MDEV-22555)
- Gnu Diff, needed if you run mysql-test-run.pl tests.

Verify that bison.exe, or git.exe, cmake.exe and perl.exe can be found in the PATH environment variable with "`where bison`", "`where git`", "`where perl`" etc. from the command line prompt.

Building Windows Binaries

The above instructions assume MariaDB 10.2 or higher.

Branch the MariaDB repository, or unpack the source archive. On the command prompt, switch to your source directory, then execute:

```
mkdir bld
cd bld
cmake ..
cmake --build . --config RelWithDebInfo
```

The above example builds a release configured for 64 bit systems in a subdirectory named `bld`. "`cmake ...`" is the configuration step, "`cmake --build . --config Relwithdebinfo`" is the build step.

Build Variations

Debug Builds

Building Debug version is done with:

```
cmake --build . --config Debug
```

- 32bit and 64 bit Builds

Build 64 bit binary

Visual Studio 2019-2022 cmake generator will use host architecture by default, that is, with the steps above, cmake will build x64 binaries on x64 machine.

Build 32 bit binary

pass -A Win32 parameter for CMake, like this

```
cmake .. -A Win32
```

Historical note: With Visual Studio 2017 and earlier, one had to pass the name of 32bit generator ,e.g cmake .. -G "Visual Studio 15 2017"

For a complete list of available generators, call "cmake" without any parameters.

IDE Builds

Instead of calling "`cmake --build`" as above, open `MySQL.sln`. When Visual Studio starts, choose Build/Compile.

Building the ZIP Package

```
cmake --build . --config relwithdebinfo --target package
```

This is how it is "done by the book", standard cmake target.

MariaDB however uses non-standard target "win_package" for the packaging for its releases, it generates 2 ZIPs, a slim one with executables, and another one with debuginfo (.PDB files). The debuginfo is important to be able to debug released binaries, and to analyze crashes.

```
cmake --build . --config relwithdebinfo --target win_package
```

Building the MSI Package

```
cmake --build . --config relwithdebinfo
cmake --build . --config relwithdebinfo --target MSI
```

Including HeidiSQL in the MSI Installer

Starting with MariaDB 5.2.7, it is possible to build an installer which includes 3rd party products, as described in MWL#200. Currently only HeidiSQL support is implemented; it is also included in the official builds. Use the CMake parameter `-DWITH_THIRD_PARTY=HeidiSQL` to include it in the installer.

Code Signing

MariaDB builds optionally support authenticode code signing with an optional parameter `SIGNCODE`. Use `cmake -DSIGNCODE=1` during the configuration step to sign the binaries in the `ZIP` and `MSI` packages.

Important: for `SIGNCODE=1` to work, the user that runs the build needs to install a valid authenticode digital certificate into their certificate store, otherwise the packaging step will fail.

Building Packages for MariaDB Releases

The full script to create the release in an out-of-source build with Visual Studio with signed binaries might look like:

```
mkdir bld
cd bld
cmake .. -DSIGNCODE=1 -DWITH_THIRD_PARTY=HeidiSQL
cmake --build . --config relwithdebinfo --target win_package
cmake --build . --config relwithdebinfo  --target MSI
```

This command sequence will produce a ZIP package (e.g mariadb-5.2.6-win32.zip) and MSI package (e.g mariadb-5.2.6-win32.msi) in the `bld` directory.

Running Tests

- **Important: Do not use Cygwin** bash, MinGW bash, Git bash, WSL bash, or any other bash when running the test suite. You will then very likely use the wrong version of Perl too (a "Unix-flavoured" one on Windows), and spend a lot of time trying to figure out why this version of Perl does not work for the test suite. Use native perl, in cmd.exe , or powershell instead,
- Switch mysql-test subdirectory of the build directory

```
cd C:\server\bld\mysql-test
```

- Run the test suite

```
perl mysql-test-run.pl --suite=main --parallel=auto
```

Running a Test Under Debugger

Assuming VS is installed on the machine

```
perl mysql-test-run.pl  <test_name> --vsjitdebugger
```

If vsjitdebugger does not start, you can edit AeDebug registry key as mentioned in

https://docs.microsoft.com/en-us/visualstudio/debugger/debug-using-the-just-in-time-debugger?view=vs-2019#jit_errors

Alternatively:

```
perl mysql-test-run.pl  <test_name> --devenv
```

(devenv.exe needs to be in PATH)

or, if you prefer WinDBG

```
perl mysql-test-run.pl  <test_name> --windbg
```

2.1.2.8.18 Creating a Debian Repository

Below are instructions for creating your own Debian repository. The instructions are based on http://www.debian.org/doc/manuals/repository-howto/repository-howto.en.html

```
REPO_DIR={pick some location}
mkdir $REPO_DIR
mkdir $REPO_DIR/binary
mkdir $REPO_DIR/source
cp *.deb *.ddeb $REPO_DIR/binary
cd $REPO_DIR
dpkg-scanpackages binary  /dev/null | gzip -9c > binary/Packages.gz
dpkg-scansources  source  /dev/null | gzip -9c > source/Sources.gz
```

Using the Debian repository you just created

One needs to add a new file to the `/etc/apt/sources.list.d/` directory. For instance a new file called `mariadb.list`

```
# sergey's MariaDB repository
#
deb file:///home/psergey/testrepo binary/
deb-src file:///home/psergey/testrepo source/
```

after which one can run

```
apt-get update  # Let apt learn about the new repository
apt-get install mariadb-server
```

and collect bugs :-).

"apt-get install" will spray output of scripts and servers all over /var/log. It is also possible to set DEBIAN_SCRIPT_DEBUG=1 to get some (not all) of it to stdout.

Cleaning up after failed installation

Run

```
dpkg --get-selections | grep mariadb
dpkg --get-selections | grep mysql
```

to see what is installed, and then

```
 dpkg --purge <packages>
```

until the former produces empty output. Note: after some failures, /etc/mysql and /var/lib/mysql are not cleaned and still need to be removed manually.

2.1.2.8.19 Building MariaDB From Source Using musl-based GNU/Linux

Instructions on compiling MariaDB on musl-based operating systems (Alpine)

Contents

The instructions on this page will help you compile MariaDB from source. Links to more complete instructions for specific platforms can be found on the source page.

- First, get a copy of the MariaDB source.

- Next, prepare your system to be able to compile the source.

Using cmake

MariaDB 10.1 and above is compiled using *cmake*. You can configure your build simply by running *cmake* using special option, i.e.

```
cmake . -DWITHOUT_TOKUDB=1
```

To build and install MariaDB after running *cmake* use

```
make
sudo make install
```

Note that building with MariaDB this way will disable tokuDB, till tokuDB becomes fully supported on musl.

2.1.2.8.20 Compiling MariaDB for Debugging

Contents

Compiling MariaDB with full debug information includes all code symbols and also new code to do internal testing of structures and allow one to trace MariaDB execution. A full debug binary will be notably slower than a normal binary (30%).

Compiling MariaDB for Debugging Using the `CMAKE_BUILD_TYPE` Option

On Unix systems, you can build a debug build by executing `cmake` and by setting the `CMAKE_BUILD_TYPE` option to `Debug`. For example:

```
cmake -DCMAKE_BUILD_TYPE=Debug .
```

Compiling MariaDB for Debugging Using Build Scripts

The other option is to use the scripts in the BUILD directory that will compile MariaDB with most common debug options and plugins:

```
./BUILD/compile-pentium64-debug-max
```

or alternatively if you want to use the Valgrind memory checking tool with the MariaDB test system:

```
./BUILD/compile-pentium64-valgrind-max
```

There are separate build scripts for different configurations in the BUILD directory.

You can find a list of the needed packages/libraries for building on Linux here.

Example of Compiling MariaDB for Debugging Using Build Scripts

- Scripts containing "debug" in the name are for developers that want to build, develop and test MariaDB.
- Scripts containing "valgrind" in the name are for running mysqld under [[|http://valgrind.org|valgrind]]. Compiling for valgrind means that we are using another implementation of MEM_ROOT to allow valgrind to better detect memory overruns. In addition, some memory areas are marked as used/not used to remove some false positives.

- Scripts containing "max" in the name include all normal plugins.

Here is an example of how to compile MariaDB for debugging in your home directory with MariaDB 5.2.9 as an example:

```
cd ~
mkdir mariadb
cd mariadb
tar xvf mariadb-5.2.9.tar.gz
ln -s mariadb-5.2.9 current
cd current
./BUILD/compile-pentium64-debug-max
```

The last command will produce a debug version of `sql/mysqld`. If you have a system other than 64 bit Intel/AMD on Linux you can use a different `BUILD/...-debug-max` file. If this fails, you can try with:

```
cmake --build  .  --config Debug
make -j4
```

Building Optimized Build With Debug Symbols

To build MariaDB with symbols, to get better stack traces and to be able to debug the binary with `gdb`, you need to supply the `-g3` option to the `gcc` compiler.

Just compiling with `-g3` will make the binary much bigger but the slowdown of the server should be negligible.

One way to do this is to edit the script

```
BUILD/compile-pentium64-max
```

and add the -g3 last on the line with `extra_flags`, like this:

```
extra_flags="$pentium64_cflags $fast_cflags -g3"
```

After that you can compile MariaDB with debugging symbols by just execution the above script.

Doing a Debug Build on Debian/Ubuntu

To build a "mysqld" binary with debugging enabled that uses the same parameters as the ones used in Debian/Ubuntu binary packages, you must do as follows (you must have a deb-src line of one of the MariaDB repositories on your /etc/apt/sources.list in order to do that):

```
apt-get build-dep mariadb-10.2
apt-get install cmake libaio1 libaio-dev
apt-get source mariadb-10.2
cd mariadb-10.2*
./debian/rules configure
./BUILD/compile-pentuim64-debug-all
```

Then you will have your "debugging enabled" mysqld binary in the sql/ directory.

This binary can directly replace the one provided by the binary package that is placed on "/usr/bin/mysqld".

Temporarily Installing your Debug Build

The commands shown below replace the release `mysqld` binary with the debug `mysqld` binary that you compiled. Most importantly, they replace the binary in a way which makes it trivial to revert back to the original release `mysqld` binary.

First, stop MariaDB.

Then, use the `mv` utility to rename the release `mysqld` binary:

```
sudo mv /usr/sbin/mysqld /usr/sbin/mysqld-orig
```

Note: Do not use the `cp` utility because that will change the file modification timestamp.

Then, install the debug `mysqld` binary from your source tree:

```
sudo install ~/mariadb-10.3.14/sql/mysqld /usr/sbin/mysqld-debug
```

Then, link the `mysqld` path to the path of your debug `mysqld` binary:

```
sudo ln -s /usr/sbin/mysqld-debug /usr/sbin/mysqld
```

Then, start MariaDB.

Be sure to replace `/usr/sbin/mysqld` with the path to your `mysqld` binary and to also replace `~/mariadb-10.3.14/sql/mysqld` with the path to your debug `mysqld` binary.

Reinstalling your Release Build

If you want to restore your original `mysqld` binary, you can do it with the following process::

First, stop MariaDB.

Then, execute the following command to delete the symbolic link:

```
sudo rm /usr/sbin/mysqld
```

Then, execute the following command to move the original `mysqld` release binary back into place:

```
sudo mv /usr/sbin/mysqld-orig /usr/sbin/mysqld
```

Then, start MariaDB.

Be sure to replace `/usr/sbin/mysqld` with the path to your `mysqld` binary

Notice that the debug `mysqld` binary located at `/usr/sbin/mysqld-debug` was not deleted. Only the symbolic link to this file was deleted. The debug `mysqld` binary is still present if it is needed again in the future.

Different Compilation Options

Changing DBUG_ASSERT to Print to Error Log

A debug binary has lots of code checks and asserts, that are not checked in production. This is done to get more performance when running in production. In some cases, when one is trying to find a hard-to-repeat bug, it could be beneficial to have these checks in production builds too.

Compiling with `-DDBUG_ASSERT_AS_PRINTF` will change DBUG_ASSERT() to print any failed check to the error log.

```
cmake . -DDBUG_ASSERT_AS_PRINTF
```

Enabling the above option should not have any notable impact on performance (probably < 1% slowdown). This is achieved by grouping asserts in MariaDB server code into two groups:

- Fast checks, using `DBUG_ASSERT()` : These are converted to printing to error log.
- Slow checks, using `DBUG_SLOW_ASSERT()` . These will always be removed in production builds.

2.1.2.8.21 Cross-compiling MariaDB

Buildroot

Buildroot is a way to cross compile MariaDB and other packages into a root filesystem. In the menuconfig you need to enable "Enable C++ Support" first under "Toolchain". After C++ is enabled MariaDB is an option under "Target Packages" ->"Libraries" -> "Databases" -> "mysql support" -> "mysql variant" -> "mariadb". Also enable the "mariadb server" below the "mysql support" option.

Details

To cross-compile with cmake you will need a toolchain file. See, for example, here. Besides cmake specific variables it

should have, at least

```
SET(STACK_DIRECTION -1)
SET(HAVE_IB_GCC_ATOMIC_BUILTINS 1)
```

with appropriate values for your target architecture. Normally these MariaDB configuration settings are detected by running a small test program, and it cannot be done when cross-compiling.

Note that during the build few helper tools are compiled and then immediately used to generate more source files for this very build. When cross-compiling these tools naturally should be built for the host architecture, not for the target architecture. Do it like this (assuming you're in the mariadb source tree):

```
$ mkdir host
$ cd host
$ cmake ..
$ make import_executables
$ cd ..
```

Now the helpers are built and you can cross-compile:

```
$ mkdir target
$ cd target
$ cmake .. -DCMAKE_TOOLCHAIN_FILE=/path/to/toolchain/file.cmake -
DIMPORT_EXECUTABLES=../host/import_executables.cmake
$ make
```

Here you invoke cmake, specifying the path to your toolchain file and the path to the `import_executables.cmake` that you have just built on the previous step. Of course, you can also specify any other cmake parameters that could be necessary for this build, for example, enable or disable specific storage engines.

See also https://lists.launchpad.net/maria-discuss/msg02911.html

2.1.2.8.22 MariaDB Source Configuration Options

Contents

All CMake configuration options for MariaDB can be displayed with:

```
cmake . -LH
```

2.1.2.8.23 Building RPM Packages From Source

To generate RPM packages from the build, supply the `-DRPM=xxx` flag to CMake, where the value `xxx` is the name of the distribution you're building for. For example, centos7 or rocky8 or fedora39 or sles15.

What these do are controlled in the following CMake files:

- `cmake/cpack_rpm.cmake`
- `cmake/build_configurations/mysql_release.cmake`
- `cmake/mariadb_connector_c.cmake`

To build the packages you should execute:

```
$ umask 0022
$ cmake --build . --target package
```

2.1.2.8.24 Compile and Using MariaDB with Sanitizers (ASAN, UBSAN, TSAN, MSAN)

Contents

What are Sanitizers?

Sanitizers are open source runtime error detectors developed by Google that are enabled during the compile step. These sanitizers add extra code during compilation that will throw exceptions when certain errors are detected.

AddressSanitizer (aka ASAN) is a memory error detector for C/C++. It finds a lot of the same things as valgrind, but with a lot less overhead.

- Use after free (dangling pointer dereference)
- Heap buffer overflow
- Stack buffer overflow
- Global buffer overflow
- Use after return
- Use after scope
- Initialization order bugs
- Memory leaks

To use ASAN you need a gcc version that supports ASAN. gcc 4.8.5 and up are known to work.

In addition to ASAN there are sanitizers for Undefined Behaviour, Thread and Memory related errors.

UndefinedBehaviourSanitizer (aka UBSAN)

ThreadSanitizer (aka TSAN)

MemorySanitizer (aka MSAN)

How to Compile MariaDB with Sanitizers

Before using ASAN locally, please ensure that it is installed on the system:

```
yum install -y /usr/lib64/libasan.so.6.0.0
```

ASAN is supported in MariaDB 10.1 and up.

You can use one of the two following build commands:

```
cmake . -DWITH_ASAN=ON
```

or from MariaDB 10.2 and up:

```
./BUILD/compile-pentium64-asan-max
```

Additionally, UBSAN, TSAN, and MSAN can be enabled in a similar way:

UBSAN:

```
yum install -y /usr/lib64/libubsan.so.1.0.0
cmake . -DWITH_UBSAN=ON
```

TSAN:

```
yum install -y /usr/lib64/libtsan.so.0.0.0
cmake . -DWITH_TSAN=ON
```

MSAN:

Note: keep in mind that only clang supports MSAN, g++ or other compilers will not work.

```
cmake . -DWITH_MSAN=ON
```

Running an MSAN Build

The time consuming aspect of building with MSAN is having instrumented system libraries. As MariaDB Foundation have built the MSAN container already for buildbot, this is how you re-use this for building locally.

First, run the container where your current directory is the source directory:

```
podman run -v $PWD:/source:z -ti --user buildbot --entrypoint bash --shm-size 5G --env MSAN_SYMBO
```

Note: docker can be used instead of the lighter weight podman if you so desire.

The shm-size is for the MTR tests which exceed the 64k default shm-size.

All the following instructions are executed within the container. Now run the configure stage of cmake:

```
cmake /source -DCMAKE_C_COMPILER=clang-15 -DCMAKE_CXX_COMPILER=clang++-15 '-DCMAKE_C_FLAGS=-O2 -W
```

Run the build stage:

```
cmake --build . --parallel
...
[100%] Built target mariadbd
[100%] Linking CXX executable mariadb-backup
Creating mariabackup link
[100%] Built target mariadb-backup
```

As the important MTR step needs to use the instrumented libraries MTR is run using the LD_LIBRARY_PATH to use those.

```
LD_LIBRARY_PATH=/msan-libs mysql-test/mtr  --mem --big-test --force --retry=0 --skip-test=.*compr
```

As newer versions occur and improvements happen these instructions may change. Look at the execution on the buildbot builder to see if any changes have been made.

Running an ASAN Build

To run mysqld with instrumentation you have to set the `ASAN_OPTIONS` environment variable before starting `mysqld`. Either in your shell or in your mysqld_safe script.

```
export ASAN_OPTIONS=abort_on_error=1
```

The above command will abort any instrumented executable if any errors are found, which is good for debugging. If you set abort_on_error=0 all server errors are logged to your error log file (mysqld.err).

To catch errors for other processes than the server, you can set more options, like this:

```
export ASAN_OPTIONS=abort_on_error=1:log_path=/tmp/asan
```

If you are seeing an incomplete stack trace for a memory allocation, you may rerun the failing test with

```
export ASAN_OPTIONS=abort_on_error=1:log_path=/tmp/asan:fast_unwind_on_malloc=0
```

To get core dumps of failures:

```
export ASAN_OPTIONS=abort_on_error=1:disable_coredump=0
```

To see all the options (or to check if an executable is instrumented), you may try the following:

```
ASAN_OPTIONS=help=1 extra/perror 0
```

Using Valgrind

The MariaDB test system can use Valgrind for finding memory leaks and wrong memory accesses. Valgrind is an

instrumentation framework for building dynamic analysis tools. If Valgrind is installed on your system, you can simply use mysql-test-run --valgrind to run the test under Valgrind.

2.1.2.9 Distributions Which Include MariaDB

Contents

The following is a partial list of distributions which include MariaDB in their package repositories. For these you can use the distribution's management system to install MariaDB.

The term "default" in the list below refers to the distribution's default relational or MySQL-type database.

Linux Distributions

- AlmaLinux — MariaDB 10.5 was included since 8.8.
- Alpine Linux — Defaults to MariaDB. MariaDB 10.6 has been available 3.12.11 and MariaDB 10.11 since 3.18.2.
- 4mLinux — Defaults to MariaDB. MariaDB 10.6 has been available since 37.0
- ALT Linux — MariaDB 5.5 included in 7.0.0, MariaDB is default from 8.1, which includes MariaDB 10.1, 9.1 includes MariaDB 10.4
- Arch Linux — Features MariaDB 10.11, and replaced MySQL as a default
- Austrumi — Defaulted to MariaDB 5.3 in 2.4.8, 3.5.8 includes MariaDB 10.1
- BlackArch — Defaulted to MariaDB 5.5 in 2014.07.01, 2020.12.01 includes MariaDB 10.5
- BlueOnyx — 5209 defaults to MariaDB 5.5, 5210R to MariaDB 10.3
- BlueStar — 4.8.4 defaults to MariaDB 10.1, 5.4.2 to MariaDB 10.5
- CentOS — MariaDB 5.5 replaced MySQL in CentOS 7
- The Chakra Project — MariaDB replaced MySQL as default in 2013.05. 2016.02 includes MariaDB 10.1
- Debian — Debian 10 "Buster" includes MariaDB 10.3, Debian 11 "Bullseye" MariaDB 10.5, Debian 12 "Bookworm" MariaDB 10.11 as default.
- Elastix —Defaulted to MariaDB 5.5 in 4.0.76
- Exherbo —Includes MariaDB 10.4
- Fedora — MariaDB 5.5 became the default relational database in Fedora 19. Fedora 30 includes MariaDB 10.3, Fedora 34 includes MariaDB 10.5.
- Funtoo —Includes MariaDB 5.5
- Gentoo Linux
- Guix —11.2.0 includes MariaDB 10.1
- Hanthana — 19.1 defaulted to MariaDB 5.5, 21 includes MariaDB 10.0, 28.1.2 includes MariaDB 10.2, 30 includes MariaDB 10.3
- KaOS —Defaulted to MariaDB 5.5 in 2014.12, 2019.04 includes MariaDB 10.3
- Kali Linux —2017.3 Included MariaDB 10.1, 2023.1 includes MariaDB 10.11
- GNU/Linux KDu — MariaDB 5.5 replaced MySQL as a default in 2.0 Final.
- Korora —Defaulted to MariaDB in 19.1, 26 includes MariaDB 10.1
- Linux from Scratch —10.1-BLFS defaults to MariaDB 10.5
- Lunar —1.7.0 includes MariaDB 5.5, Moonbase includes MariaDB 10.5
- Mageia — MariaDB 5.5 replaced MySQL as default in version 3, MariaDB 10.3 from version 7.1, MariaDB 10.5 from 8, MariaDB 10.11 from 9.
- Manjaro Linux — Defaulted to MariaDB 5.5 in 0.8.11, 16.10.3 includes MariaDB 10.1.
- NixOS —14.0.4.630 included MariaDB 10.0, 18.09 includes MariaDB 10.2, Stable includes MariaDB 10.5
- Network Security Toolkit —20-5663 defaulted to MariaDB 5.5, 32-11992 includes MariaDB 10.4
- NuTyX —14.11 included MariaDB 10.0, defaulted to MariaDB 10.1 in 8.2.1, 20.12.1 includes MariaDB 10.5
- OpenELEC
- OpenEuler —21.9 includes 10.5, 22.03 LTS includes MariaDB 10.5
- Open Mandriva —Defaulted to MariaDB 10.0 in 2014.2, includes MariaDB 10.5 in 4.2
- openSUSE — MariaDB 5.5 became the default relational database in openSUSE 12.3, and MariaDB 10.6 the default since 15.5
- Oracle Linux — 7.3 includes MariaDB 5.5. 8.0 includes MariaDB 10.3
- Paldo —Defaults to MariaDB 10.5 in Stable
- Parabola GNU/Linux —Includes MariaDB 10.1 since 3.7
- Parrot Security —Based on Debian's testing branch (stretch), Parrot Security switched from MySQL to MariaDB 10.0 in 3.1, 4.7 includes MariaDB 10.3
- Parted Magic —Defaulted to MariaDB 5.5 in 2015_11_13
- PCLinuxOS —Replaced MySQL with MariaDB 10.1 in 2017.03
- Pisi Linux —Defaulted to MariaDB 10.0 in 1.1
- Plamo —Defaulted to MariaDB 5.5 in 5.3.1, 7.3 includes MariaDB 10.2
- PoliArch —Defaulted to MariaDB 5.5 in 13.1, 15.1 includes MariaDB 10.0
- Red Hat Enterprise Linux — MariaDB 5.5 was included as the default "MySQL" database since RHEL 7, RHEL 9 includes MariaDB 10.5
- Rocky Linux — MariaDB 10.5 was included since 8.7.
- ROSA —Defaulted to MariaDB 10.0 in R4 and MariaDB 10.1 in R9.
- Sabayon —Included MariaDB 10.0 in 14.12, includes MariaDB 10.3 since 19.03

- Scientific Linux —Defaulted to MariaDB 5.5 in 7.3
- Slackware — MariaDB 5.5 replaced MySQL as default in 14.1. 15.0 includes MariaDB 10.5, current includes MariaDB 10.11
- SliTaz GNU/Linux —Includes MariaDB 10.0 in 5.0-rolling
- SME Server— started to use MariaDB 5.5 from 10.0
- Springdale Linux —Defaulted to MariaDB 5.5 in 7.2, 8.1 includes MariaDB 10.3
- SuliX — Defaults to MariaDB 5.5 in 8.
- SUSE Linux Enterprise — MariaDB 10.6 is the default relational database option on 15-SP6
- Ubuntu —MariaDB 5.5 was included in Trusty Tahr 14.04. 20.04 includes MariaDB 10.3, and 22.04 includes 10.6.
- Void — Includes MariaDB 10.5 in current
- Wifislax — Defaulted to MariaDB 10.0 in 4.11.1

BSD Distributions

- Dragonfly BSD — 3.8 included MariaDB 5.5. 5.8.0 includes MariaDB 10.4.
- FreeBSD — MariaDB is available in the ports tree and the FreeBSD Manual has instructions on Installing Applications: Packages and Ports. MariaDB 10.5 is included in 12.2
  - MariaDB on FreshPorts
- NetBSD — 6.1 and 7.0 include MariaDB 5.5.
- OpenBSD — MariaDB 10.0 was included in 5.7, 6.8 includes MariaDB 10.5.

macOS

- Homebrew — If you have Homebrew installed, you can install MariaDB Server by executing `brew install mariadb`. Find out more at Installing MariaDB Server on macOS Using Homebrew.
- MacPorts —This provides mariadb and mariadb-server. A quick guide on how to install it.

2.1.2.10 Running Multiple MariaDB Server Processes

Contents

It is possible to run multiple MariaDB Server processes on the same server, but there are certain things that need to be kept in mind. This page will go over some of those things.

Configuring Multiple MariaDB Server Processes

If multiple MariaDB Server process are running on the same server, then at minimum, you will need to ensure that the different instances do not use the same `datadir`, `port`, and `socket`. The following example shows these options set in an option file:

```
[client]
# TCP port to use to connect to mysqld server
port=3306
# Socket to use to connect to mysqld server
socket=/tmp/mysql.sock
[mariadb]
# TCP port to make available for clients
port=3306
#  Socket to make available for clients
socket=/tmp/mysql.sock
# Where MariaDB should store all its data
datadir=/usr/local/mysql/data
```

The above values are the defaults. If you would like to run multiple MariaDB Server instances on the same server, then you

will need to set unique values for each instance.

There may be additional options that also need to be changed for each instance. Take a look at the full list of options for `mysqld`.

To see the current values set for an instance, see Checking Program Options for how to do so.

To list the default values, check the end of:

```
mysqld --help --verbose
```

Starting Multiple MariaDB Server Processes

There are several different methods to start or stop the MariaDB Server process. There are two primary categories that most of these methods fall into: starting the process with the help of a service manager, and starting the process manually. See Starting and Stopping MariaDB for more information.

Service Managers

sysVinit and systemd are the most common Linux service managers. launchd is used in MacOS X. Upstart is a less common service manager.

Systemd

RHEL/CentOS 7 and above, Debian 8 Jessie and above, and Ubuntu 15.04 and above use systemd by default.

For information on how to start and stop multiple MariaDB Server processes on the same server with this service manager, see systemd: Interacting with Multiple MariaDB Server Processes.

Starting the Server Process Manually

mysqld

`mysqld` is the actual MariaDB Server binary. It can be started manually on its own.

If you want to force each instance to read only a single option file, then you can use the `--defaults-file` option:

```
mysqld --defaults-file=/etc/my_instance1.cnf
```

mysqld_safe

`mysqld_safe` is a wrapper that can be used to start the `mysqld` server process. The script has some built-in safeguards, such as automatically restarting the server process if it dies. See mysqld_safe for more information.

If you want to force each instance to read only a single option file, then you can use the `--defaults-file` option:

```
mysqld_safe --defaults-file=/etc/my_instance1.cnf
```

mysqld_multi

`mysqld_multi` is a wrapper that can be used to start the `mysqld` server process if you plan to run multiple server processes on the same host. See mysqld_multi for more information.

Other Options

In some cases, there may be easier ways to run multiple MariaDB Server instances on the same server, such as:

- Using dbdeployer.
- Starting multiple Docker containers.

2.1.2.11 Installing MariaDB Alongside MySQL

MariaDB was originally designed as a drop-in replacement of MySQL, with more features, new storage engines, fewer bugs, and better performance, but you can also install it alongside MySQL. (This can be useful, for example, if you want to migrate databases/applications one by one.)

Here are the steps to install MariaDB near an existing MySQL installation.

- Download the compiled binary tar.gz that contains the latest version (mariadb-5.5.24-linux-x86_64.tar.gz - as of writing this article) and extract the files in a directory of your choice. I will assume for this article that the directory was **/opt**.

```
[root@mariadb-near-mysql ~]# cat /etc/issue
CentOS release 6.2 (Final)

[root@mariadb-near-mysql ~]# rpm -qa mysql*
mysql-5.1.61-1.el6_2.1.x86_64
mysql-libs-5.1.61-1.el6_2.1.x86_64
mysql-server-5.1.61-1.el6_2.1.x86_64

[root@mariadb-near-mysql ~]# ps axf | grep mysqld
 2072 pts/0    S+     0:00          \_ grep mysqld
 1867 ?        S      0:01 /bin/sh /usr/bin/mysqld_safe --datadir=/var/lib/mysql --
socket=/var/lib/mysql/mysql.sock ...
 1974 ?        Sl     0:06  \_ /usr/libexec/mysqld --basedir=/usr --datadir=/var/lib/mysql --
user=mysql ...
```

- Create data directory and symlinks as below:

```
[root@mariadb-near-mysql opt]# mkdir mariadb-data
[root@mariadb-near-mysql opt]# ln -s mariadb-5.5.24-linux-x86_64 mariadb
[root@mariadb-near-mysql opt]# ls -al
total 20
drwxr-xr-x.  5 root root 4096 2012-06-06 07:27 .
dr-xr-xr-x. 23 root root 4096 2012-06-06 06:38 ..
lrwxrwxrwx.  1 root root   27 2012-06-06 07:27 mariadb -> mariadb-5.5.24-linux-x86_64
drwxr-xr-x. 13 root root 4096 2012-06-06 07:07 mariadb-5.5.24-linux-x86_64
drwxr-xr-x.  2 root root 4096 2012-06-06 07:26 mariadb-data
```

- Create group **mariadb** and user **mariadb** and set correct ownerships:

```
[root@mariadb-near-mysql opt]# groupadd --system mariadb
[root@mariadb-near-mysql opt]# useradd -c "MariaDB Server" -d /opt/mariadb -g mariadb --system
mariadb
[root@mariadb-near-mysql opt]# chown -R mariadb:mariadb mariadb-5.5.24-linux-x86_64/
[root@mariadb-near-mysql opt]# chown -R mariadb:mariadb mariadb-data/
```

- Create a new **my.cnf** in **/opt/mariadb** from support files:

```
[root@mariadb-near-mysql opt]# cp mariadb/support-files/my-medium.cnf mariadb-data/my.cnf
[root@mariadb-near-mysql opt]# chown mariadb:mariadb mariadb-data/my.cnf
```

- Edit the file **/opt/mariadb-data/my.cnf** and add custom paths, socket, port, user and the most important of all: data directory and base directory. Finally the file should have at least the following:

```
[client]
port  = 3307
socket  = /opt/mariadb-data/mariadb.sock

[mysqld]
datadir         = /opt/mariadb-data
basedir         = /opt/mariadb
port  = 3307
socket  = /opt/mariadb-data/mariadb.sock
user            = mariadb
```

- Copy the init.d script from support files in the right location:

```
[root@mariadb-near-mysql opt]# cp mariadb/support-files/mysql.server /etc/init.d/mariadb
[root@mariadb-near-mysql opt]# chmod +x /etc/init.d/mariadb
```

- Edit **/etc/init.d/mariadb** replacing **mysql** with **mariadb** as below:

```
- # Provides: mysql
+ # Provides: mariadb
- basedir=
+ basedir=/opt/mariadb
- datadir=
+ datadir=/opt/mariadb-data
- lock_file_path="$lockdir/mysql"
+ lock_file_path="$lockdir/mariadb"
```

The trickiest part will be the last changes to this file. You need to tell mariadb to use only one cnf file. In the **start** section after **$bindir/mysqld_safe** add **--defaults-file=/opt/mariadb-data/my.cnf**. Finally the lines should look like:

```
# Give extra arguments to mysqld with the my.cnf file. This script
# may be overwritten at next upgrade.
$bindir/mysqld_safe --defaults-file=/opt/mariadb-data/my.cnf --datadir="$datadir" --pid-
file="$mysqld_pid_file_path" $other_args >/dev/null 2>&1 &
```

The same change needs to be made to the mariadb-admin command below in the wait_for_ready() function so that the mariadb start command can properly listen for the server start. In the **wait_for_ready()** function, after **$bindir/mariadb-admin** add **--defaults-file=/opt/mariadb-data/my.cnf**. The lines should look like:

```
wait_for_ready () {
[...]
    if $bindir/mariadb-admin --defaults-file=/opt/mariadb-data/my.cnf ping >/dev/null 2>&1;
then
```

- Run mariadb-install-db by explicitly giving it the my.cnf file as argument:

```
[root@mariadb-near-mysql opt]# cd mariadb
[root@mariadb-near-mysql mariadb]# scripts/mariadb-install-db --defaults-file=/opt/mariadb-
data/my.cnf
```

- Now you can start MariaDB by

```
[root@mariadb-near-mysql opt]# /etc/init.d/mariadb start
Starting MySQL...                                          [  OK  ]
```

- Make MariaDB start at system start:

```
[root@mariadb-near-mysql opt]# cd /etc/init.d
[root@mariadb-near-mysql init.d]# chkconfig --add mariadb
[root@mariadb-near-mysql init.d]# chkconfig --levels 3 mariadb on
```

- Finally test that you have both instances running:

```
[root@mariadb-near-mysql ~]# mysql -e "SELECT VERSION();"
+-----------+
| VERSION() |
+-----------+
| 5.1.61    |
+-----------+
[root@mariadb-near-mysql ~]# mysql -e "SELECT VERSION();" --socket=/opt/mariadb-
data/mariadb.sock
+----------------+
| VERSION()      |
+----------------+
| 5.5.24-MariaDB |
+----------------+
```

What about MariaDB Upgrades ?

By having the **mariadb.socket**, **my.cnf** file and **databases** in **/opt/mariadb-data** if you want to upgrade the MariaDB version you will will only need to:

- extract the new version from the archive in **/opt** near the current version
- stop MariaDB

- change the symlink **mariadb** to point to the new directory
- start MariaDB
- run upgrade script... but remember you will need to provide the socket option **--socket=/opt/mariadb-data/mariadb.sock**

2.1.2.12 GPG

Contents

The MariaDB project signs their MariaDB packages for Debian, Ubuntu, Fedora, CentOS, and Red Hat.

Debian / Ubuntu key

Our repositories for Debian "Sid" and the Ubuntu 16.04 and beyond "Xenial" use the following GPG signing key. As detailed in MDEV-9781, APT 1.2.7 (and later) prefers SHA2 GPG keys and now prints warnings when a repository is signed using a SHA1 key like our previous GPG key. We have created a SHA2 key for use with these.

Information about this key:

- The short Key ID is: `0xC74CD1D8`
- The long Key ID is: `0xF1656F24C74CD1D8`
- The full fingerprint of the key is: `177F 4010 FE56 CA33 3630 0305 F165 6F24 C74C D1D8`
- The key can be added on Debian-based systems using the following command:

```
sudo apt-key adv --recv-keys --keyserver hkp://keyserver.ubuntu.com:80 0xF1656F24C74CD1D8
```

- Usage of the `apt-key` command is deprecated in the latest versions of Debian and Ubuntu, and the replacement method is to download the keyring file to the `/etc/apt/trusted.gpg.d/` directory. This can be done with the following:

```
sudo curl -LsSo /etc/apt/trusted.gpg.d/mariadb-keyring-2019.gpg https://supplychain.mariadb.
```

RPM / Source Key 2023+

Beginning in 2023 we migrated the key used to sign our yum/dnf/zypper repositories and to sign our source code and binary tarballs to the same key we use for Debian and Ubuntu. This unifies our GPG signing and enables our repositories to be compatible with FIPS and other regulations that mandate a stronger signing key.

The key can be imported on RPM-based systems using the following command:

```
sudo rpm --import https://supplychain.mariadb.com/MariaDB-Server-GPG-KEY
```

or

```
sudo rpmkeys --import https://supplychain.mariadb.com/MariaDB-Server-GPG-KEY
```

RPM / Source key pre-2023

The GPG Key ID of the MariaDB signing key we used for yum/dnf/zypper repositories and to sign our source code tarballs until the end of 2022 was `0xCBCB082A1BB943DB`. The short form of the id is `0x1BB943DB` and the full key fingerprint is:

```
1993 69E5 404B D5FC 7D2F E43B CBCB 082A 1BB9 43DB
```

This key was used by the yum/dnf/zypper repositories for RedHat, CentOS, Fedora, openSUSE, and SLES.

If you configure the mariadb.org rpm repositories using the repository configuration tool (see below) then your package manager will prompt you to import the key the first time you install a package from the repository.

You can also import the key directly using the following command:

```
sudo rpmkeys --import https://supplychain.mariadb.com/MariaDB-Server-GPG-KEY
```

Configuring Repositories

See the this page for details on using the `mariadb_repo_setup` script to configure repositories that use these keys.

See the this page for details on configuring MariaDB Foundation repositories that use these keys.

2.1.2.13 MariaDB Platform Deprecation Policy

Contents

The MariaDB Foundation tries to support as many different Operating Systems, Linux Distributions, and processor architectures as possible. However, when a distribution or OS stops receiving security and other updates it becomes difficult for the MariaDB project to provide freely packages for that platform. In such cases, our policy is to deprecate the platform and stop providing binary packages for it.

This policy and related deprecated dates are from the MariaDB Foundation. For information on the MariaDB Corporation's policies related to supporting software, see the Engineering Policies page.

Current Package Platforms

The MariaDB project builds packages for the following:

Platform	Planned Deprecation Date
Windows Server 2019	Jan 2024
Ubuntu 23.04 "Lunar"	Jan 2024
Red Hat Enterprise Linux 7.x (x86_64 only)	Jun 2024
CentOS 7.x (x86_64 only)	Jun 2024
Debian 10 "Buster" (i386, amd64 and arm64)	Jun 2024
Ubuntu 23.10 "Mantic"	Jul 2024
SLES 12.x	Oct 2024
Ubuntu 20.04 "Focal"	Apr 2025
Windows 10	Oct 2025
Debian 11 "Bullseye"	Jun 2026
Windows Server 2022	Oct 2026
Ubuntu 22.04 "Jammy"	Apr 2027
SLES 15.x	Jul 2028
Red Hat Enterprise Linux 8.x	Jun 2029
Red Hat Enterprise Linux 9.x	Jun 2032
Debian 12 "Bookworm"	TBD
Fedora 37	approximately 1 month after release of Fedora 39
Fedora 38	approximately 1 month after release of Fedora 40
Fedora 39	approximately 1 month after release of Fedora 41
SLES 12.5	6 months after release of SLES 12.6
SLES 15.0	6 months after release of SLES 15.1
Windows 11	TBC

- Ubuntu Release Information (End of Standard Support)

- Debian LTS Information (i386, amd64 and arm64 only)
- General Debian Release Information
- Red Hat Enterprise Linux Release Information
- Fedora Release Information
- FreeBSD Security Information
- openSUSE Lifetime Information
- SLES Lifecycle Information
- Windows client Lifecycle Information
- Windows Server Lifecycle Information

Deprecated Package Platforms

The MariaDB project no longer builds packages for the following Operating Systems and Linux Distributions:

Platform	Deprecation Date	Final MariaDB Version(s)
Ubuntu 18.04 LTS "Bionic"	Jun 2023	MariaDB 10.4.31, MariaDB 10.5.22, MariaDB 10.6.15, MariaDB 10.7.8, MariaDB 10.8.8, MariaDB 10.9.8, MariaDB 10.10.6, MariaDB 10.11.5, MariaDB 11.0.3
Ubuntu 22.10 "Kinetic"	Jul 2023	MariaDB 10.6.15, MariaDB 10.7.8, MariaDB 10.8.8, MariaDB 10.9.8, MariaDB 10.10.6, MariaDB 10.11.5, MariaDB 11.0.3
Fedora 36	May 2023	MariaDB 10.6.13, MariaDB 10.8.8, MariaDB 10.9.6, MariaDB 10.10.4, MariaDB 10.11.3
Fedora 35	Jan 2023	MariaDB 10.5.19, MariaDB 10.6.12, MariaDB 10.7.8, MariaDB 10.9.5, MariaDB 10.10.3
Windows 8.1	Jan 2023	MariaDB 10.5.19, MariaDB 10.6.12, MariaDB 10.7.8, MariaDB 10.9.5, MariaDB 10.10.3
Debian 10 "Buster" (ppc64el)	Jul 2022	MariaDB 10.3.36, MariaDB 10.4.26, MariaDB 10.5.17, MariaDB 10.6.9, MariaDB 10.7.5, MariaDB 10.8.4
Ubuntu 21.10 "Impish"	Jul 2022	MariaDB 10.5.16, MariaDB 10.6.8, MariaDB 10.7.4, MariaDB 10.8.3, MariaDB 10.9.1
Debian 9 "Stretch"	Jun 2022	MariaDB 10.2.44, MariaDB 10.3.35, MariaDB 10.4.25, MariaDB 10.5.16, MariaDB 10.6.8, MariaDB 10.7.4, MariaDB 10.8.3, MariaDB 10.9.1
Red Hat Enterprise Linux 7.x (non-x86_64)	May 2022	MariaDB 10.2.44, MariaDB 10.3.35, MariaDB 10.4.25, MariaDB 10.5.16, MariaDB 10.6.8, MariaDB 10.7.4, MariaDB 10.8.3, MariaDB 10.9.1
CentOS 7.x (non-x86_64)	May 2022	MariaDB 10.2.44, MariaDB 10.3.35, MariaDB 10.4.25, MariaDB 10.5.16, MariaDB 10.6.8, MariaDB 10.7.4, MariaDB 10.8.3, MariaDB 10.9.1
Fedora 34	May 2022	MariaDB 10.5.16, MariaDB 10.6.8, MariaDB 10.7.4, MariaDB 10.8.3, MariaDB 10.9.1
Ubuntu 21.04 "Hirsute"	Feb 2022	MariaDB 10.7.2, MariaDB 10.6.6, MariaDB 10.5.14
Fedora 33	Feb 2022	MariaDB 10.7.2, MariaDB 10.6.6, MariaDB 10.5.14, MariaDB 10.4.23
CentOS 8.x	Feb 2022	MariaDB 10.7.2, MariaDB 10.6.6, MariaDB 10.5.14, MariaDB 10.4.23, MariaDB 10.3.33
Ubuntu 20.10 "Groovy"	Jul 2021	MariaDB 10.6.4, MariaDB 10.5.12, MariaDB 10.4.21, MariaDB 10.3.31
Fedora 32	Apr 2021	MariaDB 10.5.10, MariaDB 10.4.19, MariaDB 10.3.29
Ubuntu 16.04 LTS "Xenial"	Apr 2021	MariaDB 10.5.10, MariaDB 10.4.19, MariaDB 10.3.29, MariaDB 10.2.38
Mint 18 LTS "Serena"	Apr 2021	MariaDB 10.5.10, MariaDB 10.4.19, MariaDB 10.3.29, MariaDB 10.2.38
Fedora 31	Nov 2020	MariaDB 10.5.7, MariaDB 10.4.16, MariaDB 10.3.26
Red Hat Enterprise Linux 6.x	Nov 2020	MariaDB 10.4.16, MariaDB 10.3.26, MariaDB 10.2.35, MariaDB 10.1.48
CentOS 6.x	Nov 2020	MariaDB 10.4.16, MariaDB 10.3.26, MariaDB 10.2.35, MariaDB 10.1.48

Fedora 30	Aug 2020	MariaDB 10.5.5, MariaDB 10.4.14, MariaDB 10.3.24
Ubuntu 19.10 "Eoan"	Jul 2020	MariaDB 10.5.5, MariaDB 10.4.14, MariaDB 10.3.24
Debian 8 "Jessie"	Jun 2020	MariaDB 10.4.13, MariaDB 10.3.23, MariaDB 10.2.32, MariaDB 10.1.45
Ubuntu 19.04 "Disco"	Jan 2020	MariaDB 10.4.12, MariaDB 10.3.22
Windows Server 2008	Jan 2020	MariaDB 10.4.12, MariaDB 10.3.22
Windows Server 2008 R2	Jan 2020	MariaDB 10.4.12, MariaDB 10.3.22
Windows 7	Jan 2020	MariaDB 10.4.12, MariaDB 10.3.22
Fedora 29	Dec 2019	MariaDB 10.4.11, MariaDB 10.3.21
Ubuntu 18.10 "Cosmic"	Jul 2019	MariaDB 10.4.7, MariaDB 10.3.17, MariaDB 10.2.26, MariaDB 10.1.41
openSUSE 42.3	Jun 2019	MariaDB 10.4.7, MariaDB 10.3.16, MariaDB 10.2.25, MariaDB 10.1.41
Fedora 28	May 2019	MariaDB 10.4.5, MariaDB 10.3.15, MariaDB 10.2.24
Ubuntu 14.04 LTS "Trusty"	Apr 2019	MariaDB 10.4.4, MariaDB 10.3.14, MariaDB 10.2.23, MariaDB 10.1.40, MariaDB 5.5.64
Mint 17.1 LTS "Rebecca"	Apr 2019	MariaDB 10.4.4, MariaDB 10.3.14, MariaDB 10.2.23, MariaDB 10.1.40, MariaDB 5.5.64
Mint 17 LTS "Qiana"	Apr 2019	MariaDB 10.4.4, MariaDB 10.3.14, MariaDB 10.2.23, MariaDB 10.1.40, MariaDB 5.5.64
SLES 11.4	Mar 2019	MariaDB 10.1.40, MariaDB 5.5.64
Windows Server 2012 R2	Oct 2018	
Fedora 27	Nov 2018	MariaDB 10.3.11, MariaDB 10.2.19
Ubuntu 17.10 "Artful"	Jul 2018	MariaDB 10.3.8, MariaDB 10.2.16, MariaDB 10.1.34
Fedora 26	May 2018	MariaDB 10.3.7, MariaDB 10.2.15, MariaDB 10.1.33
Debian 7 "Wheezy"	May 2018	MariaDB 10.3.7, MariaDB 10.2.15, MariaDB 10.1.33
Fedora 25	Feb 2018	MariaDB 10.3.5, MariaDB 10.2.13, MariaDB 10.1.31
Ubuntu 17.04 "Zesty"	Jan 2018	MariaDB 10.3.4, MariaDB 10.2.12, MariaDB 10.1.30
openSUSE 42.2	Jan 2018	MariaDB 10.3.4, MariaDB 10.2.12, MariaDB 10.1.30
Red Hat Enterprise Linux 7.2	Nov 2017	MariaDB 10.3.3, MariaDB 10.2.12, MariaDB 10.1.30, MariaDB 10.0.33, MariaDB 5.5.58
CentOS 7.2	Nov 2017	MariaDB 10.3.3, MariaDB 10.2.12, MariaDB 10.1.30, MariaDB 10.0.33, MariaDB 5.5.58
Fedora 24	Aug 2017	MariaDB 10.2.8
Ubuntu 16.10 "Yakkety"	Jul 2017	MariaDB 10.2.7, MariaDB 10.1.26, MariaDB 10.0.32
Ubuntu 12.04 LTS "Precise"	Apr 2017	MariaDB 10.1.24, MariaDB 10.0.31, MariaDB 5.5.56
Mint 13 LTS "Maya"	Apr 2017	MariaDB 10.1.24, MariaDB 10.0.31, MariaDB 5.5.56
Red Hat Enterprise Linux 7.1	Mar 2017	MariaDB 10.1.24, MariaDB 10.0.31, MariaDB 5.5.56
CentOS 7.1	Mar 2017	MariaDB 10.1.24, MariaDB 10.0.31, MariaDB 5.5.56
Red Hat Enterprise Linux 5	Mar 2017	MariaDB 10.1.22, MariaDB 10.0.30, MariaDB 5.5.54
CentOS 5	Mar 2017	MariaDB 10.1.22, MariaDB 10.0.30, MariaDB 5.5.54
Fedora 23	Feb 2017	MariaDB 10.2.4, MariaDB 10.1.22, MariaDB 10.0.30
OpenSUSE 13	Jan 2017	MariaDB 10.2.4, MariaDB 10.1.22, MariaDB 10.0.30

Fedora 22	Aug 2016	MariaDB 10.1.17, MariaDB 10.0.27
Ubuntu 15.10 "Wily"	Jul 2016	MariaDB 10.1.16, MariaDB 10.0.26
Windows 2003 Server	Apr 2016	MariaDB 10.1.13, MariaDB 10.0.24, MariaDB 5.5.48
Windows XP	Apr 2016	MariaDB 10.1.13, MariaDB 10.0.24, MariaDB 5.5.48
Debian 6 "Squeeze"	Feb 2016	MariaDB 10.0.24, MariaDB 5.5.48
Ubuntu 15.04 "Vivid"	Jan 2016	MariaDB 10.1.11, MariaDB 10.0.24
Fedora 21	Dec 2015	MariaDB 10.1.10, MariaDB 10.0.23
Fedora 20	Oct 2015	MariaDB 10.0.22, MariaDB 5.5.46
Ubuntu 14.10 "Utopic"	Jul 2015	MariaDB 10.0.22, MariaDB 5.5.46
Ubuntu 10.04 LTS "Lucid"	Apr 2015	MariaDB 10.0.18, MariaDB 5.5.43
Mint 9 LTS "Isadora"	Apr 2015	MariaDB 10.0.18, MariaDB 5.5.43
Fedora 19	Apr 2015	MariaDB 10.0.18, MariaDB 5.5.43
FreeBSD 9.2	Sep 2014	
Ubuntu 13.10 "Saucy"	Jul 2014	MariaDB 10.0.14, MariaDB 5.5.40
Mint 16 "Petra"	Jul 2014	MariaDB 10.0.14, MariaDB 5.5.40
Ubuntu 12.10 "Quantal"	Apr 2014	MariaDB 10.0.11, MariaDB 5.5.37
Mint 14 "Nadia"	Apr 2014	MariaDB 10.0.11, MariaDB 5.5.37
Ubuntu 13.04 "Raring"	Jan 2014	MariaDB 10.0.8, MariaDB 5.5.35
Mint 15 "Olivia"	Jan 2014	MariaDB 10.0.8, MariaDB 5.5.35
Fedora 18	Dec 2013	MariaDB 10.0.8, MariaDB 5.5.35
Fedora 17	Aug 2013	MariaDB 10.0.5, MariaDB 5.5.34
Ubuntu 8.04 LTS "Hardy"	Apr 2013	MariaDB 10.0.2, MariaDB 5.5.31
Ubuntu 11.10 "Oneiric"	Apr 2013	MariaDB 10.0.2, MariaDB 5.5.31
Mint 12 "Lisa"	Apr 2013	MariaDB 10.0.2, MariaDB 5.5.31
Fedora 16	Feb 2013	MariaDB 10.0.1, MariaDB 5.5.29
Ubuntu 10.10 "Maverick"	Jan 2013	MariaDB 10.0.1, MariaDB 5.5.29
Ubuntu 11.04 "Natty"	Jan 2013	MariaDB 10.0.1, MariaDB 5.5.29
Debian 5 "Lenny"	Jan 2013	MariaDB 10.0.1, MariaDB 5.5.29
Debian 4 "Etch"		
Ubuntu 9.10 "Karmic"		
Ubuntu 9.04 "Jaunty"		
Ubuntu 8.10 "Intrepid"		

Support for Deprecated Platforms

If your chosen Linux Distribution or Operating System is deprecated, packages or support are not completely unavailable. The MariaDB Corporation provides support for all versions of MariaDB back to even very old MySQL versions. This includes packaged binaries. For specific dates related to each version and more details on the MariaDB Corporation's policies, see the Engineering Policies page.

2.1.2.14 Automated MariaDB Deployment and

Administration

It is possible to automate the deployment and administration of MariaDB servers and related technologies by using third-party software. This is especially useful when deploying and administering a large number of servers, but it also has benefits for small environments.

This section describes some automation technologies from MariaDB users perspective.

Why to Automate MariaDB Deployments and Management

The reasons to automate deployment and configuration of MariaDB.

A Comparison Between Automation Systems

A summary of the differences between automation systems, to help evaluating them.

Ansible and MariaDB

General information and hints on automating MariaDB deployments with Ansible.

Puppet and MariaDB

General information on how to automate MariaDB deployments and configuration with Puppet.

Vagrant and MariaDB

General information on how to setup development MariaDB servers with Vagrant.

MariaDB Containers

MariaDB containers and Docker Official Images.

Kubernetes and MariaDB

General information and tips on deploying MariaDB on Kubernetes.

Automating Upgrades with MariaDB.Org Downloads REST API

How to use MariaDB.Org Downloads APIs to automate upgrades.

HashiCorp Vault and MariaDB

An overview of secret management with Vault for MariaDB users.

Orchestrator Overview

Using Orchestrator to automate failover and replication operations.

Rotating Logs on Unix and Linux

Rotating logs on Unix and Linux with logrotate.

Automating MariaDB Tasks with Events

Using MariaDB events for automating tasks.

2.1.2.14.1 Why to Automate MariaDB Deployments and Management

MariaDB includes a powerful configuration system. This is enough when we need to deploy a single MariaDB instance, or a small number of instances. But many modern organisations have many database servers. Deploying and upgrading them manually could require too much time, and would be error-prone.

Contents

Infrastructure as Code

Several tools exist to deploy and manage several servers automatically. These tools operate at a higher level, and execute tasks like installing MariaDB, running queries, or generating new configuration files based on a template. Instead of upgrading servers manually, users can launch a command to upgrade a group of servers, and the automation software will run the necessary tasks.

Servers can be described in a code repository. This description can include MariaDB version, its configuration, users,

backup jobs, and so on. This code is human-readable, and can serve as a documentation of which servers exist and how they are configured. The code is typically versioned in a repository, to allow collaborative development and track the changes that occurred over time. This is a paradigm called **Infrastructure as Code**.

Automation code is high-level and one usually doesn't care how operations are implemented. Their implementation is delegated to modules that handle specific components of the infrastructure. For example a module could equally work with apt and yum package managers. Other modules can implement operations for a specific cloud vendor, so we declare we want a snapshot to be done, but we don't need to write the commands to make it happen. For special cases, it is of course possible to write Bash commands, or scripts in every language, and declare that they must be run.

Manual interventions on the servers will still be possible. This is useful, for example, to investigate performance problems. But it is important to leave the servers in the state that is described by the code.

This code is not something you write once and never touch again. It is periodically necessary to modify infrastructures to update some software, add new replicas, and so on. Once the base code is in place, making such changes is often trivial and potentially it can be done in minutes.

Automated Failover

Once replication is in place, two important aspects to automate are load balancing and failover.

Proxies can implement load balancing, redirecting the queries they receive to different server, trying to distribute the load equally. They can also monitor that MariaDB servers are running and in good health, thus avoiding sending queries to a server that is down or struggling.

However, this does not solve the problem with replication: if a primary server crashes, its replicas should point to another server. Usually this means that an existing replica is promoted to a master. This kind of changes are possible thanks to MariaDB GTID.

One can promote a replica to a primary by making change to existing automation code. This is typically simple and relatively quick to do for a human operator. But this operation takes time, and in the meanwhile the service could be down.

Automating failover will minimise the time to recover. A way to do it is to use Orchestrator, a tool that can automatically promote a replica to a primary. The choice of the replica to promote is done in a smart way, keeping into account things like the servers versions and the binary log format.

Resources

- Continuous configuration automation on Wikipedia.
- Infrastructure as code on Wikipedia.

Content initially contributed by Vettabase Ltd.

2.1.2.14.2 A Comparison Between Automation Systems

This page compares the automation systems that are covered by this section of the MariaDB Knowledge Base. More information about these systems are presented in the relevant pages, and more systems may be added in the future.

Contents

Code Structure Differences

Different automation systems provide different ways to describe our infrastructure. Understanding how they work is the first step to evaluate them and choose one for our organization.

Ansible Code Structure

Ansible code consists of the following components:

- An **inventory** determines which **hosts** Ansible should be able to deploy. Each host may belong to one or more **groups**. Groups may have **children**, forming a hierarchy. This is useful because it allows us to deploy on a group, or to assign variables to a group.
- A **role** describes the state that a host, or group of hosts, should reach after a deploy.
- A **play** associates hosts or groups to their roles. Each role/group can have more than one role.
- A role consists of a list of **tasks**. Despite its name a task is not necessarily something to do, but something that must exist in a certain state.
- Tasks can use **variables**. They can affect how a task is executed (for example a variable could be a file name), or even whether a task is executed or not. Variables exist at role, group or host level. Variables can also be passed by the user when a play is applied.
- **Playbooks** are the code that is used to define tasks and variables.
- **Facts** are data that Ansible retrieves from remote hosts before deploying. This is a very important step, because facts may determine which tasks are executed or how they are executed. Facts include, for example, the operating system family or its version. A playbook sees facts as pre-set variables.
- **Modules** implement **actions** that tasks can use. Action examples are **file** (to declare that files and directories must exist) or **mysql_variables** (to declare MySQL/MariaDB variables that need to be set).

See Ansible Overview - concepts for more details and an example.

Puppet Code Structure

Puppet code consists of the following components:

- An **inventory file** defines a set of **groups** and their **targets** (the members of a group). **plugins** can be used to retrieve groups and target dynamically, so they are equivalent to Ansible dynamic inventories.
- A **manifest** is a file that describes a configuration.
- A **resource** is a component that should run on a server. For example, "file" and "service" are existing support types.
- An **attribute** relates to a resource and affects the way it is applied. For example, a resource of type "file" can have attributes like "owner" and "mode".
- A **class** groups resources and variables, describing a logical part of server configuration. A class can be associated to several servers. A class is part of a manifest.
- A **module** is a set of manifests and describes an infrastructure or a part of it.
- Classes can have typed **parameters** that affect how they are applied.
- **Properties** are variables that are read from the remote server, and cannot be arbitrarily assigned.
- **Facts** are pre-set variables collected by Puppet before applying or compiling a manifest.

Architectural Differences

The architecture of the various systems is different. Their architectures determine how a deploy physically works, and what is needed to be able to deploy.

Ansible Architecture

Ansible architecture is simple. Ansible can run from any host, and can apply its playbooks on remote hosts. To do this, it runs commands via SSH. In practice, in most cases the commands will be run as superuser via `sudo`, though this is not always necessary.

Inventories can be dynamic. In this case, when we apply a playbook Ansible connects to remote services to discover hosts.

Ansible playbooks are applied via the `ansible-playbook` binary. Changes to playbooks are only applied when we perform this operation.

To recap, Ansible does not need to be installed on the server is administers. It needs an SSH access, and normally its user needs to be able to run `sudo`. It is also possible to configure a dynamic inventory, and a remote service to be used for this purpose.

Puppet Architecture

Puppet supports two types of architecture: agent-master or standalone. The agent-master architecture is recommended by Puppet Labs, and it is the most popular among Puppet users. For this reason, those who prefer a standalone architecture

tend to prefer Ansible.

Agent-Master Architecture

When this architecture is chosen, manifests are sent to the **Puppet master**. There can be more than one master, for high availability reasons. All target hosts run a **Puppet agent**. Normally this is a service that automatically starts at system boot. The agent contacts a master at a given interval. It sends facts, and uses them to compile a **catalog** from the manifests. A catalog is a description of what exactly an individual server should run. The agent receives the catalog and checks if there are differences between its current configuration and the catalog. If differences are found, the agent applies the relevant parts of the catalog.

An optional component is **PuppetDB**. This is a central place where some data are stored, including manifests, retrieved facts and logs. PuppetDB is based on PostgreSQL and there are no plans to support MariaDB or other DBMSs.

If a manual change is made to a remove server, it will likely be overwritten the next time Puppet agent runs. To avoid this, the Puppet agent service can be stopped.

Standalone Architecture

As mentioned, this architecture is not recommended by Puppet Labs nor popular amongst Puppet users. It is similar to Ansible architecture.

Users can apply manifests from any host with Puppet installed. This could be their laptop but, in order to emulate the behavior of an agent-master architecture, normally Puppet runs on a dedicated node as a cronjob. The **Puppet apply** application will require facts from remote hosts, it will compile a catalog for each host, will check which parts of it need to be applied, and will apply them remotely.

If a manual change is made to a remove server, it will be overwritten the next time Puppet apply runs. To avoid this, comment out any cron job running Puppet apply, or comment out the target server in the inventory.

Inventory

As mentioned, Puppet supports plugins to retrieve the inventory dynamically from remote services. In an agent-master architecture, one has to make sure that each target host has access to these services. In a standalone architecture, one has to make sure that the hosts running Puppet apply have access to these services.

Storing Secrets

Often our automation repositories need to contain secrets, like MariaDB user passwords or private keys for SSH authentication.

Both Ansible and Puppet support integration with secret stores, like Hashicorp Vault. For Puppet integration, see Integrations with secret stores.

In the simplest case, Ansible allows encrypting secrets in playbooks and decrypting them during execution using ansible-vault. This implies a minimal effort to handle secrets. However, it is not the most secure way to store secrets. The passwords to disclose certain secrets need to be shared with the users who have the right to use them. Also, brute force attacks are possible.

Ecosystems and Communities

Automation software communities are very important, because they make available a wide variety of modules to handle specific software.

Ansible Ecosystem

Ansible is open source, released under the terms of the GNU GPL. It is produced by RedHat. RedHat has a page about Red Hat Ansible Automation Platform Partners, who can provide support and consulting.

Ansible Galaxy is a big repository of Ansible roles produced by both the vendor and the community. Ansible comes with `ansible-galaxy`, a tool that can be used to create roles and upload them to Ansible Galaxy.

At the time of this writing, Ansible does not have specific MariaDB official modules. MySQL official modules can be used. However, be careful not try to use features that only apply to MySQL. There are several community-maintained MariaDB roles.

Puppet Ecosystem

Puppet is open source, released under the GNU GPL. It is produced by a homonym company. The page Puppet Partners

lists partners that can provide support and consulting about Puppet.

Puppet Forge is a big repository of modules produced by the vendor and by the community, as well as how-to guides.

Currently Puppet has many MariaDB modules.

2.1.2.14.3 Ansible and MariaDB

General information and hints on how to automate MariaDB deployments and configuration with Ansible.

Ansible is an open source tool to automate deployment, configuration and operations.

Ansible Overview for MariaDB Users

Overview of Ansible and how it works with MariaDB.

Deploying to Remote Servers with Ansible

How to invoke Ansible to run commands or apply roles on remote hosts.

Deploying Docker Containers with Ansible

How to deploy and manage Docker containers with Ansible.

Existing Ansible Modules and Roles for MariaDB

Links to existing Ansible modules and roles for MariaDB.

Installing MariaDB .deb Files with Ansible

How to install MariaDB from .deb files using Ansible.

Running mariadb-tzinfo-to-sql with Ansible

Updating the timezone tables with mariadb-tzinfo-to-sql using Ansible.

Managing Secrets in Ansible

How to store passwords as part of an Ansible repository.

2.1.2.14.3.1 Ansible Overview for MariaDB Users

Ansible is a tool to automate servers configuration management. It is produced by Red Hat and it is open source software released under the terms of the GNU GPL.

It is entirely possible to use Ansible to automate MariaDB deployments and configuration. This page contains generic information for MariaDB users who want to learn, or evaluate, Ansible.

For information about how to install Ansible, see Installing Ansible in Ansible documentation.

Contents

Automation Hubs

Normally, Ansible can run from any computer that has access to the target hosts to be automated. It is not uncommon that all members of a team has Ansible installed on their own laptop, and use it to deploy.

Red Hat offers a commercial version of Ansible called Ansible Tower. It consists of a REST API and a web-based interface that work as a hub that handles all normal Ansible operations.

An alternative is AWX. AWX is the open source upstream project from which many Ansible Tower features are originally developed. AWX is released under the terms of the Apache License 2.0. However, Red Hat does not recommend to run AWX in production.

AWX development is fast. It has several features that may or may not end up in Ansible Tower. Ansible Tower is more focused on making AWS features more robust, providing a stable tool to automate production environments.

Design Principles

Ansible allows us to write **playbooks** that describe how our servers should be configured. Playbooks are lists of **tasks**.

Tasks are usually **declarative**. You don't explain *how* to do something, you declare *what* should be done.

Playbooks are **idempotent**. When you apply a playbook, tasks are only run if necessary.

Here is a task example:

```
- name: Install Perl
  package:
    name: perl
    state: present
```

"Install Perl" is just a description that will appear on screen when the task is applied. Then we use the `package` module to declare that a package called "perl" should be installed. When we apply the playbook, if Perl is already installed nothing happens. Otherwise, Ansible installs it.

When we apply a playbook, the last information that appears on the screen is a recap like the following:

```
PLAY RECAP
*********************************************************************************************
****
mariadb-01        : ok=6    changed=2    unreachable=0    failed=0    skipped=0    rescued=0
ignored=0
```

This means that six tasks were already applied (so no action was taken), and two tasks were applied.

As the above example shows, Ansible playbooks are written in YAML.

Modules (like `package`) can be written in any language, as long as they are able to process a JSON input and produce a JSON output. However the Ansible community prefers to write them in Python, which is the language Ansible is written in.

Concepts

A piece of Ansible code that can be applied to a server is called a **playbook**.

A **task** is the smallest brick of code in a playbook. The name is a bit misleading, though, because an Ansible task should not be seen as "something to do". Instead, it is a minimal description of a component of a server. In the example above, we can see a task.

A task uses a single **module**, which is an interface that Ansible uses to interact with a specific system component. In the example, the module is "package".

A task also has attributes, that describe what should be done with that module, and how. In the example above, "name" and "state" are both attributes. The `state` attribute exists for every module, by convention (though there may be exceptions). Typically, it has at least the "present" and "absent" state, to indicate if an object should exist or not.

Other important code concepts are:

- An **inventory** determines which **hosts** Ansible should be able to deploy. Each host may belong to one or more **groups**. Groups may have **children**, forming a hierarchy. This is useful because it allows us to deploy on a group, or to assign variables to a group.
- A **role** describes the state that a host, or group of hosts, should reach after a deploy.
- A **play** associates hosts or groups to their roles. Each role/group can have more than one role.
- A role is a playbook that describes how certain servers should be configured, based on the logical role they have in the infrastructure. Servers can have multiple roles, for example the same server could have both the "mariadb" and the "mydumper" role, meaning that they run MariaDB and they have mydumper installed (as shown later).
- Tasks can use **variables**. They can affect how a task is executed (for example a variable could be a file name), or even whether a task is executed or not. Variables exist at role, group or host level. Variables can also be passed by the user when a play is applied.
- **Facts** are data that Ansible retrieves from remote hosts before deploying. This is a very important step, because facts may determine which tasks are executed or how they are executed. Facts include, for example, the operating system family or its version. A playbook sees facts as pre-set variables.
- **Modules** implement **actions** that tasks can use. Action examples are **file** (to declare that files and directories must exist) or **mysql_variables** (to declare MySQL/MariaDB variables that need to be set).

Example

Let's describe a hypothetical infrastructure to find out how these concepts can apply to MariaDB.

The **inventory** could define the following groups:

- "db-main" for the cluster used by our website. All nodes belong to this group.
- "db-analytics" for our replicas used by data analysts.
- "dump" for one or more servers that take dumps from the replicas.
- "proxysql" for one or more hosts that run ProxySQL.

Then we'll need the following nodes:

- "mariadb-node" for the nodes in "db-main". This role describes how to setup nodes of a cluster using Galera.
- "mariadb-replica" for the members of "db-analytics". It describes a running replica, and it includes the tasks that are necessary to provision the node if the data directory is empty when the playbook is applied. The hostname of the primary server is defined in a variable.
- "mariadb". The aforementioned "mariadb-node" and "mariadb-replica" can be children of this group. They have many things in common (filesystem for the data directory, some basic MariaDB configuration, some installed tools...), so it could make sense to avoid duplication and describe the common traits in a super-role.
- A "mariabackup" role to take backups with Mariabackup, running jobs during the night. We can associate this role to the "db-main" group, or we could create a child group for servers that will take the backups.
- "mariadb-dump" for the server that takes dumps with mariadb-dump. Note that we may decide to take dumps on a replica, so the same host may belong to "db-analytics" and "mariadb-dump".
- "proxysql" for the namesake group.

Architecture

Ansible architecture is extremely simple. Ansible can run on any host. To apply playbooks, it connects to the target hosts and runs system commands. By default the connection happens via ssh, though it is possible to develop connection plugins to use different methods. Applying playbooks locally without establishing a connection is also possible.

Modules can be written in any language, though Python is the most common choice in the Ansible community. Modules receive JSON "requests" and facts from Ansible core, they are supposed to run useful commands on a target host, and then they should return information in JSON. Their output informs Ansible whether something has changed on the remote server and if the operations succeeded.

Ansible is not centralized. It can run on any host, and it is common for a team to run it from several laptops. However, to simplify things and improve security, it may be desirable to run it from a dedicated host. Users will connect to that host, and apply Ansible playbooks.

Ansible Resources and References

- Ansible.com
- AWX
- Ansible Tower
- Ansible Galaxy
- Ansible on Wikipedia
- Ansible Automation Platform YouTube channel
- Ansible: Getting Started
- MariaDB Deployment and Management with Ansible (video)

Further information about the concepts discussed in this page can be found in Ansible documentation:

- Basic Concepts.
- Glossary.

Content initially contributed by Vettabase Ltd.

2.1.2.14.3.2 Deploying to Remote Servers with Ansible

If we manage several remote servers, running commands on them manually can be frustrating and time consuming. Ansible allows one to run commands on a whole group of servers.

This page shows some examples of ansible-playbook invocations. We'll see how to deploy roles or parts of them to remote servers. Then we'll see how to run commands on remote hosts, and possibly to get information from them. Make sure to read Ansible Overview first, to understand Ansible general concepts.

Contents

Pinging Remote Servers

Let's start with the simplest example: we just want our local Ansible to ping remote servers to see if they are reachable. Here's how to do it:

```
ansible -i production-mariadb all -m ping
```

Before proceeding with more useful examples, let's discuss this syntax.

- **ansible** is the executable we can call to run a command from remote servers.
- **-i production-mariadb** means that the servers must be read from an inventory called production-mariadb.
- **all** means that the command must be executed against all servers from the above inventory.
- **-m ping** specifies that we want to run the ping module. This is not the ping Linux command. It tells us if Ansible is able to connect a remote server and run a simple commands on them.

To run ping on a specific group or host, we can just replace "all" with a group name or host name from the inventory:

```
ansible -i production-mariadb main_cluster -m ping
```

Running Commands on Remote Servers

The previous examples show how to run an Ansible module on remote servers. But it's also possible to run custom commands over SSH. Here's how:

```
ansible -i production-mariadb all -a 'echo $PATH'
```

This command shows the value of `$PATH` on all servers in the inventory "production-mariadb".

We can also run commands as root by adding the `-b` (or `--become`) option:

```
# print a MariaDB variable
ansible -i production-mariadb all -b -a 'mysql -e "SHOW GLOBAL VARIABLES LIKE
\'innodb_buffer_pool_size\';"'

# reboot servers
ansible -i production-mariadb all -b -a 'reboot'
```

Applying Roles to Remote Servers

We saw how to run commands on remote hosts. Applying roles to remote hosts is not much harder, we just need to add some information. An example:

```
ansible-playbook -i production-mariadb production-mariadb.yml
```

Let's see what changed:

- **ansible-playbook** is the executable file that we need to call to apply playbooks and roles.
- **production-mariadb.yml** is the play that associates the servers listed in the inventory to their roles.

If we call ansible-playbook with no additional arguments, we will apply all applicable roles to all the servers mentioned in the play.

To only apply roles to certain servers, we can use the `-l` parameter to specify a group, an individual host, or a pattern:

```
# Apply to the mariadb-main role role
ansible-playbook -i production-mariadb -l mariadb-main production-mariadb.yml

# Apply to the mariadb-main-01 host
ansible-playbook -i production-mariadb -l mariadb-main-01 production-mariadb.yml

# Apply to multiple hosts whose name starts with "mariadb-main-"
ansible-playbook -i production-mariadb -l mariadb-main-* production-mariadb.yml
```

We can also apply tasks from roles selectively. Tasks may optionally have tags, and each tag corresponds to an operation that we may want to run on our remote hosts. For example, a "mariadb" role could have the "timezone-update" tag, to update the contents of the timezone tables. To only apply the tasks with the "timezone-update" tag, we can use this command:

```
ansible-playbook -i production-mariadb --tag timezone-update production-mariadb.yml
```

Using tags is especially useful for database servers. While most of the technologies typically managed by Ansible are stateless (web servers, load balancers, etc.) database servers are not. We must pay special attention not to run tasks that could cause a database server outage, for example destroying its data directory or restarting the service when it is not necessary.

Check mode

We should always test our playbooks and roles on test servers before applying them to production. However, if test servers and production servers are not exactly in the same state (which means, some facts may differ) it is still possible that applying roles will fail. If it fails in the initial stage, Ansible will not touch the remote hosts at all. But there are cases where Ansible could successfully apply some tasks, and fail to apply another task. After the first failure, ansible-playbook will show errors and exit. But this could leave a host in an inconsistent state.

Ansible has a *check mode* that is meant to greatly reduce the chances of a failure. When run in check mode, ansible-playbook will read the inventory, the play and roles; it will figure out which tasks need to be applied; then it will connect to target hosts, read facts, and value all the relevant variables. If all these steps succeed, it is unlikely that running ansible-playbook without check mode will fail.

To run ansible-playbook in check mode, just add the `--check` (or `-C`) parameter.

References

Further documentation can be found in the Ansible website:

- ansible tool.
- ansible-playbook tool.
- Validating tasks: check mode and diff mode.

Content initially contributed by Vettabase Ltd.

2.1.2.14.3.3 Deploying Docker Containers with Ansible

Ansible can be used to manage Docker container upgrades and configuration changes. Docker has native ways to do this, namely Dockerfiles and Docker Compose. But sometimes there are reasons to start basic containers from an image and then manage configuration with Ansible or similar software. See Benefits of Managing Docker Containers with Automation Software.

In this page we'll discuss how to use Ansible to manage Docker containers.

Contents

How to Deploy a Container with Ansible

Ansible has modules to manage the Docker server, Docker containers, and Docker Compose. These modules are maintained by the community.

A dynamic inventory plugin for Docker exists. It retrieves the list of existing containers from Docker.

Docker modules and the Docker inventory plugin communicate with Docker using its API. The connection to the API can use a TSL connection and supports key authenticity verification.

To communicate with Docker API, Ansible needs a proper Python module installed on the Ansible node (`docker` or `docker-py`).

Several roles exist to deploy Docker and configure it. They can be found in Ansible Galaxy.

References

Further information can be found in Ansible documentation.

- Docker Guide.
- docker_container module.

Content initially contributed by Vettabase Ltd.

2.1.2.14.3.4 Existing Ansible Modules and Roles for MariaDB

This page contains links to Ansible modules and roles that can be used to automate MariaDB deployment and configuration. The list is not meant to be exhaustive. Use it as a starting point, but then please do your own research.

Contents

Modules

At the time time of writing, there are no MariaDB-specific modules in Ansible Galaxy. MySQL modules can be used. Trying to use MySQL-specific features may result in errors or unexpected behavior. However, the same applies when trying to use a feature not supported by the MySQL version in use.

Currently, the MySQL collection in Ansible Galaxy contains at least the following modules:

- **mysql_db**: manages MySQL databases.
- **mysql_info**: gathers information about a MySQL server.
- **mysql_query**: runs SQL queries against MySQL.
- **mysql_replication**: configures and operates asynchronous replication.
- **mysql_user**: creates, modifies and deletes MySQL users.
- **mysql_variables**: manages MySQL configuration.

Note that some modules only exist as shortcuts, and it is possible to use mysql_query instead. However, it is important to notice that mysql_query is not idempotent. Ansible does not understand MySQL queries, therefore it cannot check whether a query needs to be run or not.

To install this collection locally:

```
ansible-galaxy collection install community.mysql
```

MariaDB Corporation maintains a ColumnStore playbook on GitHub.

Other Useful Modules

Let's see some other modules that are useful to manage MariaDB servers.

shell and command

Modules like shell and command allow one to run system commands.

To deploy on Windows, win_shell and win_command can be used.

Among other things, it is possible to use one of these modules to run MariaDB queries:

```
- name: Make the server read-only
  # become root to log into MariaDB with UNIX_SOCKET plugin
  become: yes
  shell: $( which mysql ) -e "SET GLOBAL read_only = 1;"
```

The main disadvantage with these modules is that they are not idempotent, because they're meant to run arbitrary system commands that Ansible can't understand. They are still useful in a variety of cases:

- To run queries, because mysql_query is also not idempotent.
- In cases when other modules do not allow us to use the exact arguments we need to use, we can achieve our goals by writing shell commands ourselves.
- To run custom scripts that implement non-trivial logic. Implementing complex logic in Ansible tasks is possible, but it can be tricky and inefficient.
- To call command-line tools. There may be specific roles for some of the most common tools, but most of the times using them is an unnecessary complication.

copy and template

An important part of configuration management is copying configuration files to remote servers.

The copy module allows us to copy files to target hosts. This is convenient for static files that we want to copy exactly as they are. An example task:

```
- name: Copy my.cnf
  copy:
    src: ./files/my.cnf.1
    dest: /etc/mysql/my.cnf
```

As you can see, the local name and the name on remote host don't need to match. This is convenient, because it makes it easy to use different configuration files with different servers. By default, files to copy are located in a `files` subdirectory in the role.

However, typically the content of a configuration file should vary based on the target host, the group and various variables. To do this, we can use the template module, which compiles and copies templates written in Jinja.

A simple template task:

```
- name: Compile and copy my.cnf
  copy:
    src: ./templates/my.cnf.j2
    dest: /etc/mysql/my.cnf
```

Again, the local and the remote names don't have to match. By default, Jinja templates are located in a `templates` subdirectory in the role, and by convention they have the `.j2` extension. This is because Ansible uses Jinja version 2 for templating, at the time writing.

A simple template example:

```
## WARNING: DO NOT EDIT THIS FILE MANUALLY !!
## IF YOU DO, THIS FILE WILL BE OVERWRITTEN BY ANSIBLE

[mysqld]
innodb_buffer_pool_size = {{ innodb_buffer_pool_size }}

{% if use_connect sameas true %}
connect_work_size = {{ connect_work_size }}
{% endif %}
```

Other Common Modules

The following modules are also often used for database servers:

- package, apt or yum. Package is package manager-agnostic. Use them to install, uninstall and upgrade packages.
- user, useful to create the system user and group that run MariaDB binary.
- file can be used to make sure that MariaDB directories (like the data directory) exist and have proper permissions. It can also be used to upload static files.

- template allows to create configuration files (like my.cnf) more dynamically, using the Jinja template language.
- service is useful after installing MariaDB as a service, to start it, restart it or stop it.

Roles

Specific roles exist for MariaDB in Ansible Galaxy. Using them for MariaDB is generally preferable, to be sure to avoid incompatibilities and to probably be able to use some MariaDB specific features. However, using MySQL or Percona Server roles is also possible. This probably makes sense for users who also administer MySQL and Percona Server instances.

To find roles that suits you, check Ansible Galaxy search page. Most roles are also available on GitHub.

You can also search roles using the ansible-galaxy tool:

```
ansible-galaxy search mariadb
```

2.1.2.14.3.5 Installing MariaDB .deb Files with Ansible

This page refers to the operations described in Installing MariaDB .deb Files. Refer to that page for a complete list and explanation of the tasks that should be performed.

Here we discuss how to automate such tasks using Ansible. For example, here we show how to install a package or how to import a GPG key; but for an updated list of the necessary packages and for the keyserver to use, you should refer to Installing MariaDB .deb Files.

Adding apt Repositories

To add a repository:

```
- name: Add specified repository into sources list
  ansible.builtin.apt_repository:
    repo: deb [arch=amd64,arm64,ppc64el]
http://sfo1.mirrors.digitalocean.com/mariadb/repo/10.3/ubuntu bionic main
    state: present
```

If you prefer to keep the repository information in a source list file in the Ansible repository, you can upload that file to the target hosts in this way:

```
- name: Create a symbolic link
  ansible.builtin.file:
    src: ./file/mariadb.list
    dest: /etc/apt/sources.list.d/
    owner: root
    group: root
    mod: 644
    state: file
```

Updating the Repository Cache

Both the Ansible modules ansible.builtin.apt and ansible.builtin.apt_repository have an `update_cache` attribute. In ansible.builtin.apt it is set to "no" by default. Whenever a task sets it to 'yes', `apt-get update` is run on the target system. You have three ways to make sure that repositories are updated.

The first is to use ansible.builtin.apt_repository to add the desired repository, as shown above. So you only need to worry about updating repositories if you use the file method.

The second is to make sure that `update_cache` is set to 'yes' when you install a repository:

```
- name: Install foo
  apt:
    name: foo
    update_cache: yes
```

But if you run certain tasks conditionally, this option may not be very convenient. So the third option is to update the

repository cache explicitly as a separate task:

```
- name: Update repositories
  apt:
    - update_cache: yes
```

Importing MariaDB GPG Key

To import the GPG key for MariaDB we can use the ansible.builtin.apt_key Ansible module. For example:

```
- name: Add an apt key by id from a keyserver
  ansible.builtin.apt_key:
    keyserver: hkp://keyserver.ubuntu.com:80
    id: 0xF1656F24C74CD1D8
```

Installing Packages

To install Deb packages into a system:

```
- name: Install software-properties-common
  apt:
    name: software-properties-common
    state: present
```

To make sure that a specific version is installed, performing an upgrade or a downgrade if necessary:

```
- name: Install foo 1.0
  apt:
    name: foo=1.0
```

To install a package or upgrade it to the latest version, use: `state: latest` .

To install multiple packages at once:

```
- name: Install the necessary packages
  apt:
    pkg:
    - pkg1
    - pkg2=1.0
```

If all your servers run on the same system, you will always use ansible.builtin.apt and the names and versions of the packages will be the same for all servers. But suppose you have some servers running systems from the Debian family, and others running systems from the Red Hat family. In this case, you may find convenient to use two different task files for two different types of systems. To include the proper file for the target host's system:

```
- include: mariadb-debian.yml
  when: "{{ ansible_facts['os_family'] }} == 'Debian'
```

The variables you can use to run the tasks related to the proper system are:

- ansible_fact['distribution']
- ansible_fact['distribution_major_version']
- ansible_fact['os_family']

There is also a system-independent package module, but if the package names depend on the target system using it may be of very little benefit.

2.1.2.14.3.6 Running mariadb-tzinfo-to-sql with Ansible

For documentation about the `mariadb-tzinfo-to-sql` utility, see mysql_tzinfo_to_sql. This page is about running it using Ansible.

Installing or Upgrading the Package

First, we should install `mariadb-tzinfo-to-sql` if it is available on our system. For example, to install it on Ubuntu, we can use this task. For other systems, use the proper module and package name.

```
- name: Update timezone info
  tags: [ timezone-update ]
  apt:
    name: tzdata
    state: latest
    install_recommends: no
  register: timezone_info
```

This task installs the latest version of the `tzdata`, unless it is already installed and up to date. We register the `timezone_info` variables, so we can only run the next task if the package was installed or updated.

We also specify a `timezone-update` tag, so we can apply the role to only update the timezone tables.

Running the Script

The next task runs `mariadb-tzinfo-to-sql`.

```
- name: Move system timezone info into MariaDB
  tags: [ timezone-update ]
  shell: >
    mysql_tzinfo_to_sql /usr/share/zoneinfo \
      | grep -v "^Warning" \
      | mysql --database=mysql
  when: timezone_info.changed
```

We use the `shell` module to run the command. Running a command in this way is not idempotent, so we specify `when: timezone_info.changed` to only run it when necessary. Some warnings may be generated, so we pipe the output of `mysql_tzinfo_to_sql` to `grep` to filter warnings out.

Using Galera

If we're using MariaDB Galera Cluster we'll want to only update the timezone tables in one node, because the other nodes will replicate the changes. For our convenience, we can run this operation on the first node. If the nodes hostnames are defined in a list called `cluster_hosts`, we can check if the current node is the first in this way:

```
  when: timezone_info.changed and inventory_hostname == cluster_hosts[0].hostname
```

Content initially contributed by Vettabase Ltd.

2.1.2.14.3.7 Managing Secrets in Ansible

An Ansible role often runs commands that require certain privileges, so it must perform some forms of login, using passwords or key pairs. In the context of database automation, we normally talk about: SSH access, sudo access, and access to MariaDB. If we write these secrets (passwords or private keys) in clear text in an Ansible repository, anyone who has access to the repository can access them, and this is not what we want.

Let's see how we can manage secrets.

Contents

The SSH Password or Keys

Most of the times, Ansible connects to the target hosts via SSH. It is common to use the system username and the SSH keys installed in `/.ssh`, which is the SSH clients default. In this case, nothing has to be done on the clients to be able to allow Ansible to use SSH, as long as they are already able to connect to the target hosts.

It is also possible to specify a different username as `ANSIBLE_REMOTE_USER` and an SSH configuration file as `ANSIBLE_NETCONF_SSH_CONFIG`. These settings can be specified in Ansible configuration file or as environment variables.

`ANSIBLE_ASK_PASS` can be specified. If this is the case, Ansible will prompt the user asking to type an SSH password.

Avoiding Sharing Secrets

As a general rule, any configuration that implies communicating sensible information to the persons who will connects to a system implies some degree of risk. Therefore, the most common choice is to allow users to login into remote systems with their local usernames, using SSH keys.

Once Ansible is able to connect remote hosts, it can also be used to install the public keys of some users to grant them access. Sharing these keys implies no risk. Sharing private keys is never necessary, and must be avoided.

MariaDB has a UNIX_SOCKET plugin that can be used to let some users avoid entering a password, as far as they're logged in the operating system. This authentication method is used by default for the root user. This is a good way to avoid having one more password and possibly writing to a `.my.cnf` file so that the user doesn't have to type it.

Even for users who connect remotely, it is normally not necessary to insert passwords in an Ansible file. When we create a user with a password, a hash of the original password is stored in MariaDB. That hash can be found in the mysql.user table. To know the hash of a password without even creating a user, we can use the PASSWORD() function:

```
SELECT PASSWORD('my_password12') AS hash;
```

When we create a user, we can actually specify a hash instead of the password the user will have to type:

```
CREATE USER user@host IDENTIFIED BY PASSWORD '*54958E764CE10E50764C2EECBB71D01F08549980';
```

ansible-vault

Even if you try to avoid sharing secrets, it's likely you'll have to keep some in Ansible. For example, MariaDB users that connect remotely have passwords, and if we want Ansible to create and manage those users, the hashes must be placed somewhere in our Ansible repository. While a hash cannot be converted back to a password, treating hashes as secrets is usually a good idea. Ansible provides a native way to handle secrets: ansible-vault.

In the simplest case, we can manage all our passwords with a single ansible-vault password. When we add or change a new password in some file (typically a file in `host_vars` or `group_vars`) we'll use ansible-vault to crypt this password. While doing so, we'll be asked to insert our ansible-vault password. When we apply a role and Ansible needs to decrypt this password, it will ask us to enter again our ansible-vault password.

ansible-vault can use more than one password. Each password can manage a different set of secrets. So, for example, some users may have the password to manage regular MariaDB users passwords, and only one may have the password that is needed to manage the root user.

Content initially contributed by Vettabase Ltd.

2.1.2.14.4 Puppet and MariaDB

General information and hints on how to automate MariaDB deployments and configuration with Puppet.

Puppet is an open source tool deployment, configuration and operations.

Puppet Overview for MariaDB Users

Overview of Puppet and how it works with MariaDB.

Bolt Examples

How to invoke Bolt to run commands or apply roles on remote hosts.

Puppet hiera Configuration System

Using hiera to handle Puppet configuration files.

Deploying Docker Containers with Puppet

How to deploy and manage Docker containers with Puppet.

2.1.2.14.4.1 Puppet Overview for MariaDB Users

Puppet is a tool to automate servers configuration management. It is produced by Puppet Inc, and released under the terms of the Apache License, version 2.

It is entirely possible to use Ansible to automate MariaDB deployments and configuration. This page contains generic information for MariaDB users who want to learn, or evaluate, Puppet.

Puppet modules can be searched using Puppet Forge. Most of them are also published on GitHub with open source licenses. Puppet Forge allows filtering modules to only view the most reliable: supported by Puppet, supported by a Puppet partner, or approved.

For information about installing Puppet, see Installing and upgrading in Puppet documentation.

Contents

Design Principles

With Puppet, you write **manifests** that describe the resources you need to run on certain servers and their **attributes**.

Therefore manifests are **declarative**. You don't write the steps to achieve the desired result. Instead, you describe the desired result. When Puppet detects differences between your description and the current state of a server, it decides what to do to fix those differences.

Manifests are also **idempotent**. You don't need to worry about the effects of applying a manifest twice. This may happen (see Architecture below) but it won't have any side effects.

Defining Resources

Here's an example of how to describe a resource in a manifest:

```
file { '/etc/motd':
  content => '',
  ensure => present,
}
```

This block describes a resource. The resource type is `file`, while the resource itself is `/etc/motd`. The description consists of a set of attributes. The most important is `ensure`, which in this case states that the file must exist. It is also common to use this resource to indicate that a file (probably created by a previous version of the manifest) doesn't exist.

These classes of resource types exist:

- **Built-in resources**, or **Puppet core resources**: Resources that are part of Puppet, maintained by the Puppet team.
- **Defined resources**: Resources that are defined as a combination of other resources. They are written in the Puppet domain-specific language.
- **Custom resources**: Resources that are written by users, in the Ruby language.

To obtain information about resources:

```
# list existing resource types
puppet resource --types
# print information about the file resource type
puppet describe file
```

To group several resources in a reusable class:

```
class ssh_server {
  file { '/etc/motd':
    content => '',
    ensure => present,
  }
  file { '/etc/issue.net':
    content => '',
    ensure => present,
  }
}
```

There are several ways to include a class. For example:

```
include Class['ssh_server']
```

Defining Nodes

Puppet has a **main manifest** that could be a `site.pp` file or a directory containing `.pp` files. For simple infrastructures, we can define the nodes here. For more complex infrastructures, we may prefer to import other files that define the nodes.

Nodes are defined in this way:

```
node 'maria-1.example.com' {
  include common
  include mariadb
}
```

The resource type is `node`. Then we specify a hostname that is used to match this node to an existing host. This can also be a list of hostnames, a regular expression that matches multiple nodes, or the `default` keyword that matches all hosts. To use a regular expression:

```
node /^(maria|mysql)-[1-3]\.example\.com$/ {
  include common
}
```

Concepts

The most important Puppet concepts are the following:

- **Target**: A host whose configuration is managed via Puppet.
- **Group**: A logical group of targets. For example there may be a `mariadb` group, and several targets may be part of this group.
- **Facts**: Information collected from the targets, like the system name or system version. They're collected by a Ruby gem called Facter. They can be core facts (collected by default) or custom facts (defined by the user).
- **Manifest**: A description that can be applied to a target.
- **Catalog**: A compiled manifest.
- **Apply**: Modifying the state of a target so that it reflects its description in a manifest.
- **Module**: A set of manifests.
- **Resource**: A minimal piece of description. A manifest consists of a piece of resources, which describe components of a system, like a file or a service.
- **Resource type**: Determines the class of a resource. For example there is a `file` resource type, and a manifest can contain any number of resources of this type, which describe different files.
- **Attribute**: It's a characteristic of a resource, like a file owner, or its mode.
- **Class**: A group of resources that can be reused in several manifests.

Architecture

Depending on how the user decides to deploy changes, Puppet can use two different architectures:

- An **Agent-master** architecture. This is the preferred way to use Puppet.
- A **standalone architecture**, that is similar to Ansible architecture.

Agent-master Architecture

A **Puppet master** stores a catalog for each target. There may be more than one Puppet master, for redundancy.

Each target runs a **Puppet agent** in background. Each Puppet agent periodically connects to the Puppet master, sending its facts. The Puppet master compiles the relevant manifest using the facts it receives, and send back a catalog. Note that it is also possible to store the catalogs in PuppetDB instead.

Once the Puppet agent receives the up-to-date catalog, it checks all resources and compares them with its current state. It applies the necessary changes to make sure that its state reflects the resources present in the catalog.

Standalone Architecture

With this architecture, the targets run **Puppet apply**. This application usually runs as a Linux cron job or a Windows scheduled task, but it can also be manually invoked by the user.

When Puppet apply runs, it compiles the latest versions of manifests using the local facts. Then it checks every resource from the resulting catalogs and compares it to the state of the local system, applying changes where needed.

Newly created or modified manifests are normally deployed to the targets, so Puppet apply can read them from the local host. However it is possible to use PuppetDB instead.

PuppetDB

PuppetDB is a Puppet node that runs a PostgreSQL database to store information that can be used by other nodes. PuppetDB can be used with both the Agent-master and the standalone architectures, but it is always optional. However it is necessary to use some advanced Puppet features.

PuppetDB stored the following information:

- The latest facts from each target.
- The latest catalogs, compiled by Puppet apply or a Puppet master.
- Optionally, the recent history of each node activities.

External Node Classifiers

With both architectures, it is possible to have a component called an External Node Classifier (ENC). This is a script or an executable written in any language that Puppet can call to determine the list of classes that should be applied to a certain target.

An ENC received a node name in input, and should return a list of classes, parameters, etc, as a YAML hash.

Bolt

Bolt can be used in both architectures to run operations against a target or a set of targets. These operations can be commands passed manually to Bolt, scripts, Puppet tasks or plans. Bolt directly connects to targets via ssh and runs system commands.

See Bolt Examples to get an idea of what you can do with Bolt.

hiera

hiera is a hierarchical configuration system that allows us to:

- Store configuration in separate files;
- Include the relevant configuration files for every server we automate with Puppet.

See Puppet hiera Configuration System for more information.

Puppet Resources

- Puppet documentation.
- forge.puppet.com.
- Puppet on GitHub.
- Puppet on Wikipedia.

More information about the topics discussed in this page can be found in the Ansible documentation:

- Puppet Glossary in Puppet documentation.
- Overview of Puppet's architecture in Puppet documentation.
- PuppetDB documentation.
- Classifying nodes in Puppet documentation.
- Hiera in Puppet documentation.
- Bolt documentation.

Content initially contributed by Vettabase Ltd.

2.1.2.14.4.2 Bolt Examples

Contents

This page shows some examples of what we can do with Bolt to administer a set of MariaDB servers. Bolt is a tool that is part of the Puppet ecosystem.

For information about installing Bolt, see Installing Bolt in Bolt documentation.

Inventory Files

The simplest way to call Bolt and instruct it to do something on some remote targets is the following:

```
bolt ... --nodes 100.100.100.100,200.200.200.200,300,300,300,300
```

However, for non-trivial setups it is usually better to use an inventory file. An example:

```
targets:
  - uri: maria-1.example.com
    name: maria_1
    alias: mariadb_main
  ...
```

In this way, it will be possible to refer the target by name or alias.

We can also define groups, followed by the group members. For example:

```
groups:
  - name: mariadb-staging
    targets:
        - uri: maria-1.example.com
        name: maria_1
        - uri: maria-2.example.com
        name: maria_2
  - name: mariadb-production
    targets:
      ...
...
```

With an inventory of this type, it will be possible to run Bolt actions against all the targets that are members of a group:

```
bolt ... --nodes mariadb-staging
```

In the examples in the rest of the page, the `--targets` parameter will be indicated in this way, for simplicity: `--targets`

<targets>.

Running Commands on Targets

The simplest way to run a command remotely is the following:

```
bolt command run 'mariadb-admin start-all-slaves' --targets <targets>
```

Copying Files

To copy a file or a whole directory to targets:

```
bolt file upload /path/to/source /path/to/destination --targets <targets>
```

To copy a file or a whole directory from the targets to the local host:

```
bolt file download /path/to/source /path/to/destination --targets <targets>
```

Running Scripts on Targets

We can use Bolt to run a local script on remote targets. Bolt will temporarily copy the script to the targets, run it, and delete it from the targets. This is convenient for scripts that are meant to only run once.

```
bolt script run rotate_logs.sh --targets <targets>
```

Running Tasks on Targets

Puppet tasks are not always as powerful as custom scripts, but they are simpler and many of them are idempotent. The following task stops MariaDB replication:

```
bolt task run mysql::sql --targets <targets> sql="STOP REPLICA"
```

Applying Puppet Code on Targets

It is also possible to apply whole manifests or portions of Puppet code (resources) on the targets.

To apply a manifest:

```
bolt apply manifests/server.pp  --targets <targets>
```

To apply a resource description:

```
bolt apply --execute "file { '/etc/mysql/my.cnf': ensure => present }" --targets <targets>
```

Bolt Resources and References

- Bolt documentation.
- Bolt on GitHub.

Further information about the concepts explained in this page can be found in Bolt documentation:

- Inventory Files in Bolt documentation.
- Applying Puppet code in Bolt documentation.

Content initially contributed by Vettabase Ltd.

2.1.2.14.4.3 Puppet hiera Configuration System

hiera is part of Puppet. It is a hierarchical configuration system that allows us to:

- Store configuration in separate files;
- Include the relevant configuration files for every server we automate with Puppet.

Contents

hiera Configuration Files

Each hierarchy allows to one choose the proper configuration file for a resource, based on certain criteria. For example criteria may include node names, node groups, operating systems, or datacenters. Hierarchies are defined in a `hiera.yaml` file, which also defines a path for the files in each hierarchy.

Puppet facts are commonly used to select the proper files to use. For example, a path may be defined as `"os/%{facts.os.name}.yaml"`. In this case, each resource will use a file named after the operating system it uses, in the os directory. You may need to use custom facts, for example to check which microservices will use a MariaDB server, or in which datacenter it runs.

We do not have to create a file for each possible value of a certain fact. We can define a default configuration file with settings that are reasonable for most resources. Other files, when included, will override some of the default settings.

A hiera configuration file will look like this:

```
version: 5
defaults:
  datadir: global
  data_hash: yaml_data

hierarchy:
  - name: "Node data"
    path: "nodes/%{trusted.certname}.yaml"

  - name: "OS data"
    path: "os/%{facts.os.family}.yaml"

  - name: "Per-datacenter business group data" # Uses custom facts.
    path: "location/%{facts.whereami}/%{facts.group}.yaml"
```

This file would include the global files, the OS-specific files and the node-specific files. Each hierarchy will override settings from previous hierarchies.

We can actually have several hiera configuration files. `hiera.yaml` is the global file. But we will typically have additional hiera configuration files for each environment. So we can include the configuration files that apply to production, staging, etc, plus global configuration files that should be included for every environment.

Importantly, we can also have hiera configuration files for each module. So, for example, a separate `mariadb/hiera.yaml` file may defined the hierarchies for MariaDB servers. This allow us to define, for example, different configuration files for MariaDB and for MaxScale, as most of the needed settings are typically different.

Configuration files

You probably noticed that, in the previous example, we defined `data_hash: yaml_data`, which indicates that configuration files are written in YAML. Other allowed formats are JSON and HOCON. The `data_hash` setting is defined in `defaults`, but it can be overridden by hierarchies.

Content initially contributed by Vettabase Ltd.

2.1.2.14.4.4 Deploying Docker Containers with Puppet

Puppet can also be used to manage Docker container upgrades and configuration changes. Docker has more specific tools for this purpose, but sometimes there are reasons to choose alternatives. See Benefits of Managing Docker Containers with Automation Software.

In this page you will find out what managing Docker with Puppet looks like. All the snippets in this page use the `docker` resource type, supported by the Puppet company.

Contents

How to Install, Upgrade or Uninstall Docker with Puppet

Installing or upgrading Docker is simple:

```
class { 'docker':
  use_upstream_package_source => false,
  version => '17.09.0~ce-0~debian',
}
```

In this example we are using our system's repositories instead of Docker official repositories, and we are specifying the desired version. To upgrade Docker later, all we need to do is to modify the version number. While specifying a version is not mandatory, it is a good idea because it makes our manifest more reproducible.

To uninstall Docker:

```
class { 'docker':
  ensure => absent
}
```

Check the `docker` resource type documentation to find out how to use more features: for example you can use Docker Enterprise Edition, or bind the Docker daemon to a TCP port.

How to Build or Pull Docker Images with Puppet

To pull an image from Dockerhub:

```
docker::image { 'mariadb:10.0': }
```

We specified the `10.0` tag to get the desired MariaDB version. If we don't, the image with the `latest` tag will be used. Note that this is not desirable in production, because it can lead to unexpected upgrades.

You can also write a Dockerfile yourself, and then build it to create a Docker image. To do so, you need to instruct Puppet to copy the Dockerfile to the target and then build it::

```
file { '/path/to/remote/Dockerfile':
  ensure => file,
  source => 'puppet:///path/to/local/Dockerfile',
}

docker::image { 'image_name':
  docker_file => '/path/to/remote/Dockerfile'
}
```

It is also possible to subscribe to Dockerfile changes, and automatically rebuild the image whenever a new file is found:

```
docker::image { 'image_name':
  docker_file => '/path/to/remote/Dockerfile'
  subscribe => File['/path/to/remote/Dockerfile'],
}
```

To remove an image that was possibly built or pulled:

```
docker::image { 'mariadb':
  ensure => absent
}
```

How to Deploy Containers with Puppet

To run a container:

```
docker::run { 'mariadb-01':
    image   => 'mariadb:10.5',
    ports   => ['3306:6606']
}
```

`mariadb-01` is the contained name. We specified the optional `10.5` tag, and we mapped the guest port 3306 to the host port 6606. In production, you normally don't map ports because you don't need to connect MariaDB clients from the host system to MariaDB servers in the containers. Third-party tools can be installed as separate containers.

References

- docker resource type documentation, in Puppet documentation.

Content initially contributed by Vettabase Ltd.

2.1.2.14.4.5 Existing Puppet Modules for MariaDB

This page contains links to Puppet modules that can be used to automate MariaDB deployment and configuration. The list is not meant to be exhaustive. Use it as a starting point, but then please do your own research.

Contents

Puppet Forge

Puppet Forge is the website to search for Puppet modules, maintained by the Puppet company. Modules are searched by the technology that needs to be automated, and the target operating system.

Search criteria include whether the modules are supported by Puppet or its partners, and whether a module is approved by Puppet. Approved modules are certified by Puppet based on their quality and maintenance standards.

Acceptance Tests

Some modules that support the Puppet Development Kit allow some types of acceptance tests.

We can run a static analysis on a module's source code to find certain bad practices that are likely to be a source of bugs:

```
pdk validate
```

If a module's authors wrote unit tests, we can run them in this way:

```
pdk test unit
```

Supported Modules for MariaDB

At the time of writing, there are no supported or approved modules for MariaDB.

However there is a mysql module supported by Puppet, that supports the Puppet Development Kit. Though it doesn't support MariaDB-specific features, it works with MariaDB. Its documentation shows how to use the module to install MariaDB on certain operating systems.

Several unsupported and not approved modules exist for MariaDB and MaxScale.

Resources and References

- Puppet Forge website.
- Puppet Development Kit documentation.

- Modules overview in Puppet documentation.
- Beginner's guide to writing modules in Puppet documentation.
- Puppet Supported Modules page in Puppet Forge.

Content initially contributed by Vettabase Ltd.

2.1.2.14.5 Vagrant and MariaDB

Vagrant is an open source tool to quickly setup machines that can be used for development and testing. These machines can be local virtual machines, Docker containers, AWS EC2 instances, and so on. Vagrant allows one to easily and quickly setup test MariaDB servers.

Vagrant Overview for MariaDB Users

Vagrant architecture, general concepts and basic usage.

Creating a Vagrantfile

How to create a new Vagrant box running MariaDB.

Vagrant Security Concerns

Security matters related to Vagrant machines.

Running MariaDB ColumnStore containers on Linux, Windows and MacOS

The ColumnStore container allows for a simple setup of a ColumnStore single...

2.1.2.14.5.1 Vagrant Overview for MariaDB Users

Vagrant is a tool to create and manage development machines (Vagrant *boxes*). They are usually virtual machines on the localhost system, but they could also be Docker containers or remote machines. Vagrant is open source software maintained by HashiCorp and released under the MIT license.

Vagrant benefits include simplicity, and a system to create test boxes that is mostly independent from the technology used.

For information about installing Vagrant, see Installation in Vagrant documentation.

In this page we discuss basic Vagrant concepts.

Contents

Vagrant Concepts

A **Vagrant machine** is compiled from a box. It can be a virtual machine, a container or a remote server from a cloud service.

A **box** is a package that can be used to create Vagrant machines. We can download boxes from app.vagrantup.com, or we can build a new box from a Vagrantfile. A box can be used as a base for another box. The base boxes are usually operating system boxes downloaded from app.vagrantup.com.

A **provider** is responsible for providing the virtualization technology that will run our machine.

A **provisioner** is responsible for installing and configuring the necessary software on a newly created Vagrant machine.

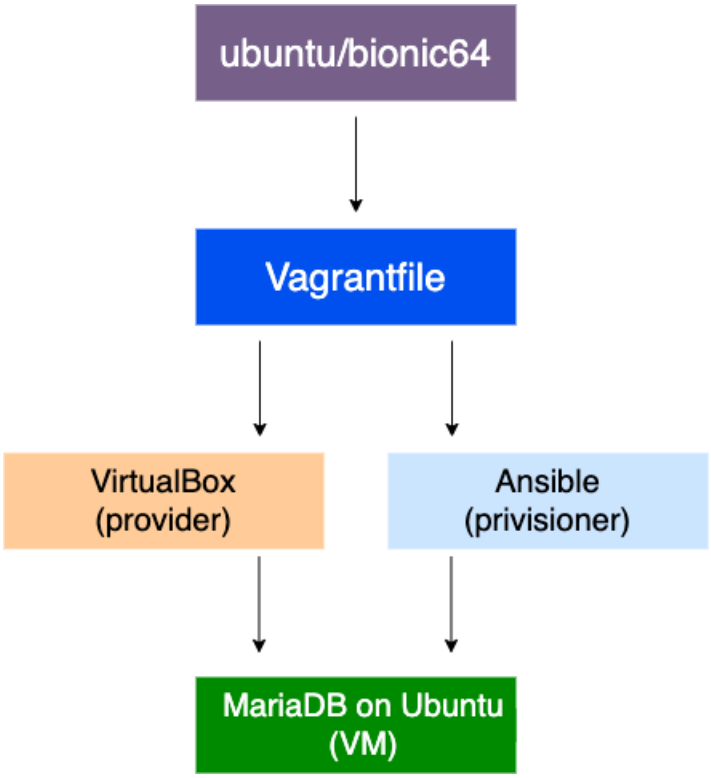
Example

The above concepts are probably easier to understand with an example.

We can use an Ubuntu box as a base to build a Vagrant machine with MariaDB. So we write a Vagrantfile for this purpose.

In the Vagrantfile we specify VirtualBox as a provider. And we use the Ansible provisioner to install and configure MariaDB. Once we finish this Vagrantfile, we can run a Vagrant command to start a Vagrant machine, which is actually a VirtualBox VM running MariaDB on Ubuntu.

The following diagram should make the example clear:



Vagrantfiles

A Vagrantfile is a file that describes how to create one or more Vagrant machines. Vagrantfiles use the Ruby language, as well as objects provided by Vagrant itself.

A Vagrantfile is often based on a box, which is usually an operating system in which we are going to install our software. For example, one can create a MariaDB Vagrantfile based on the `ubuntu/trusty64` box. A Vagrantfile can describe a box with a single server, like MariaDB, but it can also contain a whole environment, like LAMP. For most practical use cases, having the whole environment in a single box is more convenient.

Boxes can be searched in Vagrant Cloud. Most of their Vagrantfiles are available on GitHub. Searches can be made, among other things, by keyword to find a specific technology, and by provider.

Providers

A provider adds support for creating a specific type of machines. Vagrant comes with several providers, for example:

- `VirtualBox` allows one to create virtual machines with VirtualBox.
- `Microsoft-Hyper-V` allows one to create virtual machines with Microsoft Hyper-V.
- Docker allows one to create Docker containers. On non-Linux systems, Vagrant will create a VM to run Docker.

Alternative providers are maintained by third parties or sold by HashiCorp. They allow one to create different types of machines, for example using VMWare.

Some examples of useful providers, recognized by the community:

- Vagrant AWS Provider.
- Vagrant Google Compute Engine (GCE) Provider.
- Vagrant Azure Provider.
- OpenVZ.
- vagrant-lxc.

If you need to create machines with different technologies, or deploy them to unsupported cloud platforms, you can develop a custom provider in Ruby language. To find out how, see Plugin Development: Providers in Vagrant documentation. The Vagrant AWS Provider was initially written as an example provider.

Provisioners

A provisioner is a technology used to deploy software to the newly created machines.

The simplest provisioner is `shell`, which runs a shell file inside the Vagrant machine. `powershell` is also available.

Other providers use automation software to provision the machine. There are provisioners that allow one to use Ansible, Puppet, Chef or Salt. Where relevant, there are different provisioners allowing the use of these technologies in a distributed way (for example, using Puppet apply) or in a centralized way (for example, using a Puppet server).

It is interesting to note that there is both a Docker provider and a Docker provisioner. This means that a Vagrant machine can be a Docker container, thanks to the `docker` provisioner. Or it could be any virtualisation technology with Docker running in it, thanks to the `docker` provisioner. In this case, Docker pulls images and starts containers to run the software that should be running in the Vagrant machine.

If you need to use an unsupported provisioning method, you can develop a custom provisioner in Ruby language. See Plugin Development: Provisioners in Vagrant documentation.

Plugins

It is possible to install a plugin with this command:

```
vagrant plugin install <plugin_name>
```

A Vagrantfile can require that a plugin is installed in this way:

```
require 'plugin_name'
```

A plugin can be a Vagrant plugin or a Ruby gem installable from rubygems.org. It is possible to install a plugin that only exists locally by specifying its path.

Changes in Vagrant 3.0

HashiCorp published an article that describes its plans for Vagrant 3.0.

Vagrant will switch to a client-server architecture. Most of the logic will be stored in the server, while the development machines will run a thin client that communicates with the server. It will be possible to store the configuration in a central database.

Another notable change is that Vagrant is switching from Ruby to Go. For some time, it will still be possible to use Vagrantfiles and plugins written in Ruby. However, in the future Vagrantfiles and plugins should be written in one of the languages that support gRPC (not necessarily Go). Vagrantfiles can also be written in HCL, HashiCorp Configuration Language.

Vagrant Commands

This is a list of the most common Vagrant commands. For a complete list, see Command-Line Interface in Vagrant documentation.

To list the available machines:

```
vagrant box list
```

To start a machine from a box:

```
cd /box/directory
vagrant up
```

To connect to a machine:

```
vagrant ssh
```

To see all machines status and their id:

```
vagrant global-status
```

To destroy a machine:

```
vagrant destroy <id>
```

Vagrant Resources and References

Here are some valuable websites and pages for Vagrant users.

- Vagrant Up.
- app.vagrantup.com.
- Vagrant Community.
- Vagrant on Wikipedia.
- Vagrant on HashiCorp Learn.

Content initially contributed by Vettabase Ltd.

2.1.2.14.5.2 Creating a Vagrantfile

In this page we discuss how to create a Vagrantfile, which you can use to create new boxes or machines. This content is specifically written to address the needs of MariaDB users.

Contents

A Basic Vagrantfile

A Vagrantfile is a Ruby file that instructs Vagrant to create, depending on how it is executed, new Vagrant machines or boxes. You can see a box as a compiled Vagrantfile. It describes a type of Vagrant machines. From a box, we can create new Vagrant machines. However, while a box is easy to distribute to a team or to a wider public, a Vagrantfile can also directly create one or more Vagrant machines, without generating any box.

Here is a simple Vagrantfile example:

```
Vagrant.configure("2") do |config|
  config.vm.box = "hashicorp/bionic64"
  config.vm.provider "virtualbox"
  config.vm.provision :shell, path: "bootstrap.sh"
end
```

`Vagrant.configure("2")` returns the Vagrant configuration object for the new box. In the block, we'll use the `config` alias to refer this object. We are going to use version 2 of Vagrant API.

`vm.box` is the base box that we are going to use. It is Ubuntu BionicBeaver (18.04 LTS), 64-bit version, provided by HashiCorp. The schema for box names is simple: the maintainer account in Vagrant Cloud followed by the box name.

We use `vm.provision` to specify the name of the file that is going to be executed at the machine creation, to provision the machine. `bootstrap.sh` is the conventional name used in most cases.

To create new Vagrant machines from the Vagrantfile, move to the directory that contains the Vagrant project and run:

```
vagrant up
```

To compile the Vagrantfile into a box:

```
vagrant package
```

These operations can take time. To preventively check if the Vagrantfile contains syntax errors or certain types of bugs:

```
vagrant validate
```

Providers

A provider allows Vagrant to create a Vagrant machine using a certain technology. Different providers may enable a virtual machine manager (VirtualBox, VMWare, Hyper-V...), a container manager (Docker), or remote cloud hosts (AWS, Google Compute Engine...).

Some providers are developed by third parties. app.vagrant.com supports search for boxes that support the most important third parties providers. To find out how to develop a new provider, see Plugin Development: Providers.

Provider options can be specified. Options affect the type of Vagrant machine that is created, like the number of virtual CPUs. Different providers support different options.

It is possible to specify multiple providers. In this case, Vagrant will try to use them in the order they appear in the Vagrantfile. It will try the first provider; if it is not available it will try the second; and so on.

Here is an example of providers usage:

```
Vagrant.configure("2") do |config|
    config.vm.box = "hashicorp/bionic64"
    config.vm.provider "virtualbox" do |vb|
        vb.customize ["modifyvm", :id, "--memory", 1024 * 4]
    end
    config.vm.provider "vmware_fusion"
end
```

In this example, we try to use VirtualBox to create a virtual machine. We specify that this machine must have 4G of RAM (1024M * 4). If VirtualBox is not available, Vagrant will try to use VMWare.

This mechanism is useful for at least a couple of reasons:

- Different users may use different systems, and maybe they don't have the same virtualization technologies installed.
- We can gradually move from one provider to another. For a period of time, some users will have the new virtualization technology installed, and they will use it; other users will only have the old technology installed, but they will still be able to create machines with Vagrant.

Provisioners

We can use different methods for provisioning. The simplest provisioner is `shell`, that allows one to run a Bash file to provision a machine. Other provisioners allow setting up the machines using automation software, including Ansible, Puppet, Chef and Salt.

To find out how to develop a new provisioner, see Plugin Development: Provisioners.

The `shell` Provisioner

In the example above, the shell provisioner runs boostrap.sh inside the Vagrant machine to provision it. A simple bootstrap.sh may look like the following:

```
#!/bin/bash

apt-get update
apt-get install -y
```

To find out the steps to install MariaDB on your system of choice, see the Getting, Installing, and Upgrading MariaDB section.

You may also want to restore a database backup in the new Vagrant machine. In this way, you can have the database needed by the application you are developing. To find out how to do it, see Backup and Restore Overview. The most flexible type of backup (meaning that it works between different MariaDB versions, and in some cases even between MariaDB and different DBMSs) is a dump.

On Linux machines, the `shell` provisioner uses the default shell. On Windows machines, it uses PowerShell.

Uploading Files

If we use the `shell` provisioner, we need a way to upload files to the new machine when it is created. We could use the

`file` provisioner, but it works by connecting the machine via ssh, and the default user doesn't have permissions for any directory except for the synced folders. We could change the target directory owner, or we could add the default user to a group with the necessary privileges, but these are not considered good practices.

Instead, we can just put the file we need to upload somewhere in the synced folder, and then copy it with a shell command:

```
cp ./files/my.cnf /etc/mysql/conf.d/
```

Provisioning Vagrant with Ansible

Here is an example of how to provision a Vagrant machine or box by running Ansible:

```
Vagrant.configure("2") do |config|
  ...
  config.vm.provision "ansible" do |ansible|
    ansible.playbook = "vagrant.yml"
  end
end
```

With the Ansible provisioner, Ansible runs in the host system and applies a playbook in the guest system. In this example, it runs a playbook called `vagrant.yml`. The Ansible Local provisioner runs the playbook in the vagrant machine.

For more information, see Using Vagrant and Ansible in the Ansible documentation. For an introduction to Ansible for MariaDB users, see Ansible and MariaDB.

Provisioning Vagrant with Puppet

To provision a Vagrant machine or box by running Puppet:

```
Vagrant.configure("2") do |config|
  ...
  config.vm.provision "puppet" do |puppet|
    puppet.manifests_path = "manifests"
    puppet.manifest_file = "default.pp"
  end
end
```

In this example, Puppet Apply runs in the host system and no Puppet Server is needed. Puppet expects to find a `manifests` directory in the project directory. It expects it to contain `default.pp`, which will be used as an entry point. Note that `puppet.manifests_path` and `puppet.manifest_file` are set to their default values.

Puppet needs to be installed in the guest machine.

To use a Puppet server, the `puppet_server` provisioner can be used:

```
Vagrant.configure("2") do |config|
  ...
  config.vm.provision "puppet_server" do |puppet|
    puppet.puppet_server = "puppet.example.com"
  end
end
```

See the Puppet Apply provisioner and the Puppet Agent Provisioner.

For an introduction to Puppet for MariaDB users, see Puppet and MariaDB.

Sharing Files Between the Host and a Guest System

To restore a backup into MariaDB, in most cases we need to be able to copy it from the host system to the box. We may also want to occasionally copy MariaDB logs from the box to the host system, to be able to investigate problems.

The project directory (the one that contains the Vagrantfile) by default is shared with the virtual machine and mapped to the `/vagrant` directory (the synced folder). It is a good practice to put there all files that should be shared with the box when it is started. Those files should normally be versioned.

The synced folder can be changed. In the above example, we could simply add one line:

```
config.vm.synced_folder "/host/path", "/guest/path"
```

The synced folder can also be disabled:

```
config.vm.synced_folder '.', '/vagrant', disabled: true
```

Note that multiple Vagrant machines may have synced folders that point to the same directory on the host system. This can be useful in some cases, if you prefer to test some functionalities quickly, rather that replicating production environment as faithfully as possible. For example, to test if you're able to take a backup from one machine and restore it to another, you can store the backup in a common directory.

Network Communications

It is often desirable for a machine to be able to communicate with "the outside". This can be done in several ways:

- Private networks;
- Public networks;
- Exposing ports to the host.

Remembers that Vagrant doesn't create machines, but it asks a provisioner to create machines. Some provisioners support all of these communication methods, others may support some of them, or even none of them. When you create a Vagrantfile that starts machines using one of these features, it is implicit that this can only happen if the provisioner you are using supports the features you need. Check your provisioner documentation to find out which features it supports.

The default provisioner, VirtualBox, supports all these communication methods, including multiple networks.

Private Networks

A private network is a networks that can only be accesses by machines that run on the same host. Usually this also means that the machines must run on the same provisioner (for example, they all must be VirtualBox virtual machines).

Some provisioners support multiple private networks. This means that every network has a different name and can be accessed by different machines.

The following line shows how to create or join a private network called "example", where this machine's IP is assigned by the provisioner via DHCP:

```
config.vm.network 'private_network', name: 'example', type: 'dhcp'
```

While this is very convenient to avoid IP conflicts, sometimes you prefer to assign some IP's manually, in this way:

```
config.vm.network 'private_network', name: 'example', ip: '111.222.111.222'
```

Public Networks

As explained above, public networks are networks that can be accessed by machines that don't run on the same host with the same provider.

To let a machine join a public network:

```
# use provisioner DHCP:
config.vm.network "public_network", use_dhcp_assigned_default_route: true

# assign ip manually:
config.vm.network "public_network", ip: "111.222.111.222"
```

To improve security, you may want to configure a gateway:

```
config.vm.provision "shell", run: "always", inline: "route add default gw 111.222.111.222"
```

Exposing Ports

Vagrant allows us to map a TCP or UDP port in a guest system to a TCP or UDP port in the host system. For example, you can map a virtual machine port 3306 to the host port 12345. Then you can connect MariaDB in this way:

```
mariadb -hlocalhost -P12345 -u<user> -p<password>
```

You are not required to map a port to a port with a different number. In the above example, if the port 3306 in your host is

not in use, you are free to map the guest port 3306 to the host port 3306.

There are a couple of caveats:

- You can't map a single host port to multiple guest ports. If you want to expose the port 3306 from multiple Vagrant machines, you'll have to map them to different host ports. When running many machines this can be hard to maintain.
- Ports with numbers below 1024 are privileged ports. Mapping privileged ports requires root privileges.

To expose a port:

```
config.vm.network 'forwarded_port', guest: 3306, host: 3306
```

Use Cases

Suppose you run MariaDB and an application server in two separate Vagrant machines. It's usually best to let them communicate via a private network, because this greatly increases your security. The application server will still need to expose ports to the host, so the application can be tested with a web browser.

Suppose you have multiple environments of the same type, like the one described above. They run different applications that don't communicate with each other. In this case, if your provisioner supports this, you will run multiple private networks. You will need to expose the applications servers ports, mapping them to different host ports.

You may even want to implement different private networks to create an environment that reflects production complexity. Maybe in production you have a cluster of three MariaDB servers, and the application servers communicate with them via a proxy layer (ProxySQL, HAProxy, or MaxScale). So the applications can communicate with the proxies, but have no way to reach MariaDB directly. So there is a private network called "database" that can be accessed by the MariaDB servers and the proxy servers, and another private network called "application" that can be accessed by the proxy servers and the application servers. This requires that your provisioner supports multiple private networks.

Using public networks instead of private one will allow VMs that run on different hosts to be part of your topology. In general this is considered as an insecure practice, so you should probably ask yourself if you really need to do this.

References

The vagrant-mariadb-examples repository is an example of a Vagrantfile that creates a box containing MariaDB and some useful tools for developers.

Further information can be found in Vagrant documentation.

- Vagrantfile.
- Providers.
- Synced Folders.
- Ansible Provisioner.
- Puppet Apply Provisioner.
- Puppet Agent Provisioner.

See also Ruby documentation.

Content initially contributed by Vettabase Ltd.

2.1.2.14.5.3 Vagrant Security Concerns

Databases typically contain information to which access should be restricted. For this reason, it's worth discussing some security concerns that Vagrant users should be aware of.

Contents

Access to the Vagrant Machine

By default, Vagrant machines are only accessible from the localhost. SSH access uses randomly generated key pairs, and therefore it is secure.

The password for `root` and `vagrant` is "vagrant" by default. Consider changing it.

Synced Folders

By default, the project folder in the host system is shared with the machine, which sees it as `/vagrant`. This means that whoever has access to the project folder also has read and write access to the synced folder. If this is a problem, make sure to properly restrict the access to the synced folder.

If we need to exchange files between the host system and the Vagrant machine, it is not advisable to disable the synced folder. This is because the only alternative is to use the `file` provider, which works by copying files to the machine via ssh. The problem is that the default ssh user does not have permissions to write to any directory by default, and changing this would be less secure than using a synced folder.

When a machine is provisioned, it should read the needed files from the synced folder or copy them to other places. Files in the synced folder should not be accessed by the Vagrant machine during its normal activities. For example, it is fine to load a dump from the synced folder during provisioning; and it is fine to copy configuration files from the synced folder to directories in `/etc` during provisioning. But it is a bad practice to let MariaDB use table files located in the synced folder.

Reporting Security Bugs

Note that security bugs are not reported as normal bugs. Information about security bugs are not public. See Security at HashiCorp for details.

Content initially contributed by Vettabase Ltd.

2.1.2.14.5.4 Running MariaDB ColumnStore containers on Linux, Windows and MacOS

Contents

Introduction

The ColumnStore container allows for a simple and lightweight setup of a MariaDB ColumnStore single server instance for evaluation purposes. The configuration is designed for simplified developer / evaluation setup rather than production use. It allows to evaluate ColumnStore on a Windows or MacOS system, setting up a Linux system in a container. The image uses a base OS of RockyLinux.

Windows Linux Subsystem

If you have Windows 10 Creators update installed, then you can install the Ubuntu installation into the Bash console. Please follow the Ubuntu instructions in getting started. If you have recently upgraded and had Bash installed previously, ensure you uninstall and reinstall Bash first to have a clean Ubuntu installation. Note that ColumnStore will be terminated should you terminate the Bash console.

MariaDB Containers

MariaDB Containers manages lightweight containers that allows for creation of lightweight and reproducible containers with a dedicated function. On Windows and MacOS systems, Docker Engine transparently runs on a Linux virtual machine.

Since MariaDB ColumnStore relies on a Syslog daemon, the container must start both ColumnStore and rsyslogd and the runit utility is used to achieve this.

A single node image can be found at MariaDB on Docker Hub.

```
docker run -d  --name mcs mariadb/columnstore
docker exec -it mcs bash
```

A ColumnStore cluster can be brought up using a compose file provided in the ColumnStore github repository:

```
git clone https://github.com/mariadb-corporation/mariadb-columnstore-docker.git
cd mariadb-columnstore-docker/columnstore
docker-compose up -d
```

For more information about how to manage containers, see Installing and Using MariaDB via Docker.

To test an application that uses ColumnStore, it is desirable to setup several containers that will communicate with each other. To do this, we can use Docker Compose. See Setting Up a LAMP Stack with Docker Compose for more information.

2.1.2.14.6 MariaDB Containers

Containers are an OCI standard format for software images and their specified time all bundled up into a single distributable time. They can be used for production, development or testing.

Docker Inc. run a Docker Official Images program to provide users with an essential base implementation of MariaDB in a container and to exemplify best practices of a container.

The containers are available on Docker Hub as **docker.io/library/mariadb** though many container runtime implementation will fill in the **docker.io/library** where the host/path isn't specified.

The containers are in a Open Container Initiative format that allows the containers to be interoperable with a number of container runtime implementations. Docker, or more fully Docker Engine, is just one of the many available runtimes.

Many people use MariaDB Docker Official Image containers in CI systems like GitHub Actions, though its possible to use these in production environments like kubernetes.

The MariaDB Server container images are available with a number of tags:

- A full version, like 10.11.5
- A major version like 10.11
- The most recent stable GA version - **latest**
- The most recent stable LTS version - **lts**

Versions that aren't stable will be suffixed with **-rc**, or **-alpha** to clearly show their release status, and enables Renovatebot and other that follow semantic versioning to follow updates.

For a consistent application between testing an production environment using the SHA hash of the image is recommended like **docker.io/library/mariadb@sha256:29fe5062baf36bae8ec68f21a3dce4f0372dadc185e687624f1252fc49d91c67**. There is a list of mapping and history of tags to SHA hash on the Docker Library repository.

Benefits of Managing MariaDB Containers with Orchestration Software

Benefits of managing MariaDB Containers with Automation Software.

Installing and Using MariaDB via Docker

Creating and managing a MariaDB Docker container.

Container Backup and Restoration

Backup and Restore for the MariaDB Docker Official Image

Container Security Concerns

Security matters related to containers.

Adding Plugins to the MariaDB Docker Official Image

Summary of methods to install plugins in the MariaDB Docker Library Container.

Setting Up a LAMP Stack with Docker Compose

How to use Docker Compose to set up containers running a LAMP stack.

Creating a Custom Container Image

How to write a Dockerfile to create custom images.

MariaDB Server Docker Official Image Environment Variables

Environment variables can be passed on the docker run command line.

Running MariaDB ColumnStore containers on Linux, Windows and MacOS

The ColumnStore container allows for a simple setup of a ColumnStore single...

Docker Official Image Frequently Asked Questions
Frequently asked questions about the Docker Official Image.

MariaDB Container Cheat Sheet
Common commands when using MariaDB containers.

Using Healthcheck.sh
healthcheck.sh

Docker and AWS EC2
This process shows how to deploy, connect to, and create MariaDB database i...

Docker and Google Cloud
This process shows how to deploy, connect to, and create MariaDB database i...

Docker and Microsoft Azure
This process shows how to deploy, connect to, and create MariaDB database i...

There are 12 related questions.

2.1.2.14.6.1 Benefits of Managing MariaDB Containers with Orchestration Software

In this page we'll discuss why automating containers with software like Ansible or Puppet may be desirable in some cases. To talk about this, we'll first need to discuss why containers are defined *ephemeral*, and how this applies to containerized database servers (particularly MariaDB).

During the discussion, we should keep in mind that Docker Engine, CRI-I, containerd, Mirantis Container Runtime, Podman and other OCI container runtimes can be used to setup production and/or development environments. These use cases are very different from a database perspective: a production database may be big, and typically contains data that we don't want to lose. Development environments usually contain small sample data that can be rebuilt relatively quickly. This page focuses on the latter case.

Container Ephemeral Nature

Images are an OCI specified format that can be compiled from Dockerfiles as one of the ways. Containers are the OCI runtime specified way of creating a runtime version of an images. Normally, a container is not modified from the moment it is created. In other words, containers are usually designed to be **ephemeral**, meaning that they can be destroyed and replaced with new containers at any time. Provided that there is proper redundancy (for example, there are several web servers running the same services) destroying one container and starting a new one of the same type won't cause any damage.

We will discuss a bit later how this applies to MariaDB, and more generally to database servers.

When something should change, for example some software version or configuration, normally Dockerfiles are updated and containers are recreated from the latest image versions. For this reason, containers shouldn't contain anything that shouldn't be lost, and recreating them should be an extremely cheap operation. **Docker Compose** or the **Swarm mode** are used to declare which containers form a certain environment, and how they communicate with each other.

On the contrary, Ansible and Puppet are mainly built to manage the configuration of existing servers. It doesn't recreate servers, it changes their configuration. So Docker and Ansible have very different approaches. For this reason, Ansible and Puppet are not frequently used to deploy containers to production. However, using them together can bring some benefits, especially for development environments.

More on this later in the page. First, we need to understand how these concepts apply to database servers.

Stateful Technologies

Using ephemeral containers works very well for *stateless* technologies, like web servers and proxies. These technologies virtually only need binaries, configuration and small amounts of data (web pages). If some data need to be restored after a container creation, it will be a fast operation.

In the case of a database, the problem is that data can be large and need to be written somewhere. We don't want all databases to disappear when we destroy a container. Even if we had an up-to-date backup, restoring it would take time.

However, OCI Containers has features called **volumes**. A volume is a directory in the host system mapped to a directory in one or more containers. Volumes are not destroyed when containers are destroyed. They can be used to share data between any number of containers and the host system. Therefore, they are also a good way to persist data.

Suppose a MariaDB container called `mariadb-main-01` uses a volume that is mapped to `/var/docker/volumes/mariadb-main`. At some point we want to use a more recent MariaDB version. As explained earlier, the container way to do this is to destroy the container and create a new one that uses a more recent version of the MariaDB image.

So, we will destroy `mariadb-main-01`. The volume is still there. Then we create a new container with the same name, but based on a newer image. We make sure to link the volume to the new container too, so it will be able to use `/var/docker/volumes/mariadb-main` again. At this point we may want to run mariadb-upgrade, but apart from that, everything should *just work*.

The container runtime implementations also provide the opportunity to create a volume with an explicit name and this is also persistent. The actual location on the filesystem is managed by the runtime.

The above described steps are simple, but running them manually is time consuming and error-prone. Automating them with some automation software like Ansible or Puppet is often desirable.

Ways to Deploy Containers

Containers can be deployed in the following ways:

- Manually. See Installing and Using MariaDB via Docker. This is not recommended for production, or for complex environments. However, it can easily be done for the simplest cases. If we want to make changes to our custom images, we'll need to modify the Dockerfiles, destroy the containers and recreate them.
- With Docker Compose. See Setting Up a LAMP Stack with Docker Compose for a simple example. When we modify a Dockerfile, we'll need to destroy the containers and recreate them, which is usually as simple as running `docker-compose down` followed by `docker-compose-up`. After changing `docker-cmpose.yml` (maybe to add a container or a network) we'll simply need to run `docker-compose-up` again, because it is idempotent.
- Using Ansible, Puppet or other automation software, as mentioned before. We can use Ansible or Puppet to create the containers, and run them again every time we want to apply some change to the containers. This means that the containers are potentially created once and modified any number of times.

In all these cases, it is entirely possible to add Vagrant to the picture. Vagrant is a way to deploy or provision several hosts, including virtual machines (the most common case), and containers. It is agnostic in regarding the underlying technology, so it can deploy to a virtual machine, a container, or even a remote server in the same way. Containers can work with Vagrant in two ways:

- As a provisioner. In this case Vagrant will most commonly deploy a virtual machine, and will use Docker to setup the applications that need to run in it, as containers. This guarantees a higher level of isolation, compared to running the containers in the local host. Especially if you have different environments to deploy locally, because you can have them on different virtual machines.
- As a provider. Vagrant will deploy one or more containers locally. Once each container is up, Vagrant can optionally use a provisioner on it, to make sure that the container runs the proper software with proper configuration. In this case, Ansible, Puppet or other automation software can be used as a provisioner. But again, this is optional: it is possible to make changes to the Dockerfiles and recreate the containers every time.

Benefits of Managing Containers with Automation Software

Containers can be entirely managed with Docker Compose or the Swarm mode. This is often a good idea.

However, choosing to use automation software like Ansible or Puppet has some benefits too. Benefits include:

- Containers allow working without modifying the host system, and their creation is very fast. Much faster than virtual machines. This makes containers desirable for development environments.
- As explained, making all containers ephemeral and using volumes to store important data is possible. But this means adding some complexity to adapt an ephemeral philosophy to technologies that are not ephemeral by nature (databases). Also, many database professionals don't like this approach. Using automation software allows easily triggering upgrades and configuration changes in the containers, treating them as non-ephemeral systems.
- Sometimes containers are only used in development environments. If production databases are managed via Ansible, Puppet, or other automation software, this could lead to some code duplication. Dealing with configuration changes using the same procedures will reduce the cost of maintenance.
- While recreating containers is fast, being able to apply small changes with Ansible or Puppet can be more convenient in some cases: particularly if we write files into the container itself, or if recreating a container bootstrap involves some lengthy procedure.
- Trying to do something non-standard with Dockerfiles can be tricky. For example, running two processes in a

container is possible but can be problematic, as containers are designed to run single main process per container. However there are situations when this is desirable. For example PMM containers run several different processes. Launching additional processes with Ansible or Puppet may be easier than doing it with a Dockerfile.

With all this in mind, let's see some examples of cases when managing containers with Ansible, Puppet or other automation software is preferable, rather than destroying containers every time we want to make a change:

- We use Ansible or Puppet in production, and we try to keep development environments as similar as possible to production. By using Ansible/Puppet in development too, we can reuse part of the code.
- We make changes to the containers often, and recreating containers is not as fast as it should be (for example because a MariaDB dump needs to be restored).
- Creating a container implies some complex logic that does not easily fit a Dockerfile or Docker Compose (including, but not limited to, running multiple processes per container).

That said, every case is different. There are environments where these advantages do not apply, or bring a very small benefit. In those cases, the cost of adding some automation with Ansible, Puppet or similar software is probably not justified.

How to Deploy to Container from Orchestration Software

Suppose you want to manage containers configuration with Ansible.

At a first glance, the simplest way is to run Ansible in the host system. It will need to connect to the containers via SSH, so they need to expose the 22 port. But we have multiple containers, so we'll need to map the 22 port of each container to a different port in the host. This is hard to maintain and potentially insecure: in production you want to avoid exposing any container port to the host.

A better solution is to run Ansible itself in a container. The playbooks will be in a container volume, so we can access them from the host system to manage them more easily. The Ansible container will communicate with other containers using a container network, using the standard 22 port (or another port of your choice) for all containers.

2.1.2.14.6.2 Installing and Using MariaDB via Docker

Contents

Sometimes we want to install a specific version of MariaDB, MariaDB ColumnStore, or MaxScale on a certain system, but no packages are available. Or maybe, we simply want to isolate MariaDB from the rest of the system, to be sure that we won't cause any damage.

A virtual machine would certainly serve the scope. However, this means installing a system on the top of another system. It requires a lot of resources.

In many cases, the best solution is using containers. Docker is a framework that runs containers. A container is meant to run a specific daemon, and the software that is needed for that daemon to properly work. Docker does not virtualize a whole system; a container only includes the packages that are not included in the underlying system.

Docker requires a very small amount of resources. It can run on a virtualized system. It is used both in development and in production environments. Docker is an open source project, released under the Apache License, version 2.

Note that, while your package repositories could have a package called `docker`, it is probably not the Docker we are talking about. The Docker package could be called `docker.io` or `docker-engine`.

For information about installing Docker, see Get Docker in Docker documentation.

Installing Docker on Your System with the Universal Installation Script

The script below will install the Docker repositories, required kernel modules and packages on the most common Linux distributions:

```
curl -sSL https://get.docker.com/ | sh
```

Starting dockerd

On some systems you may have to start the `dockerd daemon` yourself:

```
sudo systemctl start docker
sudo gpasswd -a "${USER}" docker
```

If you don't have `dockerd` running, you will get the following error for most `docker` commands: installing-and-using-mariadb-via-docker Cannot connect to the Docker daemon at unix:*/var/run/docker.sock. Is the docker daemon running? <</code>>*

Using MariaDB Images

The easiest way to use MariaDB on Docker is choosing a MariaDB image and creating a container.

Downloading an Image

You can download a MariaDB image for Docker from the Offical Docker MariaDB, or choose another image that better suits your needs. You can search Docker Hub (the official set of repositories) for an image with this command:

```
 docker search mariadb
```

Once you have found an image that you want to use, you can download it via Docker. Some layers including necessary dependencies will be downloaded too. Note that, once a layer is downloaded for a certain image, Docker will not need to download it again for another image.

For example, if you want to install the default MariaDB image, you can type:

```
docker pull mariadb:10.4
```

This will install the 10.4 version. Versions 10.2, 10.3, 10.5 are also valid choices.

You will see a list of necessary layers. For each layer, Docker will say if it is already present, or its download progress.

To get a list of installed images:

```
docker images
```

Creating a Container

An image is not a running process; it is just the software needed to be launched. To run it, we must create a container first. The command needed to create a container can usually be found in the image documentation. For example, to create a container for the official MariaDB image:

```
docker run --name mariadbtest -e MYSQL_ROOT_PASSWORD=mypass -p 3306:3306 -d
docker.io/library/mariadb:10.3
```

`mariadbtest` is the name we want to assign the container. If we don't specify a name, an id will be automatically generated.

10.2 and 10.5 are also valid target versions:

```
docker run --name mariadbtest -e MYSQL_ROOT_PASSWORD=mypass -p 3306:3306 -d
docker.io/library/mariadb:10.2
```

```
docker run --name mariadbtest -e MYSQL_ROOT_PASSWORD=mypass -p 3306:3306  -d
docker.io/library/mariadb:10.5
```

Optionally, after the image name, we can specify some options for mysqld. For example:

```
docker run --name mariadbtest -e MYSQL_ROOT_PASSWORD=mypass -p 3306:3306 -d mariadb:10.3 --log-
bin --binlog-format=MIXED
```

Docker will respond with the container's id. But, just to be sure that the container has been created and is running, we can get a list of running containers in this way:

```
docker ps
```

We should get an output similar to this one:

```
CONTAINER ID        IMAGE                     COMMAND                CREATED
STATUS              PORTS               NAMES
819b786a8b48        mariadb                   "/docker-entrypoint.   4 minutes ago       Up 4
minutes        3306/tcp            mariadbtest
```

Running and Stopping the Container

Docker allows us to restart a container with a single command:

```
docker restart mariadbtest
```

The container can also be stopped like this:

```
docker stop mariadbtest
```

The container will not be destroyed by this command. The data will still live inside the container, even if MariaDB is not running. To restart the container and see our data, we can issue:

```
docker start mariadbtest
```

With `docker stop`, the container will be gracefully terminated: a `SIGTERM` signal will be sent to the `mysqld` process, and Docker will wait for the process to shutdown before returning the control to the shell. However, it is also possible to set a timeout, after which the process will be immediately killed with a `SIGKILL`. Or it is possible to immediately kill the process, with no timeout.

```
docker stop --time=30 mariadbtest
docker kill mariadbtest
```

In case we want to destroy a container, perhaps because the image does not suit our needs, we can stop it and then run:

```
docker rm mariadbtest
```

Note that the command above does not destroy the data volume that Docker has created for /var/lib/mysql. If you want to destroy the volume as well, use:

```
docker rm -v mariadbtest
```

Automatic Restart

When we start a container, we can use the `--restart` option to set an automatic restart policy. This is useful in production.

Allowed values are:

- `no` : No automatic restart.
- `on-failure` : The container restarts if it exits with a non-zero exit code.
- `unless-stopped` : Always restart the container, unless it was explicitly stopped as shown above.
- `always` : Similar to `unless-stopped`, but when Docker itself restarts, even containers that were explicitly stopped

will restart.

It is possible to change the restart policy of existing, possibly running containers:

```
docker update --restart always mariadb
# or, to change the restart policy of all containers:
docker update --restart always $(docker ps -q)
```

A use case for changing the restart policy of existing containers is performing maintenance in production. For example, before upgrading the Docker version, we may want to change all containers restart policy to `always`, so they will restart as soon as the new version is up and running. However, if some containers are stopped and not needed at the moment, we can change their restart policy to `unless-stopped`.

Pausing Containers

A container can also be frozen with the `pause` command. Docker will freeze the process using croups. MariaDB will not know that it is being frozen and, when we `unpause` it, MariaDB will resume its work as expected.

Both `pause` and `unpause` accept one or more container names. So, if we are running a cluster, we can freeze and resume all nodes simultaneously:

```
docker pause node1 node2 node3
docker unpause node1 node2 node3
```

Pausing a container is very useful when we need to temporarily free our system's resources. If the container is not crucial at this moment (for example, it is performing some batch work), we can free it to allow other programs to run faster.

Troubleshooting a Container

If the container doesn't start, or is not working properly, we can investigate with the following command:

```
docker logs mariadbtest
```

This command shows what the daemon sent to the stdout since the last attempt of starting - the text that we typically see when we invoke `mysqld` from the command line.

On some systems, commands such as `docker stop mariadbtest` and `docker restart mariadbtest` may fail with a permissions error. This can be caused by AppArmor, and even `sudo` won't allow you to execute the command. In this case, you will need to find out which profile is causing the problem and correct it, or disable it. **Disabling AppArmor altogether is not recommended, especially in production.**

To check which operations were prevented by AppArmor, see AppArmor Failures in AppArmor documentation.

To disable a profile, create a symlink with the profile name (in this example, `mysqld`) to `etc/apparmor.d/disable`, and then reload profiles:

```
ln -s /etc/apparmor.d/usr.sbin.mysqld /etc/apparmor.d/disable/
sudo apparmor_parser -R /etc/apparmor.d/usr.sbin.mysqld
```

For more information, see Policy Layout in AppArmor documentation.

After disabling the profile, you may need to run:

```
sudo service docker restart
docker system prune --all --volumes
```

Restarting the system will then allow Docker to operate normally.

Accessing the Container

To access the container via Bash, we can run this command:

```
docker exec -it mariadbtest bash
```

Now we can use normal Linux commands like *cd*, *ls*, etc. We will have root privileges. We can even install our favorite file editor, for example:

```
apt-get update
apt-get install vim
```

In some images, no repository is configured by default, so we may need to add them.

Note that if we run mariadb-admin shutdown or the SHUTDOWN command to stop the container, the container will be deactivated, and we will automatically exit to our system.

Connecting to MariaDB from Outside the Container

If we try to connect to the MariaDB server on `localhost`, the client will bypass networking and attempt to connect to the server using a socket file in the local filesystem. However, this doesn't work when MariaDB is running inside a container because the server's filesystem is isolated from the host. The client can't access the socket file which is inside the container, so it fails to connect.

Therefore connections to the MariaDB server must be made using TCP, even when the client is running on the same machine as the server container.

Find the IP address that has been assigned to the container:

```
docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' mariadbtest
```

You can now connect to the MariaDB server using a TCP connection to that IP address.

Forcing a TCP Connection

After enabling network connections in MariaDB as described above, we will be able to connect to the server from outside the container.

On the host, run the client and set the server address ("-h") to the container's IP address that you found in the previous step:

```
mysql -h 172.17.0.2 -u root -p
```

This simple form of the connection should work in most situations. Depending on your configuration, it may also be necessary to specify the port for the server or to force TCP mode:

```
mysql -h 172.17.0.2 -P 3306 --protocol=TCP -u root -p
```

Port Configuration for Clustered Containers and Replication

Multiple MariaDB servers running in separate Docker containers can connect to each other using TCP. This is useful for forming a Galera cluster or for replication.

When running a cluster or a replication setup via Docker, we will want the containers to use different ports. The fastest way to achieve this is mapping the containers ports to different port on our system. We can do this when creating the containers (`docker run` command), by using the `-p` option, several times if necessary. For example, for Galera nodes we will use a mapping similar to this one:

```
-p 4306:3306 -p 5567:5567 -p 5444:5444 -p 5568:5568
```

Installing MariaDB on Another Image

It is possible to download a Linux distribution image, and to install MariaDB on it. This is not much harder than installing MariaDB on a regular operating system (which is easy), but it is still the hardest option. Normally we will try existing images first. However, it is possible that no image is available for the exact version we want, or we want a custom installation, or perhaps we want to use a distribution for which no images are available. In these cases, we will install MariaDB in an operating system image.

Daemonizing the Operating System

First, we need the system image to run as a daemon. If we skip this step, MariaDB and all databases will be lost when the container stops.

To demonize an image, we need to give it a command that never ends. In the following example, we will create a Debian Jessie daemon that constantly pings the 8.8.8.8 special address:

```
docker run --name debian -p 3306:3306 -d debian /bin/sh -c "while true; do ping 8.8.8.8; done"
```

Installing MariaDB

At this point, we can enter the shell and issue commands. First we will need to update the repositories, or no packages will be available. We can also update the packages, in case some of them are newer than the image. Then, we will need to install a text editor; we will need it to edit configuration files. For example:

```
# start an interactive Bash session in the container
docker exec -ti debian bash
apt-get -y update
apt-get -y upgrade
apt-get -y install vim
```

Now we are ready to install MariaDB in the way we prefer.

2.1.2.14.6.3 Container Backup and Restoration

MariaDB databases in containers need backup and restore like their non-container equivalents.

Logicial Backups

Backup

mariadb-dump is in the Docker Official Image and can be used as follows:

```
$ docker exec some-%%REPO%% mariadb-dump --all-databases -uroot -p"$MARIADB_ROOT_PASSWORD"'
> /some/path/on/your/host/all-databases.sql
```

Restoring Data from Dump Files

For restoring data, you can use the `docker exec` command with the `-i` flag, similar to the following:

```
$ docker exec -i some-%%REPO%% sh -c 'exec mariadb -uroot -p"$MARIADB_ROOT_PASSWORD"'
< /some/path/on/your/host/all-databases.sql
```

Physical Backups

mariadb-backup is in the Docker Official Image.

Backup

MariaDB Backup can create a backup as follows:

To perform a backup using Mariabackup, a second container is started that shares the original container's data directory. An additional volume for the backup needs to be included in the second backup instance. Authentication against the MariaDB database instance is required to successfully complete the backup. In the example below, a `mysql@localhost` user is used with the MariaDB server's Unix socket shared with the backup container.

```
$ docker volume create some-%%REPO%%-socket
$ docker run --name some-%%REPO%% -v /my/own/datadir:/var/lib/mysql
-v some-%%REPO%%-socket:/var/run/mysqld -e MARIADB_MYSQL_LOCALHOST_USER=1
-e MARIADB_MYSQL_LOCALHOST_GRANTS="RELOAD, PROCESS, LOCK TABLES, BINLOG MONITOR"
-e MARIADB_ROOT_PASSWORD=my-secret-pw -d %%IMAGE%%:latest
```

Note: Privileges listed here are for 10.5+. For an exact list, see Mariabackup: Authentication and Privileges.

Mariabackup will run as the `mysql` user in the container, so the permissions on `/backup` will need to ensure that it can be written to by this user:

```
$ docker volume create some-%%REPO%%-backup
$ docker run --rm some-%%REPO%%-backup
-v some-%%REPO%%-backup:/backup %%IMAGE%%:latest chown mysql: /backup
```

Restore

These steps restore the backup made with Mariabackup.

At some point before doing the restore, the backup needs to be prepared. Perform the prepare like this:

```
$ docker run --user mysql --rm -v some-%%REPO%%-backup:/backup
%%IMAGE%%:latest mariabackup --prepare --target-dir=/backup
```

Now that the image is prepared, start the container with both the data and the backup volumes and restore the backup:

```
$ docker run --user mysql --rm -v /my/new/datadir:/var/lib/mysql
-v some-%%REPO%%-backup:/backup %%IMAGE%%:latest mariabackup --copy-back --target-dir=/backup
```

With `/my/new/datadir` containing the restored backup, start normally as this is an initialized data directory:

```
$ docker run --name some-%%REPO%% -v /my/new/datadir:/var/lib/mysql -d %%IMAGE%%:latest
```

For further information on Mariabackup, see Mariabackup Overview.

2.1.2.14.6.4 Container Security Concerns

When using containers in production, it is important to be aware of container security concerns.

Contents

Host System Security

Depending on the container runtime, containers may be running on the host system's kernel or a kernel shared with other containers. If this kernel has security bugs, those bugs are also present in the containers. Malicious containers may attempt to explain a kernel vulnerability to impact the confidentiality, integrity or availability of other containers.

In particular, Linux based containers have a container runtime that can use the following features:

- Namespaces, to isolate containers from each other and make sure that a container can't establish unauthorized connections to another container.
- Seccomp security profiles.
- Rootless operation in Docker, or Rootless Podman
- cgroups, to limit the resources (CPU, memory, IO) that each container can consume.

The administrators of a system should be particularly careful to upgrade the kernel whenever security bugs to these features are fixed.

It is important to note that when we upgrade the kernel, runC or Docker itself we cause downtime for all the containers running on the system.

Images Security

Containers are built from images. If security is a major concern, you should make sure that the images you use are secure.

If you want to be sure that you are pulling authentic images, you should only pull images signed with Docker Content Trust. Signing only ensure authenticity or origin, it doesn't dictate that entity is trustworthy.

Updated images should be used. An image usually downloads packages information at build time. If the image is not recently built, a newly created container will have old packages. Updating the packages on container creation and regularly re-updating them will ensure that the container uses packages with the most recent versions. Rebuilding an image often will reduce the time necessary to update the packages the first time.

Security bugs are usually important for a database server, so you don't want your version of MariaDB to contain known security bugs. But suppose you also have a bug in Docker, in runC, or in the kernel. A bug in a user-facing application may

allow an attacker to exploit a bug in those lower level technologies. So, after gaining access to the container, an attacker may gain access to the host system. This is why system administrators should keep both the host system and the software running in the containers updated.

References

For more information, see the following links:

- Container Security from Red Hat.
- Docker security on Docker documentation.

Content initially contributed by Vettabase Ltd.

2.1.2.14.6.5 Adding Plugins to the MariaDB Docker Official Image

MariaDB has many plugins. Most are not enabled by default, some are in the `mariadb` container, while others need to be installed from additional packages.

The following methods summarize Installing plugins in the MariaDB Docker Library Container (mariadb.org blog post) on this topic.

Which Plugins Does the Container Contain?

To see which plugins are available in the mariadb:

```
$ docker run --rm mariadb:latest ls -C /usr/lib/mysql/plugin
```

Enabling a Plugin Using Flags

Using the `--plugin-load-add` flag with the plugin name (can be repeated), the plugins will be loaded and ready when the container is started:

For example, to enable the `simple\_password\_check` plugin:

```
$ docker run --name some-%%REPO%% -e MARIADB_ROOT_PASSWORD=my-secret-pw --network=host -d mariadb
```

Enabling a Plugin in the Configuration Files

plugin-load-add` can be used as a configuration option to load plugins. The example below loads the FederatedX Storage Engine.

```
$ printf "[mariadb]\nplugin-load-add=ha_federatedx\n" > /my/custom/federatedx.conf
$ docker run --name some-mariadb -v /my/custom:/etc/mysql/conf.d -e MARIADB_ROOT_PASSWORD=my-secr
```

Install a Plugin Using SQL in /docker-entrypoint-initdb.d

INSTALL SONAME can be used to install a plugin as part of the database initialization.

Create the SQL file used in initialization:

```
$ echo 'INSTALL SONAME "disks";' > my_initdb/disks.sql
```

In this case, the `my\_initdb` is a `/docker-entrypoint-initdb.d` directory per "Initializing a fresh instance" section above.

Identifying Additional Plugins in Additional Packages

A number of plugins are in separate packages to reduce their installation size. The package names of MariaDB-created plugins can be determined using the following command:

```
$ docker run --rm mariadb:latest sh -c 'apt-get update -qq && apt-cache search mariadb-plugin'
```

Creating an Image With Plugins From Additional Packages

A new image needs to be created when using additional packages. The `mariadb` image can however be used as a base:

In the following, the CONNECT Storage Engine is installed:

```
FROM mariadb:latest
RUN apt-get update && \
    apt-get install mariadb-plugin-connect -y && \
    rm -rf /var/lib/apt/lists/*
```

Installing plugins from packages creates a configuration file in the directory `/etc/mysql/mariadb.conf.d/` that loads the plugin on startup.

2.1.2.14.6.6 Setting Up a LAMP Stack with Docker Compose

Docker Compose is a tool that allows one to declare which Docker containers should run, and which relationships should exist between them. It follows the **infrastructure as code** approach, just like most automation software and Docker itself.

For information about installing Docker Compose, see Install Docker Compose in Docker documentation.

Contents

The `docker-compose.yml` File

When using Docker Compose, the Docker infrastructure must be described in a YAML file called `docker-compose.yml`.

Let's see an example:

```
version: "3"

services:
  web:
    image: "apache:${PHP_VERSION}"
    restart: 'always'
    depends_on:
      - mariadb
    restart: 'always'
    ports:
      - '8080:80'
    links:
      - mariadb
  mariadb:
    image: "mariadb:${MARIADB_VERSION}"
    restart: 'always'
    volumes:
      - "/var/lib/mysql/data:${MARIADB_DATA_DIR}"
      - "/var/lib/mysql/logs:${MARIADB_LOG_DIR}"
      - /var/docker/mariadb/conf:/etc/mysql
    environment:
      MYSQL_ROOT_PASSWORD: "${MYSQL_ROOT_PASSWORD}"
      MYSQL_DATABASE: "${MYSQL_DATABASE}"
      MYSQL_USER: "${MYSQL_USER}"
      MYSQL_PASSWORD: "${MYSQL_PASSWORD}"
```

In the first line we declare that we are using version 3 of the Docker compose language.

Then we have the list of services, namely the `web` and the `mariadb` services.

Let's see the properties of the services:

- `port` maps the 8080 container port to the 80 host system port. This is very useful for a development environment, but not in production, because it allows us to connect our browser to the containerized web server. Normally there is no need to connect to MariaDB from the host system.
- `links` declares that this container must be able to connect `mariadb`. The hostname is the container name.
- `depends_on` declares that `mariadb` needs to start before `web`. This is because we cannot do anything with our application until MariaDB is ready to accept connections.
- `restart: always` declares that the containers must restart if they crash.
- `volumes` creates volumes for the container if it is set in a service definition, or a volume that can be used by any container if it is set globally, at the same level as `services`. Volumes are directories in the host system that can be accessed by any number of containers. This allows destroying a container without losing data.
- `environment` sets environment variables inside the container. This is important because in setting these variables we set the MariaDB root credentials for the container.

About Volumes

It is good practice to create volumes for:

- The data directory, so we don't lose data when a container is created or replaced, perhaps to upgrade MariaDB.
- The directory where we put all the logs, if it is not the datadir.
- The directory containing all configuration files (for development environments), so we can edit those files with the editor installed in the host system. Normally no editor is installed in containers. In production we don't need to do this, because we can copy files from a repository located in the host system to the containers.

Note that Docker Compose variables are just placeholders for values. Compose does not support assignment, conditionals or loops.

Using Variables

In the above example you can see several variables, like `${MARIADB_VERSION}`. Before executing the file, Docker Compose will replace this syntax with the `MARIADB_VERSION` variable.

Variables allow making Docker Compose files more re-usable: in this case, we can use any MariaDB image version without modifying the Docker Compose file.

The most common way to pass variables is to write them into a file. This has the benefit of allowing us to version the variable file along with the Docker Compose file. It uses the same syntax you would use in BASH:

```
PHP_VERSION=8.0
MARIADB_VERSION=10.5
...
```

For bigger setups, it could make sense to use different environment files for different services. To do so, we need to specify the file to use in the Compose file:

```
services:
  web:
    env_file:
      - web-variables.env
...
```

Docker Compose Commands

Docker Compose is operated using `docker-compose`. Here we'll see the most common commands. For more commands and for more information about the commands mentioned here, see the documentation.

Docker Compose assumes that the Composer file is located in the current directory and it's called `docker-compose.yml`. To use a different file, the `-f <filename>` parameter must be specified.

To pull the necessary images:

```
docker-compose pull
```

Containers described in the Compose file can be created in several ways.

To create them only if they do not exist:

```
docker-compose up --no-recreate
```

To create them if they do not exist and recreate them if their image or configuration have changed:

```
docker-compose up
```

To recreate containers in all cases:

```
docker-compose up --force-recreate
```

Normally `docker-compose up` starts the containers. To create them without starting them, add the `--no-start` option.

To restart containers without recreating them:

```
docker-compose restart
```

To kill a container by sending it a `SIGKILL`:

```
docker-compose kill <service>
```

To instantly remove a running container:

```
docker-compose rm -f <serice>
```

To tear down all containers created by the current Compose file:

```
docker-compose down
```

Docker Compose Resources and References

- Overview of Docker Compose in the Docker documentation.
- Compose file in the Docker documentation.
- Docker Compose on GitHub.

Further information about the concepts explained in this page can be found in Docker documentation:

- Environment variables in Compose.
- Overview of docker-compose CLI.
- Compose command-line reference.

Content initially contributed by Vettabase Ltd.

2.1.2.14.6.7 Creating a Custom Container Image

OCI containers, frequently and incorrectly called Docker containers, are created from OCI images. An image contains software that can be launched, including the underlying system. A container is an instance of that software.

When we want to automate MariaDB, creating an image with MariaDB and the desired configuration, we may want to create an image by ourselves, which fulfils our needs.

Contents

Images Architecture

One "source code" of an image is a Dockerfile. A Dockerfile is written in Docker specific language, and can be compiled into an image by the `docker` binary, using the `docker build` command. It can also be compiled by `buildah` using `buildah bud`.

Most images are based on another image. The base image is specified at the beginning of the Dockerfile, with the `FROM` directive. If the base image is not present in the local system, it is downloaded from the repository specified, or if not specified, from the default repository of the build program. This is often Docker Hub. For example, we can build a `mariadb-rocksdb:10.5` image starting from the `debian:13` image. In this way, we'll have all the software included in a standard Debian image, and we'll add MariaDB and its configuration upon that image.

All the following Dockerfile directives are compiled into a new Docker image, identified by an SHA256 string. Each of these images is based on the image compiled from the previous directive. A physical compiled image can serve as a base for any number of images. This mechanism saves a lot of disk space, download time and build time.

The following diagram shows the relationship between Dockerfiles, images and containers:

Dockerfile Syntax

Here's a simple Dockerfile example:

```
FROM ubuntu:20.04

RUN apt-get update
RUN apt-get install -y mariadb-server

EXPOSE 3306

LABEL version="1.0"
LABEL description="MariaDB Server"

HEALTHCHECK --start-period=5m \
  CMD mariadb -e 'SELECT @@datadir;' || exit 1

CMD ["mariadbd"]
```

This example is not very good for practical purposes, but it shows what a Dockerfile looks like.

First, we declare that the base image to use is `ubuntu:20.04`.

Then we run some commands to install MariaDB from the Ubuntu default repositories and stop the MariaDB service.

We define some metadata about the image with `LABEL`. Any label is valid.

We declare that the port 3306 (MariaDB default port) should be exposed. However, this has no effect if the port is not exposed at container creation.

We also define a healthcheck. This is a command that is run to check if the container is healthy. If the return code is 0 the healthcheck succeeds, if it's 1 it fails. In the MariaDB specific case, we want to check that it's running and able to answer a simple query. This is better than just checking that MariaDB process is running, because MariaDB could be running but unable to respond, for example because max_connections was reached or data si corrupted. We read a system variable, because we should not assume that any user-created table exists. We also specify `--start-period` to allow some time for MariaDB to start, keeping in mind that restarting it may take some time if some data is corrupted. Note that there can be only one healthcheck: if the command is specified multiple times, only the last occurrence will take effect.

Finally, we start the container command: mariadbd. This command is run when a container based on this image starts. When the process stops or crashes, the container will immediately stop.

Note that, in a container, we normally run mariadbd directly or in an entrypoint script `exec mariadbd`, rather than running mysqld_safe or running MariaDB as a service. Containers restart can be handled by the container service. See automatic restart.

See the documentation links below to learn the syntax allowed in a Dockerfile.

Using Variables

It is possible to use variables in a Dockerfile. This allows us, for example, to install different packages, install different versions of a package, or configure software differently depending on how variables are set, without modifying the Dockerfile itself.

To use a variable, we can do something like this:

```
FROM ubuntu:20.04

ARG MARIADB_CONFIG_FILE

...

ENTRYPOINT mariadbd --defaults-file=$MARIADB_CONFIG_FILE
```

Here `ARG` is used after the `FROM` directive, thus the variable cannot be used in `FROM`. It is also possible to declare a variable before `FROM`, so we can use a variable to select the base image to use or its tag, but in this case the variable cannot be used after the `FROM` directive, unless `ARG` is re-declared after the `FROM`. Here is an example:

```
ARG UBUNTU_VERSION
FROM ubuntu:$UBUNTU_VERSION

# Uncomment for the build error to be avoided
# ARG UBUNTU_VERSION

# But this will cause a build error:
RUN echo 'Ubuntu version: $UBUNTU_VERSION' > /var/build_log
```

We'll have to assign variables a value when we build the Dockerfile, in this way:

```
docker build --build-arg UBUNTU_VERSION=20.04 .
```

Note that Dockerfile variables are just placeholders for values. Dockerfiles do not support assignment, conditionals or loops.

Versioning and Deploying Images

Dockerfiles are normally versioned, as well as the files that are copied to the images.

Once an image is built, it can be pushed to a container registry. Whenever an image is needed on a host to start containers from it, it is pulled from the registry.

Container registries

A default container registry for OCI images is Docker Hub. It contains Docker Official Images maintained by the Docker Library team and the community. Any individual or organization can open an account and push images to Docker Hub. Most Docker images are open source: the Dockerfiles and the needed files to build the images are usually on GitHub.

It is also possible to setup a self-hosted registry. Images can be pushed to that registry and pulled from it, instead of using Docker Hub. If the registry is not publicly accessible, it can be used to store images used by the organization without making them publicly available.

But a self-hosted registry can also be useful for open source images: if an image is available on Docker Hub and also on a self-hosted registry, in case Docker Hub is down or not reachable, it will still be possible to pull images.

Choosing Image Names and Tags

The names of images developed by the community follow this schema:

```
repository/maintainer/technology
```

It doesn't matter if the maintainer is an individual or an organization. For images available on Docker Hub, the maintainer is the name of a Docker Hub account.

Official images maintained by the Docker Library maintainers have the implicit name of `library` filled in by the container fetching tool. For example, the official MariaDB image is called `mariadb` which is an alias for `docker.io/library/mariadb`.

All images have a tag, which identifies the version or the variant of an image. For example, all MariaDB versions available on Docker are used as image tags. MariaDB 10.11 is called `mariadb:10.11`.

By conversion, tags form a hierarchy. So for example, there is a `10.1.1` tag whose meaning will not change over time. `10.5` will always identify the latest stable version in the 10.5 branch. For some time it was `10.5.1`, then it became `10.5.2`, and so on.

When we pull an image without specifying a tag (ie, `docker pull mariadb`), we are implicitly requiring the image with the `latest` tag. This is even more mutable: at different periods of time, it pointed to the latest `10.0` version, to the latest `10.1` version, and so on.

In production, it is always better to know for sure which version we are installing. Therefore it is better to specify a tag whose meaning won't change over time, like `10.5.21`. To keep to a latest LTS version, the `lts` can be used.

Pushing and Pulling Images

To pull an image from Docker Hub or a self-hosted registry, we use the `docker pull` command. For example:

```
docker pull mariadb:10.5
```

This command downloads the specified image if it is not already present in the system, or if the local version is not up to date.

After modifying a Dockerfile, we can build an image in this way:

```
docker build .
```

This step can be automated by services like Docker Hub and GitHub. Check those service's documentation to find out how this feature works.

Once an image is created, it can be pushed to a registry. We can do it in this way:

```
docker push <image_name>:<tag>
```

Docker Content Trust

Docker has a feature called Docker Content Trust (DCT). It is a system used to digitally sign images, based on PEM keys. For environments where security is a major concern, it is important to sign images before pushing them. This can be done with both Docker Hub and self-hosted registries.

Good Practices and Caveats

As mentioned, a Dockerfile is built by creating a new image for each directive that follows `FROM`. This leads to some considerations.

- Sometimes it can be a good idea to run several shell commands in a single `RUN` directive to avoid creating images that are not useful.
- Modifying a directive means that all subsequent directives also need to be rebuilt. When possible, directives that are expected to change often should follow directives that will change seldom.
- Directives like `LABEL` or `EXPOSE` should be placed close to the end of Dockerfiles. In this way they will be rebuilt often, but this operation is cheap. On the other side, changing a label should not trigger a long rebuild process.
- Variables should be used to avoid Dockerfiles proliferation. But if a variable is used, changing its value should be tested. So, be sure not to use variables without a good reason.
- Writing logic into a Dockerfile is impossible or very hard. Call shell scripts instead, and write your logic into them. For example, in a shell script it is easy to perform a certain operation only if a variable is set to a certain value.
- If you need MariaDB containers with different configurations or different sets of plugins, use the method explained above. Do not create several Dockerfiles, with different tags, for each desired configuration or plugin set. This may lead to undesired code duplication and increased maintenance costs.

References

More details can be found in the Docker documentation:

- Dockerfile reference.
- docker build.
- Repositories.

- Deploy a registry server.
- Content trust in Docker.

See also:

- Privacy-Enhanced Mail on Wikipedia.

Content initially contributed by Vettabase Ltd.

2.1.2.14.6.8 MariaDB Server Docker Official Image Environment Variables

Contents

When you start the image, you can adjust the initialization of the MariaDB Server instance by passing one or more environment variables on the docker run command line. Do note that all of the variables below, except `MARIADB_AUTO_UPGRADE`, will have now effect if you start the container with a data directory that already contains a database: any pre-existing database will always be left untouched on container startup.

From tag 10.2.38, 10.3.29, 10.4.19, 10.5.10 onwards, and all 10.6 and later tags, the MARIADB_* equivalent variables are provided. MARIADB_* variants will always be used in preference to MYSQL_* variants.

One of MARIADB_ROOT_PASSWORD_HASH, MARIADB_ROOT_PASSWORD, MARIADB_ALLOW_EMPTY_ROOT_PASSWORD, or MARIADB_RANDOM_ROOT_PASSWORD (or equivalents, including *_FILE), is required. The other environment variables are optional.

MARIADB_ROOT_PASSWORD_HASH / MARIADB_ROOT_PASSWORD / MYSQL_ROOT_PASSWORD

This specifies the password that will be set for the MariaDB root superuser account.

MARIADB_ALLOW_EMPTY_ROOT_PASSWORD / MYSQL_ALLOW_EMPTY_PASSWORD

Set to a non-empty value, like `1`, to allow the container to be started with a blank password for the root user. NOTE: Setting this variable to yes is not recommended unless you really know what you are doing, since this will leave your MariaDB instance completely unprotected, allowing anyone to gain complete superuser access.

MARIADB_RANDOM_ROOT_PASSWORD / MYSQL_RANDOM_ROOT_PASSWORD

Set to a non-empty value, like yes, to generate a random initial password for the root user. The generated root password will be printed to stdout (GENERATED ROOT PASSWORD: .....).

MARIADB_ROOT_HOST / MYSQL_ROOT_HOST

This is the hostname part of the root user created. By default this is %, however it can be set to any default MariaDB allowed hostname component. Setting this to localhost will prevent any root user being accessible except via the unix socket.

MARIADB_DATABASE / MYSQL_DATABASE

This variable allows you to specify the name of a database to be created on image startup.

MARIADB_USER / MYSQL_USER, MARIADB_PASSWORD_HASH / MARIADB_PASSWORD / MYSQL_PASSWORD

Both user and password variables, along with a database, are required for a user to be created. This user will be granted all access (corresponding to GRANT ALL) to the MARIADB_DATABASE database.

Do not use this mechanism to create the root superuser, that user gets created by default with the password specified by the MARIADB_ROOT_PASSWORD / MYSQL_ROOT_PASSWORD variable.

MARIADB_MYSQL_LOCALHOST_USER / MARIADB_MYSQL_LOCALHOST_GRANTS

Set MARIADB_MYSQL_LOCALHOST_USER to a non-empty value to create the mysql@locahost database user. This user is especially useful for a variety of health checks and backup scripts.

The mysql@localhost user gets `USAGE` privileges by default. If more access is required, additional global privileges in the form of a comma separated list can be provided. If you are sharing a volume containing MariaDB's unix socket (/var/run/mysqld by default), privileges beyond `USAGE` can result in confidentiality, integrity and availability risks, so use a minimal set. Its also possible to use for Mariadb-backup. The healthcheck.sh script also documents the required privileges for each health check test.

MARIADB_HEALTHCHECK_GRANTS

Set MARIADB_HEALTHCHECK_GRANTS to the grants required to be given to the `healtcheck@localhost`, `healtcheck@127.0.0.1`, `healtcheck@::1`, users. When not specified the default grant is `USAGE`.

The main value used here will be `REPLICA MONITOR for the healthcheck --replication test.`

MARIADB_INITDB_SKIP_TZINFO / MYSQL_INITDB_SKIP_TZINFO

By default, the entrypoint script automatically loads the timezone data needed for the CONVERT_TZ() function. If it is not needed, any non-empty value disables timezone loading.

MARIADB_AUTO_UPGRADE / MARIADB_DISABLE_UPGRADE_BACKUP

Set MARIADB_AUTO_UPGRADE to a non-empty value to have the entrypoint check whether mariadb-upgrade needs to run, and if so, run the upgrade before starting the MariaDB server.

Before the upgrade, a backup of the system database is created in the top of the datadir with the name system_mysql_backup_*.sql.zst. This backup process can be disabled with by setting MARIADB_DISABLE_UPGRADE_BACKUP to a non-empty value.

MARIADB_MASTER_HOST

When specified, the container will connect to this host and replicate from it.

MARIADB_REPLICATION_USER / MARIADB_REPLICATION_PASSWORD_HASH / MARIADB_REPLICATION_PASSWORD

When MARIADB_MASTER_HOST is specified, MARIADB_REPLICATION_USER and MARIADB_REPLICATION_PASSWORD will be used to connect to the master.

When not specified, the MARIADB_REPLICATION_USER will be created with the REPLICATION REPLICA grants required to a client to start replication.

2.1.2.14.5.4 Running MariaDB ColumnStore Docker containers on Linux, Windows and MacOS

2.1.2.14.6.10 Docker Official Image Frequently Asked Questions

Contents

Frequently asked questions about the Docker Official Image

How to Reset Passwords

If you have an existing data directory and wish to reset the root and user passwords, and to create a database which the user can fully modify, perform the following steps.

First create a `passwordreset.sql` file:

```
CREATE USER IF NOT EXISTS root@localhost IDENTIFIED BY 'thisismyrootpassword';
SET PASSWORD FOR root@localhost = PASSWORD('thisismyrootpassword');
GRANT ALL ON *.* TO root@localhost WITH GRANT OPTION;
GRANT PROXY ON ''@'%' ON root@localhost WITH GRANT OPTION;
CREATE USER IF NOT EXISTS root@'%' IDENTIFIED BY 'thisismyrootpassword';
SET PASSWORD FOR root@'%' = PASSWORD('thisismyrootpassword');
GRANT ALL ON *.* TO root@'%' WITH GRANT OPTION;
GRANT PROXY ON ''@'%' ON root@'%' WITH GRANT OPTION;
CREATE USER IF NOT EXISTS myuser@'%' IDENTIFIED BY 'thisismyuserpassword';
SET PASSWORD FOR myuser@'%' = PASSWORD('thisismyuserpassword');
CREATE DATABASE IF NOT EXISTS databasename;
GRANT ALL ON databasename.* TO myuser@'%';
```

Adjust `myuser`, `databasename` and passwords as needed.

Then:

```
$ docker run --rm -v /my/own/datadir:/var/lib/mysql -v /my/own/passwordreset.sql:/passwordreset.s
```

On restarting the MariaDB container in this `/my/own/datadir`, the `root` and `myuser` passwords will be reset.

Temp Server Start Timeout

Question, are you getting errors like the following where a temporary server start fails to succeed in 30 seconds?

Example of log:

```
2023-01-28 12:53:42+00:00 [Note] [Entrypoint]: Starting temporary server
2023-01-28 12:53:42+00:00 [Note] [Entrypoint]: Waiting for server startup
2023-01-28 12:53:42 0 [Note] mariadbd (server 10.10.2-MariaDB-1:10.10.2+maria~ubu2204) starting a
....
2023-01-28 12:53:42 0 [Note] InnoDB: Setting file './ibtmp1' size to 12.000MiB. Physically writin
2023-01-28 12:54:13 0 [Note] mariadbd: ready for connections.
Version: '10.10.2-MariaDB-1:10.10.2+maria~ubu2204'  socket: '/run/mysqld/mysqld.sock'  port: 0  m
2023-01-28 12:54:13+00:00 [ERROR] [Entrypoint]: Unable to start server.
```

The timeout on a temporary server start is a quite generous 30 seconds.

The lack of a message like the following indicates it failed to complete writing a temporary file of 12MiB in 30 seconds.

```
2023-01-28 12:53:46 0 [Note] InnoDB: File './ibtmp1' size is now 12.000MiB.
```

If the datadir where this is stored is remote storage maybe it's a bit slow. It's ideal to have an InnoDB temporary path local

so this can be configured using the command or configuration setting:

```
innodb_temp_data_file_path=/dev/shm/ibtmp1:12M:autoextend
```

Note: depending on container runtime this space may be limited.

Creating a replication pair

`MARIADB_REPLICATION_USER` / `MARIADB_REPLICATION_PASSWORD` specify the authentication for the connection. The `MARIADB_MASTER_HOST` is the indicator that it is a replica and specifies the container aka hostname, of the master.

A `docker-compose.yml` example:

```
version: "3"
services:
  master:
    image: mariadb:latest
    command: --log-bin --log-basename=mariadb
    environment:
      - MARIADB_ROOT_PASSWORD=password
      - MARIADB_USER=testuser
      - MARIADB_PASSWORD=password
      - MARIADB_DATABASE=testdb
      - MARIADB_REPLICATION_USER=repl
      - MARIADB_REPLICATION_PASSWORD=replicationpass
    healthcheck:
      test: ["CMD", "healthcheck.sh", "--connect", "--innodb_initialized"]
      interval: 10s
      timeout: 5s
      retries: 3
  replica:
    image: mariadb:latest
    command: --server-id=2 --log-basename=mariadb
    environment:
      - MARIADB_ROOT_PASSWORD=password
      - MARIADB_MASTER_HOST=master
      - MARIADB_REPLICATION_USER=repl
      - MARIADB_REPLICATION_PASSWORD=replicationpass
      - MARIADB_HEALTHCHECK_GRANTS=REPLICA MONITOR
    healthcheck:
      test: ["CMD", "healthcheck.sh", "--connect", "--replication_io", "--replication_sql", "--re
      interval: 10s
      timeout: 5s
      retries: 3
    depends_on:
      master:
        condition: service_healthy
```

InnoDB: Upgrade after a crash is not supported. The redo log was created with MariaDB X.Y.Z

This will show up in the error log as:

```
2022-05-23 12:29:20 0 [ERROR] InnoDB: Upgrade after a crash is not supported. The redo log was cr
2022-05-23 12:29:20 0 [ERROR] InnoDB: Plugin initialization aborted with error Generic error
```

This is attempting to start on a higher MariaDB version when the shutdown of the previous version crashed.

By crashed, it means the MariaDB was force killed or had a hard power failure. MariaDB, being a durable database, can recover from these, if started with the same version. The redo log however is a less stable format, so the recovery has to be on the same Major.Minor version, in this case 10.5. This error message is saying that you when from force killed MariaDB to a later version.

So whenever you encounter this message. Start with the again with the tag set to the version in the error message, like 10.5.4, or as the redo long format is consistent in the Major.Minor version 10.5 is sufficient. After this has been started correctly, cleanly shut the service down and it will be recovered.

The logs on shutdown should have a message like:

```
2023-11-06 10:49:23 0 [Note] InnoDB: Shutdown completed; log sequence number 84360; transaction i
2023-11-06 10:49:23 0 [Note] mariadbd: Shutdown complete
```

After you see this, you can update your MariaDB tag to a later version.

Every MariaDB start is a crash recovery

Do you get on every start:

```
db-1  | 2023-02-25 19:10:02 0 [Note] Starting MariaDB 10.11.2-MariaDB-1:10.11.2+maria~ubu2204-log
db-1  | 2023-02-25 19:10:02 0 [Note] mariadbd: Aria engine: starting recovery
db-1  | tables to flush: 3 2 1 0
db-1  |  (0.0 seconds);
db-1  | 2023-02-25 19:10:02 0 [Note] mariadbd: Aria engine: recovery done
...
db-1  | 2023-02-26 13:03:29 0 [Note] InnoDB: Initializing buffer pool, total size = 32.000GiB, ch
db-1  | 2023-02-26 13:03:29 0 [Note] InnoDB: Completed initialization of buffer pool
db-1  | 2023-02-26 13:03:29 0 [Note] InnoDB: File system buffers for log disabled (block size=512
db-1  | 2023-02-26 13:03:29 0 [Note] InnoDB: Starting crash recovery from checkpoint LSN=19379687
```

Container runtimes are assume to start and stop very quickly. Check the the shutdown logs. They may be a log like:

```
db-1  | 2023-02-26 13:03:17 0 [Note] InnoDB: Starting shutdown...
db-1  | 2023-02-26 13:03:17 0 [Note] InnoDB: Dumping buffer pool(s) to /var/lib/mysql/ib_buffer_p
db-1  | 2023-02-26 13:03:17 0 [Note] InnoDB: Restricted to 519200 pages due to innodb_buf_pool_du
db-1  | 2023-02-26 13:03:17 0 [Note] InnoDB: Buffer pool(s) dump completed at 230226 13:03:17
db-1 exited with code 0
```

Note that the logs didn't include the following messages:

```
db-1  | 2023-02-26 13:03:43 0 [Note] InnoDB: Shutdown completed; log sequence number 46590; trans
db-1  | 2023-02-26 13:03:43 0 [Note] mariadbd: Shutdown complete
```

As these messages aren't here, the container was killed off before it could just down cleanly. When this happens, the startup will be a crash recovery and you won't be able to upgrade your MariaDB instance (previous FAQ) to the next Major.Minor version.

Solution is to extend the timeout in the container runtime to allow MariaDB to complete its shutdown.

How do I create a MariaDB-backup of the data?

```
docker volume create backup
docker run --name mdb -v backup:/backup -v datavolume:/var/lib/mysql mariadb
docker exec mdb mariadb-backup --backup --target-dir=/backup/d --user root --password soverysecre
docker exec mdb mariadb-backup --prepare --target-dir=/backup/d
docker exec mdb sh -c '[ ! -f /backup/d/.my-healthcheck.cnf ] && cp /var/lib/mysql/.my-healthchec
docker exec --workdir /backup/d mdb tar -Jcf ../backup.tar.xz .
docker exec mdb  rm -rf /backup/d
```

How do I restore from a MariaDB-backup

With the backup prepared like previously:

```
docker run -v backup:/docker-entrypoint-initdb.d -v newdatavolume:/var/lib/mysql mariadb
```

2.1.2.14.6.11 MariaDB Container Cheat Sheet

Contents

Get the images

Images can be found on MariaDB Docker Hub.
To get the list of images run

```
$ docker images -a
```

Create the network

```
$ docker network create mynetwork
```

It is good practice to create the container network and attach the container to the network.

```
Start the container with server options
```

To start the container in the background with the MariaDB server image run:

```
$ docker run --rm --detach \
--env MARIADB_ROOT_PASSWORD=sosecret \
--network mynetwork \
--name mariadb-server \
mariadb:latest
```

Additionally |environment variables are also provided.

Get the list of running containers

```
$ docker ps
CONTAINER ID   IMAGE            COMMAND                  CREATED          STATUS
PORTS      NAMES
ad374ec8a272   mariadb:latest   "docker-entrypoint.s…"   3 seconds ago    Up 1 second
3306/tcp   mariadb-server
```

Note: specify the flag **-a** in case you want to see all containers

Start the client from the container

To start the *mariadb* client inside the created container and run specific commands, run the following:

```
$ docker exec -it mariadb-server mariadb -psosecret -e "SHOW PLUGINS"
```

Inspect logs of a container

```
$ docker logs mariadb-server
```

In the logs you can find status information about the server, plugins, generated passwords, errors and so on.

Restart the container

```
$ docker restart mariadb-server
```

Run commands within the container

```
$ docker exec -it mariadb-server bash
```

Use a volume to specify configuration options

```
$ docker run --detach --env MARIADB_USER=anel \
  --env MARIADB_PASSWORD=anel \
  --env MARIADB_DATABASE=my_db \
  --env MARIADB_RANDOM_ROOT_PASSWORD=1 \
  --volume $PWD/my_container_config:/etc/mysql/conf.d:z \
  --network mynetwork \
  --name mariadb-server1 \
   mariadb:latest
```

One can specify custom configuration files through the **/etc/mysql/conf.d** volume during container startup.

Use a volume to specify grants during container start

```
$ docker run --detach --env MARIADB_USER=anel\
  --env MARIADB_PASSWORD=anel \
  --env MARIADB_DATABASE=my_db \
  --env MARIADB_RANDOM_ROOT_PASSWORD=1 \
  --volume $PWD/my_init_db:/docker-entrypoint-initdb.d \
  --network mynetwork \
  --name mariadb-server1 \
   mariadb:latest
```

User created with the environment variables has full grants only to the *MARIADB_DATABASE*. In order to override those grants, one can specify grants to a user, or execute any SQL statements from host file to **docker-entrypoint-initdb.d**. In the *local_init_dir* directory we can find the file, created like this:

```
$ echo "GRANT ALL PRIVILEGES ON *.* TO anel;" > my_init_db/my_grants.sql
```

2.1.2.14.6.12 Using Healthcheck.sh

The healthcheck.sh script is part of the Docker Official Images of MariaDB Server. The script is part of the respository of the Docker Official Image of MariaDB Server.

The script processes a number of argument and tests, together, in strict order. Arguments pertaining to a test must occur before the test name. If a test fails, no further processing is performed. Both arguments and tests begin with a double-hyphen.

By default, (since 2023-06-27), official images will create healthcheck@localhost, healthcheck@127.0.0.1, healthcheck@::1 users with a random password and USAGE privileges. `MARIADB_HEALTHCHECK_GRANTS` can be used for `--replication` where additional grants are required. This is stored in .my-healthcheck.cnf in the datadir of the container and passed as the `--defaults-extra-file` to the healthcheck.sh script if it exists. The `.my-healthcheck.cnf` also sets `protocol=tcp` for the `mariadb` so `--connect` is effectively there on all tests.

The `MARIADB_MYSQL_LOCALHOST_USER=1, MARIADB_MYSQL_LOCALHOST_GRANTS` environment variables can also be used, but with the creation of the healthcheck user, these are backwards compatible.

Tests

--connect

This is active when a external user can connect to the TCP port of MariaDB Server. This strictly tests just the TCP connection and not if any authentication works.

--innodb_initialized

This test is true when InnoDB has completed initializing. This includes any rollback or crash recovery that may be occurring in the background as MariaDB is starting.

The connecting user must have USAGE privileges to perform this test.

--innodb_buffer_pool_loaded

This indicates that the buffer pool dump previously saved has been completed loaded into the InnoDB Buffer Pool and as such the server has a hot cache ready for use. This checks the innodb_buffer_pool_load_status for a "complete" indicator.

This test doesn't check if innodb-system-variables/#innodb_buffer_pool_load_at_startupinnodb_buffer_pool_load_at_startup is set at startup.

The connecting user must have USAGE privileges to perform this test.

--galera_online

This indicates that the galera node is online by the wsrep_local_state variable. This includes states like "joining" and "donor" where it cannot serve SQL queries.

The connecting user must have USAGE privileges to perform this test.

--replication

This tests a replica based on the `--replication_*` parameters. The replica test must pass all of the subtests to be true. The subtests are:

- io - the IO thread is running
- sql - the sql thread is running
- seconds_behind_master - the replica is less than X seconds behind the master.
- sql_remaining_delay - the delayed replica is less than X seconds behind the master's execution of the same SQL.

These are tested for all connections, if `--replication_all` is set (default), or `--replication_name`.

The connecting user must have REPLICATION_CLIENT if using a version less than MariaDB 10.5, or REPLICA MONITOR for MariaDB 10.5 or later.

--mariadbupgrade

This healthcheck indicates that the mariadb is upgrade to the current version.

Parameters

--replication_all

Checks all replication sources

--replication_name=n

Sets the multisource connection name tested. Unsets `--replication_all`.

--replication_io

IO thread is running

--replication_sql

SQL thread is running

--replication_seconds_behind_master=n

Less than or equal this seconds of delay

--replication_sql_remaining_delay=n

Less than or equal this seconds of remaining delay

--su=n

Change to this user. Can only be done once as the root user is default for healthchecks.

--su-mysql

Change to the `mysql` unix user. Like `--su` this respawns the script so will reset all parameters. Should be the first argument. The `MARIADB_MYSQL_LOCALHOST_USER=1` environment variable is designed around usage here.

--datadir=n

For the `--mariadbupgrade` test where the upgrade file is located.

--no-defaults --defaults-file --defaults-extra-file --defaults-group-suffix

These are passed to mariadb shell for all tests except `--mariadbupgrade`

Examples

```
healthcheck.sh --su-mysql --connect --innodb_initialized
```

Switch to `mysql` user, and check if can connect and the innodb is initialized.

```
healthcheck.sh --su-mysql --connect --replication_io --replication_sql --replication_seconds_behi
```

Switch to `mysql` user, check if connections can be made, for the replication channel "archive", ensure io and sql threads are running and the seconds behind master < 600 seconds and the sql remaining delay < 30 seconds. For the "channel1", the seconds behind master is limit to 10 seconds maximum.

2.1.2.14.6.13 Docker and AWS EC2

This process shows how to deploy MariaDB in a Docker container running on an EC2 instance. First we'll create the EC2 VM, then we'll deploy Docker to it. After that, we'll pull the MariaDB Docker image which we'll use to create a running container with a MariaDB instance. Finally, we'll load a sample database into the MariaDB instance.

Create a VM in AWS EC2

1. Install MariaDB client on your local machine, either bundled with Maria DB server or standalone.

2. Login to AWS, navigate to EC2 service home

3. Choose Region for EC2 in the upper right corner of the console

4. Launch (1) Instance, giving instance a name (e.g. mrdb-ubuntu-docker-use1) and create or re-use a key pair

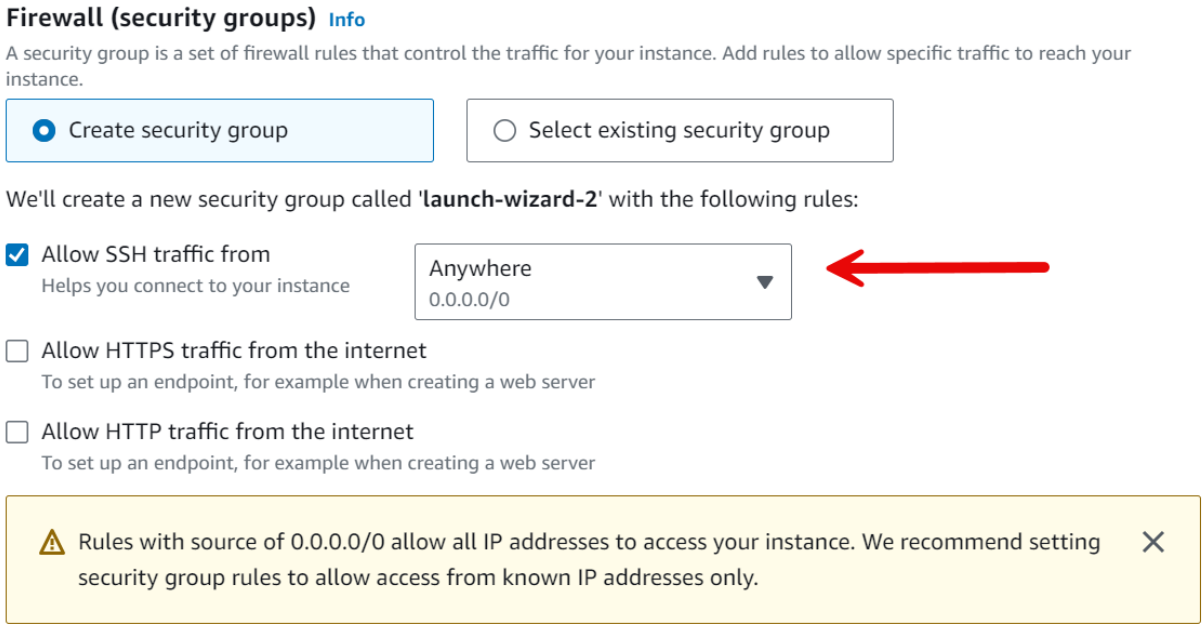
5. Choose Ubuntu 22.04 or similar free tier instance

6. Choose hardware, t2.micro or similar free tier instance

7. Create Key Pair with name (e.g. mrdb-docker-aws-pk.pem if using openSSH at the command line, or mrdb- docker-aws-pk..ppk for use with programs like PuTTY.)

9. Create or select a security group where SSH is allowed from anywhere 0.0.0.0/0. If you'd like to make this more secure, it

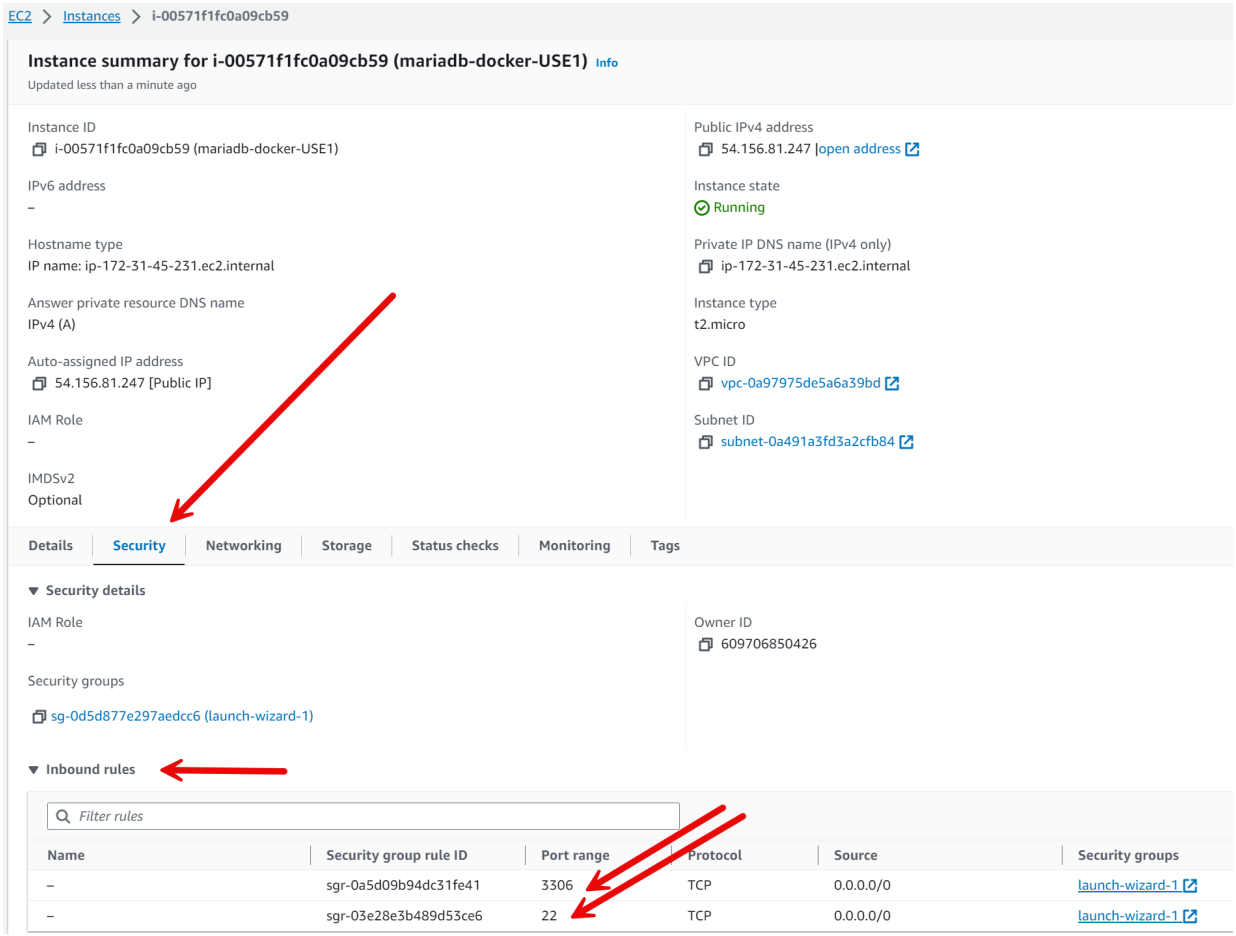
can be restricted to a specific IP address or CIDR block.



10. Accept remaining instance creation defaults and click “launch instance”.

11. Save the *.pem or *.ppk keyfile on your local hard drive when prompted. You will need it later. If you’re on Linux, don’t forget to change permissions on the downloaded *.pem / *.ppk key file: $ chmod 400 mrdb-docker-pk.pem

12. Click into the instance summary (EC2 > Instances > Instance ID) and click on the “security” tab towards the bottom.

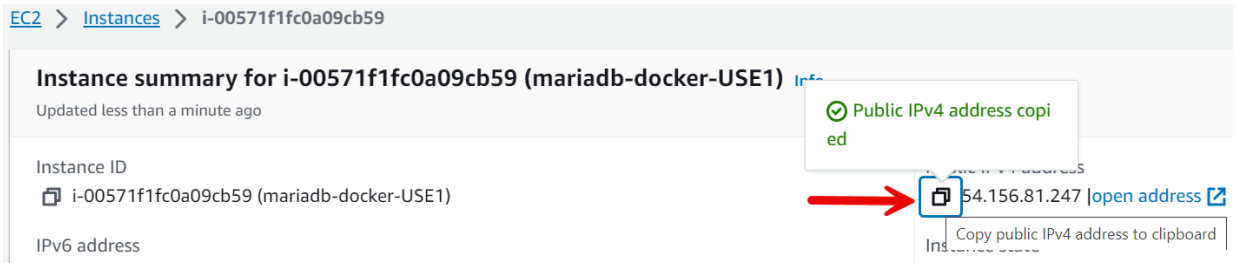


13. In the relevant security group for your instance, Create an inbound rule so that TCP port 3306 is open, allowing external connections to Maria DB (like your local command line client for MariaDB). Double check that port 22 is open while you're there for SSH.

Install Docker on the EC2 VM

For more detailed instructions, refer to Installing and Using MariaDB via Docker

14. Back in the instance summary (EC2 > Instances > Instance ID), copy the public IP (e.g. ww.xx.yyy.zzz)



15. Open terminal window, navigate to the directory with private key (*.pem or *.ppk) file and start a SSH remote shell session by typing:

$ ssh -i mrdb-docker-pk.pem ubuntu@ww.xx.yyy.zzz

(switch ww.xx.yyy.zzz for your IP address from step 14).

16. Are you sure you want to continue connecting (yes/no/[fingerprint])? Say yes

17. Escalate to root

$ sudo su

18. Install Docker

$ curl -fsSL https://get.docker.com | sudo sh

Pull the MariaDB Docker image and create the container

19. Pull MariaDB Docker image

$ docker pull mariadb:lts

20. Start MDRB docker process

at your terminal / command line, type:

$ docker run --detach --name mariadb-docker -v \Users\YouUID\Documents\YourDirName:/var/lib/mysql:Z -p 3306:3306 -e MARIADB_ROOT_PASSWORD=yoursecurepassword mariadb:lts

The -v flag mounts a directory that you choose as /var/lib/mysql will ensure that the volume is persistent. Windows file paths like C:\Users\YouUID\Documents\YourDirName should be represented as above. Linux file paths should also be absolute vs. relative. Obviously replace the root password with something that is a bit more secure than you see in this example for anything other than development purposes.

21. Shell into container

$ docker exec -it mariadb-docker bash

22. Login to MRDB inside container

Using the root password specified in step 20, type:

$ mariadb -pyoursecurepassword

23. Setup admin account with permission for remote connection, configure access control

MariaDB [(none)]> CREATE USER 'admin'@'%' IDENTIFIED BY 'admin';

MariaDB [(none)]> GRANT ALL ON *.* to 'admin'@'%' WITH GRANT OPTION;

MariaDB [(none)]> SHOW GRANTS FOR admin;

Obviously replace these passwords with something that is a bit more secure than you see in this example for anything other than development purposes.

24. Setup service account for your app with permission for remote connection, configure access control

MariaDB [(none)]> CREATE USER 'yourappname'@'%' IDENTIFIED BY 'yoursecurepassword';

MariaDB [(none)]> GRANT INSERT, UPDATE, DELETE ON *.* to 'yourappname'@'%';

MariaDB [(none)]> SHOW GRANTS FOR yourappname;

Obviously replace these passwords with something that is a bit more secure than you see in this example for anything other than development purposes.

25. Load up your database from your preexisting SQL script that contains CREATE DATABASE; USE DATABASE; and CREATE TABLE statements.

In a new local terminal window, not your SSH session, change directory to the directory containing your database creation script, say, init.sql in this example. Type:

$ mariadb --host=ww.xx.yyy.zzz --port=3306 --user=admin --password=admin -e “SOURCE init.sql”

(switch ww.xx.yyy.zzz for your IP address from step 14).

2.1.2.14.6.14 Docker and Google Cloud

This process shows how to deploy MariaDB in a Docker container running on an GCE instance. First we'll create the GCE VM, then we'll deploy Docker to it. After that, we'll pull the MariaDB Docker image which we'll use to create a running container with a MariaDB instance. Finally, we'll load a sample database into the MariaDB instance.

Create a VM in Google Cloud Compute Engine

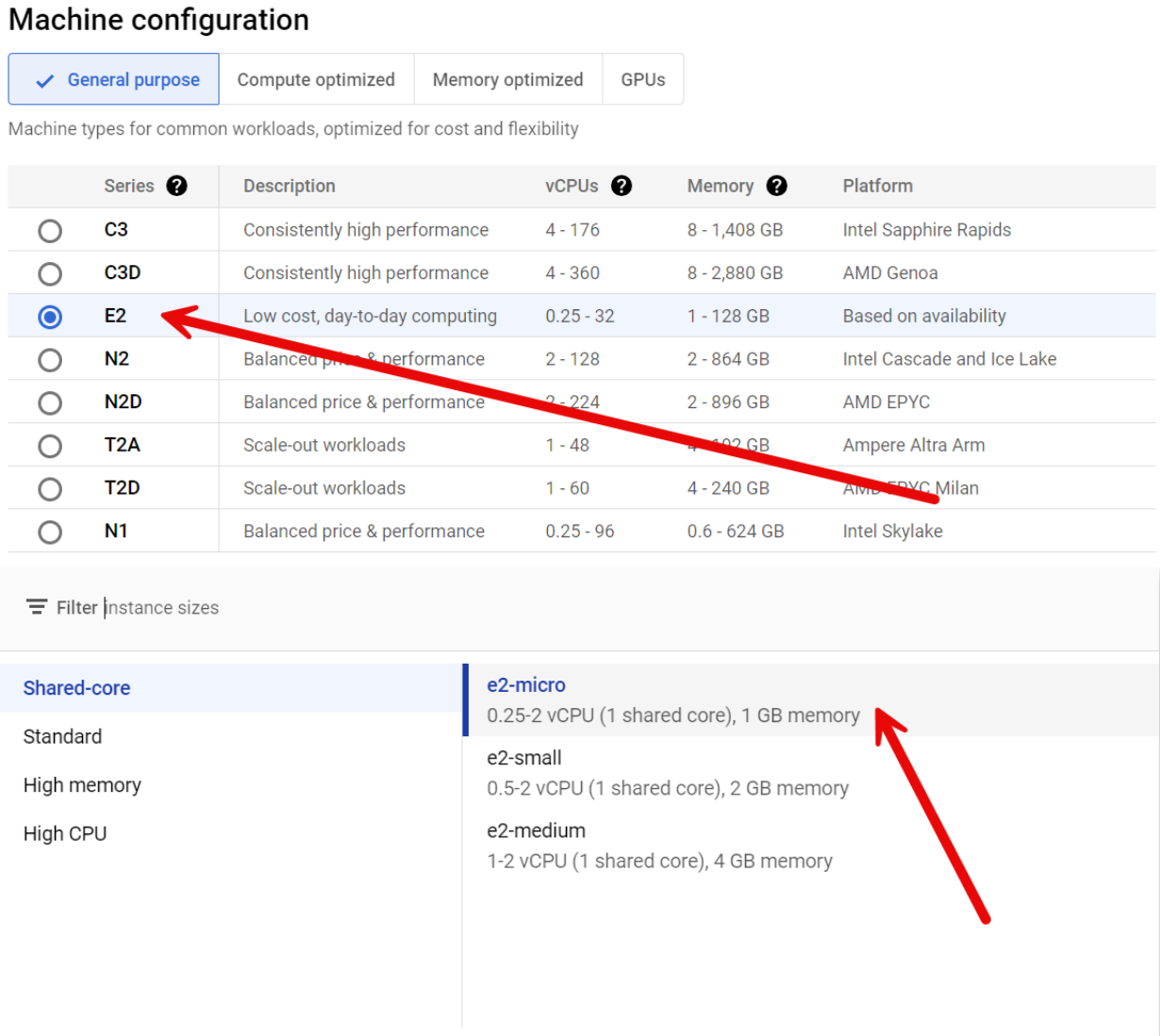
1. Install MariaDB client on your local machine, either bundled with Maria DB server or standalone.

2. Login to Google Cloud, navigate to VM instances

3. Enable Compute Engine API if you haven't already.

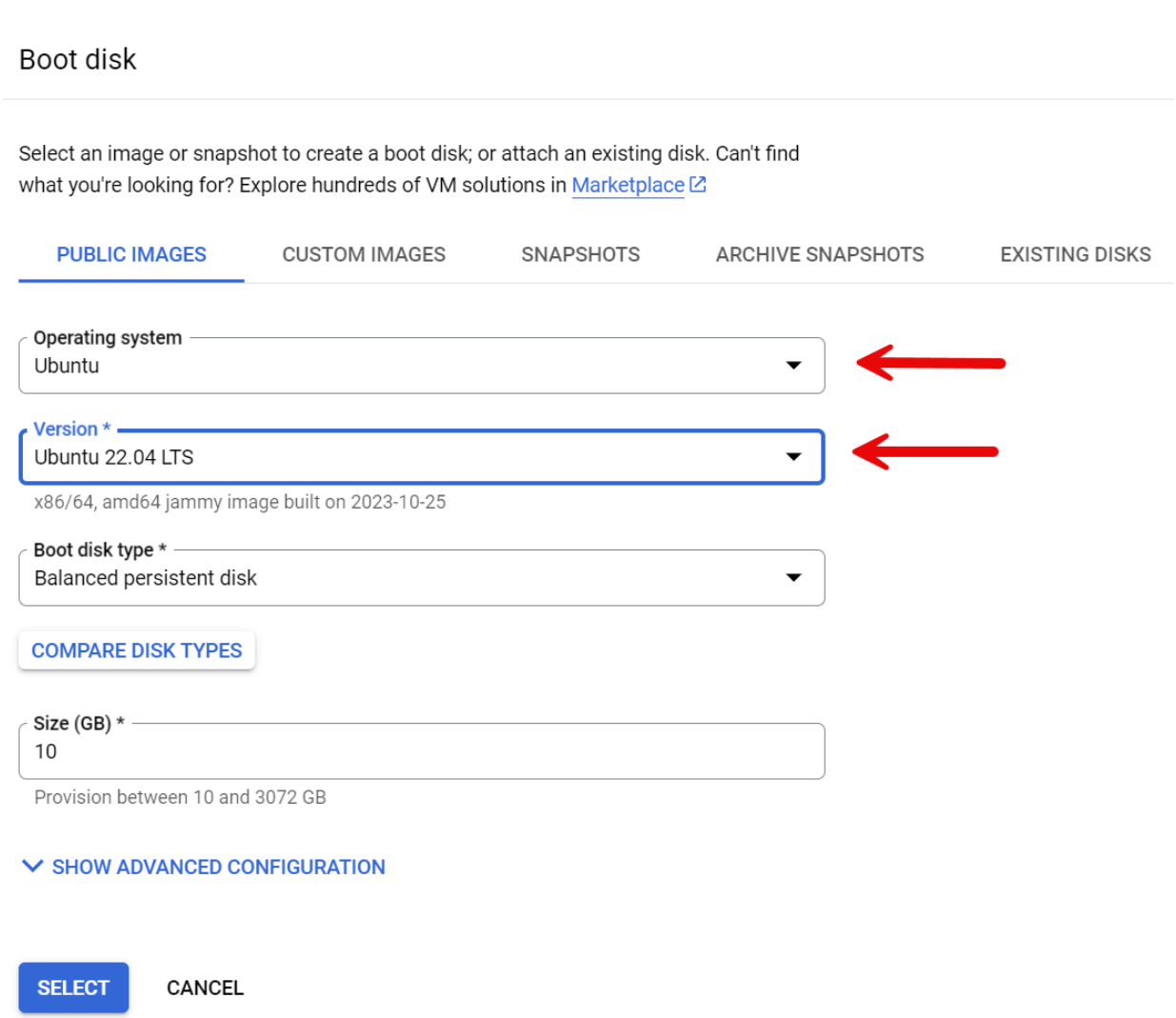
4. Click create instance, give instance a name (e.g. mrdb-ubuntu-docker-use1b), choose a region and zone.

5. Machine configuration: Choose general-purpose / E2 micro

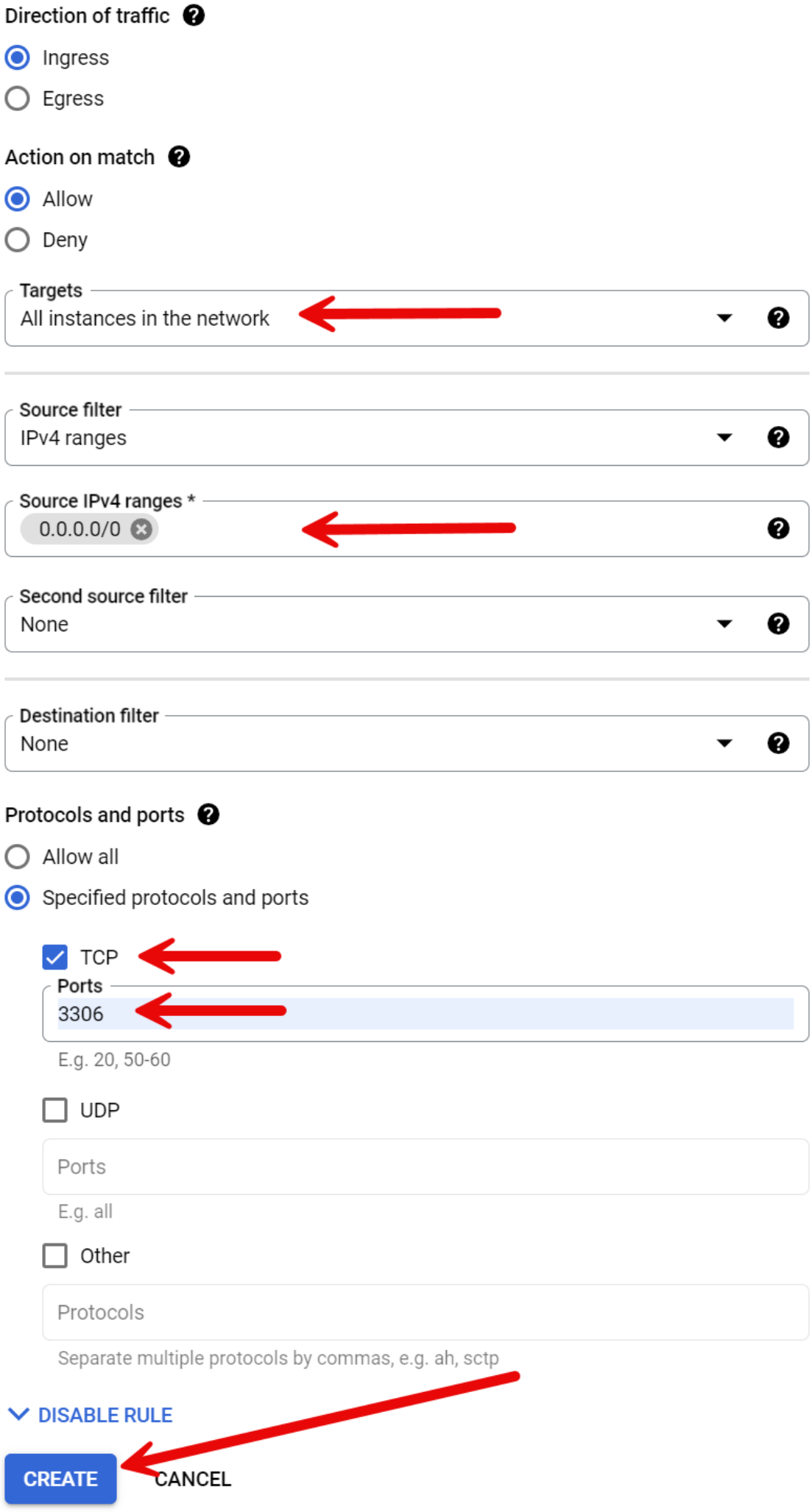


6. Boot Disk > Change

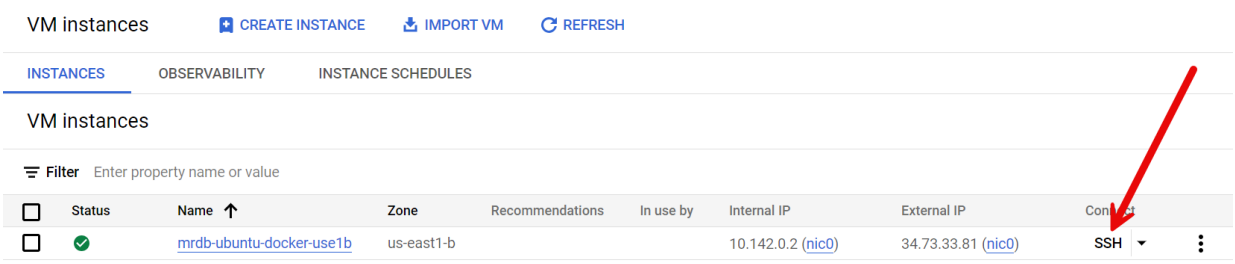
Switch the operating system to a modern Ubuntu release x86/64 CPU architecture, or similar free tier offering.



7. Create a firewall rule in the Firewall Policies section of the console. After naming it, change the targets, add 0.0.0.0/0 as a source IP range, and open TCP port 3306. Then Click create.



8. Connect using Google Cloud's built in browser SSH. Accept all prompts for authorization.



Install Docker on the GCE VM

For more detailed instructions, refer to Installing and Using MariaDB via Docker

9. Escalate to root Escalate to root

$ sudo su

10. Install Docker

$ curl -fsSL https://get.docker.com | sudo sh

11. Pull Docker image

$ docker pull mariadb:lts

12. Start MDRB docker process

at your terminal / command line, type:

$ docker run --detach --name mariadb-docker -v \Users\YouUID\Documents\YourDirName:/var/lib/mysql:Z -p 3306:3306 -e MARIADB_ROOT_PASSWORD=yoursecurepassword mariadb:lts

The -v flag mounts a directory that you choose as /var/lib/mysql will ensure that the volume is persistent. Windows file paths like C:\Users\YouUID\Documents\YourDirName should be represented as above. Linux file paths should also be absolute vs. relative. Obviously replace the root password with something that is a bit more secure than you see in this example for anything other than development purposes.

13. Shell into container $ docker exec -it mariadb-docker bash

14. Login to MRDB inside container

Using the root password specified in step 12, type:

$ mariadb -pyoursecurepassword

15. Setup admin account with permission for remote connection, configure access control Execute these SQL commands in sequence:

MariaDB [(none)]> CREATE USER 'admin'@'%' IDENTIFIED BY 'admin';

MariaDB [(none)]> GRANT ALL ON *.* to 'admin'@'%' WITH GRANT OPTION;

MariaDB [(none)]> SHOW GRANTS FOR admin;

Obviously replace these passwords with something that is a bit more secure than you see in this example for anything other than development purposes.

16. Setup service account for your app with permission for remote connection, configure access control Execute these SQL commands in sequence:

MariaDB [(none)]> CREATE USER 'yourappname'@'%' IDENTIFIED BY 'yoursecurepassword';

MariaDB [(none)]> GRANT INSERT, UPDATE, DELETE ON *.* to 'yourappname'@'%';

MariaDB [(none)]> SHOW GRANTS FOR yourappname;

Obviously replace these passwords with something that is a bit more secure than you see in this example for anything other than development purposes.

17. Load up your database from your preexisting SQL script that contains CREATE DATABASE; USE DATABASE; and CREATE TABLE statements.

Copy the external IP address of your VM instance from the Console in the VM instances list.

In a new local terminal window, not your SSH session, change directory to the directory containing your database creation script, say, init.sql in this example.

Type: $ mariadb --host=ww.xx.yyy.zzz --port=3306 --user=admin --password=admin -e “SOURCE init.sql” (switch ww.xx.yyy.zzz for your IP address from step 17.

2.1.2.14.6.15 Docker and Microsoft Azure

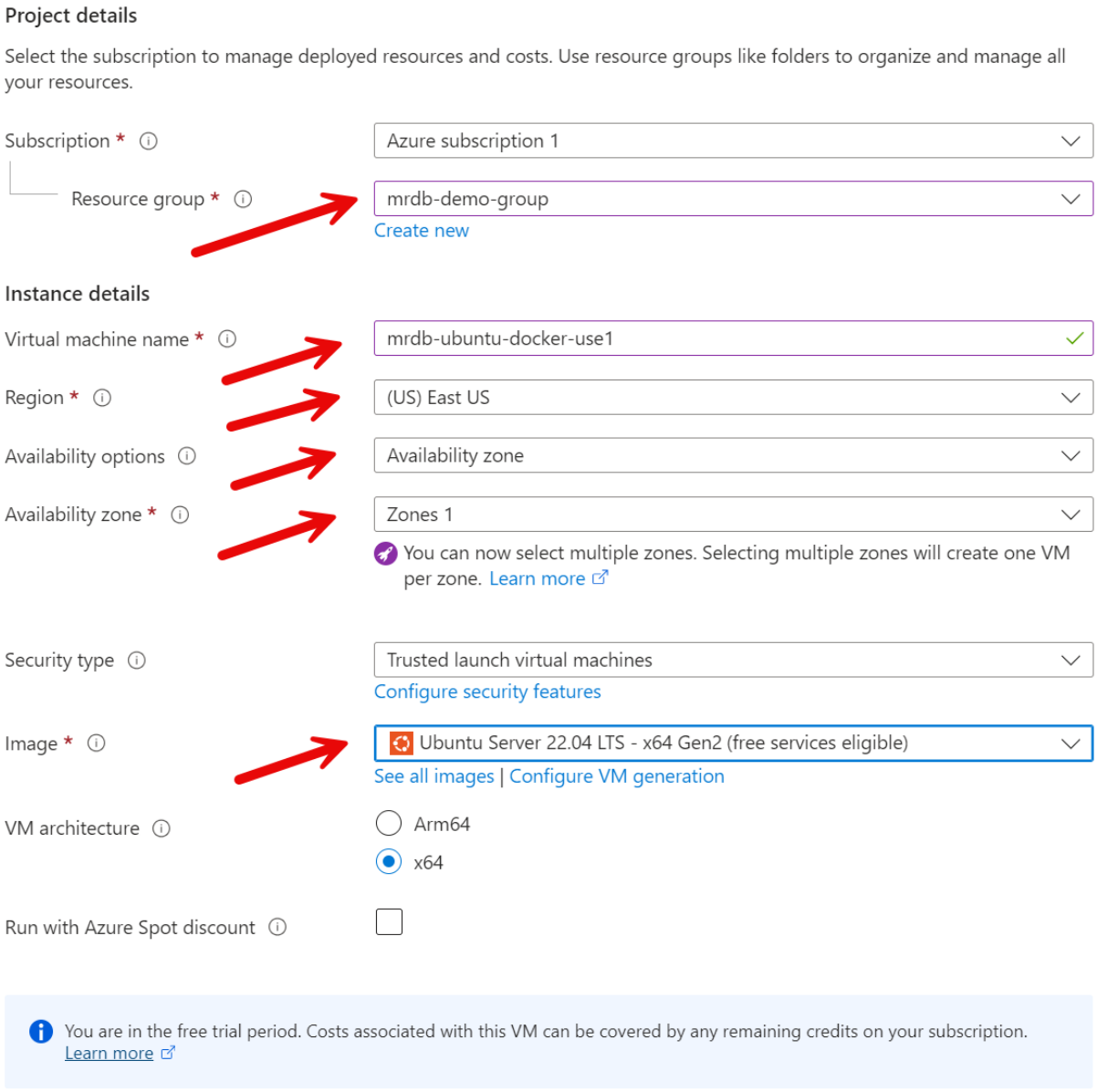
This process shows how to deploy MariaDB in a Docker container running on an Azure VM instance. First we'll create the Azure VM, then we'll deploy Docker to it. After that, we'll pull the MariaDB Docker image which we'll use to create a running container with a MariaDB instance. Finally, we'll load a sample database into the MariaDB instance.

Create a VM in Azure

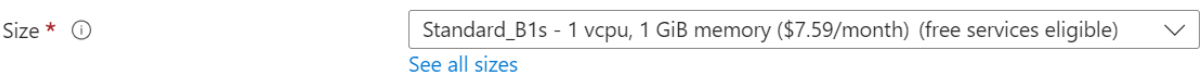
1. Install MariaDB client on your local machine, either bundled with Maria DB server or standalone.

2. Login to Azure, navigate to Azure Virtual Machine

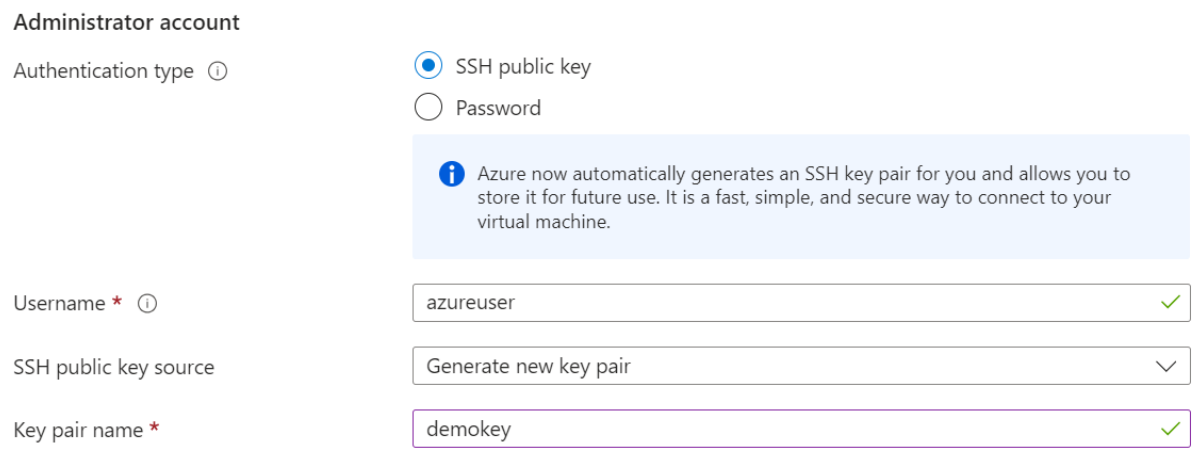
3. Create VM. Give the VM a name (e.g. mrdb-ubuntu-docker-use1), and create new or use an existing resource group. Selection region and availability zone, and choose Ubuntu 22.04 LTS x64 (free services eligible).



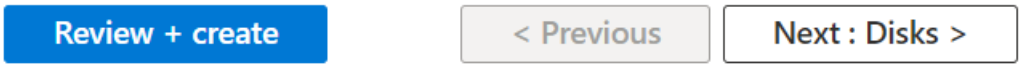
4. Choose the VM instance size, like a B1s or similar free tier. Note that Azure free works on a credit based system for new accounts



5. Configure an administrator account and generate a new key pair, and give the key pair a name.

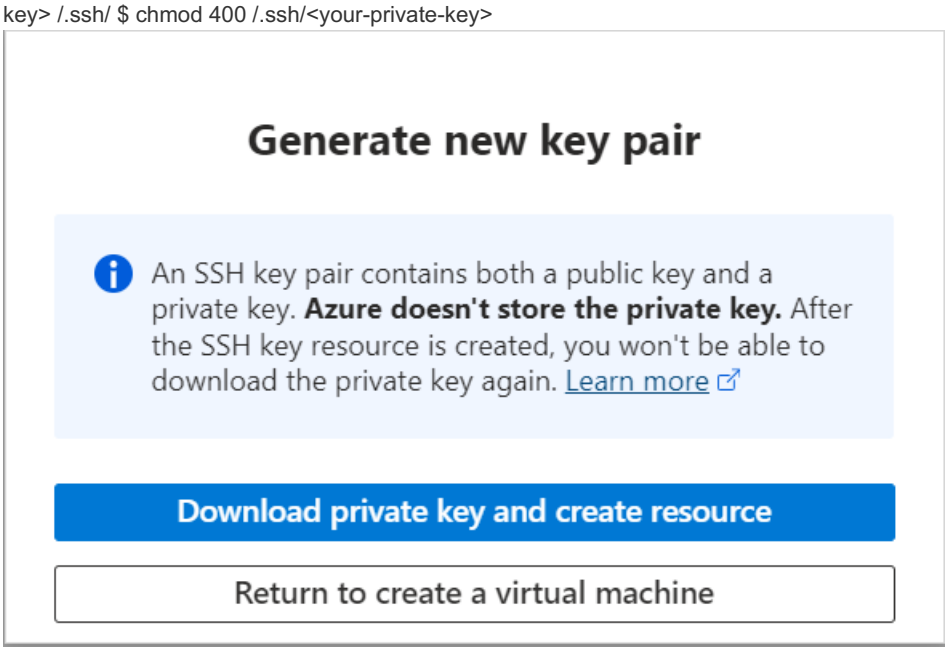


6. Click "Review + Create" at the very bottom of the "create virtual machine" page to create the VM.

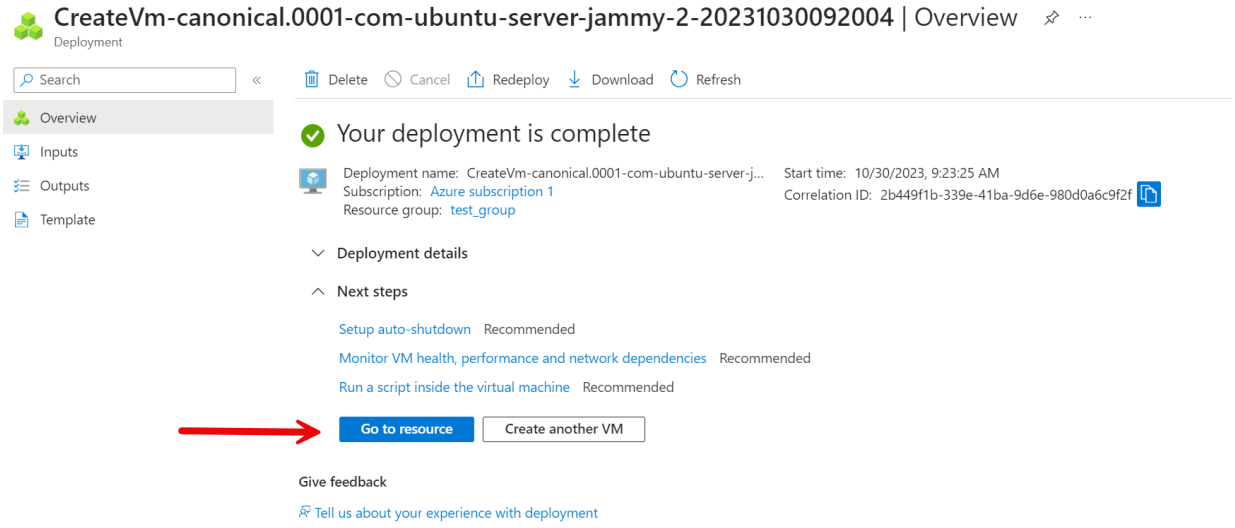


7. Download the SSH keys and them in a safe place, you will need them later. For this example, let's name the key file mrdb-docker-pk.pem.

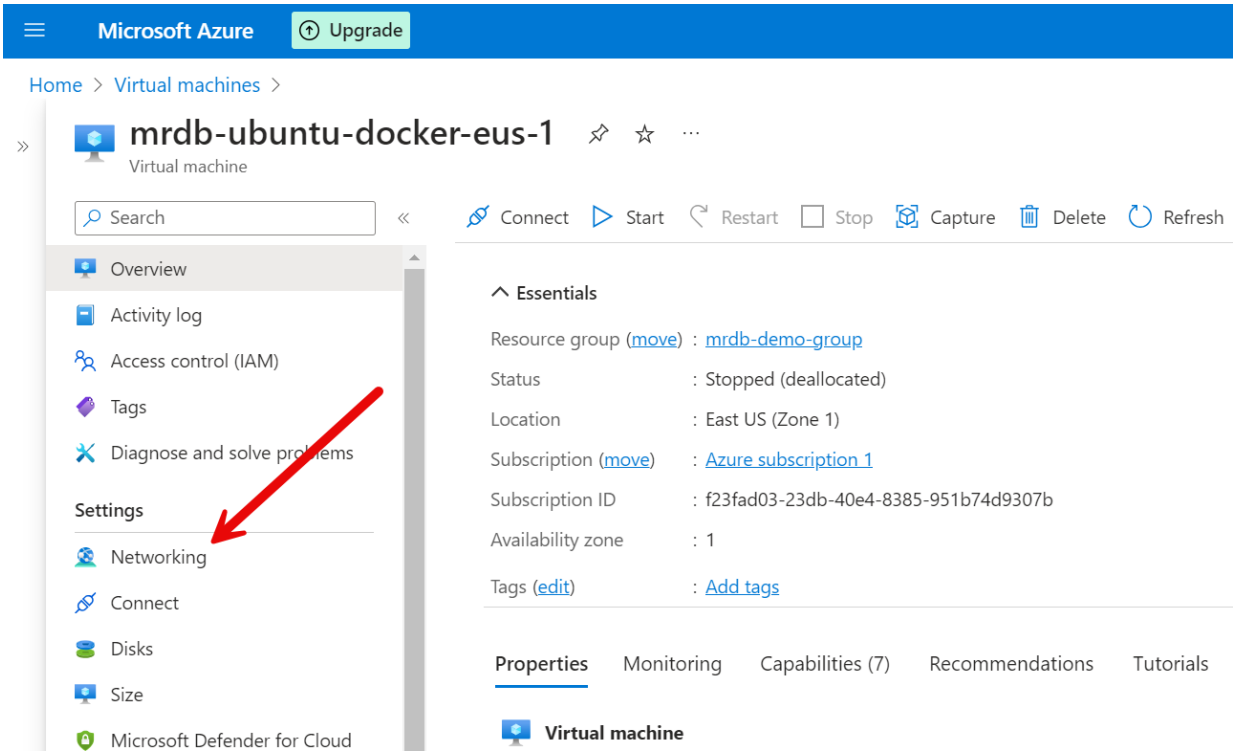
If your local machine is Linux or you are using WSL on Windows, open a terminal window and: $ mv /mnt/c/<your-private-key> /.ssh/ $ chmod 400 /.ssh/<your-private-key>



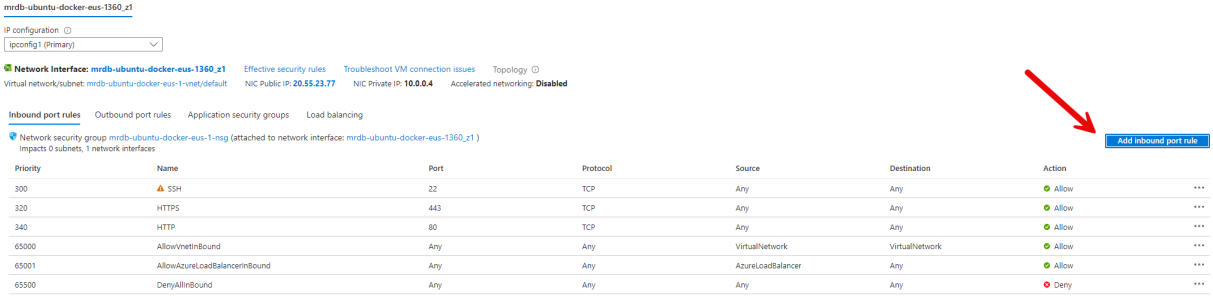
8. Once the VM is deployed, "click to resource" to get back to the virtual machine's overview page.



9. From the overview page, the left-hand navigation, choose settings > networking.



10. Click "add inbound port rule"



11. Configure the port rule to allow port TCP 3306 inbound (mySQL) so that you can make external connections from your local Maria DB command line client, to the dockerized Maria DB instance in your Azure Linux VM.

AllowCidrBlockCustom3306Inbound

mrdb-ubuntu-docker-eus-1-nsg

Source

Any

Source port ranges *

*

Destination

Any

Service

MySQL

Destination port ranges

3306

Protocol

- Any
- TCP
- UDP
- ICMP

Action

- Allow
- Deny

Priority *

350

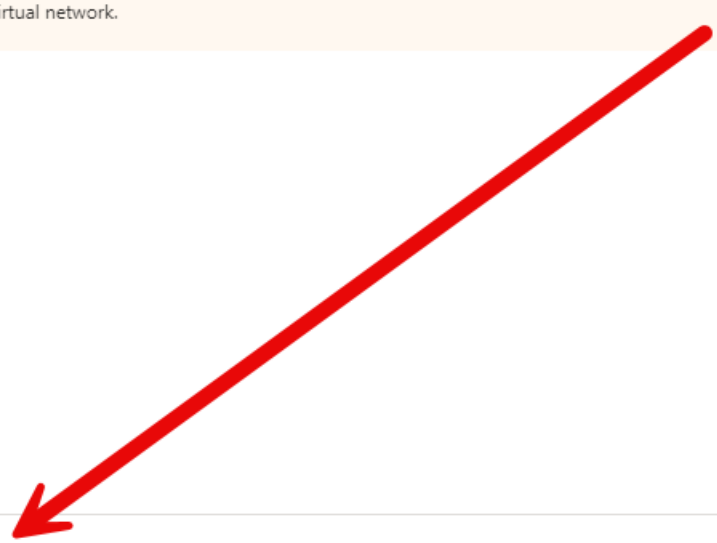
Name

AllowCidrBlockCustom3306Inbound

Description

MariaDB

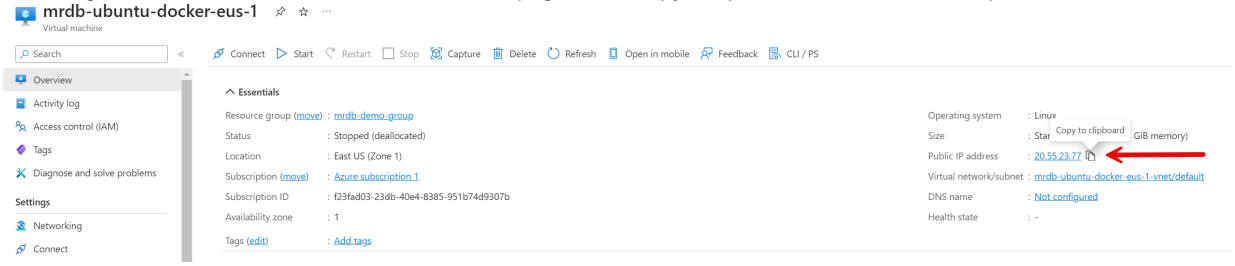
Mysql DB port 3306 is exposed to the Internet. We do not recommend exposing database ports to the Internet and suggest only exposing them to your front-end tier inside your virtual network.



Save | Cancel | Give feedback

Save

12. Navigate back to the virtual machine's overview page. Then copy the public IP address to the clipboard.



Install Docker on the Azure VM

For more detailed instructions, refer to Installing and Using MariaDB via Docker

16. Open terminal window, referencing the path to the private key (*.pem or *.ppk) file, and start a SSH remote shell session by typing:

$ ssh -i /.ssh/mrdb-docker-pk.pem azureuser@ww.xx.yyy.zzz

(switch ww.xx.yyy.zzz for your IP address from step 12, and replace "mrdb-docker-pk.pem" with your keyfile name if you chose something different).

If you forget your administrator account details, simply go to the left-hand navigation and choose settings > connect, and Azure will display the public IP address, admin username, and port for you.

17. Are you sure you want to continue connecting (yes/no/[fingerprint])? Say yes

18. Escalate to root

$ sudo su

19. Microsoft Azure on two machines come with docker preinstalled. For any reason you need to reinstall it , chose another machine type is not have docker preinstalled, you can install docker inside your SSH session with cURL by typing:

$ curl -fsSL https://get.docker.com | sudo sh

Pull the MariaDB Docker image and create the container

20. Pull MariaDB Docker image

$ docker pull mariadb:lts

21. Start MDRB docker process

at your terminal / command line, type:

$ docker run --detach --name mariadb-docker -v \Users\YouUID\Documents\YourDirName:/var/lib/mysql:Z -p 3306:3306 -e MARIADB_ROOT_PASSWORD=yoursecurepassword mariadb:lts

The -v flag mounts a directory that you choose as /var/lib/mysql will ensure that the volume is persistent. Windows file paths like C:\Users\YouUID\Documents\YourDirName should be represented as above. Linux file paths should also be absolute vs. relative. Obviously replace the root password with something that is a bit more secure than you see in this example for anything other than development purposes.

22. Shell into container

$ docker exec -it mariadb-docker bash

23. Login to MRDB inside container

Using the root password specified in step 20, type:

$ mariadb -pyoursecurepassword

24. Setup admin account with permission for remote connection, configure access control

MariaDB [(none)]> CREATE USER 'admin'@'%' IDENTIFIED BY 'admin';

MariaDB [(none)]> GRANT ALL ON *.* to 'admin'@'%' WITH GRANT OPTION;

MariaDB [(none)]> SHOW GRANTS FOR admin;

Obviously replace these passwords with something that is a bit more secure than you see in this example for anything other than development purposes.

25. Setup service account for your app with permission for remote connection, configure access control

MariaDB [(none)]> CREATE USER 'yourappname'@'%' IDENTIFIED BY 'yoursecurepassword';

MariaDB [(none)]> GRANT INSERT, UPDATE, DELETE ON *.* to 'yourappname'@'%';

MariaDB [(none)]> SHOW GRANTS FOR yourappname;

Obviously replace these passwords with something that is a bit more secure than you see in this example for anything other than development purposes.

26. Load up your database from your preexisting SQL script that contains CREATE DATABASE; USE DATABASE; and CREATE TABLE statements.

In a new local terminal window, not your SSH session, change directory to the directory containing your database creation script, say, init.sql in this example. Then type:

$ mariadb --host=ww.xx.yyy.zzz --port=3306 --user=admin --password=admin -e “SOURCE init.sql”

(switch ww.xx.yyy.zzz for your IP address from step 12).

2.1.2.14.7 Kubernetes and MariaDB

General information and hints on how to deploy MariaDB Kubernetes containers.

Kubernetes is an open source containers orchestration system. It automates deployments, horizontal scaling, configuration and operations. It is often referred to as K8s.

Kubernetes Overview for MariaDB Users

An overview of Kubernetes and how it works with MariaDB.

Kubernetes Operators for MariaDB

An overview of Kubernetes operators that can be used with MariaDB

2.1.2.14.8 Kubernetes Overview for MariaDB Users

Kubernetes, or K8s, is software to orchestrate containers. It is released under the terms of an open source license, Apache License 2.0.

Kubernetes was originally developed by Google. Currently it is maintained by the Cloud Native Computing Foundation (CNCF), with the status of Graduated Project.

For information about how to setup a learning environment or a production environment, see Getting started in Kubernetes documentation.

Contents

Architecture

Kubernetes runs in a **cluster**. A cluster runs a **workload**: a set of servers that are meant to work together (web servers, database servers, etc).

A Kubernetes cluster consists of the following components:

- **Nodes** run containers with the servers needed by our applications.
- **Controllers**constantly check the cluster nodes current state, and compare it with the desired state.
- A **Control Plane** is a set of different components that store the cluster desired state and take decisions about the nodes. The Control Plane provides an API that is used by the controllers.

For more information on Kubernetes architecture, see Concepts and Kubernetes Components in Kubernetes documentation.

Nodes

A node is a system that is responsible to run one or more pods. A pod is a set of containers that run a Kubernetes workload or part of it. All containers that run in the same pod are also located on the same node. Usually identical pods run on different nodes for fault tolerance.

For more details, see Nodes in the Kubernetes documentation.

Every node must necessarily have the following components:

- **kubelet**
- **kube-proxy**
- A **container runtime**

kubelet

kubelet has a set of **PodSpecs** which describe the desired state of pods. It checks that the current state of the pods matches the desired state. It especially takes care that containers don't crash.

kube-proxy

In a typical Kubernetes cluster, several containers located in different pods need to connect to other containers, located in the same pods (for performance and fault tolerance reasons). Therefore, when we develop and deploy an application, we can't know in advance the IPs of the containers to which it will have to connect. For example, an application server may need to connect to MariaDB, but the MariaDB IP will be different for every pod.

The main purpose of kube-proxy is to implement the concept of Kubernetes **services**. When an application needs to connect to MariaDB, it will connect to the MariaDB service. kube-proxy will receive the request and will redirect it to a running MariaDB container in the same pod.

Container Runtime

Kubernetes manages the containers in a pod via a container runtime, or container manager, that supports the Kubernetes Container Runtime Interface (CRI). Container runtimes that meet this requisite are listed in the Container runtimes page in the Kubernetes documentation. More information about the Container Runtime Interface can be found on GitHub.

Originally, Kubernetes used Docker as a container runtime. This was later deprecated, but Docker images can still be used using any container runtime.

Controllers

Controllers constantly check if there are differences between the pod's current state and their desired state. When differences are found, controllers try to fix them. Each node type controls one or more resource types. Several types of controllers are needed to run a cluster.

Most of the actions taken by the controllers user the API server in the Control Plane. However, this is not necessarily true for custom controllers. Also, some actions cannot be performed via the Control Plane. For example, if some nodes crashed, adding new nodes involves taking actions outside of the Kubernetes cluster, and controllers will have to do this themselves.

It is possible to write custom controllers to perform checks that require knowledge about a specific technology. For example, a MariaDB custom controller may want to check if replication is working by issuing SHOW REPLICA STATUS commands. This logic is specific to the way MariaDB works, and can only be implemented in a customer controller. Custom controllers are usually part of operators.

For more information, see Controllers in the Kubernetes documentation.

Control Plane

The control plane consists of the following components.

For more information about the control plane, see Control Plane Components in Kubernetes documentation.

API Server

An API Server exposes API functions both internally and externally. It is essential to coordinate Kubernetes components so that they react to node's change of state, and it allows the user to send commands.

The default implementation of the API Server is kube-apiserver. It is able to scale horizontally and to balance the load

between its instances.

kube-controller-manager

Most controllers run in this component.

etcd

etcd contains all data used by a Kubernetes cluster. It is a good idea to take regular backups of etcd data.

kube-scheduler

When a new pod is created, kube-scheduler decides which node should host it. The decision is made based on several criteria, like the resource requirements for the pod.

cloud-controller-manager

cloud-controller-manager implements the logic and API of a cloud provider. It receives requests from the API Server and performs specific actions, like creating an instance in AWS. It also runs controllers that are specific to a cloud vendor.

Clients and Tools

Kubernetes comes with a set of tools that allow us to communicate with the API server and test a cluster.

kubectl

kubectl allows communication with the API server and run commands on a Kubernetes cluster.

kubeadm

kubeadm allows creating a Kubernetes cluster that is ready to receive commands from kubectl.

kind and minikube

These tools are meant to create and manage test clusters on a personal machine. They work on Linux, MacOS and Windows. kind creates a cluster that consists of Docker containers, therefore it requires Docker to be installed. minikube runs a single-node cluster on the local machine.

Kubernetes Resources and References

- Kubernetes website.
- Kubernetes on Wikipedia.
- Kubernetes organization on GitHub.
- OperatorHub.io
- Kubernetes Community Forums.
- (video) MariaDB database clusters on Kubernetes, by Pengfei Ma, at MariaDB Server Fest 2020.
- Series of posts by Anel Husakovic on the MariaDB Foundation blog:
  - Start MariaDB in K8s
  - MariaDB & K8s: Communication between containers/Deployments
  - MariaDB & K8s: Create a Secret and use it in MariaDB deployment
  - MariaDB & K8s: Deploy MariaDB and WordPress using Persistent Volumes
  - Create statefulset MariaDB application in K8s
  - MariaDB replication using containers
  - MariaDB & K8s: How to replicate MariaDB in K8s

Content initially contributed by Vettabase Ltd.

2.1.2.14.9 Kubernetes Operators for MariaDB

Operators basically instruct Kubernetes about how to manage a certain technology. Kubernetes comes with some default operators, but it is possible to create custom operators. Operators created by the community can be found on OperatorHub.io.

Contents

Custom Operators

Kubernetes provides a declarative API. To support a specific (i.e. MariaDB) technology or implement a desired behavior (i.e. provisioning a replica), we extend Kubernetes API. This involves creating two main components:

- A custom resource.
- A custom controller.

A custom resource adds an API endpoint, so the resource can be managed via the API server. It includes functionality to get information about the resource, like a list of the existing servers.

A custom controller implements the checks that must be performed against the resource to check if its state should be corrected using the API. In the case of MariaDB, some reasonable checks would be verifying that it accepts connections, replication is running, and a server is (or is not) read only.

MariaDB Operator

mariadb-operator is a Kubernetes operator that allows you to run and operate MariaDB in a cloud native way. It aims for declaratively managing your MariaDB using Kubernetes CRDs rather than imperative commands.

It's available in both Artifact Hub and Operator Hub and supports the following features:

- Provisioning highly configurable MariaDB servers
- Multiple HA modes supported: SemiSync Replication and Galera. Automatic primary failover.
- Take and restore backups. Scheduled backups. Backup rotation
- PVCs and Kubernetes volumes (i.e. NFS) backup storage
- Bootstrap new instances from backups and volumes (i.e NFS)
- Manage users, grants and logical databases
- Configure connections for your applications
- Orchestrate and schedule sql scripts
- Prometheus metrics
- Validation webhooks to provide CRD inmutability
- Additional printer columns to report the current CRD status
- CRDs designed according to the Kubernetes API conventions
- GitOps friendly
- Multi-arch distroless based image
- Install it using kubectl, helm or OLM

This operator is open source and released under the terms of the MIT license. The source code is available on GitHub.

Other Operators

If you know about other MariaDB operators, feel free to add them to this page (see Writing and Editing Knowledge Base Articles).

MySQL and Percona Server operators should work as well, though some changes may be necessary to fix incompatibilities or take advantage of certain MariaDB features.

Content initially contributed by Vettabase Ltd.

2.1.2.14.10 Automating Upgrades with MariaDB.Org Downloads REST API

The MariaDB Foundation maintains a Downloads REST API. See the Downloads API documentation to find out all the tasks that you can accomplish with this API. Generally speaking, we can say that it provides information about MariaDB products and available versions. This allows to easily automate upgrades for MariaDB and related products.

The Downloads API exposes HTTPS endpoints that return information in JSON format. HTTP and JSON are extremely common standards that can be easily used with any programming language. All the information provided by the API is public, so no authentication is required.

How to Use the API with a Unix Shell

Linux shells are great for writing simple scripts. They are compatible to each other to some extent, so simple scripts can be run on almost any Unix/Linux system. In the following examples we'll use Bash.

On Linux, some programs you'll generally need to work with any REST API are:

- curl, to call HTTP URLs and get their output.
- js, to extract or transform information from a JSON document.

Example: Check When a New Version Becomes GA

A trivial use case is to write a script that checks the list of MariaDB GA major versions and, when something changes, send us an email. So we can test the newest GA version and eventually install it.

The script in this example will be extremely simple. We'll do it this way:

- Retrieve the JSON object describing all MariaDB versions.
- For each element of the array, only show the `release_id` and `release_status` properties, and concatenate them.
- Apply a filter, so we only select the rows containing 'stable' but not 'old' (so we exclude 'Old Stable').
- From the remaining rows, only show the first column (the version number).
- If the results we obtained are different from the previously written file (see last point) send an email.
- Save the results into a file.

This is something that we can easily do with a Unix shell:

```
#!/bin/bash

current_ga_versions=$(
    curl https://downloads.mariadb.org/rest-api/mariadb/ | \
    jq -r '.major_releases[] | .release_id + " " + .release_status' | \
    grep -i 'stable' | grep -vi 'old' | \
    cut -d ' ' -f 1
)

# create file if it doesn't exist, then compare version lists
touch ga_versions
previous_ga_versions=$( cat ga_versions )

echo "$current_ga_versions" > ga_versions

if [ "$current_ga_versions" != "$previous_ga_versions" ];
then
    mail -s 'NOTE: New MariaDB GA Versions' devops@example.com <<< 'There seems to be a new
MariaDB GA version! Yay!'
fi
```

The only non-standard command here is jq. It is a great way to manipulate JSON documents, so if you don't know it you may want to take a look at jq documentation.

How to Use the API with a Python Script

To use the API with Python, we need a module that is able to send HTTP requests and parse a JSON output. The `requests` module has both these features. It can be installed as follows:

```
pip install requests
```

The following script prints stable versions to the standard output:

```
#!/usr/bin/env python

import requests

response = requests.get('https://downloads.mariadb.org/rest-api/mariadb/').json()

for x in response['major_releases']:
    if x['release_status'] == 'Stable':
        print(x['release_id'])
```

`requests.get()` makes an HTTP call of type GET, and `requests.json()` returns a dictionary representing the previously obtained JSON document.

Content initially contributed by Vettabase Ltd.

2.1.2.14.11 HashiCorp Vault and MariaDB

Vault is open source software for secret management provided by HashiCorp. It is designed to avoid sharing secrets of various types, like passwords and private keys. When building automation, Vault is a good solution to avoid storing secrets in plain text in a repository.

MariaDB and Vault may relate to each other in several ways:

- MariaDB has a Hashicorp Key Management plugin, to manage and rotate SSH keys.
- Users passwords can be stored in Vault.
- MariaDB (and MySQL) can be used as a secret engine, a component which stores, generates, or encrypts data.
- MariaDB (and MySQL) can be used as a backend storage, providing durability for Vault data.

For information about how to install Vault, see Install Vault, as well as MySQL/MariaDB Database Secrets Engine.

Contents

Vault Features

Vault is used via an HTTP/HTTPS API.

Vault is identity-based. Users login and Vault sends them a token that is valid for a certain amount of time, or until certain conditions occur. Users with a valid token may request to obtain secrets for which they have proper permissions.

Vault encrypts the secrets it stores.

Vault can optionally audit changes to secrets and secrets requests by the users.

Vault Architecture

Vault is a server. This allows decoupling the secrets management logic from the clients, which only need to login and keep a token until it expires.

The sever can actually be a cluster of servers, to implement high availability.

The main Vault components are:

- **Storage Backed**: This is where the secrets are stored. Vault only send encrypted data to the backend storage.
- **HTTP API**: This API is used by the clients, and provides an access to Vault server.
- **Barrier**: Similarly to an actual barrier, it protects all inner Vault components. The HTTP API and the storage backend are outside of the barrier and could be accessed by anyone. All communications from and to these components have to pass through the barrier. The barrier verifies data and encrypts it. The barrier can have two states: *sealed* or *unsealed*. Data can only pass through when the barrier is unsealed. All the following components are located inside the barrier.
- **Auth Method**: Handles login attempts from clients. When a login succeeds, the auth method returns a list of security policies to Vault core.
- **Token Store**: Here the tokens generated as a result of a succeeded login are stored.
- **Secrets Engines**: These components manage secrets. They can have different levels of complexity. Some of them simply expect to receive a key, and return the corresponding secret. Others may generate secrets, including one-time-passwords.
- **Audit Devices**: These components log the requests received by Vault and the responses sent back to the clients.There may be multiple devices, in which case an **Audit Broker** sends the request or response to the proper device.

Dev Mode

It is possible to start Vault in dev mode:

```
vault server -dev
```

Dev mode is useful for learning Vault, or running experiments on some particular features. It is extremely insecure, because dev mode is equivalent to starting Vault with several insecure options. This means that Vault should never run in production in dev mode. However, this also means that all the regular Vault features are available in dev mode.

Dev mode simplifies all operations. Actually, no configuration is necessary to get Vault up and running in dev mode. It makes it possible to communicate with the Vault API from the shell without any authentication. Data is stored in memory by default. Vault is unsealed by default, and if explicitly sealed, it can be unsealed using only one key.

For more details, see "Dev" Server Mode in Vault documentation.

Vault Resources and References

- Documentation.
- MySQL/MariaDB Database Secrets Engine.
- MySQL Storage Backend.

Content initially contributed by Vettabase Ltd.

2.1.2.14.12 Orchestrator Overview

Orchestrator is a MySQL and MariaDB high availability and replication management tool. It is released by Shlomi Noach under the terms of the Apache License, version 2.0.

Orchestrator provides automation for MariaDB replication in the following ways:

- It can be used to perform certain operations, like repairing broken replication or moving a replica from one master to another. These operations can be requested using CLI commands, or via the GUI provided with Orchestrator. The actual commands sent to MariaDB are automated by Orchestrator, and the user doesn't have to worry about the details.
- Orchestrator can also automatically perform a failover in case a master crashes or is unreachable by its replicas. If that is the case, Orchestrator will promote one of the replicas to a master. The replica to promote is chosen based on several criteria, like the server versions, the binary log formats in use and the datacenters locations.

Note that, if we don't want to use Orchestrator to automate operations, we can still use it as a dynamic inventory. Other tools can use it to obtain a list of existing MariaDB servers via its REST API or CLI commands.

Orchestrator has several big users, listed in the documentation Users page. It is also included in the PMM monitoring solution.

To install Orchestrator, see:

- The install.md for a manual installation;
- The links in README.md, to install Orchestrator using automation tools.

Contents

Supported Topologies

Currently, Orchestrator fully supports MariaDB GTID, replication, and semi-synchronous replication. While Orchestrator does not support Galera specific logic, it works with Galera clusters. For details, see Supported Topologies and Versions in Orchestrator documentation.

Architecture

Orchestrator consists of a single executable called `orchestrator`. This is a process that periodically connects to the target servers. It will run SQL queries against target servers, so it needs a user with proper permissions. When the process is running, a GUI is available via a web browser, at the URL 'https://localhost:3000'. It also exposes a REST API (see Using the web API in the Orchestrator documentation).

Orchestrator expects to find a JSON configuration file called `orchestrator.conf.json`, in `/etc`.

A database is used to store the configuration and the state of the target servers. By default, this is done using built-in SQLite. However, it is possible to use an external MariaDB or MySQL server instance.

If a cluster of Orchestrator instances is running, only one central database is used. One Orchestrator node is active, while the others are passive and are only used for failover. If the active node crashes or becomes unreachable, one of the other nodes becomes the active instance. The `active_node` table shows which node is active. Nodes communicate between them using the Raft protocol.

CLI Examples

As mentioned, Orchestrator can be used from the command-line. Here you can find some examples.

List clusters:

```
orchestrator -c clusters
```

Discover a specified instance and add it to the known topology:

```
orchestrator -c discover -i <host>:<port>
```

Forget about an instance:

```
orchestrator -c topology -i <host>:<port>
```

Move a replica to a different master:

```
orchestrator -c move-up -i <replica-host>:<replica-port> -d <master-host>:<master-port>
```

Move a replica up, so that it becomes a "sibling" of its master:

```
orchestrator -c move-up -i <replica-host>:<replica-port>
```

Move a replica down, so that it becomes a replica of its"sibling":

```
orchestrator -c move-below -i <replica-host>:<replica-port> -d <master-host>:<master-port>
```

Make a node read-only:

```
orchestrator -c set-read-only -i <host>:<port>
```

Make a node writeable:

```
orchestrator -c set-writeable -i <host>:<port>
```

The `--debug` and `--stack` options can be added to the above commands to make them more verbose.

Orchestrator Resources and References

- Orchestrator on GitHub.
- Documentation.
- Raft consensus protocol website.

The README.md file lists some related community projects, including modules to automate Orchestrator with Puppet and other technologies.

On GitHub you can also find links to projects that allow the use of automation software to deploy and manage Orchestrator.

Content initially contributed by Vettabase Ltd.

2.4.6 Rotating Logs on Unix and Linux

2.1.2.14.14 Automating MariaDB Tasks with Events

MariaDB has an event scheduler that can be used to automate tasks, making them run at regular intervals of time. This page is about using events for automation. For more information about events themselves, and how to work with them, see event scheduler.

Pros and Cons of Using Events for Automation

Events can be compared to Unix cron jobs or Windows scheduled tasks. MariaDB events have at least the following benefits compared to those tools:

- Events are system-independent. The same code can run on any system.
- Events are written in procedural SQL. There is no need to install other languages or libraries.
- If you use user-defined functions, you can still take advantage of them in your events.
- Events run in MariaDB. An implication, for example, is that the results of queries remain in MariaDB itself and are not sent to a client. This means that network glitches don't affect events, there is no overhead due to data roundtrip, and therefore locks are held for a shorter time.

Some drawbacks of using events are the following:

- Events can only perform tasks that can be developed in SQL. So, for example, it is not possible to send alerts. Access to files or remote databases is limited.
- The event scheduler runs as a single thread. This means that events that are scheduled to run while another event is running will wait until the other event has stopped. This means that there is no guarantee that an event will run on exactly it's scheduled time. This should not be a problem as long as one ensures that events are short lived.
- For more events limitations, see Event Limitations.

In many cases you may prefer to develop scripts in an external programming language. However, you should know that simple tasks consisting of a few queries can easily be implemented as events.

Good Practices

When using events to automate tasks, there are good practices one may want to follow.

Move your SQL code in a stored procedure. All the event will do is to call a stored procedures. Several events may call the same stored procedure, maybe with different parameters. The procedure may also be called manually, if necessary. This will avoid code duplication. This will separate the logic from the schedule, making it possible to change an event without a risk of making changes to the logic, and the other way around.

Just like cron jobs, events should log whether if they succeed or not. Logging debug messages may also be useful for non-trivial events. This information can be logged into a dedicated table. The contents of the table can be monitored by a monitoring tool like Grafana. This allows to visualize in a dashboard the status of events, and send alerts in case of a failure.

Examples

Some examples of tasks that could easily be automated with events:

- Copying data from a remote table to a local table by night, using the CONNECT storage engine. This can be a good idea if many rows need be copied, because data won't be sent to an external client.
- Periodically delete historical data. For example, rows that are older than 5 years. Nothing prevents us from doing this with an external script, but probably this wouldn't add any value.
- Periodically delete invalid rows. In an e-commerce, they could be abandoned carts. In a messaging system, they could be messages to users that don't exist anymore.
- Add a new partition to a table and drop the oldest one (partition rotation).

Content initially contributed by Vettabase Ltd.

2.1.2.15 MariaDB Package Repository Setup and Usage

If you are looking to set up MariaDB Server, it is often easiest to use a repository. The MariaDB Foundation has a repository configuration tool at https://mariadb.org/download/ and MariaDB Corporation provides a convenient shell script to configure access to their MariaDB Package Repositories. It is available at:

- https://r.mariadb.com/downloads/mariadb_repo_setup

The script by default sets up 3 different repositories in a single repository configuration file. The repositories are

- MariaDB Server Repository
- MariaDB MaxScale Repository
- MariaDB Tools Repository

The script can be executed in the following way:

```
curl -LsS https://r.mariadb.com/downloads/mariadb_repo_setup | sudo bash
```

For the script to work, the `curl` and `ca-certificates` packages need to be installed on your system. Additionally on Debian and Ubuntu the `apt-transport-https` package needs to be installed. The script will check if these are installed and let you know before it attempts to create the repository configuration on your system.

Contents

Repositories

The script will will set up 2 different repositories in a single repository configuration file.

MariaDB Repository

The **MariaDB Repository** contains software packages related to MariaDB Server, including the server itself, clients and utilities, client libraries, plugins, and Mariabackup.

The binaries in MariaDB Corporation's **MariaDB Repository** are currently identical to the binaries in MariaDB Foundation's MariaDB Repository that is configured with the MariaDB Repository Configuration Tool.

By default, the script will configure your system to install from the repository of the latest GA version of MariaDB. That is currently MariaDB 10.7. If a new major GA release occurs and you would like to upgrade to it, then you will need to either manually edit the repository configuration file to point to the new version, or run the MariaDB Package Repository setup script again.

The script can also configure your system to install from the repository of a different version of MariaDB if you use the `--mariadb-server-version` option.

If you would not like to configure the **MariaDB Repository** on your system, then you can use the `--skip-server` option to prevent the MariaDB Package Repository setup script from configuring it.

MariaDB MaxScale Repository

The **MariaDB MaxScale Repository** contains software packages related to MariaDB MaxScale.

By default, the script will configure your system to install from the repository of the *latest* GA version of MariaDB MaxScale. When a new major GA release occurs, the repository will automatically switch to the new version. If instead you would like to stay on a particular version you will need to manually edit the repository configuration file and change ' `latest` ' to the version you want (e.g. ' `6.1` ') or run the MariaDB Package Repository setup script again, specifying the particular version or series you want.

Older versions of the MariaDB Package Repository setup script would configure a specific MariaDB MaxScale series in the repository (i.e. '`2.4`'), so if you used the script in the past to set up your repository and want MariaDB MaxScale to automatically use the latest GA version then change '`2.4`' or '`2.3`' in the repository configuration to '`latest`'. Or download the current version of the script and re-run it to set up the repository again.

The script can configure your system to install from the repository of an older version of MariaDB MaxScale if you use the `--mariadb-maxscale-version` option. For example, `--mariadb-maxscale-version=2.4` if you want the latest release in the MariaDB MaxScale 2.4.x series.

If you do not want to configure the **MariaDB MaxScale Repository** on your system, then you can use the `--skip-maxscale` option to prevent the MariaDB Package Repository setup script from configuring it.

MariaDB MaxScale is licensed under the Business Source License 1.1, so it is not entirely free to use for organizations who do not have a subscription with MariaDB Corporation. If you would like more information, see the information at MariaDB Business Source License (BSL): Frequently Asked Questions. If you would like to know how much a subscription to use MariaDB MaxScale would cost, see MariaDB Corporation's subscription pricing.

Supported Distributions

The script supports Linux distributions that are officially supported by MariaDB Corporation's MariaDB TX subscription. However, a MariaDB TX subscription with MariaDB Corporation is not required to use the MariaDB Package Repository.

The distributions currently supported by the script include:

- Red Hat Enterprise Linux (RHEL) 7 and 8
- CentOS 7
- Debian 10 (Buster), 11 (Bullseye), 12 (Bookworm)
- Ubuntu 18.04 LTS (Bionic), and 20.04 LTS (Focal)
- SUSE Linux Enterprise Server (SLES) 12 and 15

To install MariaDB on distributions not supported by the MariaDB Package Repository setup script, please consider using MariaDB Foundation's MariaDB Repository Configuration Tool. Some Linux distributions also include MariaDB in their own repositories.

Using the MariaDB Package Repository Setup Script

The script can be executed in the following way:

```
curl -LsS https://r.mariadb.com/downloads/mariadb_repo_setup | sudo bash
```

The script will have to set up package repository configuration files, so it will need to be executed as root.

The script will also install the GPG public keys used to verify the signature of MariaDB software packages. If you want to avoid that, then you can use the `--skip-key-import` option.

If the script tries to create the repository configuration file and one with that name already exists, then the script will rename the existing file with an extension in the format ".old_[0-9]+", which would make the OS's package manager ignore the file. You can safely remove those files after you have confirmed that the updated repository configuration file works..

If you want to see the repository configuration file that would be created without actually doing so, then you can use the `--write-to-stdout` option. This also prevents the need to run the script as root,

If you want to download the script, rather than executing it, then you can do so in the following way:

```
curl -LO https://r.mariadb.com/downloads/mariadb_repo_setup
```

Options

To provide options to the script, you must tell bash to expect them by executing bash with the options `-s --`, for example:

```
curl -LsS https://r.mariadb.com/downloads/mariadb_repo_setup | sudo bash -s -- --help
```

Option	Description
`--help`	Display a usage message and exit.

`--mariadb-server-version=<version>`	Override the default MariaDB Server version.
`--mariadb-maxscale-version=<version>`	Override the default MariaDB MaxScale version. By default, the script will use 'latest'.
`--os-type=<type>`	Override detection of OS type. Acceptable values include `debian`, `ubuntu`, `rhel`, and `sles`.
`--os-version=<version>`	Override detection of OS version. Acceptable values depend on the OS type you specify.
`--skip-key-import`	Skip importing GPG signing keys.
`--skip-maxscale`	Skip the 'MaxScale' repository.
`--skip-server`	Skip the 'MariaDB Server' repository.
`--skip-tools`	Skip the 'Tools' repository.
`--skip-verify`	Skip verification of MariaDB Server versions. Use with caution as this can lead to an invalid repository configuration file being created.
`--skip-check-installed`	Skip tests for required prerequisites for this script.
`--skip-eol-check`	Skip tests for versions being past their EOL date
`--skip-os-eol-check`	Skip tests for operating system versions being past EOL date
`--write-to-stdout`	Write output to stdout instead of to the OS's repository configuration file. This will also skip importing GPG public keys and updating the package cache on platforms where that behavior exists.

`--mariadb-server-version`

By default, the script will configure your system to install from the repository of the latest GA version of MariaDB. If a new major GA release occurs and you would like to upgrade to it, then you will need to either manually edit the repository configuration file to point to the new version, or run the MariaDB Package Repository setup script again.

The script can also configure your system to install from the repository of a different version of MariaDB if you use the `--mariadb-server-version` option.

The string `mariadb-` has to be prepended to the version number. For example, to configure your system to install from the repository of MariaDB 10.6, that would be:

```
curl -LsS https://r.mariadb.com/downloads/mariadb_repo_setup | sudo bash -s -- --mariadb-server-version="mariadb-10.6"
```

The following MariaDB versions are currently supported:

- `mariadb-10.4`
- `mariadb-10.5`
- `mariadb-10.6`
- `mariadb-10.10`
- `mariadb-10.11`
- `mariadb-11.0`
- `mariadb-11.1`
- `mariadb-11.2`

If you want to pin the repository of a specific minor release, such as MariaDB 10.6.14, then you can also specify the minor release. For example, `mariadb-10.6.14`. This may be helpful if you want to avoid upgrades. However, avoiding upgrades is not recommended, since minor maintenance releases may contain important bug fixes and fixes for security vulnerabilities.

`--mariadb-maxscale-version`

By default, the script will configure your system to install from the repository of the latest GA version of MariaDB MaxScale.

If you would like to pin the repository to a specific version of MariaDB MaxScale then you will need to either manually edit the repository configuration file to point to the desired version, or use the `--mariadb-maxscale-version` option.

For example, to configure your system to install from the repository of MariaDB MaxScale 6.1, that would be:

```
curl -LsS https://r.mariadb.com/downloads/mariadb_repo_setup | sudo bash -s -- --mariadb-
maxscale-version="6.1"
```

The following MariaDB MaxScale versions are currently supported:

- MaxScale 1.4
- MaxScale 2.0
- MaxScale 2.1
- MaxScale 2.2
- MaxScale 2.3
- MaxScale 2.4
- MaxScale 2.5
- MaxScale 6.1
- MaxScale 6.2

The special identifiers `latest` (for the latest GA release) and `beta` (for the latest beta release) are also supported. By default the `mariadb_repo_setup` script uses `latest` as the version.

`--os-type` and `--os-version`

If you want to run this script on an unsupported OS that you believe to be package-compatible with an OS that is supported, then you can use the `--os-type` and `--os-version` options to override the script's OS detection. If you use either option, then you must use both options.

The supported values for `--os-type` are:

- `rhel`
- `debian`
- `ubuntu`
- `sles`

If you use a non-supported value, then the script will fail, just as it would fail if you ran the script on an unsupported OS.

The supported values for `--os-version` are entirely dependent on the OS type.

For Red Hat Enterprise Linux (RHEL) and CentOS, `7` and `8` are valid options.

For Debian and Ubuntu, the version must be specified as the codename of the specific release. For example, Debian 9 must be specified as `stretch`, and Ubuntu 18.04 must be specified as `bionic`.

These options can be useful if your distribution is a fork of another distribution. As an example, Linux Mint 8.1 is based on and is fully compatible with Ubuntu 16.04 LTS (Xenial). Therefore, If you are using Linux Mint 8.1, then you can configure your system to install from the repository of Ubuntu 16.04 LTS (Xenial). If you would like to do that, then you can do so by specifying `--os-type=ubuntu` and `--os-version=xenial` to the MariaDB Package Repository setup script.

For example, to manually set the `--os-type` and `--os-version` to RHEL 8 you could do:

```
curl -LsS https://r.mariadb.com/downloads/mariadb_repo_setup | sudo bash -s -- --os-type=rhel -
-os-version=8
```

`--write-to-stdout`

The `--write-to-stdout` option will prevent the script from modifying anything on the system. The repository configuration will not be written to the repository configuration file. Instead, it will be printed to standard output. That allows the configuration to be reviewed, redirected elsewhere, consumed by another script, or used in some other way.

The `--write-to-stdout` option automatically enables `--skip-key-import`.

For example:

```
curl -LsS https://r.mariadb.com/downloads/mariadb_repo_setup | sudo bash -s -- --write-to-
stdout
```

Platform-Specific Behavior

Platform-Specific Behavior on RHEL and CentOS

On Red Hat Enterprise Linux (RHEL) and CentOS, the MariaDB Package Repository setup script performs the following

tasks:

1. Creates a repository configuration file at `/etc/yum.repos.d/mariadb.repo`.
2. Imports the GPG public key used to verify the signature of MariaDB software packages with `rpm --import` from `downloads.mariadb.com`.

Platform-Specific Behavior on Debian and Ubuntu

On Debian and Ubuntu, the MariaDB Package Repository setup script performs the following tasks:

1. Creates a repository configuration file at `/etc/apt/sources.list.d/mariadb.list`.
2. Creates a package preferences file at `/etc/apt/preferences.d/mariadb-enterprise.pref`, which gives packages from MariaDB repositories a higher priority than packages from OS and other repositories, which can help avoid conflicts. It looks like the following:

   ```
   Package: *
   Pin: origin downloads.mariadb.com
   Pin-Priority: 1000
   ```

3. Imports the GPG public key used to verify the signature of MariaDB software package with `apt-key` from the `keyserver.ubuntu.com` key server.
4. Updates the package cache with package definitions from the MariaDB Package Repository with `apt-get update`.

Platform-Specific Behavior on SLES

On SUSE Linux Enterprise Server (SLES), the MariaDB Package Repository setup script performs the following tasks:

1. Creates a repository configuration file at `/etc/zypp/repos.d/mariadb.repo`.
2. Imports the GPG public key used to verify the signature of MariaDB software packages with `rpm --import` from `downloads.mariadb.com`.

Installing Packages with the MariaDB Package Repository

After setting up the MariaDB Package Repository, you can install the software packages in the supported repositories.

Installing Packages on RHEL and CentOS

To install MariaDB on Red Hat Enterprise Linux (RHEL) and CentOS, see the instructions at Installing MariaDB Packages with YUM. For example:

```
sudo yum install MariaDB-server MariaDB-client MariaDB-backup
```

To install MariaDB MaxScale on Red Hat Enterprise Linux (RHEL) and CentOS, see the instructions at MariaDB MaxScale Installation Guide. For example:

```
sudo yum install maxscale
```

Installing Packages on Debian and Ubuntu

To install MariaDB on Debian and Ubuntu, see the instructions at Installing MariaDB Packages with APT. For example:

```
sudo apt-get install mariadb-server mariadb-client mariadb-backup
```

To install MariaDB MaxScale on Debian and Ubuntu, see the instructions at MariaDB MaxScale Installation Guide. For example:

```
sudo apt-get install maxscale
```

Installing Packages on SLES

To install MariaDB on SUSE Linux Enterprise Server (SLES), see the instructions at Installing MariaDB Packages with

ZYpp. For example:

```
sudo zypper install MariaDB-server MariaDB-client MariaDB-backup
```

To install MariaDB MaxScale on SUSE Linux Enterprise Server (SLES), see the instructions at MariaDB MaxScale Installation Guide. For example:

```
sudo zypper install maxscale
```

Versions

Version	sha256sum
2023-11-21	2d7291993f1b71b5dc84cc1d23a65a5e01e783aa765c2bf5ff4ab62814bb5da1
2023-08-21	935944a2ab2b2a48a47f68711b43ad2d698c97f1c3a7d074b34058060c2ad21b
2023-08-14	f5ba8677ad888cf1562df647d3ee843c8c1529ed63a896bede79d01b2ecc3c1d
2023-06-09	3a562a8861fc6362229314772c33c289d9096bafb0865ba4ea108847b78768d2
2023-02-16	ad125f01bada12a1ba2f9986a21c59d2cccbe8d584e7f55079ecbeb7f43a4da4
2022-11-17	367a80b01083c34899958cdd62525104a3de6069161d309039e84048d89ee98b
2022-08-22	733cf126b03f73050e242102592658913d10829a5bf056ab77e7f864b3f8de1f
2022-08-15	f99e1d560bd72a3a23f64eaede8982d5494407cafa8f995de45fb9a7274ebc5c
2022-06-14	d4e4635eeb79b0e96483bd70703209c63da55a236eadd7397f769ee434d92ca8
2022-02-08	b9e90cde27affc2a44f9fc60e302ccfcacf71f4ae02071f30d570e6048c28597
2022-01-18	c330d2755e18e48c3bba300a2898b0fc8ad2d3326d50b64e02fe65c67b454599

2.1.3 Upgrading MariaDB

Upgrading Between Major MariaDB Versions
MariaDB is designed for easy upgrades.

Upgrading Between Minor Versions on Linux
Upgrading between minor versions of MariaDB, e.g. from MariaDB 10.4.12 to MariaDB 10.4.13

Downgrading between Major Versions of MariaDB
Downgrading MariaDB is not officially supported between major versions.

Upgrading from MariaDB 10.11 to MariaDB 11.0
How to upgrade from MariaDB 10.11 to MariaDB 11.0.

Upgrading from MariaDB 10.6 to MariaDB 10.11
How to upgrade from MariaDB 10.6 to MariaDB 10.11

Upgrading from MariaDB 10.7 to MariaDB 10.8
How to upgrade from MariaDB 10.7 to MariaDB 10.8.

Upgrading from MariaDB 10.6 to MariaDB 10.7
How to upgrade from MariaDB 10.6 to MariaDB 10.7.

Upgrading from MariaDB 10.5 to MariaDB 10.6
How to upgrade from MariaDB 10.5 to MariaDB 10.6.

Upgrading from MariaDB 10.4 to MariaDB 10.5
How to upgrade from MariaDB 10.4 to MariaDB 10.5.

Upgrading from MariaDB 10.3 to MariaDB 10.4

How to upgrade from MariaDB 10.3 to MariaDB 10.4.

Upgrading from MariaDB 10.2 to MariaDB 10.3

How to upgrade from MariaDB 10.2 to MariaDB 10.3.

Upgrading MariaDB on Windows

Upgrading MariaDB on Windows.

Upgrading Galera Cluster

Upgrading MariaDB Galera Cluster.

Upgrading from MySQL to MariaDB

Upgrading from MySQL to MariaDB.

Upgrading to Unmaintained MariaDB Releases

Upgrading to unmaintained MariaDB releases

There are 25 related questions.

2.1.3.1 Upgrading Between Major MariaDB Versions

Contents

MariaDB is designed to allow easy upgrades. You should be able to trivially upgrade from ANY earlier MariaDB version to the latest one (for example MariaDB 10.3.x to MariaDB 10.11.x), usually in a few seconds. This is also mainly true for any MySQL version < 8.0 to MariaDB 10.4 and up.

Upgrades are normally easy because:

- All MariaDB table data files are backward compatible
- The MariaDB connection protocol is backward compatible. You don't normally need to upgrade any of your old clients to be able to connect to a newer MariaDB version.
- The MariaDB replica can be of any newer version than the primary.

MariaDB Corporation regularly runs tests to check that one can upgrade from MariaDB 5.5 to the latest MariaDB version without any trouble. All older versions should work too (as long as the storage engines you were using are still around).

Note that if you are using MariaDB Galera Cluster, you have to follow the Galera upgrading instructions!

Requirements for Doing an Upgrade Between Major Versions

- Go through the individual version upgrade notes (listed below) to look for any major changes or configuration options that have changed.
- Ensure that the target MariaDB version supports the storage engines you are using. For example, in 10.5 TokuDB is not supported.
- Backup the database (just in case). At least, take a copy of the `mysql` data directory with mariadb-dump --add-drop-table mysql (called mysqldump in MariaDB 10.3 and earlier) as most of the upgrade changes are done there (adding new fields and new system tables etc).
- Cleanly shutdown the server. This is necessary because even if data files are compatible between versions, recovery logs may not be.
    - Ensure that the innodb_fast_shutdown variable is not 2 (fast crash shutdown) or 3. The default of this variable is 1. The safest and recommended option for upgrades is 0. The shutdown time may be notably larger with 0

than for 1 as there are a lot more cleanups done for 0, however when preparing for an upgrade this should not be an issue.
    - innodb_force_recovery must be less than `3`.

Note that rpms don't support upgrading between major versions, only minor like 10.4.1 to 10.4.2. If you are using rpms, you should de-install the old MariaDB rpms and install the new MariaDB rpms before running mariadb-upgrade. Note that when installing the new rpms, mariadb-upgrade may be run automatically. There is no problem with running mariadb-upgrade many times.

Recommended Steps

- If you have a primary-replica setup, first upgrade one replica and when you have verified that the replica works well, upgrade the rest of the replicas (if any). Then upgrade one replica to primary, upgrade the primary, and change the replica to a primary.
- If you don't have a primary-replica setup, then take a backup, shutdown MariaDB and do the upgrade.

Step by Step Instructions for Upgrades

- Upgrade MariaDB binaries and libraries, preferably without starting MariaDB.
- If the MariaDB server process, mysqld or mariadbd was not started as part of the upgrade, start it by executing `mysqld --skip-grant-tables`. This may produce some warnings about some system tables not being up to date, but you can ignore these for now as mariadb-upgrade will fix that.
- Run mariadb-upgrade
- Restart MariaDB server.

Work Done by mariadb-upgrade

The main work done when upgrading is done by running mariadb-upgrade. The main things it does are:

- Updating the system tables in the `mysql` database to the newest version. This is very quick.
- mariadb-upgrade also runs mariadb-check --check-upgrade to check if there have been any collation changes between the major versions. This recreates indexes in old tables that are using any of the changed collations. This can take a bit of time if there are a lot of tables or there are many tables which used the changed collation. The last time a collation changed was in MariaDB/MySQL 5.1.23.

Post Upgrade Work

Check the MariaDB error log for any problems during upgrade. If there are any warnings in the log files, do your best to get rid of them!

The common warnings/errors are:

- Using obsolete options. If this is the case, remove them from your my.cnf files.
- Check the manual for new features that have been added since your last MariaDB version.
- Test that your application works as before. The main difference from before is that because of optimizer improvements your application should work better than before, but in some rare cases the optimizer may get something wrong. In this case, you can try to use explain, optimizer trace or optimizer_switch to fix the queries.

If Something Goes Wrong

- First, check the MariaDB error log to see if you are using configure options that are not supported anymore.
- Check the upgrade notices for the MariaDB release that you are upgrading to.
- File an issue in the MariaDB bug tracker so that we know about the issue and can provide a fix to make upgrades even better.
- Add a comment to this manual entry for how we can improve it.

Disaster Recovery

In the unlikely event something goes wrong, you can try the following:

- Remove the InnoDB tables from the `mysql` data directory. They are:
    - gtid_slave_pos
    - innodb_table_stats
    - innodb_index_stats
    - transaction_registry

- Move the `mysql` data directory to `mysql-old` and run mariadb-install-db to generate a new one.
- After the above, you have to add back your old users.
- When done, delete the `mysql-old` data directory.

Downgrading

MariaDB server is not designed for downgrading. That said, in most cases, as long as you haven't run any ALTER TABLE or CREATE TABLE statements and you have a mariadb-dump of your old `mysql` database , you should be able to downgrade to your previous version by doing the following:

- Do a clean shutdown. For this special case you have to set innodb_fast_shutdown to 0,before taking down the new MariaDB server, to ensure there are no redo or undo logs that need to be applied on the downgraded server.
- Delete the tables in the `mysql` database (if you didn't use the option `--add-drop-table` to mariadb-dump)
- Delete the new MariaDB installation
- Install the old MariaDB version
- Start the server with mysqld --skip-grant-tables
- Install the old `mysql` database
- Execute in the mariadb client `FLUSH PRIVILEGES`

2.1.3.2 Upgrading Between Minor Versions on Linux

> For Windows, see Upgrading MariaDB on Windows instead.

> For MariaDB Galera Cluster, see Upgrading Between Minor Versions with Galera Cluster instead.

> Before you upgrade, it would be best to take a backup of your database. This is always a good idea to do before an upgrade. We would recommend Mariabackup.

To upgrade between minor versions of MariaDB on Linux/Unix (for example from MariaDB 10.11.2 to MariaDB 10.11.3), the following procedure is suggested:

1. Stop MariaDB.
2. Uninstall the old version of MariaDB.
3. Install the new version of MariaDB.
    - On Debian, Ubuntu, and other similar Linux distributions, see Installing MariaDB Packages with APT for more information.
    - On RHEL, CentOS, Fedora, and other similar Linux distributions, see Installing MariaDB Packages with YUM for more information.
    - On SLES, OpenSUSE, and other similar Linux distributions, see Installing MariaDB Packages with ZYpp for more information.
4. Make any desired changes to configuration options in option files, such as `my.cnf` .
5. Start MariaDB.
6. Run mariadb-upgrade.
    - `mariadb-upgrade` does two things:
        1. Ensures that the system tables in the mysql database are fully compatible with the new version.
        2. Does a very quick check of all tables and marks them as compatible with the new version of MariaDB .
    - In most cases this should be a fast operation (depending of course on the number of tables).

To upgrade between major versions, see the following:

- Upgrading Between Major MariaDB Versions
- Upgrading from MariaDB 10.6 to MariaDB 10.11
- Upgrading from MariaDB 10.5 to MariaDB 10.6
- Upgrading from MariaDB 10.4 to MariaDB 10.5
- Upgrading from MariaDB 10.3 to MariaDB 10.4
- Upgrading from MariaDB 10.2 to MariaDB 10.3

2.1.3.3 Upgrading from MariaDB 10.11 to

MariaDB 11.0

Contents

This page includes details for upgrading from MariaDB 10.11 to MariaDB 11.0. It is currently incomplete. Note that MariaDB 10.11 is maintained for five years, while MariaDB 11.0 is a short-term maintenance release, only maintained for one year.

How to Upgrade

For Windows, see Upgrading MariaDB on Windows.

Before you upgrade, it would be best to take a backup of your database. This is always a good idea to do before an upgrade. We would recommend Mariabackup.

The suggested upgrade procedure is:

1. Modify the repository configuration, so the system's package manager installs MariaDB 11.0. For example,
   - On Debian, Ubuntu, and other similar Linux distributions, see Updating the MariaDB APT repository to a New Major Release for more information.
   - On RHEL, CentOS, Fedora, and other similar Linux distributions, see Updating the MariaDB YUM repository to a New Major Release for more information.
   - On SLES, OpenSUSE, and other similar Linux distributions, see Updating the MariaDB ZYpp repository to a New Major Release for more information.
2. Stop MariaDB.
3. Uninstall the old version of MariaDB.
   - On Debian, Ubuntu, and other similar Linux distributions, execute the following:
     ```
     sudo apt-get remove mariadb-server
     ```
   - On RHEL, CentOS, Fedora, and other similar Linux distributions, execute the following:
     ```
     sudo yum remove MariaDB-server
     ```
   - On SLES, OpenSUSE, and other similar Linux distributions, execute the following:
     ```
     sudo zypper remove MariaDB-server
     ```
4. Install the new version of MariaDB.
   - On Debian, Ubuntu, and other similar Linux distributions, see Installing MariaDB Packages with APT for more information.
   - On RHEL, CentOS, Fedora, and other similar Linux distributions, see Installing MariaDB Packages with YUM for more information.
   - On SLES, OpenSUSE, and other similar Linux distributions, see Installing MariaDB Packages with ZYpp for more information.
5. Make any desired changes to configuration options in option files, such as `my.cnf`. This includes removing any options that are no longer supported.
6. Start MariaDB.
7. Run mariadb-upgrade.
   - `mariadb-upgrade` does two things:
     1. Ensures that the system tables in the mysql database are fully compatible with the new version.
     2. Does a very quick check of all tables and marks them as compatible with the new version of MariaDB .

Incompatible Changes Between 10.11 and 11.0

On most servers upgrading from 10.11 should be painless. However, there are some things that have changed which could affect an upgrade:

Options That Have Changed Default Values

Option	Old default	New default

innodb_undo_tablespaces	0	3
histogram_type	DOUBLE_PREC_HB	JSON_HB

Options That Have Been Removed or Renamed

The following options should be removed or renamed if you use them in your option files:

Option	Reason
innodb_change_buffer_max_size	InnoDB Change Buffer removed
innodb_change_buffering	InnoDB Change Buffer removed

Deprecated Options

The following options have been deprecated. They have not yet been removed, but will be in a future version, and should ideally no longer be used.

Option	Reason
innodb_defragment	InnoDB Defragmentation is not particularly useful and causes a maintenance burden.
innodb_defragment_n_pages	
innodb_defragment_stats_accuracy	
innodb_defragment_fill_factor_n_recs	
innodb_defragment_fill_factor	
innodb_defragment_frequency	
innodb_file_per_table	
innodb_flush_method	
innodb_file_per_table	Has been set for many releases. Unsetting (the original InnoDB default) is no longer useful)
innodb_flush_method	Mapped it to 4 new boolean parameters that can be changed while the server is running

2.1.3.4 Upgrading from MariaDB 10.6 to MariaDB 10.11

Contents

This page includes details for upgrading from MariaDB 10.6 to the subsequent long-term maintenance version, MariaDB 10.11. It is currently incomplete.

How to Upgrade

For Windows, see Upgrading MariaDB on Windows.

Before you upgrade, it would be best to take a backup of your database. This is always a good idea to do before an upgrade. We would recommend Mariabackup.

The suggested upgrade procedure is:

1. Modify the repository configuration, so the system's package manager installs MariaDB 10.11. For example,
   - On Debian, Ubuntu, and other similar Linux distributions, see Updating the MariaDB APT repository to a New Major Release for more information.
   - On RHEL, CentOS, Fedora, and other similar Linux distributions, see Updating the MariaDB YUM repository to a New Major Release for more information.
   - On SLES, OpenSUSE, and other similar Linux distributions, see Updating the MariaDB ZYpp repository to a New Major Release for more information.
2. Stop MariaDB.
3. Uninstall the old version of MariaDB.
   - On Debian, Ubuntu, and other similar Linux distributions, execute the following:
     ```
     sudo apt-get remove mariadb-server
     ```
   - On RHEL, CentOS, Fedora, and other similar Linux distributions, execute the following:
     ```
     sudo yum remove MariaDB-server
     ```
   - On SLES, OpenSUSE, and other similar Linux distributions, execute the following:
     ```
     sudo zypper remove MariaDB-server
     ```
4. Install the new version of MariaDB.
   - On Debian, Ubuntu, and other similar Linux distributions, see Installing MariaDB Packages with APT for more information.
   - On RHEL, CentOS, Fedora, and other similar Linux distributions, see Installing MariaDB Packages with YUM for more information.
   - On SLES, OpenSUSE, and other similar Linux distributions, see Installing MariaDB Packages with ZYpp for more information.
5. Make any desired changes to configuration options in option files, such as `my.cnf`. This includes removing any options that are no longer supported.
6. Start MariaDB.
7. Run mariadb-upgrade.
   - `mariadb-upgrade` does two things:
     1. Ensures that the system tables in the mysql database are fully compatible with the new version.
     2. Does a very quick check of all tables and marks them as compatible with the new version of MariaDB .

Incompatible Changes Between 10.6 and 10.11

On most servers upgrading from 10.6 should be painless. However, there are some things that have changed which could affect an upgrade:

Compression

If a non-zlib compression algorithm was used in InnoDB or Mroonga before upgrading to 10.11, those tables will be unreadable until the appropriate compression library is installed. See Compression Plugins#Upgrading.

Options That Have Changed Default Values

Option	Old default	New default
innodb_buffer_pool_chunk_size	134217728	Autosized
spider_auto_increment_mode	-1	0
spider_bgs_first_read	-1	2
spider_bgs_mode	-1	0
spider_bgs_second_read	-1	100
spider_bka_mode	-1	1
spider_bka_table_name_type	-1	1
spider_buffer_size	-1	16000
spider_bulk_size	-1	16000
spider_bulk_update_mode	-1	0
spider_bulk_update_size	-1	16000
spider_casual_read	-1	0
spider_connect_timeout	-1	6

spider_crd_bg_mode	-1	2
spider_crd_interval	-1	51
spider_crd_mode	-1	1
spider_crd_sync	-1	0
spider_crd_type	-1	2
spider_crd_weight	-1	2
spider_delete_all_rows_type	-1	1
spider_direct_dup_insert	-1	0
spider_direct_order_limit	-1	9223372036854775807
spider_error_read_mode	-1	0
spider_error_write_mode	-1	0
spider_first_read	-1	0
spider_init_sql_alloc_size	-1	1024
spider_internal_limit	-1	9223372036854775807
spider_internal_offset	-1	0
spider_internal_optimize	-1	0
spider_internal_optimize_local	-1	0
spider_load_crd_at_startup	-1	1
spider_load_sts_at_startup	-1	1
spider_low_mem_read	-1	1
spider_max_order	-1	32767
spider_multi_split_read	-1	100
spider_net_read_timeout	-1	600
spider_net_write_timeout	-1	600
spider_quick_mode	-1	3
spider_quick_page_byte	-1	10485760
spider_quick_page_size	-1	1024
spider_read_only_mode	-1	0
spider_reset_sql_alloc	-1	1
spider_second_read	-1	0
spider_selupd_lock_mode	-1	1
spider_semi_split_read	-1	2
spider_semi_split_read_limit	-1	1
spider_semi_table_lock_connection	-1	1
spider_semi_table_lock	1	0

Options That Have Been Removed or Renamed

The following options should be removed or renamed if you use them in your option files:

Option	Reason
innodb_log_write_ahead_size	On Linux and Windows, the physical block size of the underlying storage is instead detected and used.
innodb_version	Redundant
wsrep_replicate_myisam	Use wsrep_mode instead.

wsrep_strict_ddl	Use wsrep_mode instead.

Deprecated Options

The following options have been deprecated. They have not yet been removed, but will be in a future version, and should ideally no longer be used.

Option	Reason
keep_files_on_create	MariaDB now deletes orphan files, so this setting should never be necessary.

2.1.3.5 Upgrading from MariaDB 10.7 to MariaDB 10.8

Contents

Note that MariaDB 10.8 is only maintained for one year. MariaDB 10.6 is currently the latest long-term maintenance release.

How to Upgrade

For Windows, see Upgrading MariaDB on Windows.

Before you upgrade, it would be best to take a backup of your database. This is always a good idea to do before an upgrade. We would recommend Mariabackup.

The suggested upgrade procedure is:

1. Modify the repository configuration, so the system's package manager installs MariaDB 10.8. For example,
   - On Debian, Ubuntu, and other similar Linux distributions, see Updating the MariaDB APT repository to a New Major Release for more information.
   - On RHEL, CentOS, Fedora, and other similar Linux distributions, see Updating the MariaDB YUM repository to a New Major Release for more information.
   - On SLES, OpenSUSE, and other similar Linux distributions, see Updating the MariaDB ZYpp repository to a New Major Release for more information.
2. Stop MariaDB.
3. Uninstall the old version of MariaDB.
   - On Debian, Ubuntu, and other similar Linux distributions, execute the following:
     ```
     sudo apt-get remove mariadb-server
     ```
   - On RHEL, CentOS, Fedora, and other similar Linux distributions, execute the following:
     ```
     sudo yum remove MariaDB-server
     ```
   - On SLES, OpenSUSE, and other similar Linux distributions, execute the following:
     ```
     sudo zypper remove MariaDB-server
     ```
4. Install the new version of MariaDB.
   - On Debian, Ubuntu, and other similar Linux distributions, see Installing MariaDB Packages with APT for more information.
   - On RHEL, CentOS, Fedora, and other similar Linux distributions, see Installing MariaDB Packages with YUM for more information.
   - On SLES, OpenSUSE, and other similar Linux distributions, see Installing MariaDB Packages with ZYpp for more information.
5. Make any desired changes to configuration options in option files, such as `my.cnf`. This includes removing any options that are no longer supported.
6. Start MariaDB.
7. Run mariadb-upgrade.

- `mariadb-upgrade` does two things:
  1. Ensures that the system tables in the mysql database are fully compatible with the new version.
  2. Does a very quick check of all tables and marks them as compatible with the new version of MariaDB .

Incompatible Changes Between 10.7 and 10.8

On most servers upgrading from 10.7 should be painless. However, there are some things that have changed which could affect an upgrade:

Options That Have Changed Default Values

Option	Old default value	New default value
innodb_buffer_pool_chunk_size	134217728	Autosized
spider_semi_table_lock	1	0

Options That Have Been Removed or Renamed

The following options should be removed or renamed if you use them in your option files:

Option	Reason
innodb_log_write_ahead_size	On Linux and Windows, the physical block size of the underlying storage is instead detected and used.

Deprecated Options

The following options have been deprecated. They have not yet been removed, but will be in a future version, and should ideally no longer be used.

Option	Reason
keep_files_on_create	MariaDB now deletes orphan files, so this setting should never be necessary.

Major New Features To Consider

You might consider using the following major new features in MariaDB 10.8:

- Stored procedures already have support for the IN, OUT and INOUT parameter qualifiers. Added as well for stored functions and (IN only) cursors (MDEV-10654).
- Individual columns in the index can now be explicitly sorted in the ascending or descending order. This can be useful for optimizing certain ORDER BY cases (MDEV-13756, MDEV-26938, MDEV-26939, MDEV-26996).
- See also System Variables Added in MariaDB 10.8.

2.1.3.6 Upgrading from MariaDB 10.6 to MariaDB 10.7

Contents

Note that MariaDB 10.7 is only maintained for one year. MariaDB 10.6 is currently the latest long-term maintenance release.

How to Upgrade

For Windows, see Upgrading MariaDB on Windows.

Before you upgrade, it would be best to take a backup of your database. This is always a good idea to do before an upgrade. We would recommend Mariabackup.

The suggested upgrade procedure is:

1. Modify the repository configuration, so the system's package manager installs MariaDB 10.7. For example,
   - On Debian, Ubuntu, and other similar Linux distributions, see Updating the MariaDB APT repository to a New Major Release for more information.
   - On RHEL, CentOS, Fedora, and other similar Linux distributions, see Updating the MariaDB YUM repository to a New Major Release for more information.
   - On SLES, OpenSUSE, and other similar Linux distributions, see Updating the MariaDB ZYpp repository to a New Major Release for more information.
2. Stop MariaDB.
3. Uninstall the old version of MariaDB.
   - On Debian, Ubuntu, and other similar Linux distributions, execute the following:
     ```
     sudo apt-get remove mariadb-server
     ```
   - On RHEL, CentOS, Fedora, and other similar Linux distributions, execute the following:
     ```
     sudo yum remove MariaDB-server
     ```
   - On SLES, OpenSUSE, and other similar Linux distributions, execute the following:
     ```
     sudo zypper remove MariaDB-server
     ```
4. Install the new version of MariaDB.
   - On Debian, Ubuntu, and other similar Linux distributions, see Installing MariaDB Packages with APT for more information.
   - On RHEL, CentOS, Fedora, and other similar Linux distributions, see Installing MariaDB Packages with YUM for more information.
   - On SLES, OpenSUSE, and other similar Linux distributions, see Installing MariaDB Packages with ZYpp for more information.
5. Make any desired changes to configuration options in option files, such as `my.cnf`. This includes removing any options that are no longer supported.
6. Start MariaDB.
7. Run mariadb-upgrade.
   - `mariadb-upgrade` does two things:
     1. Ensures that the system tables in the mysql database are fully compatible with the new version.
     2. Does a very quick check of all tables and marks them as compatible with the new version of MariaDB .

Incompatible Changes Between 10.6 and 10.7

On most servers upgrading from 10.6 should be painless. However, there are some things that have changed which could affect an upgrade:

Reserved Words

- ROW_NUMBER is now a reserved word.

Compression

If a non-zlib compression algorithm was used in InnoDB or Mroonga before upgrading to 10.7, those tables will be unreadable until the appropriate compression library is installed. See Compression Plugins#Upgrading.

Options That Have Changed Default Values

Option	Old default	New default
spider_auto_increment_mode	-1	0
spider_bgs_first_read	-1	2
spider_bgs_mode	-1	0
spider_bgs_second_read	-1	100
spider_bka_mode	-1	1
spider_bka_table_name_type	-1	1
spider_buffer_size	-1	16000

spider_bulk_size	-1	16000
spider_bulk_update_mode	-1	0
spider_bulk_update_size	-1	16000
spider_casual_read	-1	0
spider_connect_timeout	-1	6
spider_crd_bg_mode	-1	2
spider_crd_interval	-1	51
spider_crd_mode	-1	1
spider_crd_sync	-1	0
spider_crd_type	-1	2
spider_crd_weight	-1	2
spider_delete_all_rows_type	-1	1
spider_direct_dup_insert	-1	0
spider_direct_order_limit	-1	9223372036854775807
spider_error_read_mode	-1	0
spider_error_write_mode	-1	0
spider_first_read	-1	0
spider_init_sql_alloc_size	-1	1024
spider_internal_limit	-1	9223372036854775807
spider_internal_offset	-1	0
spider_internal_optimize	-1	0
spider_internal_optimize_local	-1	0
spider_load_crd_at_startup	-1	1
spider_load_sts_at_startup	-1	1
spider_low_mem_read	-1	1
spider_max_order	-1	32767
spider_multi_split_read	-1	100
spider_net_read_timeout	-1	600
spider_net_write_timeout	-1	600
spider_quick_mode	-1	3
spider_quick_page_byte	-1	10485760
spider_quick_page_size	-1	1024
spider_read_only_mode	-1	0
spider_reset_sql_alloc	-1	1
spider_second_read	-1	0
spider_selupd_lock_mode	-1	1
spider_semi_split_read	-1	2
spider_semi_split_read_limit	-1	1
spider_semi_table_lock_connection	-1	1
spider_reset_sql_alloc	-1	1

Options That Have Been Removed or Renamed

The following options should be removed or renamed if you use them in your option files:

Option	Reason
wsrep_replicate_myisam	Use wsrep_mode instead.
wsrep_strict_ddl	Use wsrep_mode instead.

2.1.3.7 Upgrading from MariaDB 10.5 to MariaDB 10.6

Contents

How to Upgrade

For Windows, see Upgrading MariaDB on Windows.

Before you upgrade, it would be best to take a backup of your database. This is always a good idea to do before an upgrade. We would recommend Mariabackup.

The suggested upgrade procedure is:

1. Modify the repository configuration, so the system's package manager installs MariaDB 10.6. For example,
   - On Debian, Ubuntu, and other similar Linux distributions, see Updating the MariaDB APT repository to a New Major Release for more information.
   - On RHEL, CentOS, Fedora, and other similar Linux distributions, see Updating the MariaDB YUM repository to a New Major Release for more information.
   - On SLES, OpenSUSE, and other similar Linux distributions, see Updating the MariaDB ZYpp repository to a New Major Release for more information.
2. Stop MariaDB.
3. Uninstall the old version of MariaDB.
   - On Debian, Ubuntu, and other similar Linux distributions, execute the following:
     ```
     sudo apt-get remove mariadb-server
     ```
   - On RHEL, CentOS, Fedora, and other similar Linux distributions, execute the following:
     ```
     sudo yum remove MariaDB-server
     ```
   - On SLES, OpenSUSE, and other similar Linux distributions, execute the following:
     ```
     sudo zypper remove MariaDB-server
     ```
4. Install the new version of MariaDB.
   - On Debian, Ubuntu, and other similar Linux distributions, see Installing MariaDB Packages with APT for more information.
   - On RHEL, CentOS, Fedora, and other similar Linux distributions, see Installing MariaDB Packages with YUM for more information.
   - On SLES, OpenSUSE, and other similar Linux distributions, see Installing MariaDB Packages with ZYpp for more information.
5. Make any desired changes to configuration options in option files, such as `my.cnf`. This includes removing any options that are no longer supported.
6. Start MariaDB.
7. Run mariadb-upgrade.
   - `mariadb-upgrade` does two things:
     1. Ensures that the system tables in the mysql database are fully compatible with the new version.
     2. Does a very quick check of all tables and marks them as compatible with the new version of MariaDB.

Incompatible Changes Between 10.5 and 10.6

On most servers upgrading from 10.5 should be painless. However, there are some things that have changed which could

affect an upgrade:

The bahaviour of sorting non-deterministic variables in a Select query can be changed , see (MDEV-27745)

Reserved Word

- New reserved word: OFFSET. This can no longer be used as an identifier without being quoted.

InnoDB COMPRESSED Row Format

From MariaDB 10.6.0 until MariaDB 10.6.5, tables that are of the `COMPRESSED` row format are read-only by default. This was intended to be the first step towards removing write support and deprecating the feature.

This plan has been scrapped, and from MariaDB 10.6.6, `COMPRESSED` tables are no longer read-only by default.

From MariaDB 10.6.0 to MariaDB 10.6.5, set the innodb_read_only_compressed variable to `OFF` to make the tables writable.

Character Sets

From MariaDB 10.6, the `utf8` character set (and related collations) is by default an alias for `utf8mb3` rather than the other way around. It can be set to imply `utf8mb4` by changing the value of the old_mode system variable.

Options That Have Changed Default Values

Option	Old default value	New default value
character_set_client	utf8	utf8mb3
character_set_connection	utf8	utf8mb3
character_set_results	utf8	utf8mb3
character_set_system	utf8	utf8mb3
innodb_flush_method	fsync	O_DIRECT
old_mode	Empty	UTF8_IS_UTF8MB3

Options That Have Been Removed or Renamed

The following options should be removed or renamed if you use them in your option files:

Option	Reason
innodb_adaptive_max_sleep_delay	
innodb_background_scrub_data_check_interval	
innodb_background_scrub_data_compressed	
innodb_background_scrub_data_interval	
innodb_background_scrub_data_uncompressed	
innodb_buffer_pool_instances	
innodb_checksum_algorithm	The variable is still present, but the `*innodb` and `*none` options have been removed as the `crc32` algorithm only is supported from MariaDB 10.6.
innodb_commit_concurrency	
innodb_concurrency_tickets	
innodb_file_format	
innodb_large_prefix	
innodb_lock_schedule_algorithm	
innodb_log_checksums	
innodb_log_compressed_pages	
innodb_log_files_in_group	

innodb_log_optimize_ddl	
innodb_page_cleaners	
innodb_replication_delay	
innodb_scrub_log	
innodb_scrub_log_speed	
innodb_sync_array_size	
innodb_thread_concurrency	
innodb_thread_sleep_delay	
innodb_undo_logs	

Deprecated Options

The following options have been deprecated. They have not yet been removed, but will be in a future version, and should ideally no longer be used.

Option	Reason
wsrep_replicate_myisam	Use wsrep_mode instead.
wsrep_strict_ddl	Use wsrep_mode instead.

Major New Features To Consider

- See also System Variables Added in MariaDB 10.6.

2.1.3.8 Upgrading from MariaDB 10.4 to MariaDB 10.5

Contents

How to Upgrade

For Windows, see Upgrading MariaDB on Windows instead.

Before you upgrade, it would be best to take a backup of your database. This is always a good idea to do before an upgrade. We would recommend Mariabackup.

The suggested upgrade procedure is:

1. Modify the repository configuration, so the system's package manager installs MariaDB 10.5. For example,
   - On Debian, Ubuntu, and other similar Linux distributions, see Updating the MariaDB APT repository to a New Major Release for more information.
   - On RHEL, CentOS, Fedora, and other similar Linux distributions, see Updating the MariaDB YUM repository to a New Major Release for more information.
   - On SLES, OpenSUSE, and other similar Linux distributions, see Updating the MariaDB ZYpp repository to a New Major Release for more information.
2. Stop MariaDB.
3. Uninstall the old version of MariaDB.
   - On Debian, Ubuntu, and other similar Linux distributions, execute the following:

```
sudo apt-get remove mariadb-server
```

- On RHEL, CentOS, Fedora, and other similar Linux distributions, execute the following:

```
sudo yum remove MariaDB-server
```

- On SLES, OpenSUSE, and other similar Linux distributions, execute the following:

```
sudo zypper remove MariaDB-server
```

4. Install the new version of MariaDB.
   - On Debian, Ubuntu, and other similar Linux distributions, see Installing MariaDB Packages with APT for more information.
   - On RHEL, CentOS, Fedora, and other similar Linux distributions, see Installing MariaDB Packages with YUM for more information.
   - On SLES, OpenSUSE, and other similar Linux distributions, see Installing MariaDB Packages with ZYpp for more information.
5. Make any desired changes to configuration options in option files, such as `my.cnf`. This includes removing any options that are no longer supported.
6. Start MariaDB.
7. Run mysql_upgrade.
   - `mysql_upgrade` does two things:
     1. Ensures that the system tables in the#mysql database are fully compatible with the new version.
     2. Does a very quick check of all tables and marks them as compatible with the new version of MariaDB .

Incompatible Changes Between 10.4 and 10.5

On most servers upgrading from 10.4 should be painless. However, there are some things that have changed which could affect an upgrade:

Binary name changes

All binaries previously beginning with mysql now begin with mariadb, with symlinks for the corresponding mysql command.

Usually that shouldn't cause any changed behavior, but when starting the MariaDB server via systemd, or via the mysqld_safe script symlink, the server process will now always be started as `mariadbd`, not `mysqld`.

So anything looking for the `mysqld` name in the system process list, like e.g. monitoring solutions, now needs for `mariadbd` instead when the server / service is not started directly, but via `mysqld_safe` or as a system service.

GRANT PRIVILEGE changes

A number of statements changed the privileges that they require. The old privileges were historically inappropriately chosen in the upstream. 10.5.2 fixes this problem. Note, these changes are incompatible to previous versions. A number of GRANT commands might be needed after upgrade.

- `SHOW BINLOG EVENTS` now requires the `BINLOG MONITOR` privilege (requred `REPLICATION SLAVE` prior to 10.5.2).
- `SHOW SLAVE HOSTS` now requires the `REPLICATION MASTER ADMIN` privilege (required `REPLICATION SLAVE` prior to 10.5.2).
- `SHOW SLAVE STATUS` now requires the `REPLICATION SLAVE ADMIN` or the `SUPER` privilege (required `REPLICATION CLIENT` or `SUPER` prior to 10.5.2).
- `SHOW RELAYLOG EVENTS` now requires the `REPLICATION SLAVE ADMIN` privilege (required `REPLICATION SLAVE` prior to 10.5.2).

Options That Have Changed Default Values

Option	Old default value	New default value
innodb_adaptive_hash_index	ON	OFF
innodb_checksum_algorithm	crc32	full_crc32
innodb_log_optimize_ddl	ON	OFF
slave_parallel_mode	conservative	optimistic
performance_schema_max_cond_classes	80	90
performance_schema_max_file_classes	50	80
performance_schema_max_mutex_classes	200	210
performance_schema_max_rwlock_classes	40	50
performance_schema_setup_actors_size	100	-1

performance_schema_setup_objects_size	100	-1

Options That Have Been Removed or Renamed

The following options should be removed or renamed if you use them in your option files:

Option	Reason
innodb_checksums	Deprecated and functionality replaced by innodb_checksum_algorithms in MariaDB 10.0.
idle_flush_pct	Has had no effect since merging InnoDB 5.7 from mysql-5.7.9 (MariaDB 10.2.2).
innodb_locks_unsafe_for_binlog	Deprecated in MariaDB 10.0. Use READ COMMITTED transaction isolation level instead.
innodb_rollback_segments	Deprecated and replaced by innodb_undo_logs in MariaDB 10.0.
innodb_stats_sample_pages	Deprecated in MariaDB 10.0. Use innodb_stats_transient_sample_pages instead.
max_long_data_size	Deprecated and replaced by max_allowed_packet in MariaDB 5.5.
multi_range_count	Deprecated and has had no effect since MariaDB 5.3.
thread_concurrency	Deprecated and has had no effect since MariaDB 5.5.
timed_mutexes	Deprecated and has had no effect since MariaDB 5.5.

Deprecated Options

The following options have been deprecated. They have not yet been removed, but will be in a future version, and should ideally no longer be used.

Option	Reason
innodb_adaptive_max_sleep_delay	No need for thread throttling any more.
innodb_background_scrub_data_check_interval	Problematic 'background scrubbing' code removed.
innodb_background_scrub_data_interval	Problematic 'background scrubbing' code removed.
innodb_background_scrub_data_compressed	Problematic 'background scrubbing' code removed.
innodb_background_scrub_data_uncompressed	Problematic 'background scrubbing' code removed.
innodb_buffer_pool_instances	Having more than one buffer pool is no longer necessary.
innodb_commit_concurrency	No need for thread throttling any more.
innodb_concurrency_tickets	No need for thread throttling any more.
innodb_log_files_in_group	Redo log was unnecessarily split into multiple files. Limited to 1 from MariaDB 10.5.
innodb_log_optimize_ddl	Prohibited optimizations.
innodb_page_cleaners	Having more than one page cleaner task no longer necessary.
innodb_replication_delay	No need for thread throttling any more.
innodb_scrub_log	Never really worked as intended, redo log format is being redone.
innodb_scrub_log_speed	Never really worked as intended, redo log format is being redone.
innodb_thread_concurrency	No need for thread throttling any more.
innodb_thread_sleep_delay	No need for thread throttling any more.
innodb_undo_logs	It always makes sense to use the maximum number of rollback segments.
large_page_size	Unused since multiple page size support was added.

Major New Features To Consider

You might consider using the following major new features in MariaDB 10.5:

- The S3 storage engine allows one to archive MariaDB tables in Amazon S3, or any third-party public or private cloud

that implements S3 API.
- ColumnStore columnar storage engine.
- See also System Variables Added in MariaDB 10.5.

2.1.3.9 Upgrading from MariaDB 10.3 to MariaDB 10.4

Contents

How to Upgrade

For Windows, see Upgrading MariaDB on Windows instead.

For MariaDB Galera Cluster, see Upgrading from MariaDB 10.3 to MariaDB 10.4 with Galera Cluster instead.

Before you upgrade, it would be best to take a backup of your database. This is always a good idea to do before an upgrade. We would recommend Mariabackup.

The suggested upgrade procedure is:

1. Modify the repository configuration, so the system's package manager installs MariaDB 10.4. For example,
   - On Debian, Ubuntu, and other similar Linux distributions, see Updating the MariaDB APT repository to a New Major Release for more information.
   - On RHEL, CentOS, Fedora, and other similar Linux distributions, see Updating the MariaDB YUM repository to a New Major Release for more information.
   - On SLES, OpenSUSE, and other similar Linux distributions, see Updating the MariaDB ZYpp repository to a New Major Release for more information.
2. Stop MariaDB.
3. Uninstall the old version of MariaDB.
   - On Debian, Ubuntu, and other similar Linux distributions, execute the following:
     ```
     sudo apt-get remove mariadb-server
     ```
   - On RHEL, CentOS, Fedora, and other similar Linux distributions, execute the following:
     ```
     sudo yum remove MariaDB-server
     ```
   - On SLES, OpenSUSE, and other similar Linux distributions, execute the following:
     ```
     sudo zypper remove MariaDB-server
     ```
4. Install the new version of MariaDB.
   - On Debian, Ubuntu, and other similar Linux distributions, see Installing MariaDB Packages with APT for more information.
   - On RHEL, CentOS, Fedora, and other similar Linux distributions, see Installing MariaDB Packages with YUM for more information.
   - On SLES, OpenSUSE, and other similar Linux distributions, see Installing MariaDB Packages with ZYpp for more information.
5. Make any desired changes to configuration options in option files, such as `my.cnf`. This includes removing any options that are no longer supported.
6. Start MariaDB.
7. Run `mysql_upgrade`.
   - `mysql_upgrade` does two things:
     1. Ensures that the system tables in the `mysql` database are fully compatible with the new version.
     2. Does a very quick check of all tables and marks them as compatible with the new version of MariaDB.

Incompatible Changes Between 10.3 and 10.4

On most servers upgrading from 10.3 should be painless. However, there are some things that have changed which could affect an upgrade:

Options That Have Changed Default Values

Option	Old default value	New default value
slave_transaction_retry_errors	1213,1205	1158,1159,1160,1161,1205,1213,1429,2013,12701
wsrep_debug	OFF	NONE
wsrep_load_data_splitting	ON	OFF

Options That Have Been Removed or Renamed

The following options should be removed or renamed if you use them in your option files:

Option	Reason

Authentication and TLS

- See Authentication from MariaDB 10.4 for an overview of the changes.
- The unix_socket authentication plugin is now default on Unix-like systems.
- TLSv1.0 is disabled by default in MariaDB 10.4. See tls_version and TLS Protocol Versions.

Major New Features To Consider

You might consider using the following major new features in MariaDB 10.4:

- Galera has been upgraded from Galera 3 to Galera 4.
- System-versioning extended with support for application-time periods.
- User password expiry
- Account Locking
- See also System Variables Added in MariaDB 10.4.

2.1.3.10 Upgrading MariaDB on Windows

Contents

For incompatibilities such as removed features, and changes to variables, see the pages describing changes by version on Upgrading MariaDB.

Minor Upgrades

To install a minor upgrade, e.g 10.1.27 on top of existing 10.1.26, with MSI, just download the 10.1.27 MSI and start it. It will do everything that needs to be done for minor upgrade automatically - shutdown MariaDB service(s), replace executables and DLLs, and start service(s) again.

The rest of the article is dedicated to *major* upgrades, e.g 10.1.x to 10.2.y.

General Information on Upgrade and Version Coexistence

This section assumes MSI installations.

First, check everything listed in the Incompatibilities section of the article relating to the version you are upgrading, for example, Upgrading from MariaDB 10.1 to MariaDB 10.2, to make sure you are prepared for the upgrade.

MariaDB (and also MySQL) allows different versions of the product to co-exist on the same machine, as long as these versions are different either in major or minor version numbers. For example, it is possible to have say MariaDB 5.1.51

and 5.2.6 to be installed on the same machine.

However only a single instance of 5.2 can exist. If for example 5.2.7 is installed on a machine where 5.2.6 is already installed, the installer will just replace 5.2.6 executables with 5.2.7 ones.

Now imagine, that both 5.1 and 5.2 are installed on the same machine and we want to upgrade the database instance running on 5.1 to the new version. In this case special tools are requied. Traditionally, `mysql_upgrade` is used to accomplish this. On Windows, the MySQL upgrade is a complicated multiple-step manual process.

Since MariaDB 5.2.6, the Windows distribution includes tools that simplify migration between different versions and also allow migration between MySQL and MariaDB.

Note. Automatic upgrades are only possible for DB instances that run as a Windows service.

General Recommendations

Important: Ignore any statement that tells you to *"just uninstall MySQL and install MariaDB"*. This does not work on Windows, never has, and never will. Keep your MySQL installed until after the database had been converted.

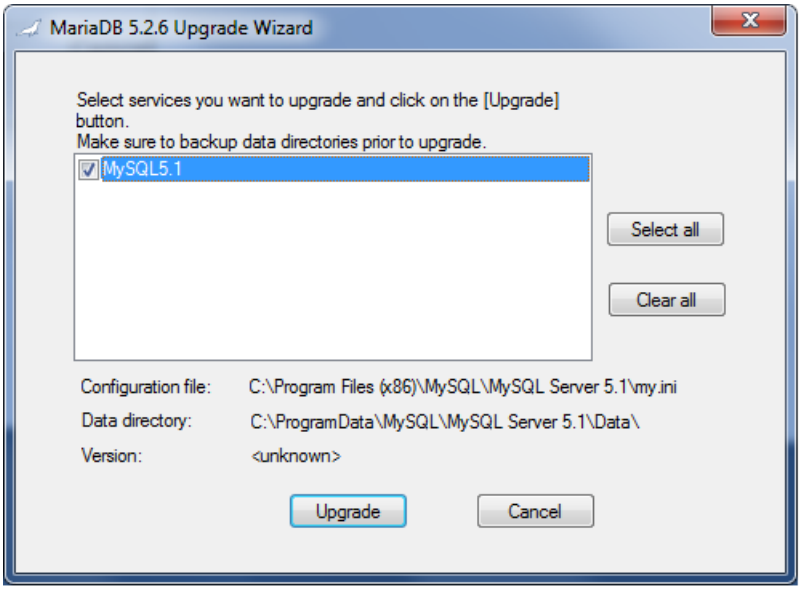
The following install/upgrade sequence is recommended in case of "major" upgrades, like going from 5.3 to 5.5

- Install new version, while still retaining the old one
- Upgrade services one by one, like described later in the document (e.g with mysql_upgrade_service). It is recommeded to have services cleanly shut down before the upgrade.
- Uninstall old version when previous step is done.

Note. This recommendation differs from the procedure on Unixes, where the upgrade sequence is "uninstall old version, install new version"

Upgrade Wizard

This is a GUI tool that is typically invoked at the end of a MariaDB installation if upgradable services are found. The UI allows you to select instances you want to upgrade.



mysql_upgrade_service

This is a command line tool that performs upgrades. The tool requires full administrative privileges (it has to start and stop services).

Example usage:

```
mysql_upgrade_service --service=MySQL
```

`mysql_upgrade_service` accepts a single parameter — the name of the MySQL or MariaDB service. It performs all the steps to convert a MariaDB/MySQL instance running as the service to the current version.

Migration to 64 bit MariaDB from 32 bit

Earlier we said that only single instance of "MariaDB <major>.<minor>" version can be installed on the same machine. This was almost correct, because MariaDB MSI installations allow 32 and 64-bit versions to be installed on the same machine, and in this case it is possible to have two instances of say 5.2 installed at the same time, an x86 one and an x64 one. One can use the x64 Upgrade wizard to upgrade an instance running as a 32-bit process to run as 64-bit.

Upgrading ZIP-based Installations.

Both UpgradeWizard and mysql_upgrade_service can also be used to upgrade database instances that were installed with the ZIP installation.

2.1.3.11 Upgrading Galera Cluster

Upgrading Between Minor Versions with Galera Cluster

Upgrading between minor versions of MariaDB with Galera Cluster, e.g. from ...

Upgrading from MariaDB 10.3 to MariaDB 10.4 with Galera Cluster

How to upgrade from MariaDB 10.3 to MariaDB 10.4 in a Galera Cluster deployment.

Upgrading from MariaDB 10.2 to MariaDB 10.3 with Galera Cluster

How to upgrade from MariaDB 10.2 to MariaDB 10.3 in a Galera Cluster deployment.

Upgrading from MariaDB 10.1 to MariaDB 10.2 with Galera Cluster

How to upgrade from MariaDB 10.1 to MariaDB 10.2 in a Galera Cluster deployment.

Upgrading from MariaDB Galera Cluster 10.0 to MariaDB 10.1 with Galera Cluster

How to upgrade from MariaDB Galera Cluster 10.0 to MariaDB 10.1 in a Galera Cluster deployment.

Upgrading from MariaDB Galera Cluster 5.5 to MariaDB Galera Cluster 10.0

How to upgrade from MariaDB Galera Cluster 5.5 to MariaDB Galera Cluster 10.0

There are 1 related questions.

2.1.3.11.1 Upgrading Between Minor Versions with Galera Cluster

Performing a Rolling Upgrade

The following steps can be used to perform a rolling upgrade between minor versions of MariaDB (for example from MariaDB 10.3.12 to MariaDB 10.3.13) when Galera Cluster is being used. In a rolling upgrade, each node is upgraded individually, so the cluster is always operational. There is no downtime from the application's perspective.

> Before you upgrade, it would be best to take a backup of your database. This is always a good idea to do before an upgrade. We would recommend Mariabackup.

For each node, perform the following steps:

1. Stop MariaDB.
2. Install the new version of MariaDB and the Galera wsrep provider.
   - On Debian, Ubuntu, and other similar Linux distributions, see Installing MariaDB Packages with APT for more information.
   - On RHEL, CentOS, Fedora, and other similar Linux distributions, see Installing MariaDB Packages with YUM for more information.
   - On SLES, OpenSUSE, and other similar Linux distributions, see Installing MariaDB Packages with ZYpp for more information.
3. Make any desired changes to configuration options in option files, such as `my.cnf`. This includes removing any system variables or options that are no longer supported.

4. Start MariaDB.
5. Run mariadb-upgrade with the `--skip-write-binlog` option.
    - `mariadb-upgrade` does two things:
        1. Ensures that the system tables in the `mysql` database are fully compatible with the new version.
        2. Does a very quick check of all tables and marks them as compatible with the new version of MariaDB .

When this process is done for one node, move onto the next node.

Note that when upgrading the Galera wsrep provider, sometimes the Galera protocol version can change. The Galera wsrep provider should not start using the new protocol version until all cluster nodes have been upgraded to the new version, so this is not generally an issue during a rolling upgrade. However, this can cause issues if you restart a non-upgraded node in a cluster where the rest of the nodes have been upgraded.

2.1.3.11.2 Upgrading from MariaDB 10.3 to MariaDB 10.4 with Galera Cluster

MariaDB starting with 10.1

Since MariaDB 10.1, the MySQL-wsrep patch has been merged into MariaDB Server. Therefore, in MariaDB 10.1 and above, the functionality of MariaDB Galera Cluster can be obtained by installing the standard MariaDB Server packages and the Galera wsrep provider library package.

Beginning in MariaDB 10.1, Galera Cluster ships with the MariaDB Server. Upgrading a Galera Cluster node is very similar to upgrading a server from MariaDB 10.3 to MariaDB 10.4. For more information on that process as well as incompatibilities between versions, see the Upgrade Guide.

Performing a Rolling Upgrade

The following steps can be used to perform a rolling upgrade from MariaDB 10.3 to MariaDB 10.4 when using Galera Cluster. In a rolling upgrade, each node is upgraded individually, so the cluster is always operational. There is no downtime from the application's perspective.

First, before you get started:

1. First, take a look at Upgrading from MariaDB 10.3 to MariaDB 10.4 to see what has changed between the major versions.
    1. Check whether any system variables or options have been changed or removed. Make sure that your server's configuration is compatible with the new MariaDB version before upgrading.
    2. Check whether replication has changed in the new MariaDB version in any way that could cause issues while the cluster contains upgraded and non-upgraded nodes.
    3. Check whether any new features have been added to the new MariaDB version. If a new feature in the new MariaDB version cannot be replicated to the old MariaDB version, then do not use that feature until all cluster nodes have been upgrades to the new MariaDB version.
2. Next, make sure that the Galera version numbers are compatible.
    1. If you are upgrading from the most recent MariaDB 10.3 release to MariaDB 10.4, then the versions will be compatible. MariaDB 10.3 uses Galera 3 (i.e. Galera wsrep provider versions 25.3.x), and MariaDB 10.4 uses Galera 4 (i.e. Galera wsrep provider versions 26.4.x). This means that upgrading to MariaDB 10.4 also upgrades the system to Galera 4. However, Galera 3 and Galera 4 should be compatible for the purposes of a rolling upgrade, as long as you are using Galera 26.4.2 or later.
    2. See What is MariaDB Galera Cluster?: Galera wsrep provider Versions for information on which MariaDB releases uses which Galera wsrep provider versions.
3. Ideally, you want to have a large enough gcache to avoid a State Snapshot Transfer (SST) during the rolling upgrade. The gcache size can be configured by setting `gcache.size` For example:
   ```
   wsrep_provider_options="gcache.size=2G"
   ```

Before you upgrade, it would be best to take a backup of your database. This is always a good idea to do before an upgrade. We would recommend Mariabackup.

Then, for each node, perform the following steps:

1. Modify the repository configuration, so the system's package manager installs MariaDB 10.4. For example,
    - On Debian, Ubuntu, and other similar Linux distributions, see Updating the MariaDB APT repository to a New Major Release for more information.
    - On RHEL, CentOS, Fedora, and other similar Linux distributions, see Updating the MariaDB YUM repository to

a New Major Release for more information.
    - On SLES, OpenSUSE, and other similar Linux distributions, see Updating the MariaDB ZYpp repository to a New Major Release for more information.
2. If you use a load balancing proxy such as MaxScale or HAProxy, make sure to drain the server from the pool so it does not receive any new connections.
3. Stop MariaDB.
4. Uninstall the old version of MariaDB and the Galera wsrep provider.
    - On Debian, Ubuntu, and other similar Linux distributions, execute the following:
      ```
      sudo apt-get remove mariadb-server galera
      ```
    - On RHEL, CentOS, Fedora, and other similar Linux distributions, execute the following:
      ```
      sudo yum remove MariaDB-server galera
      ```
    - On SLES, OpenSUSE, and other similar Linux distributions, execute the following:
      ```
      sudo zypper remove MariaDB-server galera
      ```
5. Install the new version of MariaDB and the Galera wsrep provider.
    - On Debian, Ubuntu, and other similar Linux distributions, see Installing MariaDB Packages with APT for more information.
    - On RHEL, CentOS, Fedora, and other similar Linux distributions, see Installing MariaDB Packages with YUM for more information.
    - On SLES, OpenSUSE, and other similar Linux distributions, see Installing MariaDB Packages with ZYpp for more information.
6. Make any desired changes to configuration options in option files, such as `my.cnf`. This includes removing any system variables or options that are no longer supported.
7. On Linux distributions that use `systemd` you may need to increase the service startup timeout as the default timeout of 90 seconds may not be sufficient. See Systemd: Configuring the Systemd Service Timeout for more information.
8. Start MariaDB.
9. Run `mysql_upgrade` with the `--skip-write-binlog` option.
    - `mysql_upgrade` does two things:
        1. Ensures that the system tables in the `mysql` database are fully compatible with the new version.
        2. Does a very quick check of all tables and marks them as compatible with the new version of MariaDB .

When this process is done for one node, move onto the next node.

> Note that when upgrading the Galera wsrep provider, sometimes the Galera protocol version can change. The Galera wsrep provider should not start using the new protocol version until all cluster nodes have been upgraded to the new version, so this is not generally an issue during a rolling upgrade. However, this can cause issues if you restart a non-upgraded node in a cluster where the rest of the nodes have been upgraded.

2.1.3.12 Upgrading from MySQL to MariaDB

Upgrading from MySQL to MariaDB
Upgrading from MySQL to MariaDB.

Moving from MySQL to MariaDB in Debian 9
MariaDB 10.1 is the default mysql server in Debian 9 "Stretch"

Screencast for Upgrading MySQL to MariaDB
Screencast for upgrading MySQL 5.1.55 to MariaDB

Upgrading from MySQL 5.7 to MariaDB 10.2
Following compatibility report was done on 10.2.4 and may get some fixing i...

Upgrading to MariaDB From MySQL 5.0 or Older
Upgrading to MariaDB from MySQL 5.0 (or older version)

2.1.3.12.1 Upgrading from MySQL to MariaDB

Contents

For all practical purposes, you can view MariaDB as an upgrade of MySQL:

- Before upgrading, please check if there are any known incompatibilities between your MySQL release and the MariaDB release you want to move to.
- In particular, note that the JSON type in MariaDB is a LONGTEXT, while in MySQL it's a binary type. See Making MariaDB understand MySQL JSON.
- If you are using MySQL 8.0 or above, you have to use mysqldump to move your database to MariaDB.
- For upgrading from very old MySQL versions, see Upgrading to MariaDB from MySQL 5.0 (or older version).
- Within the same base version (for example MySQL 5.5 -> MariaDB 5.5, MySQL 5.6 -> MariaDB 10.0 and MySQL 5.7 -> MariaDB 10.2) you can in most cases just uninstall MySQL and install MariaDB and you are good to go. There is no need to dump and restore databases. As with any upgrade, we recommend making a backup of your data beforehand.
- You should run mariadb-upgrade (as you would with `mysql_upgrade` in MySQL) to finish the upgrade. This is needed to ensure that your mysql privilege and event tables are updated with the new fields MariaDB uses. Note that if you use a MariaDB package, `mariadb-upgrade` is usually run automatically.
- All your old clients and connectors (PHP, Perl, Python, Java, etc.) will work unchanged (no need to recompile). This works because MariaDB and MySQL use the same client protocol and the client libraries are binary compatible. You can also use your old MySQL connector packages with MariaDB if you want.

Upgrading on Windows

On Windows, you should not uninstall MySQL and install MariaDB, this would not work, the existing database will not be found.

Thus On Windows, just install MariaDB and use the upgrade wizard which is part of installer package and is launched by MSI installer. Or, in case you prefer command line, use `mysql_upgrade_service <service_name>` on the command line.

Upgrading my.cnf

All the options in your original MySQL `my.cnf` file should work fine for MariaDB.

However as MariaDB has more features than MySQL, there is a few things that you should consider changing in your `my.cnf` file.

- MariaDB uses by default the Aria storage engine for internal temporary files instead of MyISAM. If you have a lot of temporary files, you should add and set `aria-pagecache-buffer-size` to the same value as you have for `key-buffer-size`.
- If you don't use MyISAM tables, you can set `key-buffer-size` to a very low value, like 64K.
- If using MariaDB 10.1 or earlier, and your applications often connect and disconnect to MariaDB, you should set up `thread-cache-size` to the number of concurrent queries threads you are typically running. This is important in MariaDB as we are using the jemalloc memory allocator. jemalloc usually has better performance when running many threads compared to other memory allocators, except if you create and destroy a lot of threads, in which case it will spend a lot of resources trying to manage thread specific storage. Having a thread cache will fix this problem.
- If you have a LOT of connections (> 100) that mostly run short running queries, you should consider using the thread pool. For example using : `thread_handling=pool-of-threads` and `thread_pool_size=128` could give a notable performance boost in this case. Where the `thread_pool_size` should be about `2 * number of cores on your machine`.

Other Things to Think About

- Views with definition `ALGORITHM=MERGE` or `ALGORITHM=TEMPTABLE` got accidentally swapped between MariaDB and MySQL. You have to re-create views created with either of these definitions (see MDEV-6916).
- MariaDB has LGPL versions of the C connector and Java Client. If you are shipping an application that supports MariaDB or MySQL, you should consider using these!
- You should consider trying out the MyRocks storage engine or some of the other new storage engines that MariaDB provides.

2.1.3.12.2 Moving from MySQL to MariaDB in Debian 9

MariaDB 10.1 is now the default mysql server in Debian 9 "Stretch". This page provides information on this change and instructions to help with upgrading your Debian 8 "Jessie" version of MySQL or MariaDB to MariaDB 10.1 in Debian 9 "Stretch".

Contents

Background information

The version of MySQL in Debian 8 "Jessie" is 5.5. When installing, most users will install the `mysql-server` package, which depends on the `mysql-server-5.5 package` . In Debian 9 "Stretch" the `mysql-server` package depends on a new package called `default-mysql-server` . This package in turn depends on `mariadb-server-10.1` . There is no `default-mysql-server` package in Jessie.

In both Jessie and Stretch there is also a `mariadb-server` package which is a MariaDB-specific analog to the `mysql-server` package. In Jessie this package depends on `mariadb-server-10.0` and in Stretch this package depends on `mariadb-server-10.1` (the same as the `default-mysql-server` package).

So, the main repository difference in Debian 9 "Stretch" is that when you install the `mysql-server` package on Stretch you will get MariaDB 10.1 instead of MySQL, like you would with previous versions of Debian. Note that `mysql-server` is just an empty transitional meta-package and users are encouraged to install MariaDB using the actual package `mariadb-server` .

All apps and tools, such as the popular LAMP stack, in the repositories that depend on the `mysql-server` package will continue to work using MariaDB as the database. For new installs there is nothing different that needs to be done when installing the mysql-server or mariadb-server packages.

Before you upgrade

If you are currently running MySQL 5.5 on Debian 8 "Jessie" and are planning an upgrade to MariaDB 10.1 on Debian 9 "Stretch", there are some things to keep in mind:

Backup before you begin

This is a major upgrade, and so complete database backups are strongly suggested before you begin. MariaDB 10.1 is compatible on disk and wire with MySQL 5.5, and the MariaDB developer team has done extensive development and testing to make upgrades as painless and trouble-free as possible. Even so, it's always a good idea to do regular backups, especially before an upgrade. As the database has to shutdown anyway for the upgrade, this is a good opportunity to do a backup!

Changed, renamed, and removed options

Some default values have been changed, some have been renamed, and others have been removed between MySQL 5.5 and MariaDB 10.1. The following sections detail them.

Options with changed default values

Most of the following options have increased a bit in value to give better performance. They should not use much additional memory, but some of them do use a bit more disk space.

Option	Old default value	New default value
`aria-sort-buffer-size`	`128M`	`256M`
`back_log`	`50`	`150`

innodb-concurrency-tickets	500	5000
innodb-log-file-size	5M	48M
innodb_log_compressed_pages	ON	OFF
innodb-old-blocks-time	0	1000
innodb-open-files	300	400 [2]
innodb-purge-batch-size	20	300
innodb-undo-logs	ON	20
join_buffer_size	128K	256K
max_allowed_packet	1M	4M
max-connect-errors	10	100
max-relay-log-size	0	1024M
myisam-sort-buffer-size	8M	128M
optimizer-switch	...	Added extended_keys=on, exists_to_in=on
query_alloc_block_size	8192	16384
query_cache_size	0	1M
query_cache_type	ON	OFF
query_prealloc_size	8192	24576
secure_auth	OFF	ON
sql_log_bin		No longer affects replication of events in a Galera cluster.
sql_mode	empty	NO_AUTO_CREATE_USER, NO_ENGINE_SUBSTITUTION
sync_master_info	0	10000
sync_relay_log	0	10000
sync_relay_log_info	0	10000
table_open_cache	400	2000
thread_pool_max_threads	500	1000

Options that have been removed or renamed

The following options should be removed or renamed if you use them in your config files:

Option	Reason
engine-condition-pushdown	Replaced with set optimizer_switch='engine_condition_pushdown=on'
innodb-adaptive-flushing-method	Removed by XtraDB
innodb-autoextend-increment	Removed by XtraDB
innodb-blocking-buffer-pool-restore	Removed by XtraDB
innodb-buffer-pool-pages	Removed by XtraDB
innodb-buffer-pool-pages-blob	Removed by XtraDB
innodb-buffer-pool-pages-index	Removed by XtraDB
innodb-buffer-pool-restore-at-startup	Removed by XtraDB
innodb-buffer-pool-shm-checksum	Removed by XtraDB
innodb-buffer-pool-shm-key	Removed by XtraDB
innodb-checkpoint-age-target	Removed by XtraDB

innodb-dict-size-limit	Removed by XtraDB
innodb-doublewrite-file	Removed by XtraDB
innodb-fast-checksum	Renamed to innodb-checksum-algorithm
innodb-flush-neighbor-pages	Renamed to innodb-flush-neighbors
innodb-ibuf-accel-rate	Removed by XtraDB
innodb-ibuf-active-contract	Removed by XtraDB
innodb-ibuf-max-size	Removed by XtraDB
innodb-import-table-from-xtrabackup	Removed by XtraDB
innodb-index-stats	Removed by XtraDB
innodb-lazy-drop-table	Removed by XtraDB
innodb-merge-sort-block-size	Removed by XtraDB
innodb-persistent-stats-root-page	Removed by XtraDB
innodb-read-ahead	Removed by XtraDB
innodb-recovery-stats	Removed by XtraDB
innodb-recovery-update-relay-log	Removed by XtraDB
innodb-stats-auto-update	Renamed to innodb-stats-auto-recalc
innodb-stats-update-need-lock	Removed by XtraDB
innodb-sys-stats	Removed by XtraDB
innodb-table-stats	Removed by XtraDB
innodb-thread-concurrency-timer-based	Removed by XtraDB
innodb-use-sys-stats-table	Removed by XtraDB
rpl_recovery_rank	Unused in 10.0+
xtradb-admin-command	Removed by XtraDB

Suggested upgrade procedure for replication

If you have a master-slave setup, the normal procedure is to first upgrade your slaves to MariaDB, then move one of your slaves to be the master and then upgrade your original master. In this scenario you can upgrade from MySQL to MariaDB or upgrade later to a new version of MariaDB without any downtime.

Other resources to consult before beginning your upgrade

It may also be useful to check out the Upgrading MariaDB section. It contains several articles on upgrading from MySQL to MariaDB and from one version of MariaDB to another. For upgrade purposes, MySQL 5.5 and MariaDB 5.5 are very similar. In particular, see the Upgrading from MariaDB 5.5 to MariaDB 10.0 and Upgrading from MariaDB 10.0 to MariaDB 10.1 articles.

If you need help with upgrading or setting up replication, you can always contact the MariaDB corporation to find experts to help you with this.

Upgrading to MariaDB 10.1 from MySQL 5.5

The suggested upgrade procedure is:

1. Set innodb_fast_shutdown to `0`. This is to ensure that if you make a backup as part of the upgrade, all data is written to the InnoDB data files, which simplifies any restore in the future.
2. Shutdown MySQL 5.5
3. Take a backup
    - when the server is shut down is the perfect time to take a backup of your databases
    - store a copy of the backup on external media or a different machine for safety

4. Perform the upgrade from Debian 8 to Debian 9
5. During the upgrade, the mysql_upgrade script will be run automatically; this script does two things:
    1. Upgrades the permission tables in the `mysql` database with some new fields
    2. Does a very quick check of all tables and marks them as compatible with MariaDB 10.1
        - In most cases this should be a fast operation (depending of course on the number of tables)
6. Add new options to my.cnf to enable features
    - If you change `my.cnf` then you need to restart `mysqld` with e.g. `sudo service mysql restart` or `sudo service mariadb restart`.

Upgrading to MariaDB 10.1 from an older version of MariaDB

If you have installed MariaDB 5.5 or MariaDB 10.0 on your Debian 8 "Jessie" machine from the MariaDB repositories you will need to upgrade to MariaDB 10.1 when upgrading to Debian 9 "Stretch". You can choose to continue using the MariaDB repositories or move to using the Debian repositories.

If you want to continue using the MariaDB repositories edit the MariaDB entry in your sources.list and change every instance of 5.5 or 10.0 to 10.1. Then upgrade as suggested above.

If you want to move to using MariaDB 10.1 from the Debian repositories, delete or comment out the MariaDB entries in your sources.list file. Then upgrade as suggested above.

If you are already using MariaDB 10.1 on your Debian 8 "Jessie" machine, you can choose to continue to use the MariaDB repositories or move to using the Debian repositories as with MariaDB 5.5 and 10.0. In either case, the upgrade will at most be just a minor upgrade from one version of MariaDB 10.1 to a newer version. In the case that you are already on the current version of MariaDB that exists in the Debian repositories or a newer one) MariaDB will not be upgraded during the system upgrade but will be upgraded when future versions of MariaDB are released.

> You should always perform a compete backup of your data prior to performing any major system upgrade, even if MariaDB itself is not being upgraded!

MariaDB Galera Cluster

If you have been using MariaDB Galera Cluster 5.5 or 10.0 on Debian 8 "Jessie" it is worth mentioning that Galera Cluster is included by default in MariaDB 10.1, there is no longer a need to install a separate `mariadb-galera-server` package.

Configuration options for advanced database users

To get better performance from MariaDB used in production environments, here are some suggested additions to your configuration file which in Debian is at `/etc/mysql/mariadb.d/my.cnf`:

```
[[mysqld]]
# Cache for disk based temporary files
aria_pagecache_buffer_size=128M
# If you are not using MyISAM tables directly (most people are using InnoDB)
key_buffer_size=64K
```

The reason for the above change is that MariaDB is using the newer Aria storage engine for disk based temporary files instead of MyISAM. The main benefit of Aria is that it can cache both indexes and rows and thus gives better performance than MyISAM for large queries.

Secure passwordless root accounts only on new installs

Unlike the old MySQL packages in Debian, MariaDB 10.0 onwards in Debian uses unix socket authentication on new installs to avoid root password management issues and thus be more secure and easier to use with provision systems of the cloud age.

This only affects new installs. Upgrades from old versions will continue to use whatever authentication and user accounts already existed. This is however good to know, because it can affect upgrades of dependant systems, typically e.g. require users to rewrite their Ansible scripts and similar tasks. The new feature is much easier than the old, so adjusting for it requires little work.

2.1.3.12.3 Screencast for Upgrading MySQL to MariaDB

There is a screencast for upgrading from MySQL 5.1.55 to MariaDB. Watch this example to see how easy this process is. It really is just a "drop in replacement" to MySQL.

2.1.4 Downgrading between Major Versions of MariaDB

Downgrading MariaDB is not officially supported between major versions.

For minor versions, upgrade is supported to an earlier gamma/RC/GA version as we do not change the storage format after Alpha and very rarely during Beta (it has to be a very critical bug to require such a change). There are a few very rare cases when incompatible changes happen on a GA version, for example MariaDB 10.1.21 fixed a file format incompatibility bug that prevents a downgrade to earlier MariaDB 10.1 releases. After MariaDB 10.1.21 this has not happened in a GA release.

The main reason why downgrades between major versions do not work are:

- Changes in the privilege/status tables in the mysql schema. These changes happen between most major versions as we are continuously improving the privilege system.
- Changes that affect how data is stored on disk. This happens more rarely and is usually table specific. For example, if one has used Instant add column on a table in MariaDB 10.3, that table cannot be opened in MariaDB 10.2.
- Between major releases there are often substantial changes, even if none of the new features are used. For example, both MariaDB 10.2 and MariaDB 10.3 introduce new versions of the redo log.

The only reliable way to downgrade is to restore from a full backup made before upgrading, and start the old version of MariaDB. At least one should take a backup of the mysql schema as most upgrade changes happens in this directory. This may be of help if one needs to downgrade to an earlier MariaDB version. More about this later.

Some people have reported successfully downgrading, but there are many possible things that can go wrong, and downgrading between two major versions is not tested in any way by the MariaDB developers.

In general, one can downgrade a major version to an earlier version if one has not yet run mariadb-upgrade on the new version. Note however that it's recommended that one always uses mariadb-upgrade after upgrading to a new major version as otherwise some security features in the new server may not work and tables that have indexes using a character collation that has changed may not work properly.

Assuming one **must** downgrade to an earlier major version, here is a list of things one has to do:

- MariaDB **must** be shut down cleanly. This means that:
  - One should ensure that innodb_fast_shutdown≠2.
  - One uses the SHUTDOWN command, mariadb-admin shutdown or the operating system official commands, like systemctl stop mariadb.service.
- Start the old server with --skip-privilege-tables.
- Use ALTER TABLE to restore the mysql schema tables to their original definition or drop and recreate the mysql tables. One can find the old definition by using mariadb-install-db to create a separate temporary data directory. Starting the MariaDB server on the temporary directory will allow you to use SHOW CREATE TABLE to find the old definition.
- Execute FLUSH PRIVILEGES to reload the old tables.

The cases when the above will not work are when the table format has changed in an incompatible manner. In this case the affected tables may not be usable in the earlier version.

The following is an incomplete list of when one will not be able to use a table in an earlier major version:

- MariaDB 11.0 or later
  - A downgrade to MariaDB 10.4 or earlier is not possible, because MDEV-29694 removed the InnoDB change buffer.
  - A downgrade to MariaDB 10.5 or later is only possible if innodb_change_buffering=none (the default starting with MDEV-27734).
- MariaDB 10.8 or later
  - The InnoDB redo log file `ib_logfile0` would have to be replaced with a logically equivalent file, or the shutdown LSN has to be written to the `FIL_PAGE_FILE_FLUSH_LSN` field in the system tablespace (see MDEV-27199), or the data may be accessed read-only when using innodb_force_recovery=6.
- MariaDB 10.5 → MariaDB 10.4
  - The InnoDB redo log file `ib_logfile0` has to be deleted between a clean shutdown of the newer version and a startup of the older version. This is *not recommended*.
- MariaDB 10.4 → MariaDB 10.3

- Any InnoDB table where one has used `ALTER TABLE ALGORITHM=INSTANT DROP COLUMN` while innodb_instant_alter_column_allowed=add_drop_reorder
- Any InnoDB table that was created or rebuilt while innodb_checksum_algorithm=full_crc32
- In MariaDB 10.4, the MariaDB mysql.user table was replaced by mysql.global_priv table which may cause problems if ones wants to downgrade to 10.3.

- MariaDB 10.3 → MariaDB 10.2
  - Any InnoDB table where one has used `ALTER TABLE…ADD COLUMN` (unless innodb_instant_alter_column_allowed=never).
  - A prior shutdown with innodb_fast_shutdown=0 will be needed in order to empty the undo logs whose format changed in MDEV-12288, and even then, you might need to set innodb_force_recovery=3.

2.1.2.8 Compiling MariaDB From Source

2.1.6 Starting and Stopping MariaDB

Starting and Stopping MariaDB Server

Starting MariaDB, including details on service managers.

Configuring MariaDB with Option Files

Configuring MariaDB with my.cnf and other option files.

mysqld Configuration Files and Groups

Which configuration files and groups mysqld reads.

mariadbd Options

Lists of all the options for mariadbd (previously called mysqld).

What to Do if MariaDB Doesn't Start

Troubleshooting MariaDB when it fails to start.

Running MariaDB from the Build Directory

Running mariadbd (mysqld) directly from the source directory without make install.

mysql.server

Startup script included in MariaDB distributions on Unix

mysqld_safe

Recommended way to start a mysqld server on a non-systemd Unix.

mysqladmin

Old name or symlink for mariadb-admin.

Switching Between Different Installed MariaDB Versions

Managing different installed MariaDB versions and running them one at a time

Running Multiple MariaDB Server Processes

Running multiple MariaDB Server processes on the same server.

Specifying Permissions for Schema (Data) Directories and Tables

MariaDB uses the following modes for creating directories and files

mysqld_multi

Manage several mysqld processes.

launchd

launchd is the startup service used in MacOS X.

systemd

How systemd is configured on MariaDB packages and how to alter its configuration.

sysVinit

sysVinit is one of the most common service managers for Linux and Unix.

mariadb-admin

Admin tool for monitoring, creating/dropping databases, stopping MariaDB etc.

mariadbd

Symlink or new name for mysqld.

mariadbd-multi

Symlink or new name for mysqld_multi.

mariadbd-safe

Symlink or new name for mysqld_safe.

There are 21 related questions.

2.1.6.1 Starting and Stopping MariaDB Server

Contents

There are several different methods to start or stop the MariaDB Server process. There are two primary categories that most of these methods fall into: starting the process with the help of a service manager, and starting the process manually.

Service Managers

sysVinit and systemd are the most common Linux service managers. launchd is used in MacOS X. Upstart is a less common service manager.

Systemd

RHEL/CentOS 7 and above, Debian 8 Jessie and above, and Ubuntu 15.04 and above use systemd by default.

For information on how to start and stop MariaDB with this service manager, see systemd: Interacting with the MariaDB Server Process.

SysVinit

RHEL/CentOS 6 and below, and Debian 7 Wheezy and below use sysVinit by default.

For information on how to start and stop MariaDB with this service manager, see sysVinit: Interacting with the MariaDB Server Process.

launchd

launchd is used in MacOS X.

Upstart

Ubuntu 14.10 and below use Upstart by default.

Starting the Server Process Manually

mariadbd

mariadbd is the actual MariaDB Server binary. It can be started manually on its own.

mysqld_safe

mariadbd_safe is a wrapper that can be used to start the mariadbd server process. The script has some built-in safeguards, such as automatically restarting the server process if it dies. See mariadbd_safe for more information.

mysqld_multi

mariadbd_multi is a wrapper that can be used to start the mariadbd server process if you plan to run multiple server processes on the same host. See mariadbd_multi for more information.

mysql.server

mysql.server is a wrapper that works as a standard sysVinit script. However, it can be used independently of sysVinit as a regular `sh` script. The script starts the mariadbd server process by first changing its current working directory to the MariaDB install directory and then starting mysqld_safe. The script requires the standard sysVinit arguments, such as `start`, `stop`, and `status`. See mysql.server for more information.

2.1.6.2 Configuring MariaDB with Option Files

Contents

You can configure MariaDB to run the way you want by configuring the server with MariaDB's option files. The default MariaDB option file is called `my.cnf` (or `mariadb.cnf`) on Unix-like operating systems and `my.ini` on Windows. Depending on how you've installed MariaDB, the default option file may be in a number of places, or it may not exist at all.

Global Options Related to Option Files

The following options relate to how MariaDB handles option files. These options can be used with most of MariaDB's command-line tools, not just mariadbd. They must be given as the first argument on the command-line:

Option	Description
`--print-defaults`	Read options from option files, print all option values, and then exit the program.
`--no-defaults`	Don't read options from any option file.
`--defaults-file =path`	Only read options from the given option file.

`--defaults-extra-file =path`	Read this extra option file after all other option files are read.
`--defaults-group-suffix =suffix`	In addition to the default option groups, also read option groups with the given suffix.

Default Option File Locations

MariaDB reads option files from many different directories by default. See the sections below to find out which directories are checked for which system.

For an exact list of option files read on your system by a specific program, you can execute:

```
$program --help --verbose
```

For example:

```
$ mariadbd --help --verbose
mariadbd  Ver 10.11.2-MariaDB for linux-systemd on x86_64 (MariaDB Server)
Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Starts the MariaDB database server.

Usage: mariadbd [OPTIONS]

Default options are read from the following files in the given order:
/etc/my.cnf ~/.my.cnf
The following groups are read: mysqld server mysqld-10.11 mariadb mariadb-10.11 mariadbd mariadbd
....
```

The option files are each scanned once, in the order given by `--help --verbose`. The effect of the configuration options are as if they would have been given as command line options in the order they are found.

Default Option File Locations on Linux, Unix, Mac

On Linux, Unix, or Mac OS X, the default option file is called `my.cnf`. MariaDB looks for the MariaDB option file in the locations and orders listed below.

The locations are dependent on whether the `DEFAULT_SYSCONFDIR cmake` option was defined when MariaDB was built. This option is usually defined as `/etc` when building RPM packages, but it is usually not defined when building DEB packages or binary tarballs.

- When the `DEFAULT_SYSCONFDIR cmake` option was **not** defined, MariaDB looks for the MariaDB option file in the following locations in the following order:

Location	Scope
`/etc/my.cnf`	Global
`/etc/mysql/my.cnf`	Global
`$MARIADB_HOME/my.cnf`	Server
`$MYSQL_HOME/my.cnf`	Server
defaults-extra-file	File specified with `--defaults-extra-file`, if any
`~/.my.cnf`	User

- When the `DEFAULT_SYSCONFDIR cmake` option was defined, MariaDB looks for the MariaDB option file in the following locations in the following order:

Location	Scope
`DEFAULT_SYSCONFDIR/my.cnf`	Global
`$MARIADB_HOME/my.cnf`	Server (from MariaDB 10.6)
`$MYSQL_HOME/my.cnf`	Server
defaults-extra-file	File specified with `--defaults-extra-file`, if any

`~/.my.cnf`	User

- `MARIADB_HOME` (from MariaDB 10.6) or `MYSQL_HOME` is the environment variable containing the path to the directory holding the server-specific `my.cnf` file. If `MYSQL_HOME` is not set, and the server is started with mysqld_safe, `MYSQL_HOME` is set as follows:
  - If there is a `my.cnf` file in the MariaDB data directory, but not in the MariaDB base directory, `MYSQL_HOME` is set to the MariaDB data directory.
  - Else, `MYSQL_HOME` is set to the MariaDB base directory.
- Note that if `MARIADB_HOME` is set (from MariaDB 10.6), `MYSQL_HOME` will not be used, even if set.

Default Option File Locations on Windows

On Windows, the option file can be called either `my.ini` or `my.cnf`. MariaDB looks for the MariaDB option file in the following locations in the following order:

Location	Scope
`System Windows Directory\my.ini`	Global
`System Windows Directory\my.cnf`	Global
`Windows Directory\my.ini`	Global
`Windows Directory\my.cnf`	Global
`C:\my.ini`	Global
`C:\my.cnf`	Global
`INSTALLDIR\my.ini`	Server
`INSTALLDIR\my.cnf`	Server
`INSTALLDIR\data\my.ini`	Server
`INSTALLDIR\data\my.cnf`	Server
`%MARIADB_HOME%\my.ini`	Server (from MariaDB 10.6)
`%MARIADB_HOME%\my.cnf`	Server (from MariaDB 10.6)
`%MYSQL_HOME%\my.ini`	Server
`%MYSQL_HOME%\my.cnf`	Server
defaults-extra-file	File specified with `--defaults-extra-file`, if any

- The `System Windows Directory` is the directory returned by the `GetSystemWindowsDirectory` function. The value is usually `C:\Windows`. To find its specific value on your system, open `cmd.exe` and execute:

```
echo %WINDIR%
```

- The `Windows Directory` is the directory returned by the `GetWindowsDirectory` function. The value may be a private `Windows Directory` for the application, or it may be the same as the `System Windows Directory` returned by the `GetSystemWindowsDirectory` function.
- `INSTALLDIR` is the parent directory of the directory where `mysqld.exe` is located. For example, if `mysqld.exe` is in `C:\Program Files\MariaDB 10.3\bin`, then `INSTALLDIR` would be `C:\Program Files\MariaDB 10.3`.
- `MARIADB_HOME` (from MariaDB 10.6) or `MYSQL_HOME` is the environment variable containing the path to the directory holding the server-specific `my.cnf` file.
- Note that if `MARIADB_HOME` is set (from MariaDB 10.6), `MYSQL_HOME` will not be used, even if set.

Default Option File Hierarchy

MariaDB will look in all of the above locations, in order, even if has already found an option file, and it's possible for more than one option file to exist. For example, you could have an option file in `/etc/my.cnf` with global settings for all servers, and then you could another option file in `~/.my.cnf` (i.e.your user account's home directory) which will specify additional settings (or override previously specified setting) that are specific only to that user.

Option files are usually optional. However, if the `--defaults-file` option is set, and if the file does not exist, then MariaDB will raise an error. If the `--defaults-file` option is set, then MariaDB will *only* read the option file referred to by this option.

If an option or system variable is not explicitly set, then it will be set to its default value. See Server System Variables for a

full list of all server system variables and their default values.

Custom Option File Locations

MariaDB can be configured to read options from custom options files with the following command-line arguments. These command-line arguments can be used with most of MariaDB's command-line tools, not just `mariadbd`. They must be given as the first argument on the command-line:

Option	Description
`--defaults-file=path`	Only read options from the given option file.
`--defaults-extra-file=path`	Read this extra option file after all other option files are read.

Option File Syntax

The syntax of the MariaDB option files are:

- Lines starting with # are comments.
- Empty lines are ignored.
- Option groups use the syntax `[group-name]`. See the Option Groups section below for more information on available option groups.
- The same option group can appear multiple times.
- The `!include` directive can be used to include other option files. See the Including Option Files section below for more information on this syntax.
- The `!includedir` directive can be used to include all `.cnf` files (and potentially `.ini` files) in a given directory. The option files within the directory are read in alphabetical order. See the Including Option File Directories section below for more information on this syntax.
- Dashes (`-`) and underscores (`_`) in options are interchangeable.
- Double quotes can be used to quote values
- `\n`, `\r`, `\t`, `\b`, `\s`, `\"`, `\'`, and `\\` are recognized as character escapes for new line, carriage return, tab, backspace, space, double quote, single quote, and backslash respectively.
- Certain option prefixes are supported. See the Option Prefixes section below for information about available option prefixes.
- See the Options section below for information about available options.

Option Groups

A MariaDB program can read options from one or many option groups. For an exact list of option groups read on your system by a specific program, you can execute:

```
$program --help --verbose
```

For example:

```
$ mariadbd --help --verbose
mariadbd  Ver 10.11.2-MariaDB for linux-systemd on x86_64 (MariaDB Server)
Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Starts the MariaDB database server.

Usage: mariadbd [OPTIONS]

Default options are read from the following files in the given order:
/etc/my.cnf ~/.my.cnf
The following groups are read: mysqld server mysqld-10.11 mariadb mariadb-10.11 mariadbd mariadbd
....
```

Server Option Groups

MariaDB programs reads server options from the following server option groups:

Group	Description

`[client-server]`	Options read by all MariaDB client programs and the MariaDB Server. This is useful for options like socket and port, which is common between the server and the clients.
`[server]`	Options read by MariaDB Server.
`[mysqld]`	Options read by `mysqld`, which includes both MariaDB Server and MySQL Server.
`[mysqld-X.Y]`	Options read by a specific version of `mysqld`, which includes both MariaDB Server and MySQL Server. For example, `[mysqld-10.4]`.
`[mariadb]`	Options read by MariaDB Server.
`[mariadb-X.Y]`	Options read by a specific version of MariaDB Server. For example, `[mariadb-10.4]`.
`[mariadbd]`	Options read by MariaDB Server. Available starting with MariaDB 10.4.6.
`[mariadbd-X.Y]`	Options read by a specific version of MariaDB Server. For example, `[mariadbd-10.4]`. Available starting with MariaDB 10.4.6.
`[galera]`	Options read by MariaDB Server, but only if it is compiled with Galera Cluster support. In MariaDB 10.1 and later, all builds on Linux are compiled with Galera Cluster support. When using one of these builds, options from this option group are read even if the Galera Cluster functionality is not enabled.

X.Y in the examples above refer to the base (major.minor) version of the server. For example, MariaDB 10.3.10 would read from `[mariadb-10.3]`. By using the `mariadb-X.Y` syntax, one can create option files that have MariaDB-only options in the MariaDB-specific option groups. That would allow the option file to work for both MariaDB and MySQL.

Client Option Groups

MariaDB programs reads client options from the following option groups:

Group	Description
`[client]`	Options read by all MariaDB and MySQL client programs, which includes both MariaDB and MySQL clients. For example, `mariadb-dump`.
`[client-server]`	Options read by all MariaDB client programs and the MariaDB Server. This is useful for options like socket and port, which is common between the server and the clients.
`[client-mariadb]`	Options read by all MariaDB client programs.

Tool-Specific Option Groups

Many MariaDB tools reads options from their own option groups as well. Many of them are listed below:

Group	Description
`[mysqld_safe]`	Options read by mysqld_safe, which includes both MariaDB Server and MySQL Server.
`[safe_mysqld]`	Options read by mysqld_safe, which includes both MariaDB Server and MySQL Server.
`[mariadb_safe]`	Options read by mysqld_safe from MariaDB Server.
`[mariadb-safe]`	Options read by mysqld_safe from MariaDB Server. Available starting with MariaDB 10.4.6.
`[mariabackup]`	Options read by Mariabackup. Available starting with MariaDB 10.1.31 and MariaDB 10.2.13.
`[xtrabackup]`	Options read by Mariabackup and Percona XtraBackup.
`[mysql_upgrade]`	Options read by mysql_upgrade, which includes both MariaDB Server and MySQL Server.
`[mariadb-upgrade]`	Options read by mariadb-upgrade. Available starting with MariaDB 10.4.6.
`[sst]`	Specific options read by the mariabackup SST method and the xtrabackup-v2 SST method.
`[mysql]`	Options read by mysql, which includes both MariaDB Server and MySQL Server.
`[mariadb-client]`	Options read by mariadb. Available starting with MariaDB 10.4.6.
`[mysqldump]`	Options read by mysqldump, which includes both MariaDB Server and MySQL Server.

`[mariadb-dump]`	Options read by mariadb-dump. Available starting with MariaDB 10.4.6.
`[mysqlimport]`	Options read by mysqlimport, which includes both MariaDB Server and MySQL Server.
`[mariadb-import]`	Options read by mariadb-import. Available starting with MariaDB 10.4.6.
`[mysqlbinlog]`	Options read by mysqlbinlog, which includes both MariaDB Server and MySQL Server.
`[mariadb-binlog]`	Options read by mariadb-binlog. Available starting with MariaDB 10.4.6.
`[mysqladmin]`	Options read by mysqladmin, which includes both MariaDB Server and MySQL Server.
`[mariadb-admin]`	Options read by mariadb-admin. Available starting with MariaDB 10.4.6.
`[mysqlshow]`	Options read by mysqlshow, which includes both MariaDB Server and MySQL Server.
`[mariadb-show]`	Options read by mariadb-show. Available starting with MariaDB 10.4.6.
`[mysqlcheck]`	Options read by mariadb-check, which includes both MariaDB Server and MySQL Server.
`[mariadb-check]`	Options read by mariadb-check. Available starting with MariaDB 10.4.6.
`[mysqlslap]`	Options read by mysqlslap, which includes both MariaDB Server and MySQL Server.
`[mariadb-slap]`	Options read by `mariadb-slap`. Available starting with MariaDB 10.4.6.
`[odbc]`	Options read by MariaDB Connector/ODBC, but only if the `USE_MYCNF` parameter has been set.

Custom Option Group Suffixes

MariaDB can be configured to read options from option groups with a custom suffix by providing the following command-line argument. This command-line argument can be used with most of MariaDB's command-line tools, not just `mariadbd`. It must be given as the first argument on the command-line:

Option	Description
`--defaults-group-suffix=suffix`	In addition to the default option groups, also read option groups with the given suffix.

The default group suffix can also be specified via the `MYSQL_GROUP_SUFFIX` environment variable.

Including Option Files

It is possible to include additional option files from another option file. For example, to include `/etc/mysql/dbserver1.cnf`, an option file could contain:

```
[mariadb]
...
!include /etc/mysql/dbserver1.cnf
```

Including Option File Directories

It is also possible to include all option files in a directory from another option file. For example, to include all option files in `/etc/my.cnf.d/`, an option file could contain:

```
[mariadb]
...
!includedir /etc/my.cnf.d/
```

The option files within the directory are read in alphabetical order.

All option file names must end in `.cnf` on Unix-like operating systems. On Windows, all option file names must end in `.cnf` or `.ini`.

Checking Program Options

You can check which options a given program is going to use by using the `--print-defaults` command-line argument:

Option	Description
`--print-defaults`	Read options from option files, print all option values, and then exit the program.

This command-line argument can be used with most of MariaDB's command-line tools, not just `mariadbd`. It must be given as the first argument on the command-line. For example:

```
$ mariadb-dump --print-defaults
mariadb-dump would have been started with the following arguments:
--ssl_cert=/etc/my.cnf.d/certificates/client-cert.pem --
ssl_key=/etc/my.cnf.d/certificates/client-key.pem --ssl_ca=/etc/my.cnf.d/certificates/ca.pem --
ssl-verify-server-cert --max_allowed_packet=1GB
```

You can also check which options a given program is going to use by using the `my_print_defaults` utility and providing the names of the option groups that the program reads. For example:

```
$ my_print_defaults mariadb-dump client client-server client-mariadb
--ssl_cert=/etc/my.cnf.d/certificates/client-cert.pem
--ssl_key=/etc/my.cnf.d/certificates/client-key.pem
--ssl_ca=/etc/my.cnf.d/certificates/ca.pem
--ssl-verify-server-cert
--max_allowed_packet=1GB
```

The `my_print_defaults` utility's `--mariadbd` command-line option provides a shortcut to refer to all of the server option groups:

```
$ my_print_defaults --mysqld
--log_bin=mariadb-bin
--log_slave_updates=ON
--ssl_cert=/etc/my.cnf.d/certificates/server-cert.pem
--ssl_key=/etc/my.cnf.d/certificates/server-key.pem
--ssl_ca=/etc/my.cnf.d/certificates/ca.pem
```

MySQL 5.6 Obfuscated Authentication Credential Option File

MySQL 5.6 and above support an obfuscated authentication credential option file called `.mylogin.cnf` that is created with `mysql_config_editor`.

MariaDB does not support this. The passwords in MySQL's `.mylogin.cnf` are only obfuscated, rather than encrypted, so the feature does not really add much from a security perspective. It is more likely to give users a false sense of security, rather than to seriously protect them.

Option Prefixes

MariaDB supports certain prefixes that can be used with options. The supported option prefixes are:

Option Prefix	Description
`autoset`	Sets the option value automatically. Only supported for certain options.
`disable`	For all boolean options, disables the setting (equivalent to setting it to `0`). Same as `skip`.
`enable`	For all boolean options, enables the setting (equivalent to setting it to `1`).
`loose`	Don't produce an error if the option doesn't exist.
`maximum`	Sets the maximum value for the option.
`skip`	For all boolean options, disables the setting (equivalent to setting it to `0`). Same as `disable`.

For example:

```
[mariadb]
...
# determine a good value for open_files_limit automatically
autoset_open_files_limit

# disable the unix socket plugin
disable_unix_socket

# enable the slow query log
enable_slow_query_log

# don't produce an error if these options don't exist
loose_file_key_management_filename = /etc/mysql/encryption/keyfile.enc
loose_file_key_management_filekey = FILE:/etc/mysql/encryption/keyfile.key
loose_file_key_management_encryption_algorithm = AES_CTR

# set max_allowed_packet to maximum value
maximum_max_allowed_packet

# disable external locking for MyISAM
skip_external_locking
```

Options

Dashes (`-`) and underscores (`_`) in options are interchangeable.

If an option is not explicitly set, then the server or client will simply use the default value for that option.

MariaDB Server Options

MariaDB Server options can be set in server option groups.

For a list of options that can be set for MariaDB Server, see the list of options available for `mariadbd`.

Most of the server system variables can also be set in MariaDB's option file.

MariaDB Client Options

MariaDB client options can be set in client option groups.

See the specific page for each client program to determine what options are available for that program.

Example Option Files

Most MariaDB installations include a sample MariaDB option file called `my-default.cnf`. On older releases, you would have also found the following option files:

- `my-small.cnf`
- `my-medium.cnf`
- `my-large.cnf`
- `my-huge.cnf`

However, these option files are now very dated for modern servers, so they were removed in MariaDB 10.3.1.

In source distributions, the sample option files are usually found in the `support-files` directory, and in other distributions, the option files are usually found in the `share/mysql` directory that is relative to the MariaDB base installation directory.

You can copy one of these sample MariaDB option files and use it as the basis for building your server's primary MariaDB option file.

Example Minimal Option File

The following is a minimal my.cnf file that you can use to test MariaDB.

```
[client-server]
# Uncomment these if you want to use a nonstandard connection to MariaDB
#socket=/tmp/mysql.sock
#port=3306

# This will be passed to all MariaDB clients
[client]
#password=my_password

# The MariaDB server
[mysqld]
# Directory where you want to put your data
data=/usr/local/mysql/var
# Directory for the errmsg.sys file in the language you want to use
language=/usr/local/share/mysql/english

# This is the prefix name to be used for all log, error and replication files
log-basename=mysqld

# Enable logging by default to help find problems
general-log
log-slow-queries
```

Example Hybrid Option File

The following is an extract of an option file that one can use if one wants to work with both MySQL and MariaDB.

```
# Example mysql config file.

[client-server]
socket=/tmp/mysql-dbug.sock
port=3307

# This will be passed to all mariadb clients
[client]
password=my_password

# Here are entries for some specific programs
# The following values assume you have at least 32M ram

# The MariaDB server
[mysqld]
temp-pool
key_buffer_size=16M
datadir=/my/mysqldata
loose-innodb_file_per_table

[mariadb]
datadir=/my/data
default-storage-engine=aria
loose-mutex-deadlock-detector
max-connections=20

[mariadb-5.5]
language=/my/maria-5.5/sql/share/english/
socket=/tmp/mysql-dbug.sock
port=3307

[mariadb-10.1]
language=/my/maria-10.1/sql/share/english/
socket=/tmp/mysql2-dbug.sock

[mysqldump]
quick
max_allowed_packet=16M

[mysql]
no-auto-rehash
loose-abort-source-on-error
```

2.1.6.3 mysqld Configuration Files and Groups

For all about configuring mysqld, see Configuring MariaDB with Option Files.

2.1.6.4 mariadbd Options

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadbd` is a symlink to `mysqld`.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mariadbd` is the name of the binary, with `mysqld` a symlink.

Contents

3. --auto-increment-offset
4. --binlog-alter-two-phase
5. --binlog-annotate-row-events
6. --binlog-cache-size
7. --binlog-checksum
8. --binlog-commit-wait-count
9. --binlog-commit-wait-usec
10. --binlog-direct-non-transactional-updates
11. --binlog-do-db
12. --binlog-expire-logs-seconds
13. --binlog-file-cache-size
14. --binlog-format
15. --binlog-ignore-db
16. --binlog-optimize-thread-scheduling
17. --binlog-row-event-max-size
18. --binlog-row-image
19. --binlog-row-metadata
20. --binlog-stmt-cache-size
21. --default-master-connection
22. --disconnect-slave-event-count
23. --flashback
24. --gtid-cleanup-batch-size
25. --gtid-domain-id
26. --gtid-ignore-duplicates
27. --gtid-strict-mode
28. --init-rpl-role
29. --init-slave
30. --log-basename
31. --log-bin
32. --log-bin-compress
33. --log-bin-compress-min-len
34. --log-bin-index
35. --log-bin-trust-function-creators
36. --log-bin-trust-routine-creators
37. --log-slave-updates
38. --master-host
39. --master-info-file
40. --master-password
41. --master-port
42. --master-retry-count
43. --master-ssl
44. --master-ssl-ca
45. --master-ssl-capath
46. --master-ssl-cert
47. --master-ssl-cipher
48. --master-ssl-key
49. --master-user
50. --master-verify-checksum
51. --max-binlog-cache-size
52. --max-binlog-dump-events
53. --max-binlog-size
54. --max-binlog-stmt-cache-size
55. --max-relay-log-size
56. --read-binlog-speed-limit
57. --relay-log
58. --relay-log-index
59. --relay-log-info-file
60. --relay-log-purge
61. --relay-log-recovery
62. --relay-log-space-limit
63. --replicate-annotate-row-events
64. --replicate-do-db
65. --replicate-do-table
66. --replicate-events-marked-for-skip
67. --replicate-ignore-db
68. --replicate-ignore-table
69. --replicate-rewrite_db

70. --replicate-same-server-id
71. --replicate-wild-do-table
72. --replicate-wild-ignore-table
73. --report-host
74. --report-password
75. --report-port
76. --report-user
77. --rpl-recovery-rank
78. --server-id
79. --slave-ddl-exec-mode
80. --slave-compressed-protocol
81. --slave-domain-parallel-threads
82. --slave-exec-mode
83. --slave-load-tmpdir
84. --slave-max-allowed-packet
85. --slave-max-statement-time
86. --slave-net-timeout
87. --slave-parallel-threads
88. --slave-parallel-max-queued
89. --slave-run-triggers-for-rbr
90. --slave-skip-errors
91. --slave-sql-verify-checksum
92. --slave-transaction-retries
93. --slave-transaction-retry-errors
94. --slave-transaction-retry-interval
95. --slave-type-conversions
96. --sporadic-binlog-dump-fail
97. --sync-binlog
98. --sync-master-info
99. --sync-relay-log
100. --sync-relay-log-info
101. --sysdate-is-now
102. Replication and Binary Logging Options and System Variables
103. Semisynchronous Replication Options and System Variables
    1. rpl-semi-sync-master-enabled
    2. rpl-semi-sync-master-timeout
    3. rpl-semi-sync-master-trace-level
    4. rpl-semi-sync-master-wait-no-slave
    5. rpl-semi-sync-master-wait-point
    6. rpl-semi-sync-slave-delay-master
    7. rpl-semi-sync-slave-kill-conn-timeout
    8. rpl-semi-sync-slave-enabled
    9. rpl-semi-sync-slave-trace-level
7. Optimizer Options
    1. --alter-algorithm
    2. --analyze-sample-percentage
    3. --big-tables
    4. --bulk-insert-buffer-size
    5. --expensive-subquery-limit
    6. --join-buffer-size
    7. --join-buffer-space-limit
    8. --join-cache-level
    9. --max-heap-table-size
    10. --max-join-size
    11. --max-seeks-for-key
    12. --max-sort-length
    13. --mrr-buffer-size
    14. --optimizer-extra-pruning-depth
    15. --optimizer-max-sel-arg-weight
    16. --optimizer-prune-level
    17. --optimizer-search-depth
    18. --optimizer-selectivity-sampling-limit
    19. --optimizer-switch
    20. -optimizer-trace
    21. -optimizer-trace-max-mem-size
    22. --optimizer-use-condition-selectivity
    23. --query-alloc-block-size
    24. --query-prealloc-size

28. --innodb-buffer-pool-dump-now
29. --innodb-buffer-pool-dump-pct
30. --innodb-buffer-pool-evict
31. --innodb-buffer-pool-filename
32. --innodb-buffer-pool-instances
33. --innodb-buffer-pool-load-abort
34. --innodb-buffer-pool-load-at-startup
35. --innodb-buffer-pool-load-now
36. --innodb-buffer-pool-load-pages-abort
37. --innodb-buffer-pool-populate
38. --innodb-buffer-pool-restore-at-startup
39. --innodb-buffer-pool-shm-checksum
40. --innodb-buffer-pool-shm-key
41. --innodb-buffer-pool-size
42. --innodb-change-buffer-max-size
43. --innodb-change-buffering
44. --innodb-change-buffering-debug
45. --innodb-checkpoint-age-target
46. --innodb-checksum-algorithm
47. --innodb-checksums
48. --innodb-cleaner-lsn-age-factor
49. --innodb-cmp
50. --innodb-cmp-per-index-enabled
51. --innodb-cmp-reset
52. --innodb-cmpmem
53. --innodb-cmpmem-reset
54. --innodb-commit-concurrency
55. --innodb-compression-algorithm
56. --innodb-compression-failure-threshold-pct
57. --innodb-compression-level
58. --innodb-compression-pad-pct-max
59. --innodb-concurrency-tickets
60. --innodb-corrupt-table-action
61. --innodb-data-file-buffering
62. --innodb-data-file-path
63. --innodb-data-file-write-through
64. --innodb-data-home-dir
65. --innodb-deadlock-detect
66. --innodb-deadlock-report
67. --innodb-default-encryption-key-id
68. --innodb-default-page-encryption-key
69. --innodb-default-row-format
70. --innodb-defragment
71. --innodb-defragment-fill-factor
72. --innodb-defragment-fill-factor-n-recs
73. --innodb-defragment-frequency
74. --innodb-defragment-n-pages
75. --innodb-defragment-stats-accuracy
76. --innodb-dict-size-limit
77. --innodb-disable-sort-file-cache
78. --innodb-doublewrite
79. --innodb-doublewrite-file
80. --innodb-empty-free-list-algorithm
81. --innodb-enable-unsafe-group-commit
82. --innodb-encrypt-log
83. --innodb-encrypt-tables
84. --innodb-encrypt-temporary-tables
85. --innodb-encryption-rotate-key-age
86. --innodb-encryption-rotation-iops
87. --innodb-encryption-threads
88. --innodb-extra-rsegments
89. --innodb-extra-undoslots
90. --innodb-fake-changes
91. --innodb-fast-checksum
92. --innodb-fast-shutdown
93. --innodb-fatal-semaphore-wait-threshold
94. --innodb-file-format

95. --innodb-file-format-check
96. --innodb-file-format-max
97. --innodb-file-io-threads
98. --innodb-file-per-table
99. --innodb-fillf-factor
100. --innodb-flush-log-at-trx-commi
101. --innodb-flush-method
102. --innodb-flush-neighbor-pages
103. --innodb-flush-neighbors
104. --innodb-flush-sync
105. --innodb-flushing-avg-loops
106. --innodb-force-load-corrupted
107. --innodb-force-primary-key
108. --innodb-force-recovery
109. --innodb-foreground-preflush
110. --innodb-ft-aux-table
111. --innodb-ft-cache-size
112. --innodb-ft-enable-diag-print
113. --innodb-ft-enable-stopword
114. --innodb-ft-max-token-size
115. --innodb-ft-min-token-size
116. --innodb-ft-num-word-optimize
117. --innodb-ft-result-cache-limit
118. --innodb-ft-server-stopword-table
119. --innodb-ft-sort-pll-degree
120. --innodb-ft-total-cache-size
121. --innodb-ft-user-stopword-table
122. --innodb-ibuf-accel-rate
123. --innodb-ibuf-active-contract
124. --innodb-ibuf-max-size
125. --innodb-idle-flush-pct
126. --innodb-immediate-scrub-data-uncompressed
127. --innodb-import-table-from-xtrabackup
128. --innodb-index-stats
129. --innodb-instant-alter-column-allowed
130. --innodb-instrument-semaphores
131. --innodb-io-capacity
132. --innodb-io-capacity-max
133. --innodb-large-prefix
134. --innodb-lazy-drop-table
135. --innodb-lock-schedule-algorithm
136. --innodb-lock-wait-timeout
137. --innodb-lock-waits
138. --innodb-locking-fake-changes
139. --innodb-locks
140. --innodb-locks-unsafe-for-binlog
141. --innodb-log-arch-dir
142. --innodb-log-arch-expire-sec
143. --innodb-log-archive
144. --innodb-log-block-size
145. --innodb-log-buffer-size
146. --innodb-log-checksum-algorithm
147. --innodb-log-checksums
148. --innodb-log-compressed-pages
149. --innodb-log-file-buffering
150. --innodb-log-file-size
151. --innodb-log-file-write-through
152. --innodb-log-files-in-group
153. --innodb-log-group-home-dir
154. --innodb-log-optimize-ddl
155. --innodb-log-write-ahead-size
156. --innodb-lru-flush-size
157. --innodb-lru-scan-depth
158. --innodb-max-bitmap-file-size
159. --innodb-max-changed-pages
160. --innodb-max-dirty-pages-pct
161. --innodb-max-dirty-pages-pct-lwm
162. --innodb-max-purge-lag

163. --innodb-max-purge-lag-delay
164. --innodb-max-purge-lag-wait
165. --innodb-max-undo-log-size
166. --innodb-merge-sort-block-size
167. --innodb-mirrored-log-groups
168. --innodb-monitor-disable
169. --innodb-monitor-enable
170. --innodb-monitor-reset
171. --innodb-monitor-reset-all
172. --innodb-mtflush-threads
173. --innodb-numa-interleave
174. --innodb-old-blocks-pct
175. --innodb-old-blocks-time
176. --innodb-online-alter-log-max-size
177. --innodb-open-files
178. --innodb-optimize-fulltext-only
179. --innodb-page-cleaners
180. --innodb-page-size
181. --innodb-pass-corrupt-table
182. --innodb-prefix-index-cluster-optimization
183. --innodb-print-all-deadlocks
184. --innodb-purge-batch-size
185. --innodb-purge-rseg-truncate-frequency
186. --innodb-purge-threads
187. --innodb-random-read-ahead
188. --innodb-read-ahead
189. --innodb-read-ahead-threshold
190. --innodb-read-io-threads
191. --innodb-read-only
192. --innodb-recovery-update-relay-log
193. --innodb-replication-delay
194. --innodb-rollback-on-timeout
195. --innodb-rollback-segments
196. --innodb-rseg
197. --innodb-sched-priority-cleaner
198. --innodb-safe-truncate
199. --innodb-scrub-log
200. --innodb-scrub-log-interval
201. --innodb-scrub-log-speed
202. --innodb-show-locks-held
203. --innodb-show-verbose-locks
204. --innodb-sort-buffer-size
205. --innodb-spin-wait-delay
206. --innodb-stats-auto-recalc
207. --innodb-stats-auto-update
208. --innodb-stats-include-delete-marked
209. --innodb-stats-method
210. --innodb-stats-modified-counter
211. --innodb-stats-on-metadata
212. --innodb-stats-persistent
213. --innodb-stats-persistent-sample-pages
214. --innodb-stats-sample-pages
215. --innodb-stats-traditional
216. --innodb-stats-transient-sample-pages
217. --innodb-stats-update-need-lock
218. --innodb-status-file
219. --innodb-status-output
220. --innodb-status-output-locks
221. --innodb-strict-mode
222. --innodb-support-xa
223. --innodb-sync-array-size
224. --innodb-sync-spin-loops
225. --innodb-sys-indexes
226. --innodb-sys-stats
227. --innodb-sys-tables
228. --innodb-table-locks
229. --innodb-table-stats

230. --innodb-temp-data-file-path
231. --innodb-thread-concurrency
232. --innodb-thread-concurrency-timer-based
233. --innodb-thread-sleep-delay
234. --innodb-tmpdir
235. --innodb-track-changed-pages
236. --innodb-track-redo-log-now
237. --innodb-trx
238. --innodb-truncate-temporary-tablespace-now
239. --innodb-undo-directory
240. --innodb-undo-log-truncate
241. --innodb-undo-logs
242. --innodb-undo-tablespaces
243. --innodb-use-atomic-writes
244. --innodb-use-fallocate
245. --innodb-use-global-flush-log-at-trx-commit
246. --innodb-use-mtflush
247. --innodb-use-native-aio
248. --innodb-use-purge-thread
249. --innodb-use-stacktrace
250. --innodb-use-sys-malloc
251. --innodb-use-sys-stats-table
252. --innodb-use-trim
253. --innodb-write-io-threads
254. --skip-innodb
255. --skip-innodb-checksums
256. --skip-innodb-doublewrite
257. InnoDB Storage Engine Options and System Variables

5. Aria Storage Engine Options
    1. --aria-block-size
    2. --aria-checkpoint-interval
    3. --aria-checkpoint-log-activity
    4. --aria-encrypt-tables
    5. --aria-force-start-after-recovery-failures
    6. --aria-group-commit
    7. --aria-group-commit-interval
    8. --aria-log-dir-path
    9. --aria-log-file-size
    10. --aria-log-purge-type
    11. --aria-max-sort-file-size
    12. --aria-page-checksum
    13. --aria-pagecache-age-threshold
    14. --aria-pagecache-buffer-size
    15. --aria-pagecache-division-limit
    16. --aria-pagecache-file-hash-size
    17. --aria-recover
    18. --aria-recover-options
    19. --aria-repair-threads
    20. --aria-sort-buffer-size
    21. --aria-stats-method
    22. --aria-sync-log-dir
    23. --aria-used-for-temp-tables
    24. --deadlock-search-depth-long
    25. --deadlock-search-depth-short
    26. --deadlock-timeout-long
    27. --deadlock-timeout-short
    28. Aria Storage Engine Options and System Variables
6. MyRocks Storage Engine Options
7. S3 Storage Engine Options
    1. --s3-access-key
    2. --s3-block-size
    3. --s3-bucket
    4. --s3-debug
    5. --s3-host-name
    6. --s3-pagecache-age-threshold
    7. --s3-pagecache-buffer-size
    8. --s3-pagecache-division-limit

38. --performance-schema-max-sql-text-length
39. --performance-schema-max-rwlock-classes
40. --performance-schema-max-rwlock-instances
41. --performance-schema-max-socket-classes
42. --performance-schema-max-socket-instances
43. --performance-schema-max-stage-classes
44. --performance-schema-max-statement-classes
45. --performance-schema-max-statement-stack
46. --performance-schema-max-table-handles
47. --performance-schema-max-table-instances
48. --performance-schema-max-table-lock-stat
49. --performance-schema-max-thread-classes
50. --performance-schema-max-thread-instances
51. --performance-schema-session-connect-attrs-size
52. --performance-schema-setup-actors-size
53. --performance-schema-setup-objects-size
54. --performance-schema-users-size
55. Performance Schema Options and System Variables

10. Galera Cluster Options
    1. --wsrep-allowlist
    2. --wsrep-auto-increment-control
    3. --wsrep-causal-reads
    4. --wsrep-certify-nonPK
    5. --wsrep-cluster-address
    6. --wsrep-cluster-name
    7. --wsrep-convert-LOCK-to-trx
    8. --wsrep-data-home-dir
    9. --wsrep-dbug-option
    10. --wsrep-debug
    11. --wsrep-desync
    12. --wsrep-dirty-reads
    13. --wsrep-drupal-282555-workaround
    14. --wsrep-forced-binlog-format
    15. --wsrep-gtid-domain-id
    16. --wsrep-gtid-mode
    17. --wsrep-ignore-apply-errors
    18. --wsrep-load-data-splitting
    19. --wsrep-log-conflicts
    20. --wsrep-max-ws-rows
    21. --wsrep-max-ws-size
    22. --wsrep-mode
    23. --wsrep-mysql-replication-bundle
    24. --wsrep-new-cluster
    25. --wsrep-node-address
    26. --wsrep-node-incoming-address
    27. --wsrep-node-name
    28. --wsrep-notify-cmd
    29. --wsrep-on
    30. --wsrep-OSU-method
    31. --wsrep-provider
    32. --wsrep-provider-options
    33. --wsrep-recover
    34. --wsrep-reject_queries
    35. --wsrep-replicate-myisam
    36. --wsrep-restart-slave
    37. --wsrep-retry-autocommit
    38. --wsrep-slave-FK-checks
    39. --wsrep-slave-threads
    40. --wsrep-slave-UK-checks
    41. --wsrep-sr-store
    42. --wsrep-sst-auth
    43. --wsrep-sst-donor
    44. --wsrep-sst-donor-rejects-queries
    45. --wsrep-sst-method
    46. --wsrep-sst-receive-address
    47. --wsrep-start-position
    48. --wsrep-status-file

49. --wsrep-strict-ddl
50. --wsrep-sync-wait
51. --wsrep-trx-fragment-size
52. --wsrep-trx-fragment-unit
53. Galera Cluster Options and System Variables

11. Options When Debugging mariadbd
    1. --core-file
    2. --debug
    3. --debug-assert-if-crashed-table
    4. --debug-binlog-fsync-sleep
    5. --debug-crc-break
    6. --debug-flush
    7. --debug-no-thread-alarm
    8. --debug-no-sync
    9. --debug-sync-timeout
    10. --gdb
    11. --silent-startup
    12. --sync-sys
    13. --thread-alarm
    14. Debugging Options and System Variables
12. Other Options
    1. --allow-suspicious-udfs
    2. --autocommit
    3. --automatic-sp-privileges
    4. --back-log
    5. --basedir
    6. --bind-address
    7. --block-encryption-mode
    8. --bootstrap
    9. --check-constraint-checks
    10. --chroot
    11. --column-compression-threshold
    12. --column-compression-zlib-level
    13. --column-compression-zlib-strategy
    14. --column-compression-zlib-wrap
    15. --completion-type
    16. --connect-timeout
    17. --datadir
    18. --date-format
    19. --datetime-format
    20. --deadlock-search-depth-long
    21. --deadlock-search-depth-short
    22. --deadlock-timeout-long
    23. --deadlock-timeout-short
    24. --default-password-lifetime
    25. --default-regex-flags
    26. --default-storage-engine
    27. --default-table-type
    28. --default-tmp-storage-engine
    29. --delay-key-write
    30. --des-key-file
    31. --disconnect-on-expired-password
    32. --div-precision-increment
    33. --encrypt-binlog
    34. --encrypt-tmp-disk-tables
    35. --encrypt-tmp-files
    36. --encryption-algorithm
    37. --engine-condition-pushdown
    38. --eq-range-index-dive-limit
    39. --event-scheduler
    40. --exit-info
    41. --expire-logs-days
    42. --explicit-defaults-for-timestamp
    43. --extra-max-connections
    44. --extra-port
    45. --flush
    46. --flush-time
    47. --ft-boolean-syntax

47. --ft-boolean-syntax
48. --ft-max-word-len
49. --ft-min-word-len
50. --ft-query-expansion-limit
51. --ft-stopword-file
52. --general-log
53. --general-log-file
54. --getopt-prefix-matching
55. --group-concat-max-len
56. --help
57. --histogram-size
58. --histogram-type
59. --host-cache-size
60. --idle-readonly-transaction-timeout
61. --idle-transaction-timeout
62. --idle-write-transaction-timeout
63. --ignore-db-dirs
64. --in-predicate-conversion-threshold
65. --init-connect
66. --init-file
67. --interactive-timeout
68. --large-pages
69. --local-infile
70. --lock-wait-timeout
71. --log
72. --log-disabled_statements
73. --log-error
74. --log-output
75. --log-queries-not-using-indexes
76. --log-ddl-recovery
77. --log-short-format
78. --log-slow-admin-statements
79. --log-slow-disabled-statements
80. --log-slow-file
81. --log-slow-filter
82. --log-slow-min-examined-row_limit
83. --log-slow-queries
84. --log-slow-query
85. --log-slow-query-file
86. --log-slow-query-time
87. --log-slow-rate-limit
88. --log-slow-slave-statements
89. --log-slow-time
90. --log-slow-verbosity
91. --log-tc
92. --log-tc-size
93. --log-warnings
94. --long-query-time
95. --low-priority-updates
96. --lower-case-table-names
97. --master-connect-retry
98. --max-allowed-packet
99. --max-connections
100. --max-connect-errors
101. --max-delayed-threads
102. --max-digest-length
103. --max-error-count
104. --max-length-for-sort-data
105. --max-long-data-size
106. --max-password-errors
107. --max-prepared-stmt-count
108. --max-recursive-iterations
109. --max-rowid-filter-size
110. --max-session-mem-used
111. --max-sp-recursion-depth
112. --max-statement-time
113. --max-tmp-tables
114. --max-user-connections

115. --max-write-lock-count
116. --memlock
117. --metadata-locks-cache-size
118. --metadata-locks-hash-instances
119. --min-examined-row-limit
120. --mrr-buffer-size
121. --multi-range-count
122. --mysql56-temporal-format
123. --ndb-use-copying-alter-table
124. --net-buffer-length
125. --net-read-timeout
126. --net-retry-count
127. --net-write-timeout
128. --one-thread
129. --open-files-limit
130. --pid-file
131. --plugin-load
132. --plugin-load-add
133. --plugin-dir
134. --plugin-maturity
135. --port
136. --port-open-timeout
137. --preload-buffer-size
138. --profiling-history-size
139. --progress-report-time
140. --proxy-protocol-networks
141. --query-cache-info
142. --query-cache-limit
143. --query-cache-min-res-unit
144. --query-cache-size
145. --query-cache-strip-comments
146. --query-cache-type
147. --query-cache-wlock-invalidate
148. --read-rnd-buffer-size
149. --read-only
150. --redirect-url
151. --require-secure-transport
152. --safe-show-database
153. --safe-user-create
154. --safemalloc-mem-limit
155. --secure-auth
156. --secure-file-priv
157. --secure-timestamp
158. --session-track-schema
159. --session-track-state-change
160. --session-track-system-variables
161. --session-track-transaction-info
162. --show-slave-auth-info
163. --skip-automatic-sp-privileges
164. --skip-external-locking
165. --skip-grant-tables
166. --skip-host-cache
167. --skip-large-pages
168. --skip-log-error
169. --skip-name-resolve
170. --skip-networking
171. --skip-partition
172. --skip-show-database
173. --skip-slave-start
174. --skip-ssl
175. --skip-symlink
176. --skip-thread-priority
177. --slow-launch-time
178. --slow-query-log
179. --slow-query-log-file
180. --socket
181. --sort-buffer-size

182. --sql-bin-update-same
183. --sql-if-exists
184. --sql-mode
185. --ssl
186. --ssl-ca
187. --ssl-capath
188. --ssl-cert
189. --ssl-cipher
190. --ssl-crl
191. --ssl-crlpath
192. --ssl-key
193. --stack-trace
194. --standard-compliant-cte
195. --stored-program-cache
196. --strict-password-validation
197. --symbolic-links
198. --sync-frm
199. --system-versioning-alter-history
200. --system-versioning-asof
201. --system-versioning-innodb-algorithm-simple
202. --system-versioning-insert-history
203. --table-lock-wait-timeout
204. --tc-heuristic-recover
205. --tcp-keepalive-interval
206. --tcp-keepalive-probes
207. --tcp-keepalive-time
208. --tcp-nodelay
209. --temp-pool
210. --test-expect-abort
211. --test-ignore-wrong-options
212. --thread-cache-size
213. --thread-concurrency
214. --thread-handling
215. --thread-pool-dedicated-listener
216. --thread-pool-exact-stats
217. --thread-pool-idle-timeout
218. --thread-pool-max-threads
219. --thread-pool-min-threads
220. --thread-pool-prio-kickup-timer
221. --thread-pool-priority
222. --thread-pool-size
223. --thread-pool-stall-limit
224. --thread-stack
225. --timed-mutexes
226. --time-format
227. --tls_version
228. --tmpdir
229. --transaction-isolation
230. --transaction-alloc-block-size
231. --transaction-prealloc-size
232. --transaction-read-only
233. --updatable-views-with-limit
234. --user
235. --userstat
236. --verbose
237. --version
238. --wait-timeout

13. Other Options and System Variables
14. Authentication Plugins - Options and System Variables
    1. Authentication Plugin - ed25519
        1. ed25519
    2. Authentication Plugin - gssapi
        1. gssapi
        2. gssapi_keytab_path
        3. gssapi_principal_name
        4. gssapi_mech_name
    3. Authentication Plugin - named_pipe
        1. named_pipe

This page lists all of the options for `mariadbd` (called mysqld before MariaDB 10.5), ordered by topic. For a full alphabetical list of all mariadbd options, as well as server and status variables, see Full list of MariaDB options, system and status variables.

In many cases, the entry here is a summary, and links to the full description.

By convention, server variables have usually been specified with an underscore in the configuration files, and a dash on the command line. You can however specify underscores as dashes - they are interchangeable.

See Configuring MariaDB with Option Files for which files and groups mariadbd reads for it's default options.

Option Prefixes

`--autoset-*`

- **Description:** Sets the option value automatically. Only supported for certain options.

`--disable-*`

- **Description:** For all boolean options, disables the setting (equivalent to setting it to `0`). Same as `--skip`.

`--enable-*`

- **Description:** For all boolean options, enables the setting (equivalent to setting it to `1`).

`--loose-*`

- **Description:** Don't produce an error if the option doesn't exist.

`--maximum-*`

- **Description:** Sets the maximum value for the option.

`--skip-*`

- **Description:** For all boolean options, disables the setting (equivalent to setting it to `0`). Same as `--disable`.

Option File Options

`--defaults-extra-file`

- **Commandline:** `--defaults-extra-file=name`
- **Description:** Read this extra option file after all other option files are read.
  - See Configuring MariaDB with Option Files.

`--defaults-file`

- **Commandline:** `--defaults-file=name`
- **Description:** Only read options from the given option file.
  - See Configuring MariaDB with Option Files.

`--defaults-group-suffix`

- **Commandline:** `--defaults-group-suffix=name`
- **Description:** In addition to the default option groups, also read option groups with the given suffix.
  - See Configuring MariaDB with Option Files.

`--no-defaults`

- **Commandline:** `--no-defaults`
- **Description:** Don't read options from any option file.
  - See Configuring MariaDB with Option Files.

`--print-defaults`

- **Commandline:** `--print-defaults`
- **Description:** Read options from option files, print all option values, and then exit the program.
  - See Configuring MariaDB with Option Files.

Compatibility Options

The following options have been added to MariaDB to make it more compliant with other MariaDB and MySQL versions.

Options that are also system variables are listed after:

`-a, --ansi`

- **Description:** Use ANSI SQL syntax instead of MariaDB syntax. This mode will also set transaction isolation level serializable.

`--new`

- **Description:** Use new functionality that will exist in next version of MariaDB. This function exists to make it easier to prepare for an upgrade. For version 5.1 this functions enables the LIST and RANGE partitions functions for ndbcluster.

`--old-style-user-limits`

- **Description:** Enable old-style user limits (before MySQL 5.0.3, user resources were counted per each user+host vs. per account).

`--safe-mode`

- **Description:** Disable some potential unsafe optimizations. For 5.2, INSERT DELAYED is disabled, myisam_recover_options is set to DEFAULT (automatically recover crashed MyISAM files) and the query cache is disabled. For Aria tables, disable bulk insert optimization to enable one to use aria_read_log to recover tables even if tables are deleted (good for testing recovery).

`--skip-new`

- **Description:** Disables --new in 5.2. In 5.1 used to disable some new potentially unsafe functions.

Compatibility Options and System Variables

- --old
- --old-alter-table
- --old-mode
- --old-passwords
- --show-old-temporals

Locale Options

Options that are also system variables are listed after:

`--character-set-client-handshake`

- **Commandline:** `--character-set-client-handshake`
- **Description:** Don't ignore client side character set value sent during handshake.

`--default-character-set`

- **Commandline:** `--default-character-set=name`
- **Description:** Still available as an option for setting the default character set for clients and their connections, it was deprecated and removed in MariaDB 10.2 as a server option. Use character-set-server instead.

`--language`

- **Description:** This option can be used to set the server's language for error messages. This option can be specified either as a language name or as the path to the directory storing the language's error message file. See Server Locales for a list of supported locales and their associated languages.

- This option is deprecated. Use the `lc_messages` and `lc_messages_dir` system variables instead.
- See Setting the Language for Error Messages for more information.

Locale Options and System Variables

- character-set-filesystem
- character-set-client
- character-set-connection
- character-set-database
- character-set-filesystem
- character-set-results
- character-set-server
- character-set-system
- character-sets-dir
- collation-connection
- collation-database
- collation-server
- default-week-format
- default-time-zone
- lc-messages
- lc-messages-dir
- lc-time-names

Windows Options

Options that are also system variables are listed after:

`--console`

- **Description:** Windows-only option that keeps the console window open and for writing log messages to stderr and stdout. If specified together with --log-error, the last option will take precedence.

`--install`

- **Description:** Windows-only option that installs the `mariadbd` process as a Windows service.
  - The Windows service created with this option auto-starts. If you want a service that is started on demand, then use the `--install-manual` option.
  - This option takes a service name as an argument. If this option is provided without a service name, then the service name defaults to "MARIADB".
  - This option is deprecated and may be removed in a future version. See MDEV-19358 for more information.

`--install-manual`

- **Description:** Windows-only option that installs the `mariadbd` process as a Windows service.
  - The Windows service created with this option is started on demand. If you want a service that auto-starts, use the `--install` option.
  - This option takes a service name as an argument. If this option is provided without a service name, then the service name defaults to "MARIADB".
  - This option is deprecated and may be removed in a future version. See MDEV-19358 for more information.

`--remove`

- **Description:** Windows-only option that removes the Windows service created by the `--install` or `--install-manual` options.
  - This option takes a service name as an argument. If this option is provided without a service name, then the service name defaults to "MARIADB".
  - This option is deprecated and may be removed in a future version. See MDEV-19358 for more information.

`--slow-start-timeout`

- **Description:** Windows-only option that defines the maximum number of milliseconds that the service control manager should wait before trying to kill the Windows service during startup. Defaults to `15000`.

`--standalone`

- **Description:** Windows-only option that has no effect. Kept for compatibility reasons.

Windows Options and System Variables

The following options and system variables are related to using MariaDB on Windows:

- `--named-pipe`

Replication and Binary Logging Options

The following options are related to replication and the binary log. Options that are also system variables are listed after:

`--abort-slave-event-count`

- **Commandline:** `--abort-slave-event-count=#`
- **Description:** Option used by mysql-test for debugging and testing of replication.

`--binlog-do-db`

- **Commandline:** `--binlog-do-db=name`
- **Description:** This option allows you to configure a replication master to write statements and transactions affecting databases that match a specified name into its binary log. Since the filtered statements or transactions will not be present in the binary log, its replicas will not be able to replicate them.
  - This option will **not** work with cross-database updates with statement-based logging. See the Statement-Based Logging section for more information.
  - This option can **not** be set dynamically. Available as a system variable from MariaDB 11.2.0.
  - When setting it on the command-line or in a server option group in an option file, the option does not accept a comma-separated list. If you would like to specify multiple filters, then you need to specify the option multiple times.
  - See Replication Filters for more information.

`--binlog-ignore-db`

- **Commandline:** `--binlog-ignore-db=name`
- **Description:** This option allows you to configure a replication master to **not** write statements and transactions affecting databases that match a specified name into its binary log. Since the filtered statements or transactions will not be present in the binary log, its replicas will not be able to replicate them.
  - This option will **not** work with cross-database updates with statement-based logging. See the Statement-Based Logging section for more information.
  - This option can **not** be set dynamically. Available as a system variable from MariaDB 11.2.0.
  - When setting it on the command-line or in a server option group in an option file, the option does not accept a comma-separated list. If you would like to specify multiple filters, then you need to specify the option multiple times.
  - See Replication Filters for more information.

`--binlog-row-event-max-size`

- **Commandline:** `--binlog-row-event-max-size=#`
- **Description:** The maximum size of a row-based binary log event in bytes. Rows will be grouped into events smaller than this size if possible. The value has to be a multiple of 256. Available as a system variable from MariaDB 11.2.0.
- **Default value** `8192`

`--disconnect-slave-event-count`

- **Commandline:** `--disconnect-slave-event-count=#`
- **Description:** Option used by mysql-test for debugging and testing of replication.

`--flashback`

- **Commandline:** `--flashback`
- **Description:** Setup the server to use flashback. This enables the binary log and sets `binlog_format=ROW`.
- **Introduced:** MariaDB 10.2.4

`--init-rpl-role`

- **Commandline:** `--init-rpl-role=name`
- **Description:** Set the replication role.

`--log-basename`

- **Commandline:** `--log-basename=name`
- **Description:** Basename for all log files and the .pid file. This sets all log file names at once (in 'datadir') and is normally the only option you need for specifying log files. This is especially recommended to be set if you are using replication as it ensures that your log file names are not dependent on your host name. Sets names for log-bin, log-bin-index, relay-log, relay-log-index, general-log-file, `--log-slow-query-log-file`, `--log-error-file`, and pid-file.
- **Introduced:** MariaDB 5.2

`--log-bin-trust-routine-creators`

- **Commandline:** `--log-bin-trust-routine-creators`
- **Description:** Deprecated, use log-bin-trust-function-creators.

`--master-host`

- **Commandline:** `--master-host=name`
- **Description:** Primary hostname or IP address for replication. If not set, the replica thread will not be started. Note that the setting of master-host will be ignored if there exists a valid master.info file.

`--master-info-file`

- **Commandline:** `--master-info-file=name`
- **Description:** Name and location of the file on the replica where the `MASTER_LOG_FILE` and `MASTER_LOG_POS` options (i.e. the binary log position on the primary) and most other CHANGE MASTER options are written. The replica's I/O thread keeps this binary log position updated as it downloads events.
  - See CHANGE MASTER TO: Option Persistence for more information.

`--master-password`

- **Commandline:** `--master-password=name`
- **Description:** The password the replica thread will authenticate with when connecting to the primary. If not set, an empty password is assumed. The value in master.info will take precedence if it can be read.

`--master-port`

- **Commandline:** `--master-port=#`
- **Description:** The port the master is listening on. If not set, the compiled setting of MYSQL_PORT is assumed. If you have not tinkered with configure options, this should be 3306. The value in master.info will take precedence if it can be read.

--master-retry-count

- **Commandline:** `--master-retry-count=#`
- **Description:** Number of times a replica will attempt to connect to a primary before giving up. The retry interval is determined by the MASTER_CONNECT_RETRY option for the CHANGE MASTER statement. A value of 0 means the replica will not stop attempting to reconnect. Reconnects are triggered when a replica has timed out. See slave_net_timeout.
- **Default Value:** `86400`
- **Range - 32 bit:** `0 to 4294967295`
- **Range - 64 bit:** `0 to 18446744073709551615`

--master-ssl

- **Commandline:** `--master-ssl`
- **Description:** Enable the replica to connect to the master using TLS.

--master-ssl-ca

- **Commandline:** `--master-ssl-ca[=name]`
- **Description:** Master TLS CA file. Only applies if you have enabled master-ssl.

--master-ssl-capath

- **Commandline:** `--master-ssl-capath[=name]`
- **Description:** Master TLS CA path. Only applies if you have enabled master-ssl.

--master-ssl-cert

- **Commandline:** `--master-ssl-cert[=name]`
- **Description:** Master TLS certificate file name. Only applies if you have enabled master-ssl.

--master-ssl-cipher

- **Commandline:** `--master-ssl-cipher[=name]`
- **Description:** Master TLS cipher. Only applies if you have enabled master-ssl.

--master-ssl-key

- **Commandline:** `--master-ssl-key[=name]`
- **Description:** Master TLS keyfile name. Only applies if you have enabled master-ssl.

--master-user

- **Commandline:** `--master-user=name`
- **Description:** The username the replica thread will use for authentication when connecting to the primary. The user must have FILE privilege. If the primary user is not set, user test is assumed. The value in master.info will take precedence if it can be read.

--max-binlog-dump-events

- **Commandline:** `--max-binlog-dump-events=#`
- **Description:** Option used by mysql-test for debugging and testing of replication.

--replicate-same-server-id

- **Commandline:** `--replicate-same-server-id`

- **Description:** In replication, if set to 1, do not skip events having our server id. Default value is 0 (to break infinite loops in circular replication). Can't be set to 1 if log-slave-updates is used.

`--sporadic-binlog-dump-fail`

- **Commandline:** `--sporadic-binlog-dump-fail`
- **Description:** Option used by mysql-test for debugging and testing of replication.

`--sysdate-is-now`

- **Commandline:** `--sysdate-is-now`
- **Description:** Non-default option to alias SYSDATE() to NOW() to make it safe for replication. Since 5.0, SYSDATE() has returned a `dynamic' value different for different invocations, even within the same statement.

Replication and Binary Logging Options and System Variables

The following options and system variables are related to replication and the binary log:

- auto-increment-increment
- auto-increment-offset
- binlog-alter-two-phase
- binlog-annotate-row-events
- binlog-cache-size
- binlog-checksum
- binlog-commit-wait-count
- binlog-commit-wait-usec
- binlog-direct-non-transactional-updates|
- binlog-expire-logs-seconds
- binlog-file-cache-size
- binlog-format
- binlog-optimize-thread-scheduling
- binlog-row-image
- binlog-row-metadata
- binlog-stmt-cache-size
- default-master-connection
- gtid-cleanup-batch-size
- gtid-domain-id
- gtid-ignore-duplicates
- gtid-strict-mode
- init-slave
- log-bin
- log-bin-compress
- log-bin-compress-min-len
- log-bin-index
- log-bin-trust-function-creators
- log-slave-updates
- master-verify-checksum
- max-binlog-cache-size
- max-binlog-size
- max-binlog-stmt-cache-size
- max-relay-log-size
- read-binlog-speed-limit
- relay-log
- relay-log-index
- relay-log-info-file
- relay-log-purge
- relay-log-recovery
- relay-log-space-limit
- replicate-annotate-row-events
- replicate-do-db
- replicate-do-table
- replicate-events-marked-for-skip
- replicate-ignore-db
- replicate-ignore-table

- replicate-rewrite-db
- replicate-wild-do-table
- replicate-wild-ignore-table
- report-host
- report-password
- report-port
- report-user
- rpl-recovery-rank
- server-id
- slave-compressed-protocol
- slave-ddl-exec-mode
- slave-domain-parallel-threads
- slave-exec-mode
- slave-load-tmpdir
- slave-max-allowed-packet
- slave-max-statement-time
- slave-net-timeout
- slave-parallel-max-queued
- slave-parallel-threads
- slave-run-triggers-for-rbr
- slave-skip-errors
- slave-sql-verify-checksum
- slave-transaction-retries
- slave_transaction_retry_errors
- slave_transaction_retry_interval
- slave-type-conversions
- sync-binlog
- sync-master-info
- sync-relay-log
- sync-relay-log-info

Semisynchronous Replication Options and System Variables

The options and system variables related to Semisynchronous Replication are described here.

Optimizer Options

Options that are also system variables are listed after:

`--record-buffer`

- **Commandline:** `--record-buffer=#`
- **Description:** Old alias for read_buffer_size.
- **Removed:** MariaDB 5.5

`--table-cache`

- **Commandline:** `--table-open-cache=#`
- **Description:** Removed; use --table-open-cache instead.
- **Removed:** MariaDB 5.1.3

Optimizer Options and System Variables

- alter-algorithm
- analyze-sample-percentage
- big-tables
- bulk-insert-buffer-size
- expensive-subquery-limit
- join-buffer-size
- join-buffer-space-limit
- join-cache-level
- max-heap-table-size
- max-join-size

- max-seeks-for-key
- max-sort-length
- mrr-buffer-size
- optimizer-extra-pruning-depth
- optimizer-max-sel-arg-weight
- optimizer-prune-level
- optimizer-search-depth
- optimizer-selectivity-sampling-limit
- optimizer-switch
- optimizer-trace
- optimizer-trace-max-mem-size
- optimizer-use-condition-selectivity
- query-alloc-block-size
- query-prealloc-size
- range-alloc-block-size
- read-buffer-size
- rowid-merge-buff-size
- table-definition-cache
- table-open-cache
- table-open-cache-instances
- tmp-disk-table-size
- tmp-memory-table-size
- tmp-table-size
- use-stat-tables

Storage Engine Options

`--skip-bdb`

- **Commandline:** `----skip-bdb`
- **Description:** Deprecated option; Exists only for compatibility with very old my.cnf files.
- **Removed:** MariaDB 10.5.1

`--external-locking`

- **Commandline:** `--external-locking`
- **Description:** Use system (external) locking (disabled by default). With this option enabled you can run myisamchk to test (not repair) tables while the server is running. Disable with --skip-external-locking. From MariaDB 10.2.40, MariaDB 10.3.31, MariaDB 10.4.21, MariaDB 10.5.12, MariaDB 10.6.4 and all later version, this effects InnoDB and can be used to prevent multiple instances running on the same data.

MyISAM Storage Engine Options

The options related to the MyISAM storage engine are described below. Options that are also system variables are listed after:

`--log-isam`

- **Commandline:** `--log-isam[=file_name]`
- **Description:** Enable the MyISAM log, which logs all MyISAM changes to file. If no filename is provided, the default, *myisam.log* is used.

MyISAM Storage Engine Options and System Variables

Some options and system variables related to the MyISAM storage engine can be found here. Direct links to many of them can be found below.

- concurrent-insert
- delayed-insert-limit
- delayed-insert-timeout
- delayed-queue-size
- keep-files-on-create

- key-buffer-size
- key-cache-age-threshold
- key-cache-block-size
- key-cache-division-limit
- key-cache-file-hash-size
- key-cache-segments
- myisam-block-size
- myisam-data-pointer-size
- myisam-max-sort-file-size
- myisam-mmap-size
- myisam-recover-options
- myisam-repair-threads
- myisam-sort-buffer-size
- myisam-stats-method
- myisam-use-mmap

InnoDB Storage Engine Options

The options related to the InnoDB storage engine are described below. Options that are also system variables are listed after:

`--innodb`

- **Commandline:** `--innodb=value`, `--skip-innodb`
- **Description:** This variable controls whether or not to load the InnoDB storage engine. Possible values are `ON`, `OFF`, `FORCE` or `FORCE_PLUS_PERMANENT` (from MariaDB 5.5). If set to `OFF` (the same as --skip-innodb), since InnoDB is the default storage engine, the server will not start unless another storage engine has been chosen with --default-storage-engine. `FORCE` means that the storage engine must be successfully loaded, or else the server won't start. `FORCE_PLUS_PERMANENT` enables the plugin, but if plugin cannot initialize, the server will not start. In addition, the plugin cannot be uninstalled while the server is running.

`--innodb-cmp`

- **Commandline:** `--innodb-cmp`
- **Description:**
- **Default:** `ON`

`--innodb-cmp-reset`

- **Commandline:** `--innodb-cmp-reset`
- **Description:**
- **Default:** `ON`

`--innodb-cmpmem`

- **Commandline:** `--innodb-cmpmem`
- **Description:**
- **Default:** `ON`

`--innodb-cmpmem-reset`

- **Commandline:** `--innodb-cmpmem-reset`
- **Description:**
- **Default:** `ON`

`--innodb-file-io-threads`

- **Commandline:** `--innodb-file-io-threads`
- **Description:**
- **Default:** `4`

- **Removed:** MariaDB 10.3.0

`--innodb-index-stats`

- **Commandline:** `--innodb-index-stats`
- **Description:**
- **Default:** `ON`
- **Removed:** MariaDB 10.0.0

`--innodb-lock-waits`

- **Commandline:** `--innodb-lock-waits`
- **Description:**
- **Default:** `ON`

`--innodb-locks`

- **Commandline:** `--innodb-locks`
- **Description:**
- **Default:** `ON`

`--innodb-rseg`

- **Commandline:** `--innodb-rseg`
- **Description:**
- **Default:** `ON`
- **Removed:** MariaDB 10.0.0

`--innodb-status-file`

- **Commandline:** `--innodb-status-file`
- **Description:**
- **Default:** `FALSE`

`--innodb-sys-indexes`

- **Commandline:** `--innodb-sys-indexes`
- **Description:**
- **Default:** `ON`

`--innodb-sys-stats`

- **Commandline:** `--innodb-sys-stats`
- **Description:**
- **Default:** `ON`
- **Removed:** MariaDB 10.0.0

`--innodb-sys-tables`

- **Commandline:** `--innodb-sys-tables`
- **Description:**
- **Default:** `ON`

`--innodb-table-stats`

- **Commandline:** `--innodb-table-stats`
- **Description:**
- **Default:** `ON`
- **Removed:** MariaDB 10.0.0

`--innodb-trx`

- **Commandline:** `--innodb-trx`
- **Description:**
- **Default:** `ON`

InnoDB Storage Engine Options and System Variables

Some options and system variables related to the InnoDB storage engine can be found here. Direct links to many of them can be found below.

- ignore-builtin-innodb
- innodb-adaptive-checkpoint
- innodb-adaptive-flushing
- innodb-adaptive-flushing-lwm
- innodb-adaptive-flushing-method
- innodb-adaptive-hash-index
- innodb-adaptive-hash-index-partitions
- innodb-adaptive-hash-index-parts
- innodb-adaptive-max-sleep-delay
- innodb-additional-mem-pool-size
- innodb-api-bk-commit-interval
- innodb-api-disable-rowlock
- innodb-api-enable-binlog
- innodb-api-enable-mdl
- innodb-api-trx-level
- innodb-auto-lru-dump
- innodb-autoextend-increment
- innodb-autoinc-lock-mode
- innodb-background-scrub-data-check-interval
- innodb-background-scrub-data-compressed
- innodb-background-scrub-data-interval
- innodb-background-scrub-data-uncompressed
- innodb-blocking-buffer-pool-restore
- innodb-buf-dump-status-frequency
- innodb-buffer-pool-chunk-size
- innodb-buffer-pool-dump-at-shutdown
- innodb-buffer-pool-dump-now
- innodb-buffer-pool-dump-pct
- innodb-buffer-pool-evict
- innodb-buffer-pool-filename
- innodb-buffer-pool-instances
- innodb-buffer-pool-load-abort
- innodb-buffer-pool-load-at-startup
- innodb-buffer-pool-load-now
- innodb-buffer-pool-load-pages-abort
- innodb-buffer-pool-populate
- innodb-buffer-pool-restore-at-startup
- innodb-buffer-pool-shm-checksum
- innodb-buffer-pool-shm-key
- innodb-buffer-pool-size
- innodb-change-buffer-max-size
- innodb-change-buffering
- innodb-change-buffering-debug
- innodb-checkpoint-age-target
- innodb-checksum-algorithm
- innodb-checksums
- innodb-cleaner-lsn-age-factor
- innodb-cmp-per-index-enabled
- innodb-commit-concurrency

- innodb-compression-algorithm
- innodb-compression-failure-threshold-pct
- innodb-compression-level
- innodb-compression-pad-pct-max
- innodb-concurrency-tickets
- innodb-corrupt-table-action
- innodb-data-file-buffering
- innodb-data-file-path
- innodb-data-file-write-through
- innodb-data-home-dir
- innodb-deadlock-detect
- innodb-deadlock-report
- innodb-default-encryption-key-id
- innodb-default-page-encryption-key
- innodb-default-row-format
- innodb-defragment
- innodb-defragment-fill-factor
- innodb-defragment-fill-factor-n-recs
- innodb-defragment-frequency
- innodb-defragment-n-pages
- innodb-defragment-stats-accuracy
- innodb-dict-size-limit
- innodb_disable_sort_file_cache
- innodb-doublewrite
- innodb-doublewrite-file
- innodb-empty-free-list-algorithm
- innodb-enable-unsafe-group-commit
- innodb-encrypt-log
- innodb-encrypt-tables
- innodb-encrypt-temporary-tables
- innodb-encryption-rotate-key-age
- innodb-encryption-rotation_iops
- innodb-encryption-threads
- innodb-extra-rsegments
- innodb-extra-undoslots
- innodb-fake-changes
- innodb-fast-checksum
- innodb-fast-shutdown
- innodb-fatal-semaphore-wait-threshold
- innodb-file-format
- innodb-file-format-check
- innodb-file-format-max
- innodb-file-per-table
- innodb-fill-factor
- innodb-flush-log-at-trx-commit
- innodb-flush-method
- innodb-flush-neighbor-pages
- innodb-flush-neighbors
- innodb-flush-sync
- innodb-flushing-avg-loops
- innodb-force-load-corrupted
- innodb-force-primary-key
- innodb-force-recovery
- innodb-foreground-preflush
- innodb-ft-aux-table
- innodb-ft-cache-size
- innodb-ft-enable-diag-print
- innodb-ft-enable-stopword
- innodb-ft-max-token-size
- innodb-ft-min-token-size
- innodb-ft-num-word-optimize
- innodb-ft-result-cache-limit
- innodb-ft-server-stopword-table
- innodb-ft-sort-pll-degree
- innodb-ft-total-cache-size
- innodb-ft-user-stopword-table
- innodb-ibuf-accel-rate

- innodb-ibuf-active-contract
- innodb-ibuf-max-size
- innodb-idle-flush-pct
- innodb-immediate-scrub-data-uncompressed
- innodb-import-table-from-xtrabackup
- innodb-instant-alter-column-allowed
- innodb-instrument-semaphores
- innodb-io-capacity
- innodb-io-capacity-max
- innodb-large-prefix
- innodb-lazy-drop-table
- innodb-lock-schedule-algorithm
- innodb-locking-fake-changes
- innodb-locks-unsafe-for-binlog
- innodb-log-arch-dir
- innodb-log-arch-expire-sec
- innodb-log-archive
- innodb-log-block-size
- innodb-log-buffer-size
- innodb-log-checksum-algorithm
- innodb-log-checksums
- innodb-log-compressed-pages
- innodb-log-file-buffering
- innodb-log-file-size
- innodb-log-file-write-through
- innodb-log-files-in-group
- innodb-log-group-home-dir
- innodb-log-optimize-ddl
- innodb-log-write-ahead-size
- innodb-lru-flush-size
- innodb-lru-scan-depth
- innodb-max-bitmap-file-size
- innodb-max-changed-pages
- innodb-max-dirty-pages-pct
- innodb-max-dirty-pages-pct-lwm
- innodb-max-purge-lag
- innodb-max-purge-lag-delay
- innodb-max-purge-lag-wait
- innodb-max-undo-log-size
- innodb-merge-sort-block-size
- innodb-mirrored-log-groups
- innodb-monitor-disable
- innodb-monitor-enable
- innodb-monitor-reset
- innodb-monitor-reset-all
- innodb-mtflush-threads
- innodb-numa-interleave
- innodb-old-blocks-pct
- innodb-old-blocks-time
- innodb-online-alter-log-max-size
- innodb-open-files
- innodb-optimize-fulltext-only
- innodb-page-cleaners
- innodb-page-size
- innodb-pass-corrupt-table
- innodb-prefix-index-cluster-optimization
- innodb-print-all-deadlocks
- innodb-purge-batch-size
- innodb-purge-rseg-truncate-frequency
- innodb-purge-threads
- innodb-random-read-ahead
- innodb-read-ahead
- innodb-read-ahead-threshold
- innodb-read-io-threads
- innodb-read-only
- innodb-recovery-update-relay-log
- innodb-replication-delay

- innodb-rollback-on-timeout
- innodb-rollback-segments
- innodb-safe-truncate
- innodb-sched-priority-cleaner
- innodb-scrub-log
- innodb-scrub-log-interval
- innodb-scrub-log-speed
- innodb-show-locks-held
- innodb-show-verbose-locks
- innodb-sort-buffer-size
- innodb-spin-wait-delay
- innodb-stats-auto-recalc
- innodb-stats-auto-update
- innodb-stats-include-delete-marked
- innodb-stats-method
- innodb-stats-modified-counter
- innodb-stats-on-metadata
- innodb-stats-persistent
- innodb-stats-persistent-sample-pages
- innodb-stats-sample-pages
- innodb-stats-transient-sample-pages
- innodb-stats-traditional
- innodb-stats-update-need-lock
- innodb-status-output
- innodb-status-output-locks
- innodb-strict-mode
- innodb-support-xa
- innodb-sync-array-size
- innodb-sync-spin-loops
- innodb-table-locks
- innodb-temp-data-file-path
- innodb-thread-concurrency
- innodb-thread-concurrency-timer-based
- innodb-thread-sleep-delay
- innodb-tmpdir
- innodb-track-changed-pages
- innodb-track-redo-log-now
- innodb-truncate-temporary-tablespace-now
- innodb-undo-directory
- innodb-undo-log-truncate
- innodb-undo-logs
- innodb-undo-tablespaces
- innodb-use-atomic-writes
- innodb-use-fallocate
- innodb-use-global-flush-log-at-trx-commit
- innodb-use-mtflush
- innodb-use-native_aio
- innodb-use-purge-thread
- innodb-use-stacktrace
- innodb-use-sys-malloc
- innodb-use-sys-stats-table
- innodb-use-trim
- innodb-write-io-threads
- skip-innodb
- skip-innodb-checksums
- skip-innodb-doublewrite

Aria Storage Engine Options

The options related to the Aria storage engine are described below. Options that are also system variables are listed after:

`--aria-log-dir-path`

- **Commandline:** `--aria-log-dir-path=value`
- **Description:** Path to the directory where transactional log should be stored
- **Default:** `SAME AS DATADIR`

Aria Storage Engine Options and System Variables

Some options and system variables related to the Aria storage engine can be found here. Direct links to many of them can be found below.

- aria-block-size
- aria-checkpoint-interval
- aria-checkpoint-log-activity
- aria-encrypt-tables
- aria-force-start-after-recovery-failures
- aria-group-commit
- aria-group-commit-interval
- aria-log-file-size
- aria-log-purge-type
- aria-max-sort-file-size
- aria-page-checksum
- aria-pagecache-age-threshold
- aria-pagecache-buffer-size
- aria-pagecache-division-limit
- aria-pagecache-file-hash-size
- aria-recover
- aria-recover-options
- aria-repair-threads
- aria-sort-buffer-size
- aria-stats-method
- aria-sync-log-dir
- aria-used-for-temp-tables
- deadlock-search-depth-long
- deadlock-search-depth-short
- deadlock-timeout-long
- deadlock-timeout-short

MyRocks Storage Engine Options

The options and system variables related to the MyRocks storage engine can be found here.

S3 Storage Engine Options

The options and system variables related to the S3 storage engine can be found here.

CONNECT Storage Engine Options

The options related to the CONNECT storage engine are described below.

CONNECT Storage Engine Options and System Variables

Some options and system variables related to the CONNECT storage engine can be found here. Direct links to many of them can be found below.

- connect-class-path
- connect-cond-push
- connect-conv-size
- connect-default-depth
- connect-default-prec
- connect-enable-mongo
- connect-exact-info
- connect-force_bson
- connect-indx-map
- connect-java-wrapper
- connect-json-all-path
- connect-json-grp-size
- connect-json-null
- connect-jvm-path
- connect-type-conv
- connect-use-tempfile
- connect-work-size

- connect-xtrace

Spider Storage Engine Options

The options and system variables related to the Spider storage engine can be found here.

Mroonga Storage Engine Options

The options and system variables related to the Mroonga storage engine can be found here.

TokuDB Storage Engine Options

The options and system variables related to the TokuDB storage engine can be found here.

Performance Schema Options

The options related to the Performance Schema are described below. Options that are also system variables are listed after:

`--performance-schema-consumer-events-stages-current`

- **Commandline:** `--performance-schema-consumer-events-stages-current`
- **Description:** Enable the events-stages-current consumer.
- **Default:** `OFF`

`--performance-schema-consumer-events-stages-history`

- **Commandline:** `--performance-schema-consumer-events-stages-history`
- **Description:** Enable the events-stages-history consumer.
- **Default:** `OFF`

`--performance-schema-consumer-events-stages-history-long`

- **Commandline:** `--performance-schema-consumer-events-stages-history-long`
- **Description:** Enable the events-stages-history-long consumer.
- **Default:** `OFF`

`--performance-schema-consumer-events-statements-current`

- **Commandline:** `--performance-schema-consumer-events-statements-current`
- **Description:** Enable the events-statements-current consumer. Use `--skip-performance-schema-consumer-events-statements-current` to disable.
- **Default:** `ON`

`--performance-schema-consumer-events-statements-history`

- **Commandline:** `--performance-schema-consumer-events-statements-history`
- **Description:** Enable the events-statements-history consumer.
- **Default:** `OFF`

`--performance-schema-consumer-events-statements-history-long`

- **Commandline:** `--performance-schema-consumer-events-statements-history-long`
- **Description:** Enable the events-statements-history-long consumer.
- **Default:** `OFF`

`--performance-schema-consumer-events-waits-current`

- **Commandline:** `--performance-schema-consumer-events-waits-current`
- **Description:** Enable the events-waits-current consumer.
- **Default:** `OFF`

`--performance-schema-consumer-events-waits-history`

- **Commandline:** `--performance-schema-consumer-events-waits-history`
- **Description:** Enable the events-waits-history consumer.
- **Default:** `OFF`

`--performance-schema-consumer-events-waits-history-long`

- **Commandline:** `--performance-schema-consumer-events-waits-history-long`
- **Description:** Enable the events-waits-history-long consumer.
- **Default:** `OFF`

`--performance-schema-consumer-global-instrumentation`

- **Commandline:** `--performance-schema-consumer-global-instrumentation`
- **Description:** Enable the global-instrumentation consumer. Use `--skip-performance-schema-consumer-global-instrumentation` to disable.
- **Default:** `ON`

`--performance-schema-consumer-statements-digest`

- **Commandline:** `--performance-schema-consumer-statements-digest`
- **Description:** Enable the statements-digest consumer. Use `--skip-performance-schema-consumer-statements-digest` to disable.
- **Default:** `ON`

`--performance-schema-consumer-thread-instrumentation`

- **Commandline:** `--performance-schema-consumer-thread-instrumentation`
- **Description:** Enable the statements-thread-instrumentation. Use `--skip-performance-schema-thread-instrumentation` to disable.
- **Default:** `ON`

Performance Schema Options and System Variables

Some options and system variables related to the Performance Schema can be found here. Direct links to many of them can be found below.

- performance-schema
- performance-schema-accounts-size
- performance-schema-digests-size
- performance-schema-events-stages-history-long-size
- performance-schema-events-stages-history-size
- performance-schema-events-statements-history-long-size
- performance-schema-events-statements-history-size
- performance-schema-events-waits-history-long-size
- performance-schema-events-waits-history-size
- performance-schema-hosts-size
- performance-schema-max-cond-classes
- performance-schema-max-cond-instances
- performance-schema-max-digest-length
- performance-schema-max-file-classes
- performance-schema-max-file-handles
- performance-schema-max-file-instances
- performance-schema-max-mutex-classes

- performance-schema-max-mutex-instances
- performance-schema-max-rwlock-classes
- performance-schema-max-rwlock-instances
- performance-schema-max-socket-classes
- performance-schema-max-socket-instances
- performance-schema-max-stage-classes
- performance-schema-max-statement-classes
- performance-schema-max-table-handles
- performance-schema-max-table-instances
- performance-schema-max-thread-classes
- performance-schema-max-thread-instances
- performance-schema-session-connect-attrs-size
- performance-schema-setup-actors-size
- performance-schema-setup-objects-size
- performance-schema-users-size

Galera Cluster Options

The options related to Galera Cluster are described below. Options that are also system variables are listed after:

`--wsrep-new-cluster`

- **Commandline:** `--wsrep-new-cluster`
- **Description:** Bootstrap a cluster. It works by overriding the current value of wsrep_cluster_address. It is recommended not to add this option to the config file as this will trigger bootstrap on every server start.

Galera Cluster Options and System Variables

Some options and system variables related to Galera Cluster can be found here. Direct links to many of them can be found below.

- wsrep-allowlist
- wsrep-auto-increment-control
- wsrep-causal-reads
- wsrep-certify-nonPK
- wsrep-cluster-address
- wsrep-cluster-name
- wsrep-convert-LOCK-to-trx
- wsrep-data-home-dir
- wsrep-dbug-option
- wsrep-debug
- wsrep-desync
- wsrep-dirty-reads
- wsrep-drupal-282555-workaround
- wsrep-forced-binlog-format
- wsrep-gtid-domain-id
- wsrep-gtid-mode
- wsrep-ignore-apply-errors
- wsrep-load-data-splitting
- wsrep-log-conflicts
- wsrep-max-ws-rows
- wsrep-max-ws-size
- wsrep-mode
- wsrep-mysql-replication-bundle
- wsrep-node-address
- wsrep-node-incoming-address
- wsrep-node-name
- wsrep-notify-cmd
- wsrep-on
- wsrep-OSU-method
- wsrep-provider
- wsrep-provider-options
- wsrep-recover
- wsrep-reject_queries
- wsrep-retry-autocommit

- wsrep-slave-FK-checks
- wsrep-slave-threads
- wsrep-slave-UK-checks
- wsrep-sr-store
- wsrep-sst-auth
- wsrep-sst-donor
- wsrep-sst-donor-rejects-queries
- wsrep-sst-method
- wsrep-sst-receive-address
- wsrep-start-position
- wsrep-status-file
- wsrep-strict-ddl
- wsrep-sync-wait
- wsrep-trx_fragment_size
- wsrep-trx_fragment_unit

Options When Debugging mariadbd

`--debug-assert-if-crashed-table`

- **Description:** Do an assert in handler::print_error() if we get a crashed table.

`--debug-binlog-fsync-sleep`

- **Description:** `--debug-binlog-fsync-sleep=#` If not set to zero, sets the number of micro-seconds to sleep after running fsync() on the binary log to flush transactions to disk. This can thus be used to artificially increase the perceived cost of such an fsync().

`--debug-crc-break`

- **Description:** `--debug-crc-break=#` Call my_debug_put_break_here() if crc matches this number (for debug).

`--debug-flush`

- **Description:** Default debug log with flush after write.

`--debug-no-sync`

- **Description:** `debug-no-sync[=#]` Disables system sync calls. Only for running tests or debugging!

`--debug-sync-timeout`

- **Description:** `debug-sync-timeout[=#]` Enable the debug sync facility and optionally specify a default wait timeout in seconds. A zero value keeps the facility disabled.

`--gdb`

- **Description:** Set up signals usable for debugging.

`--silent-startup`

- **Description:** Don't print Notes to the error log during startup.

`--sync-sys`

- **Description:** Enable/disable system sync calls. Syncs should only be turned off (`--disable-sync-sys`) when

running tests or debugging! Replaced by debug-no-sync from MariaDB 5.5.
- **Removed:** MariaDB 5.5

`--thread-alarm`

- **Description:** Enable/disable system thread alarm calls. Should only be turned off (`--disable-thread-alarm`) when running tests or debugging!

Debugging Options and System Variables

- core-file
- debug
- debug-no-thread-alarm

Other Options

Options that are also system variables are listed after:

`--allow-suspicious-udfs`

- **Commandline:** `--allow-suspicious-udfs`
- **Description:** Allows use of user-defined functions consisting of only one symbol `x()` without corresponding `x_init()` or `x_deinit()` . That also means that one can load any function from any library, for example `exit()` from `libc.so` . Not recommended unless you require old UDFs with one symbol that cannot be recompiled. From MariaDB 10.10, available as a system variable as well.

`--bootstrap`

- **Commandline:** `--bootstrap`
- **Description:** Used by mariadb installation scripts, such as mariadb-install-db to execute SQL scripts before any privilege or system tables exist. Do no use while an existing MariaDB instance is running.

`--chroot`

- **Commandline:** `--chroot=name`
- **Description:** Chroot mariadbd daemon during startup.

`--des-key-file`

- **Commandline:** `--des-key-file=name`
- **Description:** Load keys for des_encrypt() and des_encrypt from given file.

`--exit-info`

- **Commandline:** `--exit-info[=#]`
- **Description:** Used for debugging. Use at your own risk.

`--getopt-prefix-matching`

- **Commandline:** `--getopt-prefix-matching={0|1}`
- **Description:** Makes it possible to disable historical "unambiguous prefix" matching in the command-line option parsing.
- **Default:** TRUE
- **Introduced:** MariaDB 10.1.3

--help

- **Commandline:** `--help`
- **Description:** Displays help with many commandline options described, and exits.

--log-ddl-recovery

- **Commandline:** `--log-ddl-recovery=name`
- **Description:** Path to file used for recovery of DDL statements after a crash.
- **Default Value:** `ddl-recover.log`
- **Introduced:** MariaDB 10.6.1

--log-short-format

- **Commandline:** `--log-short-format`
- **Description:** Don't log extra information to update and slow-query logs.

--log-slow-file

- **Commandline:** `--log-slow-file=name`
- **Description:** Log slow queries to given log file. Defaults logging to hostname-slow.log

---

--log-slow-time

- **Commandline:** `--log-slow-time=#`
- **Description:** Log all queries that have taken more than long-query-time seconds to execute to the slow query log, if active. The argument will be treated as a decimal value with microsecond precision.

--log-tc

- **Commandline:** `--log-tc=name`
- **Description:** Defines the path to the memory-mapped file-based transaction coordinator log, which is only used if the binary log is disabled. If you have two or more XA-capable storage engines enabled, then a transaction coordinator log must be available. See Transaction Coordinator Log for more information. Also see the the `log_tc_size` system variable and the `--tc-heuristic-recover` option.
- **Default Value:** `tc.log`

--master-connect-retry

- **Commandline:** `--master-connect-retry=#`
- **Description:** Deprecated in 5.1.17 and removed in 5.5. The number of seconds the replica thread will sleep before retrying to connect to the master, in case the master goes down or the connection is lost.

--memlock

- **Commandline:** `--memlock`
- **Description:** Lock mariadbd in memory.

--ndb-use-copying-alter-table

- **Commandline:** `--ndb-use-copying-alter-table`
- **Description:** Force ndbcluster to always copy tables at alter table (should only be used if on-line alter table fails).

--one-thread

- **Commandline:** `--one-thread`
- **Description:** (Deprecated): Only use one thread (for debugging under Linux). Use thread-handling=no-threads

instead.
- **Removed:** MariaDB 10.0.4

`--plugin-load`

- **Commandline:** `--plugin-load=name`
- **Description:** This option can be used to configure the server to load specific plugins. This option uses the following format:
  - Plugins can be specified in the format `name=library`, where `name` is the plugin name and `library` is the plugin library. This format installs a single plugin from the given plugin library.
  - Plugins can also be specified in the format `library`, where `library` is the plugin library. This format installs all plugins from the given plugin library.
  - Multiple plugins can be specified by separating them with semicolons.
- Special care must be taken when specifying the `--plugin-load` option multiple times, or when specifying both the `--plugin-load` option and the `--plugin-load-add` option together. The `--plugin-load` option resets the plugin load list, and this can cause unexpected problems if you are not aware. The `--plugin-load-add` option does **not** reset the plugin load list, so it is much safer to use. See Plugin Overview: Specifying Multiple Plugin Load Options for more information.
- See Plugin Overview: Installing a Plugin with Plugin Load Options for more information.

`--plugin-load-add`

- **Commandline:** `--plugin-load-add=name`
- **Description:** This option can be used to configure the server to load specific plugins. This option uses the following format:
  - Plugins can be specified in the format `name=library`, where `name` is the plugin name and `library` is the plugin library. This format installs a single plugin from the given plugin library.
  - Plugins can also be specified in the format `library`, where `library` is the plugin library. This format installs all plugins from the given plugin library.
  - Multiple plugins can be specified by separating them with semicolons.
- Special care must be taken when specifying both the `--plugin-load` option and the `--plugin-load-add` option together. The `--plugin-load` option resets the plugin load list, and this can cause unexpected problems if you are not aware. The `--plugin-load-add` option does **not** reset the plugin load list, so it is much safer to use. See Plugin Overview: Specifying Multiple Plugin Load Options for more information.
- See Plugin Overview: Installing a Plugin with Plugin Load Options for more information.

`--port-open-timeout`

- **Commandline:** `--port-open-timeout=#`
- **Description:** Maximum time in seconds to wait for the port to become free. (Default: No wait).

`--safe-user-create`

- **Commandline:** `--safe-user-create`
- **Description:** Don't allow new user creation by the user who has no write privileges to the mysql.user table.

`--safemalloc-mem-limit`

- **Commandline:** `--safemalloc-mem-limit=#`
- **Description:** Simulate memory shortage when compiled with the `--with-debug=full` option.

`--show-slave-auth-info`

- **Commandline:** `--show-slave-auth-info`
- **Description:** Show user and password in SHOW SLAVE HOSTS on this primary.

`--skip-grant-tables`

- **Commandline:** `--skip-grant-tables`
- **Description:** Start without grant tables. This gives all users FULL ACCESS to all tables, which is useful in case of a lost root password. Use mariadb-admin flush-privileges, mariadb-admin reload or FLUSH PRIVILEGES to resume using the grant tables. From MariaDB 10.10, available as a system variable as well.

Because the Event Scheduler also depends on the grant tables for its functionality, it is automatically disabled when running with `--skip-grant-tables`.

`--skip-host-cache`

- **Commandline:** `--skip-host-cache`
- **Description:** Don't cache host names.

`--skip-partition`

- **Commandline:** `--skip-partition`, `--disable-partition`
- **Description:** Disables user-defined partitioning. Previously partitioned tables cannot be accessed or modifed. Tables can still be seen with SHOW TABLES or by viewing the INFORMATION_SCHEMA.TABLES table. Tables can be dropped with DROP TABLE, but this only removes .frm files, not the associated .par files, which will need to be removed manually.

`--skip-slave-start`

- **Commandline:** `--skip-slave-start`
- **Description:** If set, replica is not autostarted.

`--skip-ssl`

- **Commandline:** `--skip-ssl`
- **Description:** Disable TLS connections.

`--skip-symlink`

- **Commandline:** `--skip-symlink`
- **Description:** Don't allow symlinking of tables. Deprecated and removed in MariaDB 5.5. Use symbolic-links with the `skip` option prefix instead.
- **Removed:** MariaDB 5.5

`--skip-thread-priority`

- **Commandline:** `--skip-thread-priority`
- **Description:** Don't give threads different priorities. Deprecated and removed in MariaDB 10.0.
- **Removed:** MariaDB 10.0

`--sql-bin-update-same`

- **Commandline:** `--sql-bin-update-same=#`
- **Description:** The update log was deprecated in version 5.0 and replaced by the binary log, so this option did nothing since then. Deprecated and removed in MariaDB 5.5.
- **Removed:** MariaDB 5.5

`--ssl`

- **Commandline:** `--ssl`
- **Description:** Enable TLS for connection (automatically enabled with other flags). Disable with '`--skip-ssl`'.

`--stack-trace`

- **Commandline:** `--stack-trace`, `--skip-stack-trace`
- **Description:** Print a stack trace on failure. Enabled by default, disable with `-skip-stack-trace`.

`--symbolic-links`

- **Commandline:** `--symbolic-links`
- **Description:** Enables symbolic link support. When set, the have_symlink system variable shows as `YES`. Silently ignored in Windows. Use `--skip-symbolic-links` to disable.

`--tc-heuristic-recover`

- **Commandline:** `--tc-heuristic-recover=name`
- **Description:** If manual heuristic recovery is needed, this option defines the decision to use in the heuristic recovery process. Manual heuristic recovery may be needed if the transaction coordination log is missing or if it doesn't contain all prepared transactions. This option can be set to `OFF`, `COMMIT`, or `ROLLBACK`. The default is `OFF`. See also the `--log-tc` server option and the `log_tc_size` system variable.

`--temp-pool`

- **Commandline:** `--temp-pool`
- **Description:** Using this option will cause most temporary files created to use a small set of names, rather than a unique name for each new file. Defaults to `1` until MariaDB 10.5.6, use `--skip-temp-pool` to disable. Deprecated and defaults to `0` from MariaDB 10.5.7, as benchmarking shows it causes a heavy mutex contention.

`--test-expect-abort`

- **Commandline:** `--test-expect-abort`
- **Description:** Expect that server aborts with 'abort'; Don't write out server variables on 'abort'. Useful only for test scripts.

`--test-ignore-wrong-options`

- **Commandline:** `--test-ignore-wrong-options`
- **Description:** Ignore wrong enums values in command line arguments. Useful only for test scripts.

`--user`

- **Commandline:** `--user=name`
- **Description:** Run mariadbd daemon as user.

`--verbose`

- **Commandline:** `-v`, `--verbose`
- **Description:** Used with help option for detailed help.

Other Options and System Variables

- allow-suspicious-udfs
- automatic-sp-privileges
- back-log
- basedir
- check-constraint-checks
- column-compression-threshold
- column-compression-zlib-level

- column-compression-zlib-strategy
- column-compression-zlib-wrap
- completion-type
- connect-timeout
- datadir
- date-format
- datetime-format
- deadlock-search-depth-long
- deadlock-search-depth-short
- deadlock-timeout-long
- deadlock-timeout-short
- default-password-lifetime
- default-regex-flags
- default-storage-engine
- default-table-type
- delay-key-write
- disconnect-on-expired-password
- div-precision-increment
- enable-named-pipe
- encrypt-binlog
- encrypt-tmp-disk-tables
- encrypt-tmp-files
- encryption-algorithm
- engine-condition-pushdown
- eq-range-index-dive-limit
- event-scheduler
- expire-logs-days
- explicit-defaults-for-timestamp
- extra-max-connections
- extra-port
- flush
- flush-time
- ft-boolean-syntax
- ft-max-word-len
- ft-min-word-len
- ft-query-expansion-limit
- ft-stopword-file
- general-log
- general-log-file
- group-concat-max-len
- histogram-size
- histogram-type
- host-cache-size
- idle-readonly-transaction-timeout
- idle-transaction-timeout
- idle-write-transaction-timeout
- ignore-db-dirs
- in-predicate-conversion-threshold
- init-connect
- init-file
- interactive-timeout
- large-pages
- local-infile
- lock-wait-timeout
- log
- log-disabled-statements
- log-error
- log-output
- log-queries-not-using-indexes
- log-slow-admin-statements
- log-slow-disabled-statements
- log-slow-filter
- log-slow-min-examined-row-limit
- log-slow-queries
- log-slow-query
- log-slow-query-file
- log-slow-query-time

- log-slow-rate-limit
- log-slow-slave-statements
- log-slow-verbosity
- log-tc-size
- log-warnings
- long-query-time
- low-priority-updates
- lower-case-table-names
- max-allowed-packet
- max-connections
- max-connect-errors
- max-delayed-threads
- max-digest-length
- max-error-count
- max-length-for-sort-data
- max-long-data-size
- max-password-errors
- max-prepared-stmt-count
- max-recursive-iterations
- max-rowid-filter-size
- max-session-mem-used
- max-sp-recursion-depth
- max-statement-time
- max-tmp-tables
- max-user-connections
- max-write-lock-count
- metadata-locks-cache-size
- metadata-locks-hash-instances
- min-examined-row-limit
- mrr-buffer-size
- multi-range-count
- --mysql56-temporal-format
- net-buffer-length
- net-read-timeout
- net-retry-count
- net-write-timeout
- open-files-limit
- pid-file
- plugin-dir
- plugin-maturity
- port
- preload-buffer-size
- profiling-history-size
- progress-report-time
- proxy-protocol-networks
- query-cache-limit
- query-cache-min-res-unit
- query-cache-strip-comments
- query-cache-wlock-invalidate
- read-rnd-buffer-size
- read-only
- redirect-url
- require-secure-transport
- safe-show-database
- secure-auth
- secure-file-priv
- secure-timestamp
- session-track-schema
- session-track-state-change
- session-track-system-variables
- session-track-transaction-info
- skip-automatic-sp-privileges
- skip-external-locking
- skip-large-pages
- skip-log-error
- skip-name-resolve
- skip-networking

- skip-show-database
- slow-launch-time
- slow-query-log
- slow-query-log-file
- socket
- sort-buffer-size
- sql-if-exists
- sql-mode
- ssl-ca
- ssl-capath
- ssl-cert
- ssl-cipher
- ssl-crl
- ssl-crlpath
- ssl-key
- standards_compliant_cte
- stored-program-cache
- strict_password_validation
- sync-frm
- system-versioning-alter-history
- system-versioning-asof
- system-versioning-innodb-algorithm-simple
- system-versioning-insert-history
- table-lock-wait-timeout
- tcp-keepalive-interval
- tcp-keepalive-probes
- tcp-keepalive-time
- tcp-nodelay
- thread-cache-size
- thread-concurrency
- thread-handling
- thread-pool-dedicated-listener
- thread-pool-exact-stats
- thread-pool-idle-timeout
- thread-pool-max-threads
- thread-pool-min-threads
- thread-pool-oversubscribe
- thread-pool-prio-kickup-timer
- thread-pool-priority
- thread-pool-size
- thread-pool-stall-limit
- thread-stack
- timed-mutexes
- time-format
- tls-version
- tmpdir
- transaction-isolation
- transaction-alloc-block-size
- transaction-prealloc-size
- transaction-read-only
- updatable-views-with-limit
- userstat
- version
- wait-timeout

Authentication Plugins - Options and System Variables

Authentication Plugin - `ed25519`

The options related to the `ed25519` authentication plugin can be found here.

Authentication Plugin - `gssapi`

The system variables related to the `gssapi` authentication plugin can be found here.

The options related to the `gssapi` authentication plugin can be found here.

Authentication Plugin - `named_pipe`

The options related to the `named_pipe` authentication plugin can be found here.

Authentication Plugin - `pam`

The system variables related to the `pam` authentication plugin can be found here.

The options related to the `pam` authentication plugin can be found here.

Authentication Plugin - `unix_socket`

The options related to the `unix_socket` authentication plugin can be found here.

Encryption Plugins - Options and System Variables

Encryption Plugin - `aws_key_management`

The system variables related to the `aws_key_management` encryption plugin can be found here.

The options elated to the `aws_key_management` encryption plugin can be found here.

Encryption Plugin - `file_key_management`

The system variables related to the `file_key_management` encryption plugin can be found here.

The options related to the `file_key_management` encryption plugin can be found here.

Password Validation Plugins - Options and System Variables

Password Validation Plugin - `simple_password_check`

The system variables related to the `simple_password_check` password validation plugin can be found here.

The options related to the `simple_password_check` password validation plugin can be found here.

Password Validation Plugin - `cracklib_password_check`

The system variables related to the `cracklib_password_check` password validation plugin can be found here.

The options related to the `cracklib_password_check` password validation plugin can be found here.

Audit Plugins - Options and System Variables

Audit Plugin - `server_audit`

Options and system variables related to the `server_audit` audit plugin can be found here.

Audit Plugin - `SQL_ERROR_LOG`

The system variables related to the `SQL_ERROR_LOG` audit plugin can be found here.

The options related to the `SQL_ERROR_LOG` audit plugin can be found here.

Audit Plugin - QUERY_RESPONSE_TIME_AUDIT

The options related to the `QUERY_RESPONSE_TIME_AUDIT` audit plugin can be found here.

Daemon Plugins - Options and System Variables

Daemon Plugin - `handlersocket`

The options for the HandlerSocket plugin are all described on the HandlerSocket Configuration Option page.

Information Schema Plugins - Options and System Variables

Information Schema Plugin - `DISKS`

The options related to the `DISKS` information schema plugin can be found here.

Information Schema Plugin - `feedback`

The system variables related to the `feedback` plugin can be found here.

The options related to the `feedback` plugin can be found here.

Information Schema Plugin - `LOCALES`

The options related to the `LOCALES` information schema plugin can be found here.

Information Schema Plugin - `METADATA_LOCK_INFO`

The options related to the `METADATA_LOCK_INFO` information schema plugin can be found here.

Information Schema Plugin - `QUERY_CACHE_INFO`

The options related to the `QUERY_CACHE_INFO` information schema plugin can be found here.

Information Schema Plugin - `QUERY_RESPONSE_TIME`

The system variables related to the `QUERY_RESPONSE_TIME` information schema plugin can be found here.

The options related to the `QUERY_RESPONSE_TIME` information schema plugin can be found here.

Information Schema Plugin - `user_variables`

The options related to the `user_variables` information schema plugin can be found here.

Information Schema Plugin - `WSREP_MEMBERSHIP`

The options related to the `WSREP_MEMBERSHIP` information schema plugin can be found here.

Information Schema Plugin - `WSREP_STATUS`

The options related to the `WSREP_STATUS` information schema plugin can be found here.

Replication Plugins - Options and System Variables

Replication Plugin - `rpl_semi_sync_master`

The system variables related to the `rpl_semi_sync_master` replication plugin can be found here.

The options related to the `rpl_semi_sync_master` replication plugin can be found here.

Replication Plugin - `rpl_semi_sync_slave`

The system variables related to the `rpl_semi_sync_slave` replication plugin can be found here.

The options related to the `rpl_semi_sync_slave` replication plugin can be found here.

Default Values

You can verify the default values for an option by doing:

```
mariadbd --no-defaults --help --verbose
```

2.1.6.5 What to Do if MariaDB Doesn't Start

Contents

There could be many reasons that MariaDB fails to start. This page will help troubleshoot some of the more common reasons and provide solutions.

If you have tried everything here, and still need help, you can ask for help on IRC or on the forums - see Where to find other MariaDB users and developers - or ask a question at the Starting and Stopping MariaDB page.

The Error Log and the Data Directory

The reason for the failure will almost certainly be written in the error log and, if you are starting MariaDB manually, to the console. By default, the error log is named *host-name*.err and is written to the data directory.

Common Locations:

- /var/log/
- /var/log/mysql
- C:\Program Files\MariaDB x.y\data (x.y refers to the version number)
- C:\Program Files (x86)\MariaDB x.y\data (32bit version on 64bit Windows)

It's also possible that the error log has been explicitly written to another location. This is often done by changing the `datadir` or `log_error` system variables in an option file. See Option Files below for more information about that.

A quick way to get the values of these system variables is to execute the following commands:

```
mysqld --help --verbose | grep 'log-error' | tail -1
mysqld --help --verbose | grep 'datadir' | tail -1
```

Option Files

Another kind of file to consider when troubleshooting is option files. The default option file is called `my.cnf`. Option files contain configuration options, such as the location of the data directory mentioned above. If you're unsure where the option file is located, see Configuring MariaDB with Option Files: Default Option File Locations for information on the default locations.

You can check which configuration options MariaDB server will use from its option files by executing the following command:

```
mysqld --print-defaults
```

You can also check by executing the following command:

```
my_print_defaults --mysqld
```

See Configuring MariaDB with Option Files: Checking Program Options for more information on checking configuration options.

Invalid Option or Option Value

Another potential reason for a startup failure is that an option file contains an invalid option or an invalid option value. In those cases, the error log should contain an error similar to this:

```
140514 12:19:37 [ERROR] /usr/local/mysql/bin/mysqld: unknown variable 'option=value'
```

This is more likely to happen when you upgrade to a new version of MariaDB. In most cases the option file from the old version of MariaDB will work just fine with the new version. However, occasionally, options are removed in new versions of MariaDB, or the valid values for options are changed in new versions of MariaDB. Therefore, it's possible for an option file to stop working after an upgrade.

Also remember that option names are case sensitive.

Examine the specifics of the error. Possible fixes are usually one of the following:

- If the option is completely invalid, then remove it from the option file.
- If the option's name has changed, then fix the name.
- If the option's valid values have changed, then change the option's value to a valid one.
- If the problem is caused by a simple typo, then fix the typo.

Can't Open Privilege Tables

It is possible to see errors similar to the following:

```
System error 1067 has occurred.
Fatal error: Can't open privilege tables: Table 'mysql.host' doesn't exist
```

If errors like this occur, then critical system tables are either missing or are in the wrong location. The above error is quite common after an upgrade if the option files set the `basedir` or `datadir` to a non-standard location, but the new server is using the default location. Therefore, make sure that the `basedir` and `datadir` variables are correctly set.

If you're unsure where the option file is located, see Configuring MariaDB with Option Files: Default Option File Locations for information on the default locations.

If the system tables really do not exist, then you may need to create them with mariadb-install-db. See Installing System Tables (mariadb-install-db) for more information.

Can't Create Test File

One of the first tests on startup is to check whether MariaDB can write to the data directory. When this fails, it will log an error like this:

```
May 13 10:24:28 mariadb3 mysqld[19221]: 2019-05-13 10:24:28 0 [Warning] Can't create test file
/usr/local/data/mariadb/mariadb3.lower-test
May 13 10:24:28 mariadb3 mysqld[19221]: 2019-05-13 10:24:28 0 [ERROR] Aborting
```

This is usually a permission error on the directory in which this file is being written. Ensure that the entire `datadir` is owned by the user running `mysqld`, usually `mysql`. Ensure that directories have the "x" (execute) directory permissions for the owner. Ensure that all the parent directories of the `datadir` upwards have "x" (execute) permissions for all (`user`, `group`, and `other`).

Once this is checked look at the systemd and selinux documentation below, or AppArmor.

Can't Lock Aria Control File

On starting MariaDB, the `aria_log_control` file is locked. If a lock cannot be obtained, it will log and error like this:

```
2023-05-01 16:27:03 0 [ERROR] mariadbd: Can't lock aria control file
'/var/lib/mysql/aria_log_control' for exclusive use, error: 11. Will retry for 30 seconds
```

This almost always cause for this is that there is already an existing MariaDB service running on this data directory. Recommend aborting this startup and looking closely for the other MariaDB instance.

The less likely case is there isn't locking available which might occur on a NFS data directory with explictly disable locking.

Unable to lock ./ibdata1 error 11

Like the above for the Aria Control File, this is a attempting to exclusively lock the `ibdata1` InnoDB system tablespace. Error 11 corresponds to the system error "OS error code 11: Resource temporarily unavailable" meaning the lock cannot be created.

```
2023-05-01 16:27:34 0 [ERROR] InnoDB: Unable to lock ./ibdata1 error: 11
2023-05-01 16:27:34 0 [Note] InnoDB: Check that you do not already have another mariadbd process
using the same InnoDB data or log files.
2023-05-01 16:27:34 0 [ERROR] InnoDB: Plugin initialization aborted with error Generic error
2023-05-01 16:27:35 0 [Note] InnoDB: Starting shutdown...
```

Like the above, this is an indication that a second MariaDB instance is already running on the data directory.

InnoDB

InnoDB is probably the MariaDB component that most frequently causes a crash. In the error log, lines containing InnoDB messages generally start with "InnoDB:".

Cannot Allocate Memory for the InnoDB Buffer Pool

In a typical installation on a dedicated server, at least 70% of your memory should be assigned to InnoDB buffer pool; sometimes it can even reach 85%. But be very careful: don't assign to the buffer pool more memory than it can allocate. If it cannot allocate memory, InnoDB will use the disk's swap area, which is very bad for performance. If swapping is disabled or the swap area is not big enough, InnoDB will crash. In this case, MariaDB will probably try to restart several times, and each time it will log a message like this:

```
140124 17:29:01 InnoDB: Fatal error: cannot allocate memory for the buffer pool
```

In that case, you will need to add more memory to your server/VM or decrease the value of the innodb_buffer_pool_size variables.

Remember that the buffer pool will slightly exceed that limit. Also, remember that MariaDB also needs allocate memory for other storage engines and several per-connection buffers. The operating system also needs memory.

InnoDB Table Corruption

By default, InnoDB deliberately crashes the server when it detects table corruption. The reason for this behavior is preventing corruption propagation. However, in some situations, server availability is more important than data integrity. For this reason, we can avoid these crashes by changing the value of innodb_corrupt_table_action to 'warn'.

If InnoDB crashes the server after detecting data corruption, it writes a detailed message in the error log. The first lines are similar to the following:

```
InnoDB: Database page corruption on disk or a failed
InnoDB: file read of page 7.
InnoDB: You may have to recover from a backup.
```

Generally, it is still possible to recover most of the corrupted data. To do so, restart the server in InnoDB recovery mode and try to extract the data that you want to backup. You can save them in a CSV file or in a non-InnoDB table. Then, restart the server in normal mode and restore the data.

MyISAM

Most tables in the mysql database are MyISAM tables. These tables are necessary for MariaDB to properly work, or even

start.

A MariaDB crash could cause system tables corruption. With the default settings, MariaDB will simply not start if the system tables are corrupted. With myisam_recover_options, we can force MyISAM to repair damaged tables.

systemd

If you are using `systemd`, then there are a few relevant notes about startup failures:

- If MariaDB is configured to access files under `/home`, `/root`, or `/run/user`, then the default systemd unit file will prevent access to these directories with a `Permission Denied` error. This happens because the unit file set `ProtectHome=true`. See Systemd: Configuring Access to Home Directories for information on how to work around this.
- The default systemd unit file also sets ProtectSystem=full, which places restrictions on writing to a few other directories. Overwriting this with `ProtectSystem=off` in the same way as above will restore access to these directories.
- If MariaDB takes longer than 90 seconds to start, then the default systemd unit file will cause it to fail with an error. This happens because the default value for the `TimeoutStartSec` option is 90 seconds. See Systemd: Configuring the Systemd Service Timeout for information on how to work around this.
- The systemd journal may also contain useful information about startup failures. See Systemd: Systemd Journal for more information.

See systemd documentation for further information on systemd configuration.

SELinux

Security-Enhanced Linux (SELinux) is a Linux kernel module that provides a framework for configuring mandatory access control (MAC) system for many resources on the system. It is enabled by default on some Linux distributions, including RHEL, CentOS, Fedora, and other similar Linux distribution. SELinux prevents programs from accessing files, directories or ports unless it is configured to access those resources.

You might need to troubleshoot SELinux-related issues in cases, such as:

- MariaDB is using a non-default port.
- MariaDB is reading from or writing to some files (datadir, log files, option files, etc.) located at non-default paths.
- MariaDB is using a plugin that requires access to resources that default installations do not use.

Setting SELinux state to `permissive` is a common way to investigate what is going wrong while allowing MariaDB to function normally. `permissive` is supposed to produce a log entry every time it should block a resource access, without actually blocking it. However, there are situations when SELinux blocks resource accesses even in `permissive` mode.

See SELinux for more information.

AppArmor

Add the following to `/etc/apparmor.d/tunables/alias` if you have moved the datadir:

```
alias /var/lib/mysql/ -> /data/mariadb/,
```

The restart AppArmor:

```
sudo systemctl restart apparmor
```

2.1.6.6 Running MariaDB from the Build Directory

You can run mariadbd directly from the build directory (without doing `make install`).

Starting mariadbd After Build on Windows

On Windows, the data directory is produced during the build.

The simplest way to start database from the command line is:

1. Go to the directory where mariadbd.exe is located (subdirectory sql\Debug or sql\Relwithdebinfo of the build directory)
2. From here, execute, if you are using MariaDB 10.5 or newer,

   ```
   mariadbd.exe --console
   ```

   else

   ```
   mariadbd.exe --console
   ```

As usual, you can pass other server parameters on the command line, or store them in a my.ini configuraton file and pass `--defaults-file=path\to\my.ini`

The default search path on Windows for the my.ini file is:

- GetSystemWindowsDirectory()
- GetWindowsDirectory()
- C:\
- Directory where the executable is located

Starting mariadbd After Build on Unix

Copy the following to your '`~/.my.cnf`' file.

There are two lines you have to edit: '`datadir=`' and '`language=`'. Be sure to change them to match your environment.

```
# Example MariadB config file.
# You can copy this to one of:
# /etc/my.cnf to set global options,
# /mysql-data-dir/my.cnf to get server specific options or
# ~/my.cnf for user specific options.
#
# One can use all long options that the program supports.
# Run the program with --help to get a list of available options

# This will be passed to all MariaDB clients
[client]
#password=my_password
#port=3306
#socket=/tmp/mysql.sock

# Here is entries for some specific programs
# The following values assume you have at least 32M ram

# The mariadb server  (both [mysqld] and [mariadb] works here)
[mariadb]
#port=3306
#socket=/tmp/mysql.sock

# The following three entries caused mysqld 10.0.1-MariaDB (and possibly other versions) to abort
# skip-locking
# set-variable  = key_buffer=16M

loose-innodb_data_file_path = ibdata1:1000M
loose-mutex-deadlock-detector
gdb

######### Fix the two following paths

# Where you want to have your database
datadir=/path/to/data/dir

# Where you have your mysql/MariaDB source + sql/share/english
language=/path/to/src/dir/sql/share/english

########## One can also have a different path for different versions, to simplify development.

[mariadb-10.1]
lc-messages-dir=/my/maria-10.1/sql/share

[mariadb-10.2]
lc-messages-dir=/my/maria-10.2/sql/share

[mysqldump]
quick
set-variable = max_allowed_packet=16M

[mysql]
no-auto-rehash

[myisamchk]
set-variable= key_buffer=128M
```

With the above file in place, go to your MariaDB source directory and execute:

```
./scripts/mariadb-install-db --srcdir=$PWD --datadir=/path/to/data/dir --user=$LOGNAME
```

Above '$PWD' is the environment variable that points to your current directory. If you added `datadir` to your `my.cnf`, you don't have to give this option above. Also above, --user=$LOGNAME is necessary when using msqyld 10.0.1-MariaDB (and possibly other versions)

Now you can start `mariadbd` (or `mysqld` if you are using a version older than MariaDB 10.5) in the debugger:

```
cd sql
ddd ./mariadbd &
```

Or start mariadbd on its own:

```
cd sql
./mariadbd &
```

After starting up `mariadbd` using one of the above methods (with the debugger or without), launch the client (as root if you don't have any users setup yet).

```
../client/mariadb
```

Using a Storage Engine Plugin

The simplest case is to compile the storage engine into MariaDB:

```
cmake -DWITH_PLUGIN_<plugin_name>=1` .
```

Another option is to point `mariadbd` to the storage engine directory:

```
./mariadbd --plugin-dir={build-dir-path}/storage/connect/.libs
```

2.1.6.7 mysql.server

Contents

The `mysql.server` startup script is in MariaDB distributions on Linux and Unix. It is a wrapper that works as a standard sysVinit script. However, it can be used independently of sysVinit as a regular `sh` script. The script starts the `mysqld` server process by first changing its current working directory to the MariaDB install directory and then starting `mysqld_safe`. The script requires the standard sysVinit arguments, such as `start`, `stop`, `restart`, and `status`. For example:

```
mysql.server start
mysql.server restart
mysql.server stop
mysql.server status
```

It can be used on systems such as Linux, Solaris, and Mac OS X.

The `mysql.server` script starts `mysqld` by first changing to the MariaDB install directory and then calling `mysqld_safe`.

Using mysql.server

The command to use `mysql.server` and the general syntax is:

```
mysql.server [ start | stop | restart | status ] <options> <mysqld_options>
```

Options

If an unknown option is provided to `mysqld_safe` on the command-line, then it is passed to `mysqld_safe`.

`mysql.server` supports the following options:

Option	Description

`--basedir=path`	The path to the MariaDB installation directory.
`--datadir=path`	The path to the MariaDB data directory.
`--pid-file=file_name`	The path name of the file in which the server should write its process ID. If not provided, the default, `host_name.pid` is used.
`--service-startup-timeout=file_name`	How long in seconds to wait for confirmation of server startup. If the server does not start within this time, *mysql.server* exits with an error. The default value is 900. A value of 0 means not to wait at all for startup. Negative values mean to wait forever (no timeout).
`--use-mysqld_safe`	Use mysqld_safe to start the server. This is the default.
`--use-manager`	Use Instance Manager to start the server.
`--user=user_name`	The login user name to use for running `mysqld`.

Option Files

In addition to reading options from the command-line, `mysql.server` can also read options from option files.

The following options relate to how MariaDB command-line tools handles option files. They must be given as the first argument on the command-line:

Option	Description
`--print-defaults`	Print the program argument list and exit.
`--no-defaults`	Don't read default options from any option file.
`--defaults-file=#`	Only read default options from the given file #.
`--defaults-extra-file=#`	Read this file after the global files are read.

Option Groups

`mysql.server` reads options from the following option groups from option files:

Group	Description
`[mysql.server]`	Options read by `mysql.server`, which includes both MariaDB Server and MySQL Server.

`mysql.server` also reads options from the following server option groups from option files:

Group	Description
`[mysqld]`	Options read by `mysqld`, which includes both MariaDB Server and MySQL Server.
`[server]`	Options read by MariaDB Server.
`[mysqld-X.Y]`	Options read by a specific version of `mysqld`, which includes both MariaDB Server and MySQL Server. For example, `[mysqld-5.5]`.
`[mariadb]`	Options read by MariaDB Server.
`[mariadb-X.Y]`	Options read by a specific version of MariaDB Server.
`[client-server]`	Options read by all MariaDB client programs and the MariaDB Server. This is useful for options like socket and port, which is common between the server and the clients.
`[galera]`	Options read by a galera-capable MariaDB Server. Available on systems compiled with Galera support.

Customizing mysql.server

If you have installed MariaDB to a non-standard location, then you may need to edit the `mysql.server` script to get it to work right.

If you do not want to edit the `mysql.server` script itself, then `mysql.server` also sources a few other `sh` scripts. These files can be used to set any variables that might be needed to make the script work in your specific environment. The files are:

- /etc/default/mysql
- /etc/sysconfig/mysql
- /etc/conf.d/mysql

Installed Locations

`mysql.server` can be found in the `support-files` directory under your MariaDB installation directory or in a MariaDB source distribution.

Installed SysVinit Locations

On systems that use sysVinit, `mysql.server` may also be installed in other locations and with other names.

If you installed MariaDB on Linux using RPMs, then the `mysql.server` script will be installed into the `/etc/init.d` directory with the name `mysql`. You need not install it manually.

Manually Installing with SysVinit

If you install MariaDB from source or from a binary tarball that does not install `mysql.server` automatically, and if you are on a system that uses sysVinit, then you can manually install `mysql.server` with sysVinit. This is usually done by copying it to `/etc/init.d/` and then creating specially named symlinks in the appropriate `/etc/rcX.d/` directories (where 'X' is a number between 0 and 6).

> In the examples below we will follow the historical convention of renaming the `mysql.server` script to '`mysql`' when we copy it to `/etc/init.d/`.

The first step for most Linux distributions is to copy the `mysql.server` script to `/etc/init.d/` and make it executable:

```
cd /path/to/your/mariadb-version/support-files/
cp mysql.server /etc/init.d/mysql
chmod +x /etc/init.d/mysql
```

Now all that is needed is to create the specially-named symlinks. On both RPM and Debian-based Linux distributions there are tools which do this for you. Consult your distribution's documentation if neither of these work for you and follow their instructions for generating the symlinks or creating them manually.

On RPM-based distributions (like Fedora and CentOS), you use `chkconfig`:

```
chkconfig --add mysql
chkconfig --level 345 mysql on
```

On Debian-based distributions you use `update-rc.d`:

```
update-rc.d mysql defaults
```

On FreeBSD, the location for startup scripts is `/usr/local/etc/rc.d/` and when you copy the `mysql.server` script there you should rename it so that it matches the `*.sh` pattern, like so:

```
cd /path/to/your/mariadb/support-files/
cp mysql.server /usr/local/etc/rc.d/mysql.server.sh
```

As stated above, consult your distribution's documentation for more information on starting services like MariaDB at system startup.

See mysqld startup options for information on configuration options for `mysqld`.

2.1.6.8 mysqld_safe

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadbd-safe` is a symlink to `mysqld_safe`.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mariadbd-safe` is the name of the server, with `mysqld_safe` a symlink .

Contents

The `mysqld_safe` startup script is in MariaDB distributions on Linux and Unix. It is a wrapper that starts `mysqld` with some extra safety features. For example, if `mysqld_safe` notices that `mysqld` has crashed, then `mysqld_safe` will automatically restart `mysqld` .

`mysqld_safe` is the recommended way to start `mysqld` on Linux and Unix distributions that do not support `systemd` . Additionally, the `mysql.server` init script used by `sysVinit` starts `mysqld` with `mysqld_safe` by default.

Using mysqld_safe

The command to use `mysqld_safe` and the general syntax is:

```
mysqld_safe [ --no-defaults | --defaults-file | --defaults-extra-file | --defaults-group-suffix
| --print-defaults ] <options> <mysqld_options>
```

Options

Many of the options supported by `mysqld_safe` are identical to options supported by `mysqld` . If an unknown option is provided to `mysqld_safe` on the command-line, then it is passed to `mysqld` .

`mysqld_safe` supports the following options:

Option	Description
`--help`	Display a help message and exit.
`--autoclose`	(NetWare only) On NetWare, `mysqld_safe` provides a screen presence. When you unload (shut down) the `mysqld_safe` NLM, the screen does not by default go away. Instead, it prompts for user input: `NLM has terminated; Press any key to close the screen` . If you want NetWare to close the screen automatically instead, use the `--autoclose` option to mysqld_safe.
`--basedir=path`	The path to the MariaDB installation directory.
`--core-file-size=size`	The size of the core file that mysqld should be able to create. The option value is passed to ulimit -c.
`--crash-script=file`	Script to call in the event of mysqld crashing.
`--datadir=path`	The path to the data directory.
`--defaults-extra-file=path`	The name of an option file to be read in addition to the usual option files. This must be the first option on the command line if it is used. If the file does not exist or is otherwise inaccessible, the server will exit with an error.
`--defaults-file=file_name`	The name of an option file to be read instead of the usual option files. This must be the first option on the command line if it is used.
`--defaults-group-suffix=#`	In addition to the default option groups, also read option groups with this suffix.

`--flush-caches`	Flush and purge buffers/caches before starting the server.
`--ledir=path`	If mysqld_safe cannot find the server, use this option to indicate the path name to the directory where the server is located.
`--log-error=file_name`	Write the error log to the given file.
`--malloc-lib=lib`	Preload shared library *lib* if available. See debugging MariaDB for an example.
`--mysqld=prog_nam`	The name of the server program (in the ledir directory) that you want to start. This option is needed if you use the MariaDB binary distribution but have the data directory outside of the binary distribution. If mysqld_safe cannot find the server, use the `--ledir` option to indicate the path name to the directory where the server is located.
`--mysqld-version=suffix`	This option is similar to the `--mysqld` option, but you specify only the suffix for the server program name. The basename is assumed to be mysqld. For example, if you use `--mysqld-version=debug`, mysqld_safe starts the mysqld-debug program in the ledir directory. If the argument to `--mysqld-version` is empty, mysqld_safe uses mysqld in the ledir directory.
`--nice=priority`	Use the nice program to set the server´s scheduling priority to the given value.
`--no-defaults`	Do not read any option files. This must be the first option on the command line if it is used.
`--no-watch`, `--nowatch`, `--no-auto-restart`	Exit after starting mysqld.
`--numa-interleave`	Run mysqld with its memory interleaved on all NUMA nodes.
`--open-files-limit=count`	The number of files that mysqld should be able to open. The option value is passed to ulimit -n. Note that you need to start mysqld_safe as root for this to work properly.
`--pid-file=file_name`	The path name of the process ID file.
`--plugin-dir=dir_name`	Directory for client-side plugins.
`--port=port_num`	The port number that the server should use when listening for TCP/IP connections. The port number must be 1024 or higher unless the server is started by the root system user.
`--print-defaults`	Print the program argument list and exit.
`--skip-kill-mysqld`	Do not try to kill stray mysqld processes at startup. This option works only on Linux.
`--socket=path`	The Unix socket file that the server should use when listening for local connections.
`--syslog`, `--skip-syslog`	`--syslog` causes error messages to be sent to syslog on systems that support the logger program. `--skip-syslog` suppresses the use of syslog; messages are written to an error log file.
`--syslog-tag=tag`	For logging to syslog, messages from `mysqld_safe` and mysqld are written with a tag of mysqld_safe and mysqld, respectively. To specify a suffix for the tag, use `--syslog-tag=tag`, which modifies the tags to be `mysqld_safe-tag` and `mysqld-tag`.
`--timezone=timezone`	Set the TZ time zone environment variable to the given option value. Consult your operating system documentation for legal time zone specification formats. Also see Time Zones.
`--user={user_name or user_id}`	Run the mysqld server as the user having the name user_name or the numeric user ID user_id. (“User” in this context refers to a system login account, not a MariaDB user listed in the grant tables.)

Option Files

In addition to reading options from the command-line, `mysqld_safe` can also read options from option files. If an unknown option is provided to `mysqld_safe` in an option file, then it is ignored.

The following options relate to how MariaDB command-line tools handles option files. They must be given as the **first argument** on the command-line:

Option	Description
`--print-defaults`	Print the program argument list and exit.
`--no-defaults`	Don't read default options from any option file.
`--defaults-file=#`	Only read default options from the given file #.
`--defaults-extra-file=#`	Read this file after the global files are read.
`--defaults-group-suffix=#`	In addition to the default option groups, also read option groups with this suffix.

Option Groups

`mysqld_safe` reads options from the following option groups from option files:

Group	Description
`[mysqld_safe]`	Options read by `mysqld_safe`, which includes both MariaDB Server and MySQL Server.
`[safe_mysqld]`	Options read by `mysqld_safe`, which includes both MariaDB Server and MySQL Server.
`[mariadb_safe]`	Options read by `mysqld_safe` from MariaDB Server.
`[mariadb-safe]`	Options read by `mysqld_safe` from MariaDB Server. Available starting with MariaDB 10.4.6.

The `[safe_mysqld]` option group is primarily supported for backward compatibility. You should rename such option groups to `[mysqld_safe]` in MariaDB installations to prevent breakage in the future if this compatibility is removed.

`mysqld_safe` also reads options from the following server option groups from option files:

Group	Description
`[mysqld]`	Options read by `mysqld`, which includes both MariaDB Server and MySQL Server.
`[server]`	Options read by MariaDB Server.
`[mysqld-X.Y]`	Options read by a specific version of `mysqld`, which includes both MariaDB Server and MySQL Server. For example, `[mysqld-5.5]`.
`[mariadb]`	Options read by MariaDB Server.
`[mariadb-X.Y]`	Options read by a specific version of MariaDB Server.
`[client-server]`	Options read by all MariaDB client programs and the MariaDB Server. This is useful for options like socket and port, which is common between the server and the clients.
`[galera]`	Options read by a galera-capable MariaDB Server. Available on systems compiled with Galera support.

For example, if you specify the `log_error` option in a server option group in an option file, like this:

```
[mariadb]
log_error=error.log
```

Then `mysqld_safe` will also use this value for its own `--log-error` option:

Configuring the Open Files Limit

When using `mysqld_safe`, the system's open files limit can be changed by providing the `--open-files-limit` option either on the command-line or in an option file. For example:

```
[mysqld_safe]
open_files_limit=4294967295
```

The option value is passed to `ulimit -n`. Note that you need to start `mysqld_safe` as root for this to work properly. However, you can't currently set this to `unlimited`. See MDEV-18410 about that.

When `mysqld_safe` starts `mysqld`, it also uses this option to set the value of the `open_files_limit` system variable for `mysqld`.

Configuring the Core File Size

When using `mysqld_safe`, if you would like to enable core dumps, the system's core file size limit can be changed by providing the `--core-file-size` option either on the command-line or in an option file. For example:

```
[mysqld_safe]
core_file_size=unlimited
```

The option value is passed to `ulimit -c`. Note that you need to start `mysqld_safe` as root for this to work properly.

Configuring MariaDB to Write the Error Log to Syslog

When using `mysqld_safe`, if you would like to redirect the error log to the syslog, then that can easily be done by using the `--syslog` option. `mysqld_safe` redirects two types of log messages to the syslog--its own log messages, and log messages for `mysqld`.

- `mysqld_safe` configures its own log messages to go to the `daemon` syslog facility. The log level for these messages is either `notice` or `error`, depending on the specific type of log message. The default tag is `mysqld_safe`.
- `mysqld_safe` also configures the log messages for `mysqld` to go to the `daemon` syslog facility. The log level for these messages is `error`. The default tag is `mysqld`.

Sometimes it can be helpful to add a suffix to the syslog tag, such as if you are running multiple instances of MariaDB on the same host. To add a suffix to each syslog tag, use the `--syslog-tag` option.

Specifying mysqld

By default, `mysqld_safe` tries to start an executable named `mysqld`.

You can also specify another executable for `mysqld_safe` to start instead of `mysqld` by providing the `--mysqld` or `--mysqld-version` options either on the command-line or in an option file.

By default, it will look for `mysqld` in the following locations in the following order:

- `$BASEDIR/libexec/mysqld`
- `$BASEDIR/sbin/mysqld`
- `$BASEDIR/bin/mysqld`
- `$PWD/bin/mysqld`
- `$PWD/libexec/mysqld`
- `$PWD/sbin/mysqld`
- `@libexecdir@/mysql`

Where `$BASEDIR` is set by the `--basedir` option, `$PWD` is the current working directory where `mysqld_safe` was invoked, and `@libexecdir@` is set at compile-time by the `INSTALL_BINDIR` option for `cmake`.

You can also specify where the executable is located by providing the `--ledir` option either on the command-line or in an option file.

Specifying datadir

By default, `mysqld_safe` will look for the `datadir` in the following locations in the following order:

- `$BASEDIR/data/mysql`
- `$BASEDIR/data`
- `$BASEDIR/var/mysql`
- `$BASEDIR/var`
- `@localstatedir@`

Where `$BASEDIR` is set by the `--basedir` option, and `@localstatedir@` is set at compile-time by the `INSTALL_MYSQLDATADIR` option for `cmake`.

You can also specify where the `datadir` is located by providing the `--datadir` option either on the command-line or in an option file.

Logging

When you use `mysqld_safe` to start `mysqld`, `mysqld_safe` logs to the same destination as `mysqld`.

`mysqld_safe` has several log-related options:

- `--syslog` : Write error messages to syslog on systems that support the logger program.
- `--skip-syslog` : Do not write error messages to syslog. Messages are written to the default error log file (host_name.err in the data directory), or to a named file if the `--log-error` option is given.
- `--log-error=file_name` : Write error messages to the named error file.

If none of these options is provided, then the default is `--skip-syslog` .

If `--syslog` and `--log-error` are both provided, then a warning is issued and `--log-error` takes precedence.

`mysqld_safe` also writes notices to `stdout` and errors to `stderr` .

Editing mysqld_safe

`mysqld_safe` is a `sh` script, so if you need to change its behavior, then it can easily be edited. However, you should not normally edit the script. A lot of behavior can be changed by providing options either on the command-line or in an option file.

If you do edit `mysqld_safe` , then you should be aware of the fact that a package upgrade can overwrite your changes. If you would like to preserve your changes, be sure to have a backup.

NetWare

On NetWare, mysqld_safe is a NetWare Loadable Module (NLM) that is ported from the original Unix shell script. It starts the server as follows:

1. Runs a number of system and option checks.
2. Runs a check on MyISAM tables.
3. Provides a screen presence for the MariaDB server.
4. Starts mysqld, monitors it, and restarts it if it terminates in error.
5. Sends error messages from mysqld to the host_name.err file in the data directory.
6. Sends mysqld_safe screen output to the host_name.safe file in the data directory.

1.3.15 mysqladmin

2.1.6.10 Switching Between Different Installed MariaDB Versions

Contents

This article is about managing many different installed MariaDB versions and running them one at a time. This is useful when doing benchmarking, testing, or for when developing different MariaDB versions.

This is most easily done using the tar files from mariadb.org/download/.

Stopping a pre-installed MySQL/MariaDB from interfering with your tests

If MySQL/MariaDB is already installed and running, you have two options:

1. Use test MariaDB servers with a different port & socket.
   - In this case you are probably best off creating a specific section for MariaDB in your `~/.my.cnf` file.
2. Stop mysqld with `/etc/rc.d/mysql stop` or `mariadb-admin shutdown` .

Note that you don't have to uninstall or otherwise remove MySQL!

How to create a binary distribution (tar file)

Here is a short description of how to generate a tar file from a source distribution. If you have downloaded a binary tar file, you can skip this section.

The steps to create a binary tar file are:

- Decide where to put the source. A good place is under `/usr/local/src/mariadb-5.#`.
- Get the source
- Compile the source
- Create the binary tar ball.

You will then be left with a tar file named something like: `mariadb-11.0.1-MariaDB-linux-x86_64.tar.gz`

Creating a directory structure for the different installations

Install the binary tar files under `/usr/local/` with the following directory names (one for each MariaDB version you want to use), for example:

- `mariadb-10.5`
- `mariadb-10.6`
- `mariadb-10.11`
- `mariadb-11.0`
- `mariadb-11.1`

The above assumes you are just testing major versions of MariaDB. If you are testing specific versions, use directory names like `mariadb-11.0.1`

With the directories in place, create a sym-link named `mariadb` which points at the `mariadb-XXX` directory you are currently testing. When you want to switch to testing a different version, just update the sym-link.

Example:

```
cd /usr/local
tar xfz /tmp/mariadb-11.0.1-linux-systemd-x86_64.tar.gz
mv -vi mariadb-11.0.1-MariaDB-systemd-linux-x86_64 mariadb-11.0
ln -vs mariadb-11.0 mariadb
```

Setting up the data directory

When setting up the data directory, you have the option of either using a shared database directory or creating a unique database directory for each server version. For testing, a common directory is probably easiest. Note that you can only have one `mysqld` server running against one data directory.

Setting up a common data directory

The steps are:

1. Create the `mysql` system user if you don't have it already! (On Linux you do it with the `useradd` command).
2. Create the directory (we call it `mariadb-data` in the example below) or add a symlink to a directory which is in some other place.
3. Create the `mysql` permission tables with mariadb-install-db

```
cd /usr/local/
mkdir mariadb-data
cd mariadb
./bin/mariadb-install-db --no-defaults --datadir=/usr/local/mariadb-data
chown -R mysql mariadb-data mariadb-data/*
```

The reason to use `--no-defaults` is to ensure that we don't inherit incorrect options from some old my.cnf.

Setting up different data directories

To create a different `data` directories for each installation:

```
cd mariadb
./scripts/mariadb-install-db --no-defaults
chown -R mysql mariadb-data mariadb-data/*
```

This will create a directory `data` inside the current directory.

If you want to use another disk you should do:

```
cd mariadb
ln -s path-to-empty-directory-for-data data
./scripts/mariadb-install-db --no-defaults --datadir=./data
chown -R mysql mariadb-data mariadb-data/*
```

Running a MariaDB server

The normal steps are:

```
rm mariadb
ln -s mariadb-# mariadb
cd mariadb
./bin/mysqld_safe --no-defaults --datadir=/usr/local/mariadb-data &
```

Setting up a .my.cnf file for running multiple MariaDB main versions

If you are going to start/stop MariaDB a lot of times, you should create a `~/.my.cnf` file for the common options you are using.

The following example shows how to use a non-standard TCP-port and socket (to not interfere with a main MySQL/MariaDB server) and how to setup different options for each main server:

```
[client-server]
socket=/tmp/mysql.sock
port=3306
[mysqld]
datadir=/usr/local/mariadb-data

[mariadb-11.0]
# Options for MariaDB 11.0
[mariadb-11.1]
# Options for MariaDB 11.1
```

If you create an `~/.my.cnf` file, you should start `mysqld` with `--defaults-file=~/.my.cnf` instead of `--no-defaults` in the examples above.

2.1.6.11 Specifying Permissions for Schema (Data) Directories and Tables

Default File Permissions

By default MariaDB uses the following permissions for files and directories:

Object Type	Default Mode	Default Permissions
Files	`0660`	`-rw-rw----`
Directories	`0700`	`drwx------`

Configuring File Permissions with Environment Variables

You can configure MariaDB to use different permissions for files and directories by setting the following environment variables before you start the server:

Object Type	Environment Variable
Files	`UMASK`
Directories	`UMASK_DIR`

In other words, if you would run the following in a shell:

```
export UMASK=0640
export UMASK_DIR=0750
```

These environment variables do not set the umask. They set the default file system permissions. See MDEV-23058 for more information.

Configuring File Permissions with systemd

If your server is started by `systemd`, then there is a specific way to configure the umask. See Systemd: Configuring the umask for more information.

2.1.6.12 mysqld_multi

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadbd-multi` is a symlink to `mysqld_multi`.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mariadbd-multi` is the name of the server, with `mysqld_multi` a symlink.

Before using mysqld_multi be sure that you understand the meanings of the options that are passed to the mysqld servers and why you would want to have separate mysqld processes. Beware of the dangers of using multiple mysqld servers with the same data directory. Use separate data directories, unless you know what you are doing. Starting multiple servers with the same data directory does not give you extra performance in a threaded system.

Contents

The `mysqld_multi` startup script is in MariaDB distributions on Linux and Unix. It is a wrapper that is designed to manage several `mysqld` processes running on the same host. In order for multiple `mysqld` processes to work on the same host, these processes must:

- Use different Unix socket files for local connections.
- Use different TCP/IP ports for network connections.
- Use different data directories.
- Use different process ID files (specified by the `--pid-file` option) if using `mysqld_safe` to start `mysqld`.

`mysqld_multi` can start or stop servers, or report their current status.

Using mysqld_multi

The command to use `mysqld_multi` and the general syntax is:

```
mysqld_multi [options] {start|stop|report} [GNR[,GNR] ...]
```

`start`, `stop`, and `report` indicate which operation to perform.

You can specify which servers to perform the operation on by providing one or more `GNR` values. `GNR` refers to an option

group number, and it is explained more in the option groups section below. If there is no `GNR` list, then `mysqld_multi` performs the operation for all `GNR` values found in its option files.

Multiple `GNR` values can be specified as a comma-separated list. `GNR` values can also be specified as a range by separating the numbers by a dash. There must not be any whitespace characters in the `GNR` list.

For example:

This command starts a single server using option group `[mysqld17]`:

```
mysqld_multi start 17
```

This command stops several servers, using option groups `[mysqld8]` and `[mysqld10]` through `[mysqld13]`:

```
mysqld_multi stop 8,10-13
```

Options

`mysqld_multi` supports the following options:

Option	Description
`--example`	Give an example of a config file with extra information.
`--help`	Display help and exit.
`--log=filename`	Specify the path and name of the log file. If the file exists, log output is appended to it.
`--mysqladmin=prog_name`	The mariadb-admin binary to be used to stop servers. Can be given within groups `[mysqld#]`.
`--mysqld=prog_name`	The mysqld binary to be used. Note that you can also specify mysqld_safe as the value for this option. If you use mysqld_safe to start the server, you can include the `mysqld` or `ledir` options in the corresponding `[mysqldN]` option group. These options indicate the name of the server that mysqld_safe should start and the path name of the directory where the server is located. Example: `[mysqld38]` `mysqld = mysqld-debug` `ledir = /opt/local/mysql/libexec`.
`--no-log`	Print to stdout instead of the log file. By default the log file is turned on.
`--password=password`	The password of the MariaDB account to use when invoking mariadb-admin. Note that the password value is not optional for this option, unlike for other MariaDB programs.
`--silent`	Silent mode; disable warnings.
`--tcp-ip`	Connect to the MariaDB server(s) via the TCP/IP port instead of the UNIX socket. This affects stopping and reporting. If a socket file is missing, the server may still be running, but can be accessed only via the TCP/IP port. By default connecting is done via the UNIX socket. This option affects stop and report operations.
`--user=username`	The user name of the MariaDB account to use when invoking mariadb-admin.
`--verbose`	Be more verbose.
`--version`	Display version information and exit.
`--wsrep-new-cluster`	Bootstrap a cluster. Added in MariaDB 10.1.15.

Option Files

In addition to reading options from the command-line, `mysqld_multi` can also read options from option files. If an unknown option is provided to `mysqld_multi` in an option file, then it is ignored.

The following options relate to how MariaDB command-line tools handles option files. They must be given as the first argument on the command-line:

Option	Description

`--print-defaults`	Print the program argument list and exit.
`--no-defaults`	Don't read default options from any option file.
`--defaults-file=#`	Only read default options from the given file #.
`--defaults-extra-file=#`	Read this file after the global files are read.
`--defaults-group-suffix=#`	In addition to the default option groups, also read option groups with this suffix.

Option Groups

`mysqld_safe` reads options from the following option groups from option files:

Group	Description
`[mysqld_multi]`	Options read by `mysqld_multi`, which includes both MariaDB Server and MySQL Server.

`mysqld_multi` also searches option files for option groups with names like `[mysqldN]`, where `N` can be any positive integer. This number is referred to in the following discussion as the option group number, or `GNR`:

Group	Description
`[mysqldN]`	Options read by a `mysqld` instance managed by `mysqld_multi`, which includes both MariaDB Server and MySQL Server. The `N` refers to the instance's `GNR`.

`GNR` values distinguish option groups from one another and are used as arguments to `mysqld_multi` to specify which servers you want to start, stop, or obtain a status report for. The `GNR` value should be the number at the end of the option group name in the option file. For example, the `GNR` for an option group named `[mysqld17]` is `17`.

Options listed in these option groups are the same that you would use in the regular server option groups used for configuring `mysqld`. However, when using multiple servers, it is necessary that each one use its own value for options such as the Unix socket file and TCP/IP port number.

The `[mysqld_multi]` option group can be used for options that are needed for `mysqld_multi` itself. `[mysqldN]` option groups can be used for options passed to specific `mysqld` instances.

The regular server option groups can also be used for common options that are read by all instances:

Group	Description
`[mysqld]`	Options read by `mysqld`, which includes both MariaDB Server and MySQL Server.
`[server]`	Options read by MariaDB Server.
`[mysqld-X.Y]`	Options read by a specific version of `mysqld`, which includes both MariaDB Server and MySQL Server. For example, `[mysqld-5.5]`.
`[mariadb]`	Options read by MariaDB Server.
`[mariadb-X.Y]`	Options read by a specific version of MariaDB Server.
`[client-server]`	Options read by all MariaDB client programs and the MariaDB Server. This is useful for options like socket and port, which is common between the server and the clients.
`[galera]`	Options read by a galera-capable MariaDB Server. Available on systems compiled with Galera support.

For an example of how you might set up an option file, use this command:

```
mysqld_multi --example
```

Authentication and Privileges

Make sure that the MariaDB account used for stopping the `mysqld` processes (with the mariadb-admin utility) has the same user name and password for each server. Also, make sure that the account has the `SHUTDOWN` privilege. If the servers that you want to manage have different user names or passwords for the administrative accounts, you might want to create an account on each server that has the same user name and password. For example, you might set up a common `multi_admin` account by executing the following commands for each server:

```
shell> mysql -u root -S /tmp/mysql.sock -p
Enter password:
mysql> GRANT SHUTDOWN ON *.*
 -> TO ´multi_admin´@´localhost´ IDENTIFIED BY ´multipass´;
```

Change the connection parameters appropriately when connecting to each one. Note that the host name part of the account name must allow you to connect as `multi_admin` from the host where you want to run `mysqld_multi` .

User Account

Make sure that the data directory for each server is fully accessible to the Unix account that the specific `mysqld` process is started as. If you run the `mysqld_multi` script as the Unix `root` account, and if you want the `mysqld` process to be started with another Unix account, then you can use use the `--user` option with `mysqld` . If you specify the `--user` option in an option file, and if you did not run the `mysqld_multi` script as the Unix `root` account, then it will just log a warning and the `mysqld` processes are started under the original Unix account.

Do not run the `mysqld` process as the Unix `root` account, unless you know what you are doing.

Example

The following example shows how you might set up an option file for use with mysqld_multi. The order in which the mysqld programs are started or stopped depends on the order in which they appear in the option file. Group numbers need not form an unbroken sequence. The first and fifth [mysqldN] groups were intentionally omitted from the example to illustrate that you can have “gaps” in the option file. This gives you more flexibility.

```
# This file should probably be in your home dir (~/.my.cnf)
# or /etc/my.cnf
# Version 2.1 by Jani Tolonen
[mysqld_multi]
mysqld     = /usr/local/bin/mysqld_safe
mysqladmin = /usr/local/bin/mysqladmin
user       = multi_admin
password   = multipass
[mysqld2]
socket     = /tmp/mysql.sock2
port       = 3307
pid-file   = /usr/local/mysql/var2/hostname.pid2
datadir    = /usr/local/mysql/var2
language   = /usr/local/share/mysql/english
user       = john
[mysqld3]
socket     = /tmp/mysql.sock3
port       = 3308
pid-file   = /usr/local/mysql/var3/hostname.pid3
datadir    = /usr/local/mysql/var3
language   = /usr/local/share/mysql/swedish
user       = monty
[mysqld4]
socket     = /tmp/mysql.sock4
port       = 3309
pid-file   = /usr/local/mysql/var4/hostname.pid4
datadir    = /usr/local/mysql/var4
language   = /usr/local/share/mysql/estonia
user       = tonu
[mysqld6]
socket     = /tmp/mysql.sock6
port       = 3311
pid-file   = /usr/local/mysql/var6/hostname.pid6
datadir    = /usr/local/mysql/var6
language   = /usr/local/share/mysql/japanese
user       = jani
```

2.1.6.13 launchd

In MacOS, create a file called /Library/LaunchDaemons/com.mariadb.server.plist with the following contents (edit to suit):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>Label</key> <string>com.mariadb.server</string>
  <key>KeepAlive</key><true/>
  <key>RunAtLoad</key><true/>
  <key>LaunchOnlyOnce</key><false/>
  <key>ExitTimeOut</key><integer>600</integer>
  <key>WorkingDirectory</key><string>/usr/local/var</string>
  <key>Program</key><string>/usr/local/bin/mysqld</string>
  <key>ProgramArguments</key>
  <array>
    <string>/usr/local/bin/mysqld</string>
    <string>--user=_mysql</string>
    <string>--basedir=/usr/local/opt/mariadb</string>
    <string>--plugin-dir=/usr/local/opt/mariadb/lib/plugin</string>
    <string>--datadir=/usr/local/var/mysql</string>
    <string>--log-error=/usr/local/var/mysql/Data-Server.local.err</string>
    <string>--pid-file=/usr/local/var/mysql/Data-Server.local.pid</string>
    <string>--sql-
mode=ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION</string>
  </array>
</dict>
</plist>
```

Then from a shell, run `launchctl load /Library/LaunchDaemons/com.mariadb.server.plist` and MariaDB will run immediately, and also upon reboot.

2.1.6.14 systemd

`systemd` is a `sysVinit` replacement that is the default service manager on the following Linux distributions:

- RHEL 7 and above
- CentOS 7 and above
- Fedora 15 and above
- Debian 8 and above
- Ubuntu 15.04 and above
- SLES 12 and above
- OpenSUSE 12.2 and above

MariaDB's `systemd` unit file is included in the server packages for RPMs and DEBs. It is also included in certain binary tarballs.

The service name is `mariadb.service` .

Installing & Starting MariaDB

When installing MariaDB server rpm / dep package, it will automatically run the mariadb-install-db script, that creates the initial databases and users.

When MariaDB is started with the `systemd` unit file, it directly starts the `mariadbd` process as the `mysql` user. Unlike with `sysVinit` , the `mariadbd` process is not started with `mysqld_safe` . As a consequence, options will not be read from the `[mysqld_safe]` option group from option files.

Contents

Contents of the MariaDB Service's Unit File

The contents of the `mariadb.service` file can be examined with `systemctl show mariadb.service`.

Interacting with the MariaDB Server Process

The service can be interacted with by using the `systemctl` command.

Starting the MariaDB Server Process on Boot

MariaDB's `systemd` service can be configured to start at boot by executing the following:

```
sudo systemctl enable mariadb.service
```

Starting the MariaDB Server Process

MariaDB's `systemd` service can be started by executing the following:

```
sudo systemctl start mariadb.service
```

MariaDB's `systemd` unit file has a default startup timeout of about 90 seconds on most systems. If certain startup tasks, such as crash recovery, take longer than this default startup timeout, then `systemd` will assume that `mariadbd` has failed to startup, which causes `systemd` to kill the `mariadbd` process. To work around this, you can reconfigure the MariaDB `systemd` unit to have an infinite timeout.

Note that systemd 236 added the `EXTEND_TIMEOUT_USEC` environment variable that allows services to extend the startup timeout during long-running processes. Starting with MariaDB 10.1.33, MariaDB 10.2.15, and MariaDB 10.3.6, on systems with systemd versions that support it, MariaDB uses this feature to extend the startup timeout during certain startup processes that can run long. Therefore, if you are using `systemd` 236 or later, then you should not need to manually override `TimeoutStartSec`, even if your startup tasks, such as crash recovery, run for longer than the configured value. See MDEV-14705 for more information.

Stopping the MariaDB Server Process

MariaDB's `systemd` service can be stopped by executing the following:

```
sudo systemctl stop mariadb.service
```

Restarting the MariaDB Server Process

MariaDB's `systemd` service can be restarted by executing the following:

```
sudo systemctl restart mariadb.service
```

Checking the Status of the MariaDB Server Process

The status of MariaDB's `systemd` service can be obtained by executing the following:

```
sudo systemctl status mariadb.service
```

Interacting with Multiple MariaDB Server Processes

A `systemd` template unit file with the name `mariadb@.service` is installed in `INSTALL_SYSTEMD_UNITDIR` on some systems. See Locating the MariaDB Service's Unit File to see what directory that refers to on each distribution.

This template unit file allows you to interact with multiple MariaDB instances on the same system using the same template unit file. When you interact with a MariaDB instance using this template unit file, you have to provide an instance name as a suffix. For example, the following command tries to start a MariaDB instance with the name `node1`:

```
sudo systemctl start mariadb@node1.service
```

MariaDB's build system cannot include the `mariadb@.service` template unit file in RPM packages on platforms that have `cmake` versions older than 3.3.0, because these `cmake` versions have a bug that causes it to encounter errors when packaging a file in RPMs if the file name contains the `@` character. MariaDB's RHEL 7 and CentOS 7 RPM build hosts only got a new enough `cmake` version starting with MariaDB 10.1.39, MariaDB 10.2.23, and MariaDB 10.3.14. To use this functionality on a MariaDB version that does not have the file, you can copy the file from a package that does have the file.

Default configuration of Multiple Instances in 10.4 and Later

`systemd` will also look for an option file for a specific MariaDB instance based on the instance name.

It will use the `.%I` as the custom option group suffix that is appended to any server option group, in any configuration file included by default.

In all distributions, the `%I` is the MariaDB instance name. In the above `node1` case, it would use the option file at the path `/etc/mynode1.cnf`.

When using multiple instances, each instance will of course also need their own `datadir`, `socket` and, `port` (unless `skip_networking` is specified). As `mariadb-install-db#option-groups` reads the same sections as the server, and ExecStartPre= `run mariadb-install-db` within the service, the instances are autocreated if there is sufficient priviledges.

To use a 10.3 configuration in 10.4 or later and the following customisation in the editor after running `sudo systemctl edit mariadb@.service`:

```
[Unit]
ConditionPathExists=

[Service]
Environment='MYSQLD_MULTI_INSTANCE=--defaults-file=/etc/my%I.cnf'
```

Custom configuration of Multiple Instances in 10.4 and Later

Because users may want to do many various things with their multiple instances, we've provided a way to let the user define how they wish their multiple instances to run. The systemd environment variable `MYSQLD_MULTI_INSTANCE` can be set to anything that mariadbd and mariadb-install-db will recognise.

A hosting environment where each user has their own instance may look like (with `sudo systemctl edit mariadb@.service`):

```
[Service]
ProtectHome=false
Environment='MYSQLD_MULTI_INSTANCE=--defaults-file=/home/%I/my.cnf \
                        --user=%I \
                        --socket=/home/%I.sock \
                        --datadir=/home/%I/mariadb_data \
                        --skip-networking'
```

Here the instance name is the unix user of the service.

Configuring Multiple Instances in 10.3 and Earlier

`systemd` will also look for an option file for a specific MariaDB instance based on the instance name. By default, it will look for the option file in a directory defined at build time by the `INSTALL_SYSCONF2DIR` option provided to `cmake`.

For example, on RHEL, CentOS, Fedora, and other similar Linux distributions, `INSTALL_SYSCONF2DIR` is defined as `/etc/my.cnf.d/`, so it will look for an option file that matches the format:

- `/etc/my.cnf.d/my%I.cnf`

And on Debian, Ubuntu, and other similar Linux distributions, `INSTALL_SYSCONF2DIR` is defined as `/etc/mysql/conf.d//`, so it will look for an option file that matches the format:

- `/etc/mysql/conf.d/my%I.cnf`

In all distributions, the `%I` is the MariaDB instance name. In the above `node1` case, it would use the option file at the path `/etc/my.cnf.d/mynode1.cnf` for RHEL-like distributions and `/etc/mysql/conf.d/mynode1.cnf` for Debian-like distributions.

When using multiple instances, each instance will of course also need their own `datadir`. See mariadb-install-db for information on how to initialize the `datadir` for additional MariaDB instances.

Systemd and Galera Cluster

Bootstrapping a New Cluster

When using Galera Cluster with systemd, the first node in a cluster has to be started with `galera_new_cluster`. See Getting Started with MariaDB Galera Cluster: Bootstrapping a New Cluster for more information.

Recovering a Node's Cluster Position

When using Galera Cluster with systemd, a node's position in the cluster can be recovered with `galera_recovery`. See Getting Started with MariaDB Galera Cluster: Determining the Most Advanced Node for more information.

SSTs and Systemd

MariaDB's `systemd` unit file has a default startup timeout of about 90 seconds on most systems. If an SST takes longer than this default startup timeout on a joiner node, then `systemd` will assume that `mariadbd` has failed to startup, which

causes `systemd` to kill the `mariadbd` process on the joiner node. To work around this, you can reconfigure the MariaDB `systemd` unit to have an infinite timeout. See Introduction to State Snapshot Transfers (SSTs): SSTs and Systemd for more information.

Note that systemd 236 added the `EXTEND_TIMEOUT_USEC` environment variable that allows services to extend the startup timeout during long-running processes. Starting with MariaDB 10.1.35, MariaDB 10.2.17, and MariaDB 10.3.8, on systems with systemd versions that support it, MariaDB uses this feature to extend the startup timeout during long SSTs. Therefore, if you are using `systemd` 236 or later, then you should not need to manually override `TimeoutStartSec`, even if your SSTs run for longer than the configured value. See MDEV-15607 for more information.

Configuring the Systemd Service

You can configure MariaDB's `systemd` service by creating a "drop-in" configuration file for the `systemd` service. On most systems, the `systemd` service's directory for "drop-in" configuration files is `/etc/systemd/system/mariadb.service.d/`. You can confirm the directory and see what "drop-in" configuration files are currently loaded by executing:

```
$ sudo systemctl status mariadb.service
● mariadb.service - MariaDB 10.1.37 database server
   Loaded: loaded (/usr/lib/systemd/system/mariadb.service; enabled; vendor preset: disabled)
  Drop-In: /etc/systemd/system/mariadb.service.d
           └─migrated-from-my.cnf-settings.conf, timeoutstartsec.conf
...
```

If you want to configure the `systemd` service, then you can create a file with the `.conf` extension in that directory. The configuration option(s) that you would like to change would need to be placed in an appropriate section within the file, usually `[Service]`. If a `systemd` option is a list, then you may need to set the option to empty before you set the replacement values. For example:

```
[Service]

ExecStart=
ExecStart=/usr/bin/numactl --interleave=all  /usr/sbin/mariadbd $MYSQLD_OPTS
$_WSREP_NEW_CLUSTER $_WSREP_START_POSITION
```

After any configuration change, you will need to execute the following for the change to go into effect:

```
sudo systemctl daemon-reload
```

Useful Systemd Options

Useful `systemd` options are listed below. If an option is equivalent to a common `mysqld_safe` option, then that is also listed. Use `systemctl edit mariadb.service` to create the systemd option under a `[Service]` section header.

mysqld_safe option	systemd option	Comments
no option	`ProtectHome=false`	If any MariaDB files are in /home/
no option	`PrivateDevices=false`	If any MariaDB storage references raw block devices
no option	`ProtectSystem=`	If any MariaDB write any files to anywhere under /boot, /usr or /etc
no option	`TimeoutStartSec={time}`	Service startup timeout. See Configuring the Systemd Service Timeout.
no option (see MDEV-9264)	`OOMScoreAdjust={priority}`	e.g. -600 to lower priority of OOM killer for mariadbd
`open-files-limit`	`LimitNOFILE={limit}`	Limit on number of open files. See Configuring the Open Files Limit.

core-file-size	LimitCORE={size}	Limit on core file size. Useful when enabling core dumps. See Configuring the Core File Size.
	LimitMEMLOCK={size} or infinity	Limit on how much can be locked in memory. Useful when large-pages or memlock is used
nice	Nice={nice value}	
syslog	StandardOutput=syslog	See Configuring MariaDB to Write the Error Log to Syslog.
	StandardError=syslog	
	SyslogFacility=daemon	
	SyslogLevel=err	
syslog-tag	SyslogIdentifier	
flush-caches	ExecStartPre=/usr/bin/sync	
	ExecStartPre=/usr/sbin/sysctl -q -w vm.drop_caches=3	
malloc-lib	Environment=LD_PRELOAD=/path/to/library	
numa-interleave	NUMAPolicy=interleave	from systemd v243 onwards
	or: ExecStart=/usr/bin/numactl --interleave=all /usr/sbin/mariadbd $MYSQLD_OPTS $_WSREP_NEW_CLUSTER $_WSREP_START_POSITION	prepending ExecStart=/usr/bin/numactl --interleave=all to existing ExecStart setting
no-auto-restart	Restart={exit-status}	

Note: the systemd manual contains the official meanings for these options. The manual also lists considerably more options than the ones listed above.

There are other options and the mariadb-service-convert script will attempt to convert these as accurately as possible.

Configuring the Systemd Service Timeout

MariaDB's systemd unit file has a default startup timeout of about 90 seconds on most systems. If a service startup takes longer than this default startup timeout, then systemd will assume that mariadbd has failed to startup, which causes systemd to kill the mariadbd process. To work around this, it can be changed by configuring the TimeoutStartSec option for the systemd service.

A similar problem can happen when stopping the MariaDB service. Therefore, it may also be a good idea to set TimeoutStopSec.

For example, you can reconfigure the MariaDB systemd service to have an infinite timeout by executing one of the following commands:

If you are using systemd 228 or older, then you can execute the following to set an infinite timeout:

```
sudo systemctl edit mariadb.service

[Service]

TimeoutStartSec=0
TimeoutStopSec=0
```

Systemd 229 added the infinity option, so if you are using systemd 229 or later, then you can execute the following to set an infinite timeout:

```
sudo systemctl edit mariadb.service

[Service]

TimeoutStartSec=infinity
TimeoutStopSec=infinity
```

Note that systemd 236 added the EXTEND_TIMEOUT_USEC environment variable that allows services to extend the startup timeout during long-running processes. On systems with systemd versions that support it, MariaDB uses this feature to extend the startup timeout during certain startup processes that can run long.

Configuring the Open Files Limit

When using `systemd`, rather than setting the open files limit by setting the `open-files-limit` option for `mysqld_safe` or the `open_files_limit` system variable, the limit can be changed by configuring the `LimitNOFILE` option for the MariaDB `systemd` service. The default is set to `LimitNOFILE=16364` in `mariadb.service`.

For example, you can reconfigure the MariaDB `systemd` service to have a larger limit for open files by executing the following commands:

```
sudo systemctl edit mariadb.service

[Service]

LimitNOFILE=infinity
```

An important note is that setting `LimitNOFILE=infinity` doesn't actually set the open file limit to *infinite*.

In `systemd` 234 and later, setting `LimitNOFILE=infinity` actually sets the open file limit to the value of the kernel's `fs.nr_open` parameter. Therefore, in these `systemd` versions, you may have to change this parameter's value.

The value of the `fs.nr_open` parameter can be changed permanently by setting the value in `/etc/sysctl.conf` and restarting the server.

The value of the `fs.nr_open` parameter can be changed temporarily by executing the `sysctl` utility. For example:

```
sudo sysctl -w fs.nr_open=1048576
```

In `systemd` 233 and before, setting `LimitNOFILE=infinity` actually sets the open file limit to `65536`. See systemd issue #6559 for more information. Therefore, in these `systemd` versions, it is not generally recommended to set `LimitNOFILE=infinity`. Instead, it is generally better to set `LimitNOFILE` to a very large integer. For example:

```
sudo systemctl edit mariadb.service

[Service]
LimitNOFILE=1048576
```

Configuring the Core File Size

When using `systemd`, if you would like to enable core dumps, rather than setting the core file size by setting the `core-file-size` option for `mysqld_safe`, the limit can be changed by configuring the `LimitCORE` option for the MariaDB `systemd` service. For example, you can reconfigure the MariaDB `systemd` service to have an infinite size for core files by executing the following commands:

```
sudo systemctl edit mariadb.service

[Service]
LimitCORE=infinity
```

Configuring MariaDB to Write the Error Log to Syslog

When using `systemd`, if you would like to redirect the error log to the syslog, then that can easily be done by doing the following:

- Ensure that `log_error` system variable is **not** set.

- Set `StandardOutput=syslog`.
- Set `StandardError=syslog`.
- Set `SyslogFacility=daemon`.
- Set `SysLogLevel=err`.

For example:

```
sudo systemctl edit mariadb.service

[Service]

StandardOutput=syslog
StandardError=syslog
SyslogFacility=daemon
SysLogLevel=err
```

If you have multiple instances of MariaDB, then you may also want to set `SyslogIdentifier` with a different tag for each instance.

Configuring LimitMEMLOCK

If using --memlock or the iouring in InnoDB in MariaDB 10.6 with a Linux Kernel version < 5.12, you will need to raise the LimitMEMLOCK limit.

```
sudo systemctl edit mariadb.service

[Service]

LimitMEMLOCK=2M
```

Note: Prior to MariaDB 10.1.10, the --memlock option could not be used with the MariaDB `systemd` service.

Configuring Access to Home Directories

MariaDB's systemd unit file restricts access to `/home`, `/root`, and `/run/user` by default. This restriction can be overridden by setting the ProtectHome option to `false` for the MariaDB `systemd` service. This is done by creating a "drop-in" directory `/etc/systemd/system/mariadb.service.d/` and in it a file with a `.conf` suffix that contains the `ProtectHome=false` directive.

You can reconfigure the MariaDB `systemd` service to allow access to `/home` by executing the following commands:

```
sudo systemctl edit mariadb.service

[Service]

ProtectHome=false
```

Configuring the umask

When using `systemd`, the default file permissions of `mariadbd` can be set by setting the `UMASK` and `UMASK_DIR` environment variables for the `systemd` service. For example, you can configure the MariaDB `systemd` service's umask by executing the following commands:

```
sudo systemctl edit mariadb.service

[Service]

Environment="UMASK=0750"
Environment="UMASK_DIR=0750"
```

These environment variables do not set the umask. They set the default file system permissions. See MDEV-23058 for more information.

Keep in mind that configuring the umask this way will only affect the permissions of files created by the `mariadbd`

process that is managed by `systemd`. The permissions of files created by components that are not managed by `systemd`, such as mariadb-install-db, will not be affected.

See Specifying Permissions for Schema (Data) Directories and Tables for more information.

Configuring the data directory

When doing a standard binary tarball install the datadir will be under /usr/local/data. The default systemd service file makes the whole /usr directory tree write protected however.

So when just copying the distributed service file a tarball install will not start up, complaining e.g. about

```
[Warning] Can't create test file /usr/local/.../data/ubuntu-focal.lower-test
[ERROR] mariadbd: File '/usr/local/.../data/aria_log_control' not found (Errcode: 30 "Read-only
file system")
[ERROR] mariadbd: Got error 'Can't open file' when trying to use aria control file
'/usr/local/.../data/aria_log_control'
```

So when using a data directory under /usr/local that specific directory needs to be made writable for the service using the ReadWritePaths setting:

```
sudo systemctl edit mariadb.service

[Service]
ReadWritePaths=/usr/local/mysql/data
```

Systemd Socket Activation

MariaDB starting with 10.6.0
MariaDB can use systemd's socket activation.

This is an on-demand service for MariaDB that will activate when required.

Systemd socket activation uses a `mariadb.socket` definition file to define a set of UNIX and TCP sockets. Systemd will listen on these sockets, and when they are connected to, systemd will start the `mariadb.service` and hand over the socket file descriptors for MariaDB to process the connection.

MariaDB remains running at this point and will have all sockets available and process connections exactly like it did before 10.6.

When MariaDB is shut down, the systemd `mariadb.socket` remains active, and a new connection will restart the `mariadb.service`.

Using Systemd Socket Activation

To use MariaDB systemd socket activation, instead of enabling/starting `mariadb.service`, `mariadb.socket` is used instead.

So the following commands work exactly like the `mariadb.service` equivalents.

```
systemctl start mariadb.socket
systemctl enable mariadb.socket
```

These files alone only contain the UNIX and TCP sockets and basic network connection information to which will be listening for connections. `@mariadb` is a UNIX abstract socket, which means it doesn't appear on the filesystem. Connectors based on MariaDB Connector/C will be able to connect with these by using the socket name directly, provided the higher level implementation doesn't try to test for the file's existence first. Some connectors like PHP use mysqlnd that is a pure PHP implementation and as such will only be able to connect to on filesystem UNIX sockets.

With systemd activated sockets there is only a file descriptor limit on the number of listening sockets that can be created.

When to Use Systemd Socket Activation

A common use case for systemd socket activated MariaDB is when there needs to be a quick boot up time. MariaDB needs to be ready to run, but it doesn't need to be running.

The ideal use case for systemd socket activation for MariaDB is for infrastructure providers running many multiple instances of MariaDB, where each instance is dedicated for a user.

Downsides to Using Systemd Socket Activiation

From the time the connection occurs, the client is going to be waiting until MariaDB has fully initialized before MariaDB can process the awaiting connection. If MariaDB was previously hard shutdown and needs to perform an extensive InnoDB rollback, then the activation time may be larger than the desired wait time of the client connection.

Configuring Systemd Socket Activation

When MariaDB is run under systemd socket activation, the usual socket , port, and backlog system variables are ignored, as these settings are contained within the systemd socket definition file.

There is no configuration required in MariaDB to use MariaDB under socket activation.

The systemd options available are from the `systemd documentation`, however `ListenStream` and `BackLog` would be the most common configuration options.

As MariaDB isn't creating these sockets, the sockets don't need to be created with a `mysql` user. The sockets MariaDB may end up listening to under systemd socket activation, it may have not had the privileges to create itself.

Changes to the default `mariadb.socket` can be made in the same way as services, `systemctl edit mariadb.socket` , or using `/etc/systemd/system/mariadb.socket.d/someconfig.conf` files.

Extra Port

A systemd socket can be configured as an `extra_port`, `by using the` FileDescriptorName=extra `in the` .socket `file`.

The `mariadb-extra.socket` is already packaged and ready for use.

Multi-instance socket activation

`mariadb@.socket` is MariaDB's packaged multi-instance defination. It creates multiple UNIX sockets based on the socket file started.

Starting `mariadb@bob.socket` will use the `mariadb@.socket` defination with `%I` within the defination replaced with "bob".

When something connects to a socket defined there, the `mariadb@bob.service` will be started.

Systemd Socket Activation for Hosting Service Providers

A systemd socket activation service with multi-instance can provide an on-demand per user access to a hosting service provider's dedicated database.

"User", in this case, refers to the customer of the hosting service provider.

End User Benefits

This provides the following benefits for the user:

- Each user has their own dedicated instance with the following benefits:
  - The instance is free from the database contention of neighbors on MariaDB shared resources (table cache, connections, etc)
  - The user is free to change their own configuration of MariaDB, within the limits and permissions of the service provider.
  - Database service level backups, like mariabackup, are now directly available.
  - A user can install their own plugins.
  - The user can run a different database version to their neighbors.
  - If a user's neighbor triggers a fault in the server, the uder's instance isn't affected.
- The database runs as their unix user in the server facilitating:
  - User can directly migrate their MariaDB data directory to a different provider.
  - The user's data is protected from other users on a kernel level.

Hosting Service Provider Benefits

In addition to providing user benefits as a sales item, the following are additional benefits for the hosting service provider compared to a monolith service:

- Without passwords for the database, while still having security, support may be easier.
- When a user's database isn't active, there is no resource usage, only listening file descriptors by systemd.
- The socket activation transparently, with a minor startup time, starts the service as required.
- When the user's database hasn't had any activity after a time, it will deactivate (MDEV-25282).
- Planned enhancements in InnoDB provide:
  - an on-demand consumption of memory (MDEV-25340 .
  - a proactive reduction in memory (MDEV-25341).
  - a memory resource pressure reduction in memory use (MDEV-24670).
- The service provider can still cap the user's database memory usage in a ulimit way that a user cannot override in settings.
- The service provider may choose a CPU/memory/IO based billing to the user on Linux cgroup accounting rather than the available comprared to the rather limited options in `CREATE USER` .
- Because a user's database will shutdown when inactive, a database upgrade on the server will not take effect for the user until it passively shuts down, restarts, and then gets reactivated hence reducing user downtime..

Downsides to the Hosting Service Provider

The extra memory used by more instances. This is mitigated by the on-demand activation. The deactivation when idle, and improved InnoDB memory management.

With plenty of medium size database servers running, the Linux OOM kill has the opportunity to kill off only a small number of database servers running rather than everyones.

Example on configuration Items for a per user, systemd socket activitated multi-instance service

From a server pespective the operation would be as follows;

To make the socket ready to connect and systemd will be listening to the socket:

```
# systemctl start mariadb@username.socket
# systemctl start mariadb-extra@username.socket
```

To enable this on reboot (the same way as a systemd service):

```
# systemctl enable mariadb@username.socket
# systemctl enable mariadb-extra@username.socket
```

A MariaDB Template File

A global template file. Once installed as a user's `$HOME/.my.cnf` file, it will becomes the default for many applications, and the MariaDB server itself.

```
# cat /etc/my.cnf.templ
[client]
socket=/home/USER/mariadb.sock

[client-server]
user=USER

[mariadbd]
datadir=/home/USER/mariadb-datadir
```

Custom Configuration for the Multi-instance Service

This extends/modifies the MariaDB multi-instance service.

The feature of this extension are:

- that it will autocreate configuration file for user applications
- It will install the database on first service start
- `auth-root-*` in mariadb-install-db means that the user is their own privileged user with unix socket authentication active. This means non-that user cannot access another users service, even with access to the unix socket(s). For more information see unix socket authentication security.
- If the MariaDB version was upgrade, the upgrade changes are made automatically

- `LimitData` places a hard upper limit so the user doesn't exceed a portion of the server resources

```
# cat /etc/systemd/system/mariadb@.service.d/user.conf
[Service]
User=%I
ProtectHome=false

Environment=MYSQLD_MULTI_INSTANCE="--defaults-file=/home/%I/.my.cnf"

ExecStartPre=
ExecStartPre=/bin/sh -c "[ -f /home/%I/.my.cnf ] || sed -e \"s/USER/%I/g\" /etc/my.cnf.templ >
/home/%I/.my.cnf"
ExecStartPre=mkdir -p /home/%I/mariadb-datadir
ExecStartPre=/usr/bin/mariadb-install-db $MYSQLD_MULTI_INSTANCE --rpm \
   --auth-root-authentication-method=socket --auth-root-socket-user=%I
ExecStartPost=/usr/bin/mariadb-upgrade $MYSQLD_MULTI_INSTANCE

# To limit user based tuning
LimitData=768M
# For io_uring use by innodb on < 5.12 kernels
LimitMEMLOCK=1M
```

Custom Configuration for the Multi-instance Socket

This extends/modifies the MariaDB socket defination to be per user.

Create sockets based on the user of the istance (`%I`). Permissions are only necessary in the sense that the user can connect to them. It won't matter to the server. Access control is enforced within the server, however if the user web services are run as the user, `Mode=777` can be reduced. `@mariadb-%I` is a abstract unix socket not on the filesystem. It may help if a user is in a chroot. Not all applications can connect to abstract sockets.

```
# cat /etc/systemd/system/mariadb@.socket.d/user.conf
[Socket]
SocketUser=%I
SocketMode=777
ListenSteam=
ListenStream=@mariadb-%I
ListenStream=/home/%I/mariadb.sock
```

The extra socket provides the user the ability to access the server when all max-connections are used:

```
# cat /etc/systemd/system/mariadb-extra@.socket.d/user.conf
[Socket]
SocketUser=%I
SocketMode=777
ListenSteam=
ListenStream=@mariadb-extra-%I
ListenStream=/home/%I/mariadb-extra.sock
```

Systemd Journal

`systemd` has its own logging system called the `systemd` journal. The `systemd` journal contains information about the service startup process. It is a good place to look when a failure has occurred.

The MariaDB `systemd` service's journal can be queried by using the `journalctl` command. For example:

```
$ sudo journalctl n 20 -u mariadb.service
-- Logs begin at Fri 2019-01-25 13:49:04 EST, end at Fri 2019-01-25 18:07:02 EST. --
Jan 25 13:49:15 ip-172-30-0-249.us-west-2.compute.internal systemd[1]: Starting MariaDB 10.1.37
database server...
Jan 25 13:49:16 ip-172-30-0-249.us-west-2.compute.internal mysqld[2364]: 2019-01-25 13:49:16
140547528317120 [Note] /usr/sbin/mysqld (mysqld 10.1.37-MariaDB) starting as process 2364 ...
Jan 25 13:49:17 ip-172-30-0-249.us-west-2.compute.internal systemd[1]: Started MariaDB 10.1.37
database server.
Jan 25 18:06:42 ip-172-30-0-249.us-west-2.compute.internal systemd[1]: Stopping MariaDB 10.1.37
database server...
Jan 25 18:06:44 ip-172-30-0-249.us-west-2.compute.internal systemd[1]: Stopped MariaDB 10.1.37
database server.
Jan 25 18:06:57 ip-172-30-0-249.us-west-2.compute.internal systemd[1]: Starting MariaDB 10.1.37
database server...
Jan 25 18:08:32 ip-172-30-0-249.us-west-2.compute.internal systemd[1]: mariadb.service start-pre
operation timed out. Terminating.
Jan 25 18:08:32 ip-172-30-0-249.us-west-2.compute.internal systemd[1]: Failed to start MariaDB
10.1.37 database server.
Jan 25 18:08:32 ip-172-30-0-249.us-west-2.compute.internal systemd[1]: Unit mariadb.service
entered failed state.
Jan 25 18:08:32 ip-172-30-0-249.us-west-2.compute.internal systemd[1]: mariadb.service failed.
```

Converting mysqld_safe Options to Systemd Options

`mariadb-service-convert` is a script included in many MariaDB packages that is used by the package manager to convert `mysqld_safe` options to `systemd` options. It reads any explicit settings in the `[mysqld_safe]` option group from option files, and its output is directed to `/etc/systemd/system/mariadb.service.d/migrated-from-my.cnf-settings.conf`. This helps to keep the configuration the same when upgrading from a version of MariaDB that does not use `systemd` to one that does.

Implicitly high defaults of `open-files-limit` may be missed by the conversion script and require explicit configuration. See Configuring the Open Files Limit.

2.1.6.15 sysVinit

Contents

sysVinit is one of the most common service managers. On systems that use sysVinit, the `mysql.server` script is normally installed to `/etc/init.d/mysql`.

Interacting with the MariaDB Server Process

The service can be interacted with by using the `service` command.

Starting the MariaDB Server Process on Boot

On RHEL/CentOS and other similar distributions, the `chkconfig` command can be used to enable the MariaDB Server process at boot:

```
chkconfig --add mysql
chkconfig --level 345 mysql on
```

On Debian and Ubuntu and other similar distributions, the `update-rc.d` command can be used:

```
update-rc.d mysql defaults
```

Starting the MariaDB Server Process

```
service mysql start
```

Stopping the MariaDB Server Process

```
service mysql stop
```

Restarting the MariaDB Server Process

```
service mysql restart
```

Checking the Status of the MariaDB Server Process

```
service mysql status
```

Manually Installing mysql.server with SysVinit

If you install MariaDB from source or from a binary tarball that does not install `mysql.server` automatically, and if you are on a system that uses sysVinit, then you can manually install `mysql.server` with sysVinit. See mysql.server: Manually Installing with SysVinit for more information.

SysVinit and Galera Cluster

Bootstrapping a New Cluster

When using Galera Cluster with sysVinit, the first node in a cluster has to be started with `service mysql bootstrap`. See Getting Started with MariaDB Galera Cluster: Bootstrapping a New Cluster for more information.

1.3.25 Mariadb-admin

2.1.6.17 mariadbd

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadbd` is a symlink to mysqld, the MariaDB server.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mariadbd` is the name of the server, with `mysqld` a symlink .

See mysqld for details.

2.1.6.18 mariadbd-multi

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadbd-multi` is a symlink to `mysqld_multi`, the wrapper designed to manage several mysqld processes running on the same host.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mariadbd-multi` is the name of the server, with `mysqld_multi` a symlink .

See mysqld_multi for details.

2.1.6.19 mariadbd-safe

MariaDB starting with 10.4.6

From MariaDB 10.4.6, `mariadbd-safe` is a symlink to `mysqld_safe`, the tool for starting mysqld on Linux and Unix distributions that do not support systemd.

MariaDB starting with 10.5.2

From MariaDB 10.5.2, `mariadbd-safe` is the name of the binary, with `mysqld_safe` a symlink .

See mysqld_safe for details.

2.1.7 MariaDB Performance & Advanced Configurations

Articles of how to setup your MariaDB optimally on different systems

Fusion-io

This category contains information about Fusion-io support in MariaDB

Atomic Write Support

Enabling atomic writes to speed up InnoDB on selected SSD cards.

Configuring Linux for MariaDB

Linux kernel settings IO scheduler For optimal IO performance running a da...

Configuring MariaDB for Optimal Performance

How to get optimal performance.

Configuring Swappiness

Setting Linux swappiness.

There are 5 related questions.

2.1.7.1 Fusion-io

This category contains information about Fusion-io support in MariaDB

Fusion-io Introduction

Fusion-io PCIe SSD cards to speed up MariaDB.

Atomic Write Support

Enabling atomic writes to speed up InnoDB on selected SSD cards.

MariaDB 10.0.15 Fusion-io Release Notes

Status: | Release Date: 12 Dec 2014

MariaDB 10.0.15 Fusion-io Changelog

Status: | Release Date: 12 Dec 2014

InnoDB Page Flushing

Configuring when and how InnoDB flushes dirty pages to disk.

2.1.7.1.1 Fusion-io Introduction

Contents

Fusion-io develops PCIe based NAND flash memory cards and related software that can be used to speed up MariaDB databases.

The ioDrive branded products can be used as block devices (super-fast disks) or to extend basic DRAM memory. ioDrive is deployed by installing it on an x86 server and then installing the card driver under the operating system. All main line 64-bit operating systems and hypervisors are supported: RHEL, CentOS, SuSe, Debian, OEL etc. and VMware, Microsoft Windows/Server etc. Drivers and their features are constantly developed further.

ioDrive cards support software RAID and you can combine two or more physical cards into one logical drive. Through ioMemory SDK and its APIs, one can integrate and enable more thorough interworking between your own software and the cards - and cut latency.

The key differentiator between a Fusion-io and a legacy SSD/HDD is the following: **A Fusion-io card is connected directly on the system bus (PCIe)**, this enables high data transfer throughput (1.5 GB/s, 3.0 GB/s or 6GB/s) and the fast direct memory access (DMA) method can be used to transfer data. The ATA/SATA protocol stack is omitted and therefore latency is cut short. Fusion-io performance is dependent on server speed: the faster processors and the newer PCIe-bus version you have, the better is the ioDrive performance. Fusion-io memory is non-volatile, in other words, data remains on the card even when the server is powered off.

Use Cases

1. You can start by using ioDrive for database files that need heavy random access.
2. Whole database on ioDrive.
3. In some cases, Fusion-io devices allow for atomic writes, which allows the server to safely disable the doublewrite buffer.
4. Use ioDrive as a write-through read cache. This is possible on server level with Fusion-io directCache software or in VMware environments using ioTurbine software or the ioCache bundle product. Reads happen from ioDrive and all writes go directly to your SAN or disk.
5. Highly Available shared storage with ION. Have two different hosts, Fusion-io cards in them and share/replicate data with Fusion-io's ION software.
6. The luxurious Platinum setup: MariaDB Galera Cluster running on Fusion-io SLC cards on several hosts.

Atomic Writes

Starting with MariaDB 5.5.31, MariaDB Server supports atomic writes on Fusion-io devices that use the NVMFS (formerly called DirectFS) file system. Unfortunately, NVMFS was never offered under 'General Availability', and SanDisk declared that NVMFS would reach end-of-life in December 2015. Therefore, NVMFS support is no longer offered by SanDisk.

MariaDB Server does not currently support atomic writes on Fusion-io devices with any other file systems.

See atomic write support for more information about MariaDB Server's atomic write support.

Future Suggested Development

- Extend InnoDB disk cache to be stored on Fusion-io acting as extended memory.

Settings For Best Performance

Fusion-io memory can be formatted with different sector size of either 512 or 4096 bytes. Bigger sectors are expected to be faster, but only if I/O is done in blocks of 4KB or multiples of that. Speaking of MariaDB: if only InnoDB data files are stored in Fusion-io memory, all I/O is done in blocks of 16K and thus 4K sector size can be used. If the InnoDB redo log (I/O block size: 512 bytes) goes to the same Fusion-io memory, then short sectors should be used.

Note: XtraDB has the experimental feature of an increased InnoDB log block size of 4K. If this is enabled, then both redo log I/O and page I/O in InnoDB will match a sector size of 4K.

As of file systems: currently XFS is expected to yield the best performance with MariaDB. However depending on the exact kernel version and version of XFS code in use, one might be affected by a bug that severely limits XFS performance in

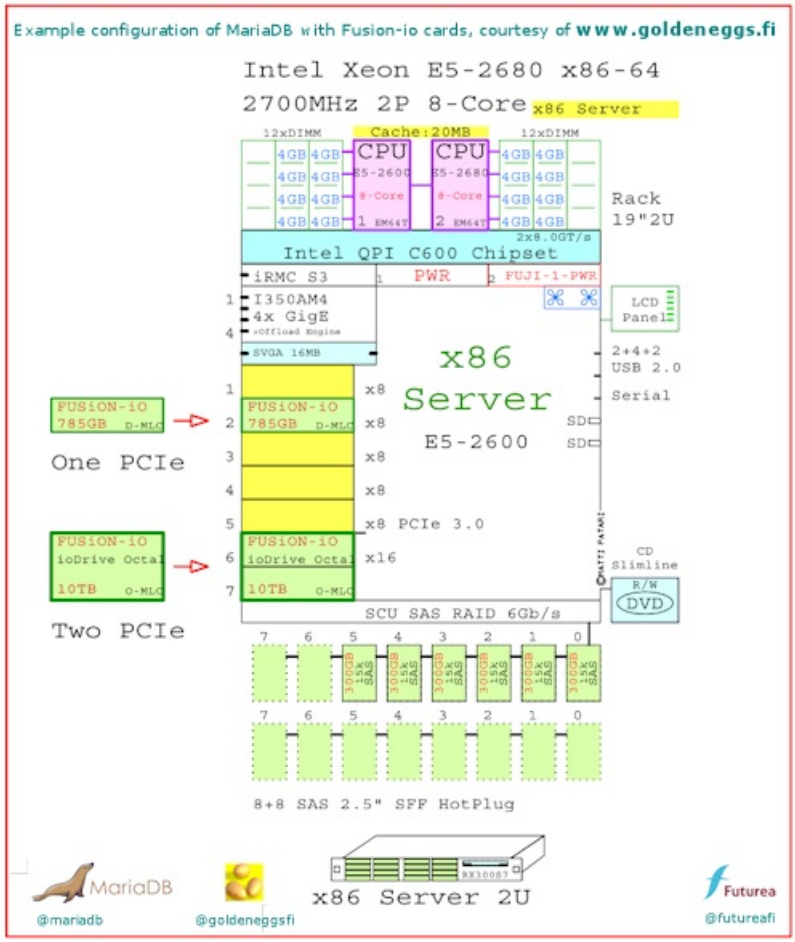
concurrent environments. This has been fixed in kernel versions above 3.5 or RHEL6 kernels kernel-2.6.32-358 or later (because of bug 807503 being fixed).

For the pitbull machine where I have run such tests, ext4 was faster than xfs for 32 or more threads:

- up to 8 threads xfs was few percent faster (10% on average).
- at 16 threads it was a draw (2036 tps vs. 2070 tps).
- at 32 threads ext4 was 28% faster (2345 tps vs. 1829 tps).
- at 64 threads ext4 was even 47% faster (2362 tps vs. 1601 tps).
- at higher concurrency ext4 lost it's bite, but was still constantly better than xfs.

Those numbers are for spinning disks. I guess for Fusion-io memory the XFS numbers will be even worse.

Example Configuration



Card Models

There are several card models. ioDrive is older generation, ioDrive2 is newer. SLC sustains more writes. MLC is good enough for normal use.

1. ioDrive2, capacities per card 365GB, 785GB, 1.2TB with MLC. 400GB and 600GB with SLC, performance up to 535000 IOPS & 1.5GB/s bandwidth
2. ioDrive2 Duo, capacities per card 2.4TB MLC and 1.2TB SLC, performance up to 935000 IOPS & 3.0GB/s bandwidth
3. ioDrive, capacities per card 320GB, 640GB MLC and 160GB, 320GB SLC, performance up to 145000 IOPS & 790MB/s bandwidth
4. ioDrive Duo, capacities per card 640GB, 1.28TB MLC and 320GB, 640GB SLC, performance up to 285000 IOPS & 1.5GB/s bandwidth
5. ioDrive Octal, capacities per card 5TB and 10TB MLC, performance up to 1350000 IOPS & 6.7GB/s bandwidth
6. ioFX, a 420GB QDP MLC workstation product, 1.4GB/s bandwidth
7. ioCache, a 600GB MLC card with ioTurbine software bundle that can be used to speed up VMware based virtual hosts.
8. ioScale, 3.2TB card, building block to enable all-flash data center build out in hyperscale web and cloud environments. Product has been developed in co-operation with Facebook.

Additional Software

- directCache - transforms ioDrive to work as a read cache in your server. Writes go directly to your SAN
- ioTurbine - read cache software for VMware
- ION - transforms ioDrive into a shareable storage
- ioSphere - software to manage and monitor several ioDrives

2.1.7.1.2 Atomic Write Support

Contents

Partial Write Operations

When Innodb writes to the filesystem, there is generally no guarantee that a given write operation will be complete (not partial) in cases of a poweroff event, or if the operating system crashes at the exact moment a write is being done.

Without detection or prevention of partial writes, the integrity of the database can be compromised after recovery.

innodb_doublewrite - an Imperfect Solution

Since its inception, Innodb has had a mechanism to detect and ignore partial writes via the InnoDB Doublewrite Buffer (also innodb_checksum can be used to detect a partial write).

Doublewrites, controlled by the innodb_doublewrite system variable, comes with its own set of problems. Especially on SSD, writing each page twice can have detrimental effects (write leveling).

Atomic Write - a Faster Alternative to innodb_doublewrite

A better solution is to directly ask the filesystem to provide an atomic (all or nothing) write guarantee. Currently this is only available on a few SSD cards.

Enabling Atomic Writes from MariaDB 10.2

When starting, MariaDB 10.2 and beyond automatically detects if any of the supported SSD cards are used.

When opening an InnoDB table, there is a check if the tablespace for the table is on a device that supports atomic writes and if yes, it will automatically enable atomic writes for the table. If atomic writes support is not detected, the doublewrite buffer will be used.

One can disable atomic write support for all cards by setting the variable innodb-use-atomic-writes to `OFF` in your my.cnf file. It's `ON` by default.

Enabling Atomic Writes in MariaDB 5.5 to MariaDB 10.1

To use atomic writes instead of the doublewrite buffer, add:

```
innodb_use_atomic_writes = 1
```

to the my.cnf config file.

Note that atomic writes are only supported on Fusion-io devices that use the NVMFS file system in these versions of MariaDB.

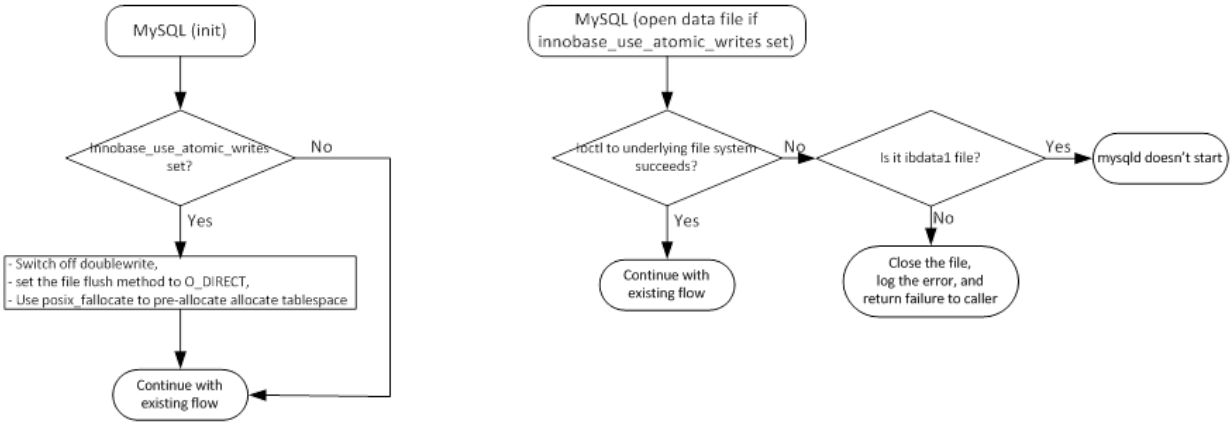
About innodb_use_atomic_writes (in MariaDB 5.5 to MariaDB 10.1)

The following happens when atomic writes are enabled

- if innodb_flush_method is neither `O_DIRECT`, `ALL_O_DIRECT`, or `O_DIRECT_NO_FSYNC`, it is switched to `O_DIRECT`

- innodb_use_fallocate is switched `ON` (files are extended using `posix_fallocate` rather than writing zeros behind the end of file)
- Whenever an Innodb datafile is opened, a special `ioctl()` is issued to switch on atomic writes. If the call fails, an error is logged and returned to the caller. This means that if the system tablespace is not located on an atomic write capable device or filesystem, InnoDB/XtraDB will refuse to start.
- if innodb_doublewrite is set to `ON`, `innodb_doublewrite` will be switched `OFF` and a message written to the error log.

Here is a flowchart showing how atomic writes work inside InnoDB:



Devices that Support Atomic Writes with MariaDB

MariaDB currently supports atomic writes on the following devices:

- Fusion-io devices with the NVMFS file system . MariaDB 5.5 and above.
- Shannon SSD . MariaDB 10.2 and above.

5.3.2.16 InnoDB Page Flushing

2.1.7.2 Configuring Linux for MariaDB

Contents

Linux kernel settings

IO scheduler

For optimal IO performance running a database we are using the *none* (previously called *noop*) scheduler. Recommended schedulers are *none* and *mq-deadline* (previously called *deadline*). You can check your scheduler setting with:

```
cat /sys/block/${DEVICE}/queue/scheduler
```

For instance, it should look like this output:

```
cat /sys/block/vdb/queue/scheduler
[none] mq-deadline kyber bfq
```

Older kernels may look like:

```
cat /sys/block/sda/queue/scheduler
[noop] deadline cfq
```

Writing the new scheduler name to the same /sys node will change the scheduler:

```
echo mq-deadline >  /sys/block/vdb/queue/scheduler
```

The impact of schedulers depend significantly on workload and hardware. You can measure the IO-latency using the biolatency bcc-tools script with an aim to keep the mean as low as possible.

Resource Limits

Configuring the Open Files Limit

By default, the system limits how many open file descriptors a process can have open at one time. It has both a soft and hard limit. On many systems, both the soft and hard limit default to 1024. On an active database server, it is very easy to exceed 1024 open file descriptors. Therefore, you may need to increase the soft and hard limits. There are a few ways to do so.

If you are using `mysqld_safe` to start `mysqld`, then see the instructions at mysqld_safe: Configuring the Open Files Limit.

If you are using `systemd` to start `mysqld`, then see the instructions at systemd: Configuring the Open Files Limit.

Otherwise, you can set the soft and hard limits for the `mysql` user account by adding the following lines to `/etc/security/limits.conf`:

```
mysql soft nofile 65535
mysql hard nofile 65535
```

After the system is rebooted, the `mysql` user should use the new limits, and the user's `ulimit` output should look like the following:

```
$ ulimit -Sn
65535
$ ulimit -Hn
65535
```

Configuring the Core File Size

By default, the system limits the size of core files that could be created. It has both a soft and hard limit. On many systems, the soft limit defaults to 0. If you want to enable core dumps, then you may need to increase this. Therefore, you may need to increase the soft and hard limits. There are a few ways to do so.

If you are using `mysqld_safe` to start `mysqld`, then see the instructions at mysqld_safe: Configuring the Core File Size.

If you are using `systemd` to start `mysqld`, then see the instructions at systemd: Configuring the Core File Size.

Otherwise, you can set the soft and hard limits for the `mysql` user account by adding the following lines to `/etc/security/limits.conf`:

```
mysql soft core unlimited
mysql hard core unlimited
```

After the system is rebooted, the `mysql` user should use the new limits, and the user's `ulimit` output should look like the following:

```
$ ulimit -Sc
unlimited
$ ulimit -Hc
unlimited
```

Swappiness

See configuring swappiness.

2.1.7.3 Configuring MariaDB for Optimal

Performance

Contents

This article is to help you configure MariaDB for optimal performance.

Note that by default MariaDB is configured to work on a desktop system and should because of this not take a lot of resources. To get things to work for a dedicated server, you have to do a few minutes of work.

For this article we assume that you are going to run MariaDB on a dedicated server.

Feel free to update this article if you have more ideas.

my.cnf Files

MariaDB is normally configured by editing the my.cnf file.

The following my.cnf example files were included with MariaDB until MariaDB 10.3.0. If present, you can examine them to see more complete examples of some of the many ways to configure MariaDB and use the one that fits you best as a base. Note that these files are now quite outdated, so what was huge a few years ago may no longer be seen as such.

- my-small.cnf
- my-medium.cnf
- my-large.cnf
- my-huge.cnf

InnoDB Storage Engine

InnoDB is normally the default storage engine with MariaDB.

- You should set innodb_buffer_pool_size to about 80% of your memory. The goal is to ensure that 80 % of your working set is in memory.

The other most important InnoDB variables are:

- innodb_log_file_size
- innodb_flush_method
- innodb_thread_sleep_delay

Some other important InnoDB variables:

- innodb_max_dirty_pages_pct_lwm
- innodb_read_ahead_threshold
- innodb_buffer_pool_instances. Deprecated and ignored from MariaDB 10.5.1.
- innodb_adaptive_max_sleep_delay. Deprecated and ignored from MariaDB 10.5.5.
- innodb_thread_concurrency. Deprecated and ignored from MariaDB 10.5.5.

Aria Storage Engine

- MariaDB uses by default the Aria storage engine for internal temporary files. If you have many temporary files, you should set aria_pagecache_buffer_size to a reasonably large value so that temporary overflow data is not flushed to disk. The default is 128M.

MyISAM

- If you don't use MyISAM tables explicitly (true for most MariaDB 10.4+ users), you can set key_buffer_size to a very low value, like 64K.

Lots of Connections

A Lot of Fast Connections + Small Set of Queries + Disconnects

- If you are doing a lot of fast connections / disconnects, you should increase back_log and if you are running MariaDB 10.1 or below thread_cache_size.
- If you have a lot (> 128) of simultaneous running fast queries, you should consider setting thread_handling to `pool_of_threads`.

Connecting From a Lot of Different Machines

- If you are connecting from a lot of different machines you should increase host_cache_size to the max number of machines (default 128) to cache the resolving of hostnames. If you don't connect from a lot of machines, you can set this to a very low value!

2.1.7.4 Configuring Swappiness

Contents

Why to Avoid Swapping

Obviously, accessing swap memory from disk is far slower than accessing RAM directly. This is particularly bad on a database server because:

- MariaDB's internal algorithms assume that memory is not swap, and are highly inefficient if it is. Some algorithms are intended to avoid or delay disk IO, and use memory where possible - performing this with swap can be worse than just doing it on disk in the first place.
- Swap increases IO over just using disk in the first place as pages are actively swapped in and out of swap. Even something like removing a dirty page that is no longer going to be stored in memory, while designed to improve efficiency, will under a swap situation cost more IO.
- Database locks are particularly inefficient in swap. They are designed to be obtained and released often and quickly, and pausing to perform disk IO will have a serious impact on their usability.

The main way to avoid swapping is to make sure you have enough RAM for all processes that need to run on the machine. Setting the system variables too high can mean that under load the server runs short of memory, and needs to use swap. So understanding what settings to use and how these impact your server's memory usage is critical.

Setting Swappiness on Linux

Linux has a swappiness setting which determines the balance between swapping out pages (chunks of memory) from RAM to a preconfigured swap space on the hard drive.

The setting is from 0 to 100, with lower values meaning a lower likelihood of swapping. The default is usually 60 - you can check this by running:

```
sysctl vm.swappiness
```

The default setting encourages the server to use swap. Since there probably won't be much else on the database server besides MariaDB processes to put into swap, you'll probably want to reduce this to zero to avoid swapping as much as possible. You can change the default by adding a line to the `sysctl.conf` file (usually found in `/etc/sysctl.conf`).

To set the swappiness to zero, add the line:

```
vm.swappiness = 0
```

This normally takes effect after a reboot, but you can change the value without rebooting as follows:

```
sysctl -w vm.swappiness=0
```

Since RHEL 6.4, setting swappiness=0 more aggressively avoids swapping out, which increases the risk of OOM killing under strong memory and I/O pressure.

A low swappiness setting is recommended for database workloads. For MariaDB databases, it is recommended to set swappiness to a value of 1.

```
vm.swappiness = 1
```

Disabling Swap Altogether

While some disable swap altogether, and you certainly want to avoid any database processes from using it, it can be prudent to leave some swap space to at least allow the kernel to fall over gracefully should a spike occur. Having emergency swap available at least allows you some scope to kill any runaway processes.

2.1.8 Troubleshooting Installation Issues

Articles relating to installation issues users might run into.

Troubleshooting Connection Issues

Common problems when trying to connect to MariaDB.

Installation issues on Windows

Issues people have encountered when installing MariaDB on Windows

Troubleshooting MariaDB Installs on Red Hat/CentOS

Issues people have encountered when installing MariaDB on Red Hat / CentOS

Installation issues on Debian and Ubuntu

Solutions to different installation issues on Debian and Ubuntu

What to Do if MariaDB Doesn't Start

Troubleshooting MariaDB when it fails to start.

Installing on an Old Linux Version

Typical errors from using an incompatible MariaDB binary on a linux system

Error: symbol mysql_get_server_name, version libmysqlclient_16 not defined

Error from using MariaDB's mysql command-line client with MySQL's libmysqlclient.so

Installation Issues with PHP5

PHP5 may give an error if used with the old connect method

There are 13 related questions.

6.2.7 Troubleshooting Connection Issues

2.1.8.2 Installation issues on Windows

Contents

MariaDB 10.4.13

MariaDB 10.4.13 may not start on Windows. See MDEV-22555.

To resolve this, download, click and install https://aka.ms/vs/16/release/vc_redist.x64.exe and then install 10.4.13.

Unsupported Versions of Windows

Recent versions of MariaDB may not install on unsupported Windows versions. See Deprecated Package Platforms to find the final supported versions.

MariaDB 5.2.5 and earlier

On Windows Vista/7 , changes to database or my.ini are not persistent, when mysqld.exe is run from the command line.

The reason for this behavior is Vista/Win7 file system redirection. Writes to protected locations (in this case a subdirectory of Program Files) are redirected to the user's so-called "Virtual Store".

Workarounds:

- Run mysqld.exe as service. See answer here on how to create a MariaDB service.
- Run mysqld.exe from the elevated command prompt.
- Change the ACL of the data directory and add full control for the current user.

The Windows installer for MariaDB 5.2.6 and higher will set the data directory ACL to include full access rights for the user who runs the setup to prevent this issue from happening.

Systems with User Account Control

Running `mysql_install_db.exe` from a standard command prompt might cause the error:

```
FATAL ERROR: OpenSCManager failed
```

To get rid of it, use the elevated command prompt, for example on Windows 7 start it via 'Run as administrator' option.

2.1.2.1.6 Troubleshooting MariaDB Installs on Red Hat/CentOS

2.1.8.4 Installation issues on Debian and Ubuntu

Solutions to different installation issues on Debian and Ubuntu

Differences in MariaDB in Debian (and Ubuntu)
MariaDB when installed from the Debian repos has a number of differences with standard MariaDB.

Moving from MySQL to MariaDB in Debian 9
MariaDB 10.1 is the default mysql server in Debian 9 "Stretch"

Creating a Debian Repository
Instructions for creating your own Debian repository

MariaDB 5.5.33 Debian and Ubuntu Installation Issues
Workarounds for some repository issues with the 5.5.33 release.

MariaDB Debian Live Images
Debian live iso images with pre-installed MariaDB (obsolete)

apt-upgrade Fails, But the Database is Running
timeout causing apt to fail while the database is running.

There are 9 related questions.

2.1.8.4.1 Differences in MariaDB in Debian (and

Ubuntu)

Contents

The `.deb` packages provided by MariaDB Foundation's and MariaDB Corporation's repositories are not identical to the official `.deb` packages provided by Debian's and Ubuntu's default repositories.

The packages provided by MariaDB Foundation's and MariaDB Corporation's repositories are generated using the Debian packaging in MariaDB's official source code. The Debian packaging scripts are specifically in the `debian/` directory.

The packages provided by Debian's and Ubuntu's default repositories are generated using the Debian packaging in Debian's mirror of MariaDB's source code, which contains some custom changes. The source tree can be found here:

- https://salsa.debian.org/mariadb-team/mariadb-server

As a consequence, MariaDB behaves a bit differently if it is installed from Debian's and Ubuntu's default repositories.

Option File Locations

- The option file located at `/etc/mysql/my.cnf` is handled by the `update-alternatives` mechanism when the `mysql-common` package is installed. It is a symbolic link that references either `mysql.cnf` or `mariadb.cnf` depending on whether MySQL or MariaDB is installed. Most of the MariaDB option files are therefore actually located in `/etc/mysql/mariadb.d/`.

System Variables

Variable	MariaDB in Debian	Standard MariaDB	Notes
character_set_server	utf8mb4	latin1	Debian sets a default character set that can support emojis etc.
collation_server	utf8mb4_general_ci	latin1_swedish_ci	

Options

Option	MariaDB in Debian	Standard MariaDB	Notes
`plugin-load-add`	`auth_socket.so`	-	Before MariaDB 10.4.3, MariaDB did not enable the `unix_socket` authentication plugin by default.This is default in Debian, allowing passwordless login.

TLS

- MariaDB binaries from `.deb` packages provided by Debian's and Ubuntu's default repositories are linked with a different TLS library than MariaDB binaries from `.deb` packages provided by MariaDB Foundation's and MariaDB Corporation's repositories.
- MariaDB Server binaries:
  - In MariaDB 10.4.6 and later, MariaDB Server is statically linked with the bundled wolfSSL libraries in `.deb` packages provided by Debian's and Ubuntu's default repositories.
  - In MariaDB 10.4.5 and before, MariaDB Server is statically linked with the bundled yaSSL libraries in `.deb` packages provided by Debian's and Ubuntu's default repositories.
  - In contrast, MariaDB Server is dynamically linked with the system's OpenSSL libraries in `.deb` packages provided by MariaDB Foundation and MariaDB Corporation.
- MariaDB client and utility binaries:
  - In MariaDB 10.4.6 and later, MariaDB's clients and utilities and MariaDB Connector/C are dynamically linked with the system's GnuTLS libraries in `.deb` packages provided by Debian's and Ubuntu's default repositories. libmysqlclient is still statically linked with the bundled wolfSSL libraries.
  - In MariaDB 10.2 and later, MariaDB's clients and utilities and MariaDB Connector/C are dynamically linked with the system's GnuTLS libraries in `.deb` packages provided by Debian's and Ubuntu's default

repositories. libmysqlclient is still statically linked with the bundled yaSSL libraries.
    - In MariaDB 10.1 and earlier, MariaDB's clients and utilities and libmysqlclient are statically linked with the bundled yaSSL libraries in `.deb` packages provided by Debian's and Ubuntu's default repositories.
    - In contrast, MariaDB's clients and utilities, libmysqlclient, and MariaDB Connector/C are dynamically linked with the system's OpenSSL libraries in `.deb` packages provided by MariaDB Foundation's and MariaDB Corporation's repositories.
- See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.

Authentication

- The `unix_socket` authentication plugin is installed by default in **new installations** that use the `.deb` packages provided by Debian's default repositories in Debian 9 and later and Ubuntu's default repositories in Ubuntu 15.10 and later.
- The `root@localhost` created by mariadb-install-db will also be created to authenticate via the unix_socket authentication plugin in these builds.

2.1.3.12.2 Upgrading from MySQL to MariaDB

2.1.2.8.18 Creating a Debian Repository

2.1.8.4.4 apt-upgrade Fails, But the Database is Running

After running `apt-upgrade mariadb`, it's possible that apt shows a fail in trying to start the server, but in fact the database is up and running, which then provokes apt to remain in a non finished state.

For example:

```
# apt-get upgrade
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
2 not fully installed or removed.
After this operation, 0 B of additional disk space will be used.
Do you want to continue? [Y/n]
Setting up mariadb-server-10.1 (10.1.10+maria-1~trusty) ...
 * Stopping MariaDB database server mysqld
   ...done.
 * Starting MariaDB database server mysqld
   ...fail!
invoke-rc.d: initscript mysql, action "start" failed.
dpkg: error processing package mariadb-server-10.1 (--configure):
 subprocess installed post-installation script returned error exit status 1
dpkg: dependency problems prevent configuration of mariadb-server:
 mariadb-server depends on mariadb-server-10.1 (= 10.1.10+maria-1~trusty); however:
  Package mariadb-server-10.1 is not configured yet.

dpkg: error processing package mariadb-server (--configure):
 dependency problems - leaving unconfigured
No apport report written because the error message indicates its a followup error from a
previous failure.
Errors were encountered while processing:
 mariadb-server-10.1
 mariadb-server
E: Sub-process /usr/bin/dpkg returned an error code (1)
```

This situation could occur if the timeout for the init script was too short. For example, see MDEV-9382, a situation where the timeout was 30 seconds, but the server was taking 48 seconds to start.

To overcome this, the timeout needs to be increased. This can be achieved as follows:

- **On systems where systemd is not enabled/supported:** The timeout can be increased by setting

MYSQLD_STARTUP_TIMEOUT either directly in the script or via the command line. In MariaDB 10.1.13 and later versions, the init script also sources /etc/default/mariadb, so it can also be used to set MYSQLD_STARTUP_TIMEOUT to persistently change the startup timeout. The default timeout has been increased from 30s to 60s in MariaDB 10.1.13.

- **On systems that support systemd**: The startup timeout can be increased by setting TimeoutStartSec systemd option.

2.1.6.5 What to Do if MariaDB Doesn't Start

2.1.8.6 Installing on an Old Linux Version

This article lists some typical errors that may happen when you try to use an incompatible MariaDB binary on a linux system:

The following example errors are from trying to install MariaDB built for SuSE 11.x on a SuSE 9.0 server:

```
> scripts/mysql_install_db
./bin/my_print_defaults: /lib/i686/libc.so.6:
  version `GLIBC_2.4' not found (required by ./bin/my_print_defaults)
```

and

```
> ./bin/mysqld --skip-grant &
./bin/mysqld: error while loading shared libraries: libwrap.so.0:
cannot open shared object file: No such file or directory
```

If you see either of the above errors, the binary MariaDB package you installed is not compatible with your system.

The options you have are:

- Find another MariaDB package or tar from the download page that matches your system.

  or

- Download the source and build it.

2.1.8.7 Error: symbol mysql_get_server_name, version libmysqlclient_16 not defined

If you see the error message:

```
symbol mysql_get_server_name, version libmysqlclient_16 not defined in file libmysqlclient.so.16
with link time reference
```

...then you are probably trying to use the mysql command-line client from MariaDB with libmysqlclient.so from MySQL.

The symbol `mysql_get_server_name()` is something present in the MariaDB source tree and not in the MySQL tree.

If you have both the MariaDB client package and the MySQL client packages installed this error will happen if your system finds the MySQL version of `libmysqlclient.so` first.

To figure out which library is being linked in dynamically (ie, the wrong one) use the 'ldd' tool.

```
ldd $(which mysql) | grep mysql
```

or

```
ldd /path/to/the/binary | grep mysql
```

For example:

```
me@mybox:~$ ldd $(which mysql)|grep mysql
        libmysqlclient.so.16 => /usr/lib/libmysqlclient.so.16 (0xb74df000)
```

You can then use your package manager's tools to find out which package the library belongs to.

On CentOS the command to find out which package installed a specific file is:

```
rpm -qf /path/to/file
```

On Debian-based systems, the command is:

```
dpkg -S /path/to/file
```

Here's an example of locating the library and finding out which package it belongs to on an Ubuntu system:

```
me@mybox:~$ ldd $(which mysql)|grep mysql
 libmysqlclient.so.16 => /usr/lib/libmysqlclient.so.16 (0xb75f8000)
me@mybox:~$ dpkg -S /usr/lib/libmysqlclient.so.16
libmariadbclient16: /usr/lib/libmysqlclient.so.16
```

The above shows that the mysql command-line client is using the library `/usr/lib/libmysqlclient.so.16` and that that library is part of the `libmariadbclient16` Ubuntu package. Unsurprisingly, the mysql command-line client works perfectly on this system.

If the answer that came back had been something other than a MariaDB package, then it is likely there would have been issues with running the MariaDB mysql client application.

If the library that the system tries to use is not from a MariaDB package, the remedy is to remove the offending package (and possibly install or re-install the correct package) so that the correct library can be used.

2.1.9 Installing System Tables (mysql_install_db)

2.1.10 mysql_install_db.exe

Contents

The `mysql_install_db.exe` utility is the Windows equivalent of mysql_install_db.

Functionality

The functionality of `mysql_install_db.exe` is comparable with the shell script `mysql_install_db` used on Unix, however it has been extended with both Windows specific functionality (creating a Windows service) and to generally useful functionality. For example, it can set the 'root' user password during database creation. It also creates the `my.ini` configuration file in the data directory and adds most important parameters to it (e.g port).

`mysql_install_db.exe` is used by the MariaDB installer for Windows if the "Database instance" feature is selected. It obsoletes similar utilities and scripts that were used in the past such as `mysqld.exe --install`, `mysql_install_db.pl`, and `mysql_secure_installation.pl`.

Parameter	Description
`-?, --help`	Display help message and exit
`-d, --datadir=name`	Data directory of the new database
`-S, --service=name`	Name of the Windows service
`-p, --password=name`	Password of the root user
`-P, --port=#`	`mysqld` port
`-W, --socket=name`	named pipe name
`-D, --default-user`	Create default user
`-R, --allow-remote-root-access`	Allow remote access from network for user root
`-N, --skip-networking`	Do not use TCP connections, use pipe instead

`-i , --innodb-page-size`	Innodb page size, since MariaDB 10.2.5
`-s , --silent`	Print less information
`-o , --verbose-bootstrap`	Include mysqld bootstrap output
`-l , --large-pages`	Use large pages, since MariaDB 10.6.1
`-c , --config`	my.ini config template file, since MariaDB 10.6.1

Note : to create a Windows service, `mysql_install_db.exe` should be run by a user with full administrator privileges (which means elevated command prompt on systems with UAC). For example, if you are running it on Windows 7, make sure that your command prompt was launched via 'Run as Administrator' option.

Example

```
mysql_install_db.exe --datadir=C:\db --service=MyDB --password=secret
```

will create the database in the directory C:\db, register the auto-start Windows service "MyDB", and set the root password to 'secret'.

To start the service from the command line, execute

```
sc start MyDB
```

Removing Database Instances

If you run your database instance as service, to remove it completely from the command line, use

```
sc stop <servicename>
sc delete <servicename>
rmdir /s /q <path-to-datadir>
```

2.1.6.2 Configuring MariaDB with Option Files

2.1.12 MariaDB Environment Variables

MariaDB makes use of numerous environment variables that may be set on your system. Environment variables have the lowest precedence, so any options set on the command line or in an option file will take precedence.

It's usually better not to rely on environment variables, and to rather set the options you need directly, as this makes the system a little more robust and easy to administer.

Here is a list of environment variables used by MariaDB.

Environment Variable	Description
CXX	Name of the C++ compiler, used for running CMake.
CC	Name of the C compiler, used for running CMake.
DBI_USER	Perl DBI default username.
DBI_TRACE	Perl DBI trace options.
HOME	Default directory for the mysql_history file.
MYSQL_DEBUG	Debug trace options used when debugging.
MYSQL_GROUP_SUFFIX	In addition to the given option groups, also read groups with this suffix.
MYSQL_HISTFILE	Path to the mysql_history file, overriding the $HOME/.mysql_history setting.
MYSQL_HOME	Path to the directory containing the my.cnf file used by the server.
MYSQL_HOST	Default host name used by the mariadb command line client.
MYSQL_PS1	Command prompt for use by the mariadb command line client.

MYSQL_PWD	Default password when connecting to mysqld. It is strongly recommended to use a more secure method of sending the password to the server.
MYSQL_TCP_PORT	Default TCP/IP port number.
MYSQL_UNIX_PORT	On Unix, default socket file used for localhost connections.
PATH	Path to directories that hold executable programs (such as the mariadb client, mariadb-admin), so that these can be run from any location.
TMPDIR	Directory where temporary files are created.
TZ	Local time zone.
UMASK	Creation mode when creating files. See Specifying Permissions for Schema (Data) Directories and Tables.
UMASK_DIR	Creation mode when creating directories. See Specifying Permissions for Schema (Data) Directories and Tables.
USER	On Windows, up to MariaDB 5.5, the default user name when connecting to the mysqld server. API GetUserName() is used in later versions.

2.1.13 MariaDB on Amazon AWS

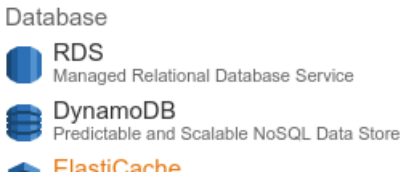
MariaDB is available on Amazon AWS through MariaDB SkySQL, as one of the database options when using Amazon's RDS service, or using a MariaDB AMI on Amazon EC2 from the AWS Marketplace.

MariaDB SkySQL

Cloud database service is available through MariaDB SkySQL on Amazon AWS. MariaDB SkySQL delivers MariaDB with enterprise features for mission-critical workloads. Support is provided directly by MariaDB. Refer to SkySQL Documentation for complete details. Get started to launch a MariaDB database on AWS in minutes.

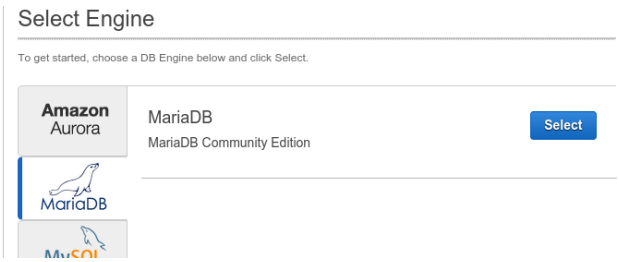
Amazon RDS

To get started with MariaDB on Amazon's RDS service, click on the RDS link in the Database section of the AWS console.



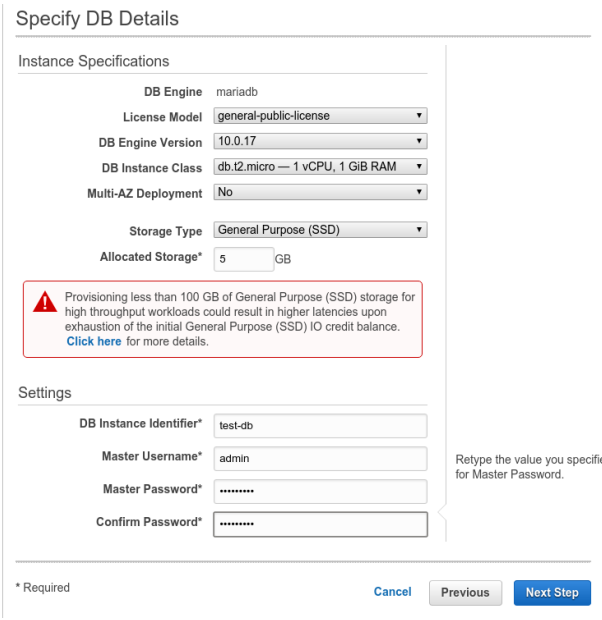
Next, click on the **Get Started Now** button. Alternatively, you can click on the **Launch DB Instance** button from the Instances section of the RDS Dashboard.

In either case, you will be brought to the page where you can select the database engine you want to use. Click on the **MariaDB** logo and then click on the **Select** button.



You will then move to step 2 where you choose whether or not you want to use your MariaDB instance for production or non-production usage. Amazon has links on this page to documentation on the various options.

After selecting the choice you want you will move to step 3 where you specify the details for your database, including setting up an admin user in the database.



You will then move to step 4 where you can configure advanced settings, including security settings, various options, backup settings, maintenance defaults, and so on.

Refer to Amazon's RDS documentation for complete documentation on all the various settings and for information on connecting to and using your RDS MariaDB instances.

AMI on EC2

MariaDB AMIs (Amazon Machine Images) are available in the AWS Marketplace. These AMIs, kept up-to-date with the most recently released versions of MariaDB, are a great way to try out the newest MariaDB versions before they make it to RDS and/or to use MariaDB in a more traditional server environment.

2.1.14 Migrating to MariaDB

Migrating to MariaDB from another DBMS.

Migrating to MariaDB from MySQL

Help with moving from MySQL to MariaDB, features and compatibility

Migrating to MariaDB from SQL Server

Guide to help you migrate from SQL Server to MariaDB.

Migrating to MariaDB from PostgreSQL

Information on migrating from PostgreSQL to MariaDB.

Migrating to MariaDB from Oracle

Help with migrating to MariaDB from Oracle

2.1.14.1 Migrating to MariaDB from MySQL

.

MariaDB versus MySQL - Features

MariaDB advantage in delivering enterprise-level high availability, scalability and security.

MariaDB versus MySQL - Compatibility

Compatibility and differences with MariaDB related to high availability, security and scalability.

Upgrading from MySQL to MariaDB

Upgrading from MySQL to MariaDB.

Incompatibilities and Feature Differences Between MariaDB 11.0 and MySQL 8.0

List of incompatibilities and feature differences between MariaDB 11.0 and MySQL 8.0.

Incompatibilities and Feature Differences Between MariaDB 10.11 and MySQL 8.0
List of incompatibilities and feature differences between MariaDB 10.11 and MySQL 8.0.

Incompatibilities and Feature Differences Between MariaDB 10.10 and MySQL 8.0
List of incompatibilities and feature differences between MariaDB 10.10 and MySQL 8.0.

Incompatibilities and Feature Differences Between MariaDB 10.6 and MySQL 8.0
List of incompatibilities and feature differences between MariaDB 10.6 and MySQL 8.0.

Incompatibilities and Feature Differences Between MariaDB 10.5 and MySQL 8.0
List of incompatibilities and feature differences between MariaDB 10.5 and MySQL 8.0.

Incompatibilities and Feature Differences Between MariaDB 10.4 and MySQL 8.0
List of incompatibilities and feature differences between MariaDB 10.4 and MySQL 8.0.

Function Differences Between MariaDB and MySQL
Functions available only in MariaDB.

System Variable Differences between MariaDB and MySQL
Comparison of variable differences between major versions of MariaDB and MySQL.

Upgrading from MySQL 5.7 to MariaDB 10.2
Following compatibility report was done on 10.2.4 and may get some fixing i...

Installing MariaDB Alongside MySQL
MariaDB was designed as a drop in place replacement for MySQL, but you can ...

Moving from MySQL to MariaDB in Debian 9
MariaDB 10.1 is the default mysql server in Debian 9 "Stretch"

Screencast for Upgrading MySQL to MariaDB
Screencast for upgrading MySQL 5.1.55 to MariaDB

Upgrading to MariaDB From MySQL 5.0 or Older
Upgrading to MariaDB from MySQL 5.0 (or older version)

Incompatibilities and Feature Differences Between MariaDB and MySQL - Unmaintained Series
Incompatibilities and feature differences between unmaintained MariaDB and MySQL series.

There are 1 related questions.

2.1.14.1.1 MySQL vs MariaDB: Performance

Title: MariaDB versus MySQL - Features

See also **MariaDB vs MySQL - Compatibility**

Differences Per Releases

For differences between specific releases, see

- Incompatibilities and Feature Differences Between MariaDB 11.1 and MySQL 8.0
- Incompatibilities and Feature Differences Between MariaDB 11.0 and MySQL 8.0
- Incompatibilities and Feature Differences Between MariaDB 10.11 and MySQL 8.0
- Incompatibilities and Feature Differences Between MariaDB 10.10 and MySQL 8.0
- Incompatibilities and Feature Differences Between MariaDB 10.9 and MySQL 8.0
- Incompatibilities and Feature Differences Between MariaDB 10.8 and MySQL 8.0
- Incompatibilities and Feature Differences Between MariaDB 10.7 and MySQL 8.0

- Incompatibilities and Feature Differences Between MariaDB 10.6 and MySQL 8.0
- Incompatibilities and Feature Differences Between MariaDB 10.5 and MySQL 8.0
- Incompatibilities and Feature Differences Between MariaDB 10.4 and MySQL 8.0
- Incompatibilities and Feature Differences Between MariaDB 10.3 and MySQL 5.7
- Incompatibilities and Feature Differences Between MariaDB 10.2 and MySQL 5.7

For a detailed breakdown of system variable differences, see:

- System variable differences between MariaDB 11.1 and MySQL 8.0
- System variable differences between MariaDB 11.0 and MySQL 8.0
- System variable differences between MariaDB 10.11 and MySQL 8.0
- System variable differences between MariaDB 10.10 and MySQL 8.0
- System variable differences between MariaDB 10.9 and MySQL 8.0
- System variable differences between MariaDB 10.8 and MySQL 8.0
- System variable differences between MariaDB 10.7 and MySQL 8.0
- System variable differences between MariaDB 10.6 and MySQL 8.0
- System variable differences between MariaDB 10.5 and MySQL 8.0
- System variable differences between MariaDB 10.4 and MySQL 8.0
- System variable differences between MariaDB 10.3 and MySQL 8.0
- System variable differences between MariaDB 10.3 and MySQL 5.7
- System variable differences between MariaDB 10.2 and MySQL 5.7
- System variable differences between MariaDB 10.1 and MySQL 5.7
- System variable differences between MariaDB 10.1 and MySQL 5.6
- System variable differences between MariaDB 10.0 and MySQL 5.6
- System variable differences between MariaDB 5.5 and MySQL 5.5

For a detailed breakdown of function differences, see:

- Function Differences Between MariaDB 11.1 and MySQL 8.0
- Function Differences Between MariaDB 11.0 and MySQL 8.0
- Function Differences Between MariaDB 10.11 and MySQL 8.0
- Function Differences Between MariaDB 10.10 and MySQL 8.0
- Function Differences Between MariaDB 10.9 and MySQL 8.0
- Function Differences Between MariaDB 10.8 and MySQL 8.0
- Function Differences Between MariaDB 10.7 and MySQL 8.0
- Function Differences Between MariaDB 10.6 and MySQL 8.0
- Function Differences Between MariaDB 10.5 and MySQL 8.0
- Function Differences Between MariaDB 10.4 and MySQL 8.0
- Function Differences Between MariaDB 10.3 and MySQL 8.0
- Function Differences Between MariaDB 10.3 and MySQL 5.7
- Function Differences Between MariaDB 10.2 and MySQL 5.7

More Storage Engines

In addition to the standard MyISAM, BLACKHOLE, CSV, MEMORY, ARCHIVE, and MERGE storage engines, the following are also included with MariaDB Source and Binary packages:

- ColumnStore, a column oriented storage engine optimized for Data warehousing.
- MyRocks, a storage engine with great compression, in 10.2
- Aria, MyISAM replacement with better caching.
- FederatedX (drop-in replacement for Federated)
- OQGRAPH (In MariaDB 5.2 and later. Disabled in MariaDB 5.5 only.)
- SphinxSE (In MariaDB 5.2 and later)
- CONNECT in MariaDB 10.0 and later.
- SEQUENCE in MariaDB 10.0 and later.
- Spider in MariaDB 10.0 and later.
- TokuDB (In MariaDB 5.5 and later, removed in 10.6)
- Cassandra (In MariaDB 10.0, removed in 10.6)

Speed Improvements

- MariaDB now provides much faster privilege checks for setups with many user accounts or many database
- The new FLUSH SSL command allows SSL certificates to be reloaded without restarting the server
- Many optimizer enhancements in MariaDB 5.3. Subqueries are now finally usable. The complete list and a comparison with MySQL is here. A benchmark can be found here.
- Faster and safer replication: Group commit for the binary log. This makes many setups that use replication and lots of updates more than 2x times faster.
- Parallel replication *— new in 10.0*
- Improvements for InnoDB asynchronous IO subsystem on Windows.

- Indexes for the MEMORY(HEAP) engine are faster. According to a simple test, 24% faster on INSERT for integer index and 60% faster for index on a CHAR(20) column. Fixed in MariaDB 5.5 and MySQL 5.7.
- Segmented Key Cache for MyISAM. Can speed up MyISAM tables with up to 4x — *new in 5.2*
- Adjustable hash size for MyISAM and Aria. This can greatly improve shutdown time (from hours to minutes) if using a lot of MyISAM/Aria tables with delayed keys — *new in 10.0.13*
- CHECKSUM TABLE is faster.
- We improved the performance of character set conversions (and removed conversions when they were not really needed). Overall speed improvement is 1-5 % (according to sql-bench) but can be higher for big result sets with all characters between 0x00-0x7f.
- Pool of Threads in MariaDB 5.1 and even better in MariaDB 5.5. This allows MariaDB to run with 200,000+ connections and with a notable speed improvement when using many connections.
- Several speed improvements when a client connects to MariaDB. Many of the improvements were done in MariaDB 10.1 and MariaDB 10.2.
- There are some improvements to the DBUG code to make its execution faster when debug is compiled in but not used.
- Our use of the Aria storage engine enables faster complex queries (queries which normally use disk-based temporary tables). The Aria storage engine is used for internal temporary tables, which should give a speedup when doing complex selects. Aria is usually faster for temporary tables when compared to MyISAM because Aria caches row data in memory and normally doesn't have to write the temporary rows to disk.
- The test suite has been extended and now runs much faster than before, even though it tests more things.

Extensions & New Features

We've added a lot of new features to MariaDB. If a patch or feature is useful, safe, and stable — we make every effort to include it in MariaDB. The most notable features are:

- Support introduced for System-versioned tables. Allows queries to access both current and historic data, aiding in managing retention, analysis and point-in-time recovery. — *new in 10.3*
- ALTER TABLE... DROP COLUMN can now run as Instant operations. Can also now change the ordering of columns. — *new in 10.4*
- Support introduced for password expiration, using the user password expiry — *new in 10.4*
- In order to support the use of multiple authentication plugins for a single user, the `mysql.user` system table has been retired in favor of the `mysql.glob_priv` system table. — *new in 10.4*
- The unix_socket authentication plugin is now the default on Unix-like systems. This represents a major change to authentication in MariaDB — *new in 10.4*
- Support introduced for Optimizer Trace, which provides detailed information on how the Optimizer processes queries. To enable Optimizer Trace, set the `optimizer_trace` system variable — *new in 10.4*
- The MariaDB SQL/PL stored procedure dialect (enabled with sql_mode=ORACLE) now supports Oracle style packages. Support for the following statements are available: CREATE PACKAGE, CREATE PACKAGE BODY, DROP PACKAGE, DROP PACKAGE BODY, SHOW CREATE PACKAGE, SHOW CREATE PACKAGE BODY — *new in 10.3*
- Automatic collection of Engine Independent Table Statistics — *new in 10.4*
- Support for the use of parentheses (brackets) for specifying precedence in the ordering of execution for `SELECT` statements and Table Value Operations, (including the use of UNION, EXCEPT, INTERSECT operations) — *new in 10.4*
- Support for anchored data types added to local stored procedure variables. — *new in 10.3*
- Support added for Stored Aggregate functions — *new in 10.3*
- Oracle compatible SUBSTR() function is available — *new in 10.3*
- Oracle compatible SEQUENCE support is provided — *new in 10.3*
- Support for anchored data types added to stored routine variables — *new in 10.3*
- Support for anchored data types added to stored routine parameters — *new in 10.3*
- Cursors with parameters are now supported — *new in 10.3*
- INVISIBLE columns are now supported — *new in 10.3*
- Instant ADD COLUMN is now available for InnoDB — *new in 10.3*
- Window functions are supported — *new in 10.2*
- Number of supported decimals in DECIMAL has increased from `30` to `38` — *new in 10.2*
- Recursive Common Table Expressions — *new in 10.2*
- New WITH statement. `WITH` is a common table expression that allows one to refer to a subquery expression many times in a query — *new in 10.2*
- CHECK CONSTRAINT — *new in 10.2*
- DEFAULT expression, including `DEFAULT` for `BLOB` and `TEXT` — *new in 10.2*
- Added catchall for list partitions — *new in 10.2*
- Oracle-style EXECUTE IMMEDIATE statement — *new in 10.2*
- Several new JSON functions — *new in 10.2*
- Microsecond Precision in Processlist
- Table Elimination
- Virtual Columns — *new in 5.2*
- Microseconds in MariaDB — *new in 5.3*

- Extended User Statistics — *new in 5.2*
- KILL all queries for a user — *new in 5.3*,
- KILL QUERY ID - terminates the query by query_id, leaving the connection intact — *new in 10.0.5*,
- Pluggable Authentication — *new in 5.2*
- Storage-engine-specific CREATE TABLE — *new in 5.2*
- Enhancements to INFORMATION SCHEMA.PLUGINS table — *new in 5.2*
- Group commit for the binary log. This makes replication notably faster! — *new in 5.3*
- Added `--rewrite-db` mysqlbinlog option to change the used database — *new in 5.2*
- Progress reporting for `ALTER TABLE` and `LOAD DATA INFILE` — *new in 5.3*
- Faster joins and subqueries — *new in 5.3*
- HandlerSocket and faster HANDLER calls — *new in 5.3*
- Dynamic Columns support — *new in 5.3*
- GIS Functionality — *new in 5.3*
- Multi-source replication — *new in 10.0*
- Global Transaction ID — *new in 10.0*
- SHOW EXPLAIN gives the EXPLAIN plan for a query running in another thread. — *new in 10.0*
- Roles — *new in 10.0*
- PCRE Regular Expressions (including `REGEXP_REPLACE()`) — *new in 10.0*
- CREATE OR REPLACE
- DELETE ... RETURNING — *new in 10.0*
- MariaDB supports more collations than MySQL.

For a full list, please see features for each release

Better Testing

- More tests in the test suite.
- Bugs in tests fixed.
- Test builds with different configure options to get better feature testing.
- Remove invalid tests. (e.g. don't test feature "X" if that feature is not in the tested build)

Fewer Warnings and Fewer Bugs

- Bugs are bad. Fix as many bugs as possible and try to not introduce new ones.
- Compiler warnings are also bad. Eliminate as many compiler warnings as possible.

Truly Open Source

- All code in MariaDB is released under GPL, LGPL or BSD.
- MariaDB does not have closed source modules like the ones that can be found in MySQL Enterprise Edition. In fact, all the closed source features in MySQL 5.5 Enterprise Edition are found in the MariaDB open source version.
- MariaDB client libraries (for C, for Java (JDBC), for Windows (ODBC)) are released under LGPL to allow linking with closed source software. MySQL client libraries are released under GPL that does not allow linking with closed source software.
- MariaDB includes test cases for all fixed bugs. Oracle doesn't provide test cases for new bugs fixed in MySQL 5.5.
- All bugs and development plans are public.
- MariaDB is developed by the community in true open source spirit.

Related Links

- Compatiblity between MariaDB and MySQL
- Moving from MySQL
- Troubleshooting Installation Issues

2.1.14.1.2 MariaDB versus MySQL: Compatibility

> See also **MariaDB vs MySQL - Features**

Contents

Replacement for MySQL

Until MariaDB 5.5, MariaDB versions functioned as a "drop-in replacement" for the equivalent MySQL version, with some limitations. From MariaDB 10.0, it is usually still very easy to upgrade from MySQL.

- MariaDB's data files are generally binary compatible with those from the equivalent MySQL version.
  - All filenames and paths are generally the same.
  - Data and table definition files (.frm) files are binary compatible.
  - See note below for an incompatibility with views!
- MariaDB's client protocol is binary compatible with MySQL's client protocol.
  - All client APIs and structs are identical.
  - All ports and sockets are generally the same.
  - All MySQL connectors (PHP, Perl, Python, Java, .NET, MyODBC, Ruby, MySQL C connector etc) work unchanged with MariaDB.
  - There are some installation issues with PHP5 that you should be aware of (a bug in how the old PHP5 client checks library compatibility).

This means that for many cases, you can just uninstall MySQL and install MariaDB and you are good to go. There is not generally any need to convert any data files.

However, you must still run mysql_upgrade to finish the upgrade. This is needed to ensure that your mysql privilege and event tables are updated with the new fields MariaDB uses.

That said, MariaDB has a lot of new options, extension, storage engines and bug fixes that are not in MySQL. You can find the feature set for the different MariaDB versions on the What is in the different MariaDB Releases page.

Drop-in Compatibility of Specific MariaDB Versions

MariaDB 10.2, MariaDB 10.3, and MariaDB 10.4 function as limited drop-in replacements for MySQL 5.7, as far as InnoDB is concerned. However, the implementation differences continue to grow in each new MariaDB version.

MariaDB 10.0 and MariaDB 10.1 function as limited drop-in replacements for MySQL 5.6, as far as InnoDB is concerned. However, there are some implementation differences in some features.

MariaDB 5.5 functions as a drop-in replacement for MySQL 5.5.

MariaDB 5.1, MariaDB 5.2, and MariaDB 5.3 function as drop-in replacements for MySQL 5.1.

Replication Compatibility

Replication compatibility depends on:

- The MariaDB Server version
- The MySQL Server version
- The role of each server

Replication compatibility details are described below for each MySQL version that is still maintained.

For replication compatibility details between MariaDB versions, see Cross-Version Replication Compatibility.

MySQL 5.7

MariaDB Server 10.2 and later can replicate from a MySQL 5.7 primary server.

MariaDB Server does not support the MySQL implementation of Global Transaction IDs (GTIDs), so the MariaDB replica server must use the binary log file and position for replication. If GTID mode is enabled on the MySQL primary server, the MariaDB replica server will remove the MySQL GTID events and replace them with MariaDB GTID events.

Although MariaDB Server and MySQL 5.7 are compatible at the replication level, they may have some incompatibilities at the SQL (detailed below). Those differences can cause replication failures in some cases. To decrease the risk of compatibility issues, it is recommended to set binlog_format to `ROW`. When you want to replicate from MySQL 5.7 to MariaDB Server, it is recommended to test your application, so that any compatibility issues can be found and fixed.

MariaDB can't make any claims about whether a MySQL 5.7 replica server can replicate from a MariaDB primary server.

MySQL 8.0

MariaDB Server cannot replicate from a MySQL 8.0 primary server, because MySQL 8.0 has a binary log format that is incompatible.

Incompatibilities between Currently Maintained MariaDB Versions and MySQL

Incompatibilities between MariaDB 11.1 and MySQL 8.0

- See Incompatibilities and Feature Differences Between MariaDB 11.1 and MySQL 8.0 for details.
- Function Differences Between MariaDB 11.1 and MySQL 8.0
- System Variable Differences Between MariaDB 11.1 and MySQL 8.0

Incompatibilities between MariaDB 11.0 and MySQL 8.0

- See Incompatibilities and Feature Differences Between MariaDB 11.0 and MySQL 8.0 for details.
- Function Differences Between MariaDB 11.0 and MySQL 8.0
- System Variable Differences Between MariaDB 11.0 and MySQL 8.0

Incompatibilities between MariaDB 10.11 and MySQL 8.0

- See Incompatibilities and Feature Differences Between MariaDB 10.11 and MySQL 8.0 for details.
- Function Differences Between MariaDB 10.11 and MySQL 8.0
- System Variable Differences Between MariaDB 10.11 and MySQL 8.0

Incompatibilities between MariaDB 10.10 and MySQL 8.0

- See Incompatibilities and Feature Differences Between MariaDB 10.10 and MySQL 8.0 for details.
- Function Differences Between MariaDB 10.10 and MySQL 8.0
- System Variable Differences Between MariaDB 10.10 and MySQL 8.0

Incompatibilities between MariaDB 10.6 and MySQL 8.0

- See Incompatibilities and Feature Differences Between MariaDB 10.6 and MySQL 8.0 for details.
- Function Differences Between MariaDB 10.6 and MySQL 8.0
- System Variable Differences Between MariaDB 10.6 and MySQL 8.0

Incompatibilities between MariaDB 10.5 and MySQL 8.0

- See Incompatibilities and Feature Differences Between MariaDB 10.5 and MySQL 8.0 for details.
- Function Differences Between MariaDB 10.5 and MySQL 8.0
- System Variable Differences Between MariaDB 10.5 and MySQL 8.0

Incompatibilities between MariaDB 10.4 and MySQL 8.0

- See Incompatibilities and Feature Differences Between MariaDB 10.4 and MySQL 8.0 for details.
- Function Differences Between MariaDB 10.4 and MySQL 8.0
- System Variable Differences Between MariaDB 10.4 and MySQL 8.0

Incompatibilities between Unmaintained MariaDB Versions and MySQL

Incompatibilities between MariaDB 10.9 and MySQL 8.0

- Function Differences Between MariaDB 10.9 and MySQL 8.0
- System Variable Differences Between MariaDB 10.9 and MySQL 8.0

Incompatibilities between MariaDB 10.8 and MySQL 8.0

- See Incompatibilities and Feature Differences Between MariaDB 10.8 and MySQL 8.0 for details.
- Function Differences Between MariaDB 10.8 and MySQL 8.0
- System Variable Differences Between MariaDB 10.8 and MySQL 8.0

Incompatibilities between MariaDB 10.7 and MySQL 8.0

- See Incompatibilities and Feature Differences Between MariaDB 10.7 and MySQL 8.0 for details.
- Function Differences Between MariaDB 10.7 and MySQL 8.0
- System Variable Differences Between MariaDB 10.7 and MySQL 8.0

Incompatibilities between MariaDB 10.3 and MySQL 5.7

- See Incompatibilities and Feature Differences Between MariaDB 10.3 and MySQL 5.7 for details.
- Function Differences Between MariaDB 10.3 and MySQL 5.7
- System Variable Differences Between MariaDB 10.3 and MySQL 5.7

Incompatibilities between MariaDB 10.2 and MySQL 5.7

- See Incompatibilities and Feature Differences Between MariaDB 10.2 and MySQL 5.7 for details.
- System Variable Differences Between MariaDB 10.2 and MySQL 5.7

Incompatibilities between MariaDB 10.1 and MySQL 5.7

- MariaDB 10.1 and above does not support MySQL 5.7's packed JSON objects. MariaDB follows the SQL standard and stores the JSON as a normal TEXT/BLOB. If you want to replicate JSON columns from MySQL to MariaDB, you should store JSON objects in MySQL in a TEXT column or use statement based replication. If you are using JSON columns and want to upgrade to MariaDB, you can either convert the JSON columns to TEXT or use mysqldump to copy these tables to MariaDB. In MySQL, JSON is compared according to json values. In MariaDB JSON strings are normal strings and compared as strings.
- MariaDB 10.1's InnoDB encryption is implemented differently than MySQL 5.7's InnoDB encryption.
- MariaDB 10.1 does not support the ngram and MeCab full-text parser plugins - MDEV-10267, MDEV-10268.
- MariaDB 10.1 does not support multiple triggers for a table - MDEV-6112. This is fixed in MariaDB 10.2
- MariaDB 10.1 does not support CREATE TABLESPACE for InnoDB.
- MariaDB 10.1 does not support MySQL 5.7's “native” InnoDB partitioning handler. Fixed in MariaDB 10.6.15.
- MariaDB does not support MySQL 5.7's X protocol.
- MariaDB 10.1 does not support the use of multiple triggers of the same type for a table. This feature was introduced in MariaDB 10.2.2.
- MariaDB 10.1 does not support MySQL 5.7's transportable tablespaces for partitioned InnoDB tables. ALTER TABLE ... {DISCARD|IMPORT} PARTITION is not supported. For a workaround see the following blog post.
- MariaDB 10.1 does not support MySQL 5.7's online undo tablespace truncation. However, this feature was added to

MariaDB 10.2.

- MySQL 5.7 features a new implementation of the `performance_schema` and a `sys` schema wrapper. These are not yet supported in MariaDB.
- MySQL 5.7 adds multi-source replication and replication channels. Multi-source replication was added to MariaDB previously, in MariaDB 10.0, and uses a different syntax.
- MySQL 5.7 adds group replication. This feature is incompatible with MariaDB's galera-cluster replication.
- MariaDB 10.1 does not support MySQL 5.7's, `ACCOUNT LOCK/UNLOCK` synax for `CREATE USER` and `ALTER USER` statements.
- MariaDB 10.1 does not support MySQL 5.7's `ALTER TABLE...RENAME INDEX` statements.
- MariaDB 10.1 does not support MySQL 5.7's `STACKED` operation for `GET DIAGNOSTICS` statements.
- MariaDB 10.1 does not support MySQL 5.7's `{WITH|WITHOUT} VALIDATION` syntax for `ALTER TABLE.. EXCHANGE PARTITION` statements.
- MariaDB does not support the optional *init_vector* argument for AES_ENCRYPT and AES_DECRYPT or the block_encryption_mode variable - MDEV-9069
- MariaDB does not support the `--initialize` option. Use `mysql_install_db` instead. - MDEV-19010
- Also see Incompatibilities between MariaDB 10.0 and MySQL 5.6.
- Also see a detailed breakdown of System variable differences between MariaDB 10.1and MySQL 5.7.

Incompatibilities between MariaDB 10.0 and MySQL 5.6

- MySQL does not support MariaDB's Spider Storage Engine.
- All MySQL binaries (`mysqld` , myisamchk etc.) give a warning if one uses a prefix of an option (such as `--big-table` instead of `--big-tables`). MariaDB binaries work in the same way as most other Unix commands and don't give warnings when using unique prefixes.
- MariaDB GTID is not compatible with MySQL 5.6. This means that one can't have MySQL 5.6 as a slave for MariaDB 10.0. However MariaDB 10.0 can be a slave of MySQL 5.6 or any earlier MySQL/MariaDB version. Note that MariaDB and MySQL also have different GTID system variables, so these need to be adjusted when migrating.
- MariaDB 10.0 multi-source replication is not supported in MySQL 5.6.
- To make CREATE TABLE ... SELECT work the same way in statement based and row based replication it's by default executed as CREATE OR REPLACE TABLE on the slave. One benefit of this is that if the slave dies in the middle of CREATE ... SELECT it will be able to continue.
    - One can use the slave-ddl-exec-mode variable to specify how `CREATE TABLE` and `DROP TABLE` is replicated.
- See also a detailed breakdown of System variable differences between MariaDB 10.0 and MySQL 5.6.
- MySQL 5.6 has performance schema enabled by default. For performance reasons MariaDB 10.0 has it disabled by default. You can enable it by starting `mysqld` with the option `--performance-schema` .
- MariaDB 10.0 does not support the MySQL Memcached plugin. However, data stored using memcached can be retrieved because the data is stored as InnoDB tables. MariaDB is able to start successfully with an error message of not being able to find libmemcached.so library.
- Users created with MySQL's SHA256 password algorithm cannot be used in MariaDB 10.0 as MariaDB does not include MySQL's `sha256_password` plugin.
- MariaDB 10.0 does not support delayed replication - MDEV-7145.
- Also see a detailed breakdown of System variable differences between MariaDB 10.0 and MySQL 5.6.
- The low-level temporal format used by TIME, DATETIME and TIMESTAMP is different in MySQL 5.6 and MariaDB 10.0. (In MariaDB 10.1, the MySQL implementation is used by default - see mysql56_temporal_format.)
- MariaDB implements some changes in the SQL query optimizer over what's available in MySQL. This can result in `EXPLAIN` statements showing different plans.
- MySQL delayed replication, (through `MASTER_DELAY`), is not supported in MariaDB 10.0, it was implemented in MariaDB 10.2.5
- MariaDB does not support the optional *init_vector* argument for AES_ENCRYPT and AES_DECRYPT or the block_encryption_mode variable - MDEV-9069

Incompatibilities between MariaDB 5.5 and MySQL 5.5

- **Views with definition ALGORITHM=MERGE or ALGORITHM=TEMPTABLE got accidentally swapped between MariaDB and MySQL! You have to re-create views created with either of these definitions!**
- INSERT IGNORE also gives warnings for duplicate key errors. You can turn this off by setting `OLD_MODE=NO_DUP_KEY_WARNINGS_WITH_IGNORE` (see OLD_MODE).
- Before MariaDB 5.5.31, `X'HHHH'` , the standard SQL syntax for binary string literals, erroneously worked in the same way as `0xHHHH` , which could work as a number or string depending on the context. In 5.5.31 this was fixed to behave as a string in all contexts (and never as a number), introducing an incompatibility with previous versions of MariaDB, and all versions of MySQL. See CAST and Hexadecimal Literals for more details and examples.
- MariaDB dynamic columns are not supported by MySQL.
- MariaDB virtual columns are not supported by MySQL.
- MariaDB's HandlerSocket plugin is not supported by MySQL.
- MariaDB's Cassandra Storage Engine is not supported by MySQL.

- As of MariaDB 5.5.35, EXTRACT (HOUR FROM ...) adheres to the SQL standard and returns a result from 0 to 23. In MySQL, and earlier versions of MariaDB, the result can be greater than 23.
- See also a detailed breakdown of System variable differences between MariaDB 5.5 and MySQL 5.5.

Incompatibilities between MariaDB 5.3 and MySQL 5.1

- **Views with definition ALGORITHM=MERGE or ALGORITHM=TEMPTABLE got accidentally swapped between MariaDB 5.2 and MariaDB 5.3! You have to re-create views created with either of these definitions!**
- A few error messages related to wrong conversions are different as MariaDB provides more information in the message about what went wrong.
- Error numbers for MariaDB-specific errors have been moved to start from 1900 so as not to conflict with MySQL errors.
- Microseconds now work in all contexts; MySQL, in some contexts, lost the microsecond part from datetime and time.
- UNIX_TIMESTAMP(constant-date-string) returns a timestamp with 6 decimals in MariaDB while MySQL returns it without a decimal. This can cause a problem if you are using UNIX_TIMESTAMP() as a partitioning function. You can fix this by using FLOOR(UNIX_TIMESTAMP(..)) or changing the date string to a date number, like 20080101000000.
- MariaDB performs stricter checking of date, datetime and timestamp values. For example UNIX_TIMESTAMP('x') now returns NULL instead of 0.
- The old `--maria-` startup options are removed. You should use the `--aria-` prefix instead. (MariaDB 5.2 supports both `--maria-` and `--aria-`)
- `SHOW PROCESSLIST` has an extra `Progress` column which shows progress for some commands. You can disable it by starting `mysqld` with either `--old-mode=NO_PROGRESS_INFO` or with the `--old` flag (see OLD_MODE).
- `INFORMATION_SCHEMA.PROCESSLIST` has three new columns for progress reporting: `STAGE` , `MAX_STAGE` , and `PROGRESS` .
- Long comments which start with `/*M!` or `/*M!#####` are executed.
- If you use max_user_connections=0 (which means any number of connections) when starting mysqld, you can't change the global variable anymore while mysqld remains running. This is because when mysqld is started with `max_user_connections=0` it does not allocate counting structures (which also involve a mutex for each connection). This would lead to wrong counters if you later changed the variable. If you want to be able to change this variable at runtime, set it to a high value at startup.
- You can set `max_user_connections` (both the global variable and the `GRANT` option) to `-1` to stop users from connecting to the server. The global `max_user_connections` variable does not affect users with the `SUPER` privilege.
- The IGNORE directive does not ignore all errors (like fatal errors), only things that are safe to ignore.

Incompatibilities between MariaDB 5.2 and MySQL 5.1

The list is the same as between MariaDB 5.1 and MySQL 5.1, with one addition:

- A new SQL_MODE value was added: `IGNORE_BAD_TABLE_OPTIONS` . If it is not set, using a table, field, or index attribute (option) that is not supported by the chosen storage engine will cause an error. This change might cause warnings in the error log about incorrectly defined tables from the `mysql` database, fix that with mysql_upgrade.

For all practical purposes, MariaDB 5.2 is a drop in replacement for MariaDB 5.1 and MySQL 5.1.

Incompatibilities between MariaDB 5.1 and MySQL 5.1

In some few cases MariaDB has to be incompatible to allow MariaDB to provide more and better information than MySQL.

Here is the list of all known user level incompatibilities you may see when using MariaDB 5.1 instead of MySQL 5.1.

- The installation package names start with MariaDB instead of MySQL.
- Timings may be different as MariaDB is in many cases faster than MySQL.
- mysqld in MariaDB also reads the `[mariadb]` sections of your my.cnf files.
- You can't use a binary only storage engine library with MariaDB if it's not compiled for exactly the same MariaDB version. (This is because the server internal structure THD is different between MySQL and MariaDB. This is common also between different MySQL versions). This should not be a problem as most people don't load new storage engines and MariaDB comes with more storage engines than MySQL.
- `CHECKSUM TABLE` may give different result as MariaDB doesn't ignore NULL's in the columns as MySQL 5.1 does (Future MySQL versions should calculate checksums the same way as MariaDB). You can get the 'old style' checksum in MariaDB by starting mysqld with the `--old` option. Note however that that the MyISAM and Aria storage engines in MariaDB are using the new checksum internally, so if you are using `--old` , the `CHECKSUM` command will be slower as it needs to calculate the checksum row by row.
- The slow query log has more information about the query, which may be a problem if you have a script which parses the slow query log.
- MariaDB by default takes a bit more memory than MySQL because we have by default enabled the Aria storage engine for handling internal temporary tables. If you need MariaDB to take very little memory (at the expense of

performance), you can set the value of `aria_pagecache_buffer_size` to `1M` (the default is `128M`).
- If you are using new command options, new features of MariaDB or new storage engines, you can't move easily back and forth between MySQL and MariaDB anymore.

Old, Unsupported Configuration Options

If you are using any of the following options in your `/etc/my.cnf` or other `my.cnf` file you should remove them. This is also true for MySQL 5.1 or newer:

- `skip-bdb`

Replacing a MySQL RPM

If you uninstalled a MySQL RPM to install MariaDB, note that the MySQL RPM on uninstall renames `/etc/my.cnf` to `/etc/my.cnf.rpmsave`.

After installing MariaDB you should do the following to restore your old configuration options:

```
mv -vi /etc/my.cnf.rpmsave /etc/my.cnf
```

Incompatibilities between MariaDB and MySQL-Proxy

A MySQL client API is able to connect to MariaDB using MySQL-Proxy but a MariaDB client API will receive progress reporting informations that MySQL-Proxy does not implement, to get full compatibility in all case just disable progress reporting on the client or server side.

Another option is to use the MariaDB MaxScale proxy, that works with both MySQL and MariaDB.

Related Links

- MariaDB vs MySQL - Features
- Moving from MySQL to MariaDB
- Troubleshooting Installation Issues
- Projects and applications that works with MariaDB

2.1.3.12.1 Upgrading from MySQL to MariaDB

2.1.14.1.4 Incompatibilities and Feature Differences Between MariaDB 11.1 and MySQL 8.0

Contents

MariaDB maintains high levels of compatibility with MySQL, and most applications that use MySQL will work seamlessly with MariaDB. However, take note of the following incompatibilities and feature differences between MariaDB 11.1 and MySQL 8.0. It is based on the versions MySQL 8.0.34 and MariaDB 11.1.2. Note that MySQL 8 is an 'evergreen' release, so features may be added or removed in later releases.

Storage Engines

In addition to the standard InnoDB, MyISAM, BLACKHOLE, CSV, MEMORY, ARCHIVE, and MERGE storage engines, the following are also available with MariaDB 11.1:

- ColumnStore utilizes a massively parallel distributed data architecture and is designed for big data scaling to process petabytes of data.
- MyRocks, a storage engine with great compression
- S3 storage engine allows one to archive MariaDB tables in Amazon S3, or any third-party public or private cloud that

implements S3 API.
- Aria, MyISAM replacement with better caching.
- CONNECT
- SEQUENCE
- Spider
- SphinxSE
- FederatedX (drop-in replacement for Federated)
- OQGRAPH

Extensions and New Features

The most notable features available in MariaDB, but not in MySQL, are:

- Galera is a standard part of MariaDB Server.
- Temporal data tables in the form of:
  - System-versioned tables (allow you to query and operate on historic data).
  - Application-time periods (allow you to query and operate on a temporal range of data), including the WITHOUT OVERLAPS clause.
  - Bitemporal tables (which combine both system-versioning and application-time periods).
- DML-only flashback, allowing instances, databases or tables to be rolled back to an old snapshot.
- Oracle compatibility mode
- MariaDB supports localization in a number of additional languages: Bulgarian, Chinese, Georgian, Hindi, Serbian, and Ukrainian.
- MariaDB has made major improvements to the optimizer.
- Sequences
- Table Value Constructors
- Dynamic Columns support
- Semi-sync plugin merged into the server
- OR REPLACE syntax for CREATE statements, such as CREATE OR REPLACE TABLE, CREATE OR REPLACE DATABASE, etc
- DELETE ... RETURNING, INSERT ... RETURNING, REPLACE ... RETURNING
- GRANT to PUBLIC - MDEV-5215 (blog post)
- WAIT syntax for setting the lock wait timeout.
- UUID data type for storing UUIDs.
- INET6 and INET4 data types for storing IPv6 and IPv4 addresses.
- SUPER privileges made more granular.
- PROXY protocol support
- Multiple compression algorithms available as plugins
- Number of supported decimals in DECIMAL has increased from `30` to `38`
- Added catchall for list partitions
- Oracle-style EXECUTE IMMEDIATE statement
- Lots of new JSON functions
- Microsecond Precision in Processlist
- Table Elimination
- Virtual Columns
- Extended User Statistics
- KILL all queries for a user
- Storage-engine-specific CREATE TABLE
- MariaDB supports more collations (506) than MySQL (266).
- FLUSH SSL command to reload SSL certificates without server restart.
- `IF NOT EXISTS` clause added to INSTALL PLUGIN and `IF EXISTS` clause added to UNINSTALL PLUGIN and UNINSTALL SONAME
- Enhancements to INFORMATION SCHEMA.PLUGINS table
- Group commit for the binary log. This makes replication notably faster!
- The binary log in MariaDB can be compressed.
- BACKUP STAGE allows one to implement very efficient backups with minimal locking.
- Progress reporting for ALTER TABLE and LOAD DATA INFILE
- SHOW EXPLAIN gives the EXPLAIN plan for a query running in another thread. MySQL introduced the EXPLAIN FOR CONNECTION syntax to do the same thing.
- PCRE Regular Expressions (including REGEXP_REPLACE())
- HandlerSocket and faster HANDLER calls
- MySQL 8 does not support PROCEDURE ANALYSE
- MySQL 8 does not support the use of `\N` as an alias for NULL

Incompatibilities

When moving from MySQL 8.0 to MariaDB 11.1, please take note of the following incompatibilities:

- For a list of function differences, see Function Differences Between MariaDB 11.1 and MySQL 8.0
- For a list of system variable differences, see System Variable Differences Between MariaDB 11.1 and MySQL 8.0
- MariaDB does not support MySQL's SET PERSIST - MDEV-16228
- MariaDB's GTID is not compatible with MySQL's. Note that MariaDB and MySQL also have different GTID system variables, so these need to be adjusted when migrating.
- The unix_socket authentication plugin is now default on Unix-like systems, which is a major change to authentication in MariaDB. See Authentication from MariaDB 10.4 for an overview of the changes.
- All mysql* binaries are now named mariadb* (the previous mysql named is retained as a symlink for compatibility purposes)
- Not all character sets and collations are supported across both MySQL and MariaDB. As of 11.1, MariaDB supports 40 character sets and 506 collations . As of 8.0.34, MySQL supports 41 character sets (`gb18030` being the additional one - MDEV-7495) and 286 collations.
- MariaDB indicates collation pad status as part of the name (e.g. `utf8mb3_unicode_nopad_ci`), while MySQL indicates pad status by means of an extra column in SHOW COLLATION.
- To make CREATE TABLE ... SELECT work the same way in statement based and row based replication it's by default executed as CREATE OR REPLACE TABLE on the replica. One benefit of this is that if the replica dies in the middle of CREATE ... SELECT it will be able to continue.
  - One can use the slave-ddl-exec-mode variable to specify how `CREATE TABLE` and `DROP TABLE` is replicated.
- Users created with MySQL's SHA256 password algorithm cannot be used in MariaDB 11.1 - MDEV-9804.
- MariaDB 11.1 does not support Lateral Derived Tables - MDEV-19078.
- MariaDB 11.1 does not support CIDR notation for user accounts - MDEV-25515.
- MariaDB stores JSON as true text, not in binary format as MySQL. MariaDB's JSON functions are much faster than MySQL's so there is no need to store in binary format, which would add complexity when manipulating JSON objects.
- For the same reason, MariaDB's JSON data type is an alias for LONGTEXT. If you want to replicate JSON columns from MySQL to MariaDB, you should store JSON objects in MySQL in a TEXT or LONGTEXT column or use statement-based replication. If you are using JSON columns and want to upgrade to MariaDB, use the mysql_json plugin to automatically convert MySQL JSON to TEXT, or alternatively you need to either convert them to TEXT or use mysqldump to copy these tables to MariaDB.
- In MySQL, JSON is compared according to json values. In MariaDB JSON strings are normal strings and compared as strings.
- MariaDB 11.1 does not support MySQL's JSON operators (`->` and `->>`) - MDEV-13594
- MariaDB 11.1 supports the standard by producing null and a warning for JSON_SEARCH when given invalid data, while MySQL produces an error.
- Roles
  - MariaDB never allows authentication via roles, while MySQL permits this.
  - MySQL permits activating multiple roles at the same time. MariaDB can achieve the same result by creating an intermediate aggregate role.
  - In the INFORMATION_SCHEMA.ENABLED_ROLES table, MySQL reports just the direct list of enabled roles, while MariaDB reports the enabled role, plus the effective inherited roles.
  - MySQL extends the INFORMATION_SCHEMA.APPLICABLE_ROLES table .
  - MySQL includes the tables INFORMATION_SCHEMA.ROLE_TABLE_GRANTS, INFORMATION_SCHEMA.ROLE_ROUTINE_GRANTS, INFORMATION_SCHEMA.ROLE_COLUMN_GRANTS, and INFORMATION_SCHEMA ADMINISTRABLE_ROLE_AUTHORIZATIONS.
- MySQL has the performance schema enabled by default. For performance reasons MariaDB 11.1 has it disabled by default. You can enable it by starting `mariadbd` with the option `--performance-schema` .
- MariaDB has removed the InnoDB Change Buffer.
- In MariaDB 11.1, using FLUSH TABLES without any table list will only close tables not in use, and tables not locked by the FLUSH TABLES connection. If there are no locked tables, FLUSH TABLES will be instant and will not cause any waits, as it no longer waits for tables in use. When a table list is provided, the server will wait for the end of any transactions that are using the tables. In MySQL, FLUSH TABLES only waits for the statements to complete.
- MariaDB binaries (`mariadbd` , myisamchk etc.) give a warning if one uses a unique prefix of an option (such as `--big-table` instead of `--big-tables`). MySQL binaries require the full option name.
- MariaDB 11.1 implements InnoDB encryption in a different way to MySQL 8.0.
- MySQL's implementation of aborting statements that exceed a certain time to execute can only kill SELECTs, while MariaDB's can kill any queries (excluding stored procedures).
- MariaDB 11.1 does not support MySQL's `SELECT /*+ MAX_EXECUTION_TIME(n) */ ...` - see Aborting Statements that Exceed a Certain Time to Execute.
- MySQL 8.0 does not support the Query Cache.
- MariaDB 11.1 does not support the MySQL Memcached plugin (which has been deprecated in MySQL 8.0). However, data stored using memcached can be retrieved because the data is stored as InnoDB tables. MariaDB is able to start successfully with an error message of not being able to find libmemcached.so library.
- In MySQL, `X'HHHH'` , the standard SQL syntax for binary string literals, erroneously works in the same way as `0xHHHH` , which could work as a number or string depending on the context. In MariaDB, this has been fixed to behave as a string in all contexts (and never as a number). See CAST and Hexadecimal Literals for more details and

examples.
- In MariaDB 11.1, SHOW CREATE TABLE does not quote the DEFAULT value of an integer. MariaDB 10.2 and earlier, and MySQL, do. Since MariaDB can support defaults for BLOB and TEXT fields, while MySQL does not, SHOW CREATE TABLE will also append `DEFAULT NULL` where no default is explicitly provided to nullable BLOB or TEXT fields in MariaDB.
- As a result of implementing Table Value Constructors, the VALUES function has been renamed to VALUE().
- MariaDB's NOWAIT supports SELECT statements, LOCK TABLES and various DDL statements, while MySQL's NOWAIT only supports SELECT.
- MariaDB's NOWAIT cannot be added on views and stored procedures while MySQL's can - MDEV-25247
- MariaDB does not support RENAME table while it is write-locked - MDEV-30814
- MariaDB returns an `ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction` when unable to lock within the time, while MySQL returns `ERROR 3572 (HY000): Statement aborted because lock(s) could not be acquired immediately and NOWAIT is set`
- MariaDB does not support the optional *init_vector* argument for AES_ENCRYPT and AES_DECRYPT or the block_encryption_mode variable - MDEV-9069
- MariaDB does not support the `--initialize` option. Use mariadb-install-db instead. - MDEV-19010
- MariaDB 11.1 does not support the ngram and MeCab full-text parser plugins - MDEV-10267, MDEV-10268.
- MariaDB 11.1 does not support the MySQL X plugin.
- MariaDB 11.1 does not support CREATE TABLESPACE for InnoDB.
- The MySQL 8.0 and MariaDB 11.1 INFORMATION_SCHEMA.COLUMNS table contain slightly different fields.
- MariaDB 11.1 client executables allow the connection protocol to be forced by specifying only connection properties on the command-line. See mariadb Command-line client
- The MySQL binary log includes the thread_id, while MariaDB's binary log does not - MDEV-7850
- The MariaDB syntax supporting Spatial Reference System IDs for spatial data type columns with REF_SYSTEM_ID is not supported by MySQL. MySQL 8 introduced `CREATE SPATIAL REFERENCE SYSTEM`, which is not supported by MariaDB - MDEV-29953
- MariaDB 11.1 does not support the RESTART statement - MDEV-30813
- MariaDB 11.1 does not support the SELECT FOR UPDATE and FOR SHARE locks - MDEV-17514

2.1.14.1.5 Incompatibilities and Feature Differences Between MariaDB 11.0 and MySQL 8.0

Contents

MariaDB maintains high levels of compatibility with MySQL, and most applications that use MySQL will work seamlessly with MariaDB. However, take note of the following incompatibilities and feature differences between MariaDB 11.0 and MySQL 8.0. It is based on the versions MySQL 8.0.34 and MariaDB 11.0.2. Note that MySQL 8 is an 'evergreen' release, so features may be added or removed in later releases.

Storage Engines

In addition to the standard InnoDB, MyISAM, BLACKHOLE, CSV, MEMORY, ARCHIVE, and MERGE storage engines, the following are also available with MariaDB 11.0:

- ColumnStore utilizes a massively parallel distributed data architecture and is designed for big data scaling to process petabytes of data.
- MyRocks, a storage engine with great compression
- S3 storage engine allows one to archive MariaDB tables in Amazon S3, or any third-party public or private cloud that implements S3 API.
- Aria, MyISAM replacement with better caching.
- CONNECT
- SEQUENCE
- Spider
- SphinxSE
- FederatedX (drop-in replacement for Federated)
- OQGRAPH

Extensions and New Features

The most notable features available in MariaDB, but not in MySQL, are:

- Galera is a standard part of MariaDB Server.
- Temporal data tables in the form of:
  - System-versioned tables (allow you to query and operate on historic data).
  - Application-time periods (allow you to query and operate on a temporal range of data), including the WITHOUT OVERLAPS clause.
  - Bitemporal tables (which combine both system-versioning and application-time periods).
- DML-only flashback, allowing instances, databases or tables to be rolled back to an old snapshot.
- Oracle compatibility mode
- MariaDB supports localization in a number of additional languages: Bulgarian, Chinese, Georgian, Hindi, Serbian, and Ukrainian.
- MariaDB has made major improvements to the optimizer.
- Sequences
- Table Value Constructors
- Dynamic Columns support
- Semi-sync plugin merged into the server
- OR REPLACE syntax for CREATE statements, such as CREATE OR REPLACE TABLE, CREATE OR REPLACE DATABASE, etc
- DELETE ... RETURNING, INSERT ... RETURNING, REPLACE ... RETURNING
- GRANT to PUBLIC - MDEV-5215 (blog post)
- WAIT syntax for setting the lock wait timeout.
- UUID data type for storing UUIDs.
- INET6 and INET4 data types for storing IPv6 and IPv4 addresses.
- SUPER privileges made more granular.
- PROXY protocol support
- Multiple compression algorithms available as plugins
- Number of supported decimals in DECIMAL has increased from `30` to `38`
- Added catchall for list partitions
- Oracle-style EXECUTE IMMEDIATE statement
- Lots of new JSON functions
- Microsecond Precision in Processlist
- Table Elimination
- Virtual Columns
- Extended User Statistics
- KILL all queries for a user
- Storage-engine-specific CREATE TABLE
- MariaDB supports more collations (506) than MySQL (266).
- FLUSH SSL command to reload SSL certificates without server restart.
- `IF NOT EXISTS` clause added to INSTALL PLUGIN and `IF EXISTS` clause added to UNINSTALL PLUGIN and UNINSTALL SONAME
- Enhancements to INFORMATION SCHEMA.PLUGINS table
- Group commit for the binary log. This makes replication notably faster!
- The binary log in MariaDB can be compressed.
- BACKUP STAGE allows one to implement very efficient backups with minimal locking.
- Progress reporting for ALTER TABLE and LOAD DATA INFILE
- SHOW EXPLAIN gives the EXPLAIN plan for a query running in another thread. MySQL introduced the EXPLAIN FOR CONNECTION syntax to do the same thing.
- PCRE Regular Expressions (including REGEXP_REPLACE())
- HandlerSocket and faster HANDLER calls
- MySQL 8 does not support PROCEDURE ANALYSE
- MySQL 8 does not support the use of `\N` as an alias for NULL

Incompatibilities

When moving from MySQL 8.0 to MariaDB 11.0, please take note of the following incompatibilities:

- For a list of function differences, see Function Differences Between MariaDB 11.0 and MySQL 8.0
- For a list of system variable differences, see System Variable Differences Between MariaDB 11.0 and MySQL 8.0
- MariaDB does not support MySQL's SET PERSIST - MDEV-16228
- MariaDB's GTID is not compatible with MySQL's. Note that MariaDB and MySQL also have different GTID system variables, so these need to be adjusted when migrating.
- The unix_socket authentication plugin is now default on Unix-like systems, which is a major change to authentication in MariaDB. See Authentication from MariaDB 10.4 for an overview of the changes.
- All mysql* binaries are now named mariadb* (the previous mysql named is retained as a symlink for compatibility purposes)
- Not all character sets and collations are supported across both MySQL and MariaDB. As of 11.0, MariaDB supports

40 character sets and 506 collations . As of 8.0.34, MySQL supports 41 character sets (`gb18030` being the additional one - MDEV-7495) and 286 collations.

- MariaDB indicates collation pad status as part of the name (e.g. `utf8mb3_unicode_nopad_ci`), while MySQL indicates pad status by means of an extra column in SHOW COLLATION.
- To make CREATE TABLE ... SELECT work the same way in statement based and row based replication it's by default executed as CREATE OR REPLACE TABLE on the slave. One benefit of this is that if the slave dies in the middle of CREATE ... SELECT it will be able to continue.
  - One can use the slave-ddl-exec-mode variable to specify how `CREATE TABLE` and `DROP TABLE` is replicated.
- Users created with MySQL's SHA256 password algorithm cannot be used in MariaDB 11.0 - MDEV-9804.
- MariaDB 11.0 does not support Lateral Derived Tables - MDEV-19078.
- MariaDB 11.0 does not support CIDR notation for user accounts - MDEV-25515.
- MariaDB stores JSON as true text, not in binary format as MySQL. MariaDB's JSON functions are much faster than MySQL's so there is no need to store in binary format, which would add complexity when manipulating JSON objects.
- For the same reason, MariaDB's JSON data type is an alias for LONGTEXT. If you want to replicate JSON columns from MySQL to MariaDB, you should store JSON objects in MySQL in a TEXT or LONGTEXT column or use statement-based replication. If you are using JSON columns and want to upgrade to MariaDB, use the mysql_json plugin to automatically convert MySQL JSON to TEXT, or alternatively you need to either convert them to TEXT or use mysqldump to copy these tables to MariaDB.
- In MySQL, JSON is compared according to json values. In MariaDB JSON strings are normal strings and compared as strings.
- MariaDB 11.0 does not support MySQL's JSON operators (`->` and `->>`) - MDEV-13594
- MariaDB 11.0 supports the standard by producing null and a warning for JSON_SEARCH when given invalid data, while MySQL produces an error.
- Roles
  - MariaDB never allows authentication via roles, while MySQL permits this.
  - MySQL permits activating multiple roles at the same time. MariaDB can achieve the same result by creating an intermediate aggregate role.
  - In the INFORMATION_SCHEMA.ENABLED_ROLES table, MySQL reports just the direct list of enabled roles, while MariaDB reports the enabled role, plus the effective inherited roles.
  - MySQL extends the INFORMATION_SCHEMA.APPLICABLE_ROLES table .
  - MySQL includes the tables INFORMATION_SCHEMA.ROLE_TABLE_GRANTS, INFORMATION_SCHEMA.ROLE_ROUTINE_GRANTS, INFORMATION_SCHEMA.ROLE_COLUMN_GRANTS, and INFORMATION_SCHEMA ADMINISTRABLE_ROLE_AUTHORIZATIONS.
- MySQL has the performance schema enabled by default. For performance reasons MariaDB 11.0 has it disabled by default. You can enable it by starting `mariadbd` with the option `--performance-schema` .
- MariaDB has removed the InnoDB Change Buffer.
- In MariaDB 11.0, using FLUSH TABLES without any table list will only close tables not in use, and tables not locked by the FLUSH TABLES connection. If there are no locked tables, FLUSH TABLES will be instant and will not cause any waits, as it no longer waits for tables in use. When a table list is provided, the server will wait for the end of any transactions that are using the tables. In MySQL, FLUSH TABLES only waits for the statements to complete.
- MariaDB binaries (`mariadbd` , myisamchk etc.) give a warning if one uses a unique prefix of an option (such as `--big-table` instead of `--big-tables`). MySQL binaries require the full option name.
- MariaDB 11.0 implements InnoDB encryption in a different way to MySQL 8.0.
- MySQL's implementation of aborting statements that exceed a certain time to execute can only kill SELECTs, while MariaDB's can kill any queries (excluding stored procedures).
- MariaDB 11.0 does not support MySQL's `SELECT /*+ MAX_EXECUTION_TIME(n) */ ...` - see Aborting Statements that Exceed a Certain Time to Execute.
- MySQL 8.0 does not support the Query Cache.
- MariaDB 11.0 does not support the MySQL Memcached plugin (which has been deprecated in MySQL 8.0). However, data stored using memcached can be retrieved because the data is stored as InnoDB tables. MariaDB is able to start successfully with an error message of not being able to find libmemcached.so library.
- In MySQL, `X'HHHH'` , the standard SQL syntax for binary string literals, erroneously works in the same way as `0xHHHH` , which could work as a number or string depending on the context. In MariaDB, this has been fixed to behave as a string in all contexts (and never as a number). See CAST and Hexadecimal Literals for more details and examples.
- In MariaDB 11.0, SHOW CREATE TABLE does not quote the DEFAULT value of an integer. MariaDB 10.2 and earlier, and MySQL, do. Since MariaDB can support defaults for BLOB and TEXT fields, while MySQL does not, SHOW CREATE TABLE will also append `DEFAULT NULL` where no default is explicitly provided to nullable BLOB or TEXT fields in MariaDB.
- As a result of implementing Table Value Constructors, the VALUES function has been renamed to VALUE().
- MariaDB's NOWAIT supports SELECT statements, LOCK TABLES and various DDL statements, while MySQL's NOWAIT only supports SELECT.
- MariaDB's NOWAIT cannot be added on views and stored procedures while MySQL's can - MDEV-25247
- MariaDB does not support RENAME table while it is write-locked - MDEV-30814
- MariaDB returns an `ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction`

when unable to lock within the time, while MySQL returns `ERROR 3572 (HY000): Statement aborted because lock(s) could not be acquired immediately and NOWAIT is set`

- MariaDB does not support the optional *init_vector* argument for AES_ENCRYPT and AES_DECRYPT or the block_encryption_mode variable - MDEV-9069
- MariaDB does not support the `--initialize` option. Use mariadb-install-db instead. - MDEV-19010
- MariaDB 11.0 does not support the ngram and MeCab full-text parser plugins - MDEV-10267, MDEV-10268.
- MariaDB 11.0 does not support the MySQL X plugin.
- MariaDB 11.0 before MariaDB 11.0.3 does not support MySQL 8's "native" InnoDB partitioning handler - MDEV-29253
- MariaDB 11.0 does not support CREATE TABLESPACE for InnoDB.
- The MySQL 8.0 and MariaDB 11.0 INFORMATION_SCHEMA.COLUMNS table contain slightly different fields.
- MariaDB 11.0 client executables allow the connection protocol to be forced by specifying only connection properties on the command-line. See mariadb Command-line client
- The MySQL binary log includes the thread_id, while MariaDB's binary log does not - MDEV-7850
- The MariaDB syntax supporting Spatial Reference System IDs for spatial data type columns with REF_SYSTEM_ID is not supported by MySQL. MySQL 8 introduced `CREATE SPATIAL REFERENCE SYSTEM`, which is not supported by MariaDB - MDEV-29953
- MariaDB 11.0 does not support the RESTART statement - MDEV-30813
- MariaDB 11.0 does not support the SELECT FOR UPDATE and FOR SHARE locks - MDEV-17514

2.1.14.1.6 Incompatibilities and Feature Differences Between MariaDB 10.11 and MySQL 8.0

Contents

MariaDB maintains high levels of compatibility with MySQL, and most applications that use MySQL will work seamlessly with MariaDB. However, take note of the following incompatibilities and feature differences between MariaDB 10.11 and MySQL 8.0. It is based on the versions MySQL 8.0.32 and MariaDB 10.11.2. Note that MySQL 8 is an 'evergreen' release, so features may be added or removed in later releases.

Storage Engines

In addition to the standard InnoDB, MyISAM, BLACKHOLE, CSV, MEMORY, ARCHIVE, and MERGE storage engines, the following are also available with MariaDB 10.11:

- ColumnStore utilizes a massively parallel distributed data architecture and is designed for big data scaling to process petabytes of data.
- MyRocks, a storage engine with great compression
- S3 storage engine allows one to archive MariaDB tables in Amazon S3, or any third-party public or private cloud that implements S3 API.
- Aria, MyISAM replacement with better caching.
- CONNECT
- SEQUENCE
- Spider
- SphinxSE
- FederatedX (drop-in replacement for Federated)
- OQGRAPH

Extensions and New Features

The most notable features available in MariaDB, but not in MySQL, are:

- Galera is a standard part of MariaDB Server.
- Temporal data tables in the form of:
  - System-versioned tables (allow you to query and operate on historic data).
  - Application-time periods (allow you to query and operate on a temporal range of data), including the WITHOUT OVERLAPS clause.
  - Bitemporal tables (which combine both system-versioning and application-time periods).
- DML-only flashback, allowing instances, databases or tables to be rolled back to an old snapshot.

- Oracle compatibility mode
- MariaDB supports localization in a number of additional languages: Bulgarian, Chinese, Hindi, Serbian, and Ukrainian.
- Sequences
- Table Value Constructors
- Dynamic Columns support
- Semi-sync plugin merged into the server
- OR REPLACE syntax for CREATE statements, such as CREATE OR REPLACE TABLE, CREATE OR REPLACE DATABASE, etc
- DELETE ... RETURNING, INSERT ... RETURNING, REPLACE ... RETURNING
- GRANT to PUBLIC - MDEV-5215 (blog post)
- WAIT syntax for setting the lock wait timeout.
- UUID data type for storing UUIDs.
- INET6 and INET4 data types for storing IPv6 and IPv4 addresses.
- SUPER privileges made more granular.
- PROXY protocol support
- Multiple compression algorithms available as plugins
- Number of supported decimals in DECIMAL has increased from `30` to `38`
- Added catchall for list partitions
- Oracle-style EXECUTE IMMEDIATE statement
- Lots of new JSON functions
- Microsecond Precision in Processlist
- Table Elimination
- Virtual Columns
- Extended User Statistics
- KILL all queries for a user
- Storage-engine-specific CREATE TABLE
- MariaDB supports more collations (506) than MySQL (266).
- FLUSH SSL command to reload SSL certificates without server restart.
- `IF NOT EXISTS` clause added to INSTALL PLUGIN and `IF EXISTS` clause added to UNINSTALL PLUGIN and UNINSTALL SONAME
- Enhancements to INFORMATION SCHEMA.PLUGINS table
- Group commit for the binary log. This makes replication notably faster!
- The binary log in MariaDB can be compressed.
- BACKUP STAGE allows one to implement very efficient backups with minimal locking.
- Progress reporting for ALTER TABLE and LOAD DATA INFILE
- SHOW EXPLAIN gives the EXPLAIN plan for a query running in another thread. MySQL introduced the EXPLAIN FOR CONNECTION syntax to do the same thing.
- PCRE Regular Expressions (including REGEXP_REPLACE())
- HandlerSocket and faster HANDLER calls
- MySQL 8 does not support PROCEDURE ANALYSE
- MySQL 8 does not support the use of `\N` as an alias for NULL

Incompatibilities

When moving from MySQL 8.0 to MariaDB 10.11, please take note of the following incompatibilities:

- For a list of function differences, see Function Differences Between MariaDB 10.11 and MySQL 8.0
- For a list of system variable differences, see System Variable Differences Between MariaDB 10.11 and MySQL 8.0
- MariaDB does not support MySQL's SET PERSIST - MDEV-16228
- MariaDB's GTID is not compatible with MySQL's. Note that MariaDB and MySQL also have different GTID system variables, so these need to be adjusted when migrating.
- The unix_socket authentication plugin is now default on Unix-like systems, which is a major change to authentication in MariaDB. See Authentication from MariaDB 10.4 for an overview of the changes.
- All mysql* binaries are now named mariadb* (the previous mysql named is retained as a symlink for compatibility purposes)
- Not all character sets and collations are supported across both MySQL and MariaDB. As of 10.11, MariaDB supports 40 character sets and 506 collations . As of 8.0.32, MySQL supports 41 character sets (`gb18030` being the additional one - MDEV-7495) and 286 collations.
- MariaDB indicates collation pad status as part of the name (e.g. `utf8mb3_unicode_nopad_ci`), while MySQL indicates pad status by means of an extra column in SHOW COLLATION.
- To make CREATE TABLE ... SELECT work the same way in statement based and row based replication it's by default executed as CREATE OR REPLACE TABLE on the slave. One benefit of this is that if the slave dies in the middle of CREATE ... SELECT it will be able to continue.
  - One can use the slave-ddl-exec-mode variable to specify how `CREATE TABLE` and `DROP TABLE` is replicated.
- Users created with MySQL's SHA256 password algorithm cannot be used in MariaDB 10.11 - MDEV-9804.

- MariaDB 10.11 does not support Lateral Derived Tables - MDEV-19078.
- MariaDB 10.11 does not support CIDR notation for user accounts - MDEV-25515.
- MariaDB stores JSON as true text, not in binary format as MySQL. MariaDB's JSON functions are much faster than MySQL's so there is no need to store in binary format, which would add complexity when manipulating JSON objects.
- For the same reason, MariaDB's JSON data type is an alias for LONGTEXT. If you want to replicate JSON columns from MySQL to MariaDB, you should store JSON objects in MySQL in a TEXT or LONGTEXT column or use statement-based replication. If you are using JSON columns and want to upgrade to MariaDB, use the mysql_json plugin to automatically convert MySQL JSON to TEXT, or alternatively you need to either convert them to TEXT or use mysqldump to copy these tables to MariaDB.
- In MySQL, JSON is compared according to json values. In MariaDB JSON strings are normal strings and compared as strings.
- MariaDB 10.11 does not support MySQL's JSON operators (`->` and `->>`) - MDEV-13594
- MariaDB 10.11 supports the standard by producing null and a warning for JSON_SEARCH when given invalid data, while MySQL produces an error.
- Roles
  - MariaDB never allows authentication via roles, while MySQL permits this.
  - MySQL permits activating multiple roles at the same time. MariaDB can achieve the same result by creating an intermediate aggregate role.
  - In the INFORMATION_SCHEMA.ENABLED_ROLES table, MySQL reports just the direct list of enabled roles, while MariaDB reports the enabled role, plus the effective inherited roles.
  - MySQL extends the INFORMATION_SCHEMA.APPLICABLE_ROLES table .
  - MySQL includes the tables INFORMATION_SCHEMA.ROLE_TABLE_GRANTS, INFORMATION_SCHEMA.ROLE_ROUTINE_GRANTS, INFORMATION_SCHEMA.ROLE_COLUMN_GRANTS, and INFORMATION_SCHEMA ADMINISTRABLE_ROLE_AUTHORIZATIONS.
- MySQL has the performance schema enabled by default. For performance reasons MariaDB 10.11 has it disabled by default. You can enable it by starting `mysqld` with the option `--performance-schema` .
- In MariaDB 10.11, using FLUSH TABLES without any table list will only close tables not in use, and tables not locked by the FLUSH TABLES connection. If there are no locked tables, FLUSH TABLES will be instant and will not cause any waits, as it no longer waits for tables in use. When a table list is provided, the server will wait for the end of any transactions that are using the tables. In MySQL, FLUSH TABLES only waits for the statements to complete.
- MariaDB binaries (`mysqld` , myisamchk etc.) give a warning if one uses a unique prefix of an option (such as `--big-table` instead of `--big-tables`). MySQL binaries require the full option name.
- MariaDB 10.11 implements InnoDB encryption in a different way to MySQL 8.0.
- MySQL's implementation of aborting statements that exceed a certain time to execute can only kill SELECTs, while MariaDB's can kill any queries (excluding stored procedures).
- MariaDB 10.11 does not support MySQL's `SELECT /*+ MAX_EXECUTION_TIME(n) */ ...` - see Aborting Statements that Exceed a Certain Time to Execute.
- MySQL 8.0 does not support the Query Cache.
- MariaDB 10.11 does not support the MySQL Memcached plugin (which has been deprecated in MySQL 8.0). However, data stored using memcached can be retrieved because the data is stored as InnoDB tables. MariaDB is able to start successfully with an error message of not being able to find libmemcached.so library.
- In MySQL, `X'HHHH'` , the standard SQL syntax for binary string literals, erroneously works in the same way as `0xHHHH` , which could work as a number or string depending on the context. In MariaDB, this has been fixed to behave as a string in all contexts (and never as a number). See CAST and Hexadecimal Literals for more details and examples.
- In MariaDB 10.11, SHOW CREATE TABLE does not quote the DEFAULT value of an integer. MariaDB 10.2 and earlier, and MySQL, do. Since MariaDB can support defaults for BLOB and TEXT fields, while MySQL does not, SHOW CREATE TABLE will also append `DEFAULT NULL` where no default is explicitly provided to nullable BLOB or TEXT fields in MariaDB.
- As a result of implementing Table Value Constructors, the VALUES function has been renamed to VALUE().
- MariaDB's NOWAIT supports SELECT statements, LOCK TABLES and various DDL statements, while MySQL's NOWAIT only supports SELECT.
- MariaDB's NOWAIT cannot be added on views and stored procedures while MySQL's can - MDEV-25247
- MariaDB does not support RENAME table while it is write-locked - MDEV-30814
- MariaDB returns an `ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction` when unable to lock within the time, while MySQL returns `ERROR 3572 (HY000): Statement aborted because lock(s) could not be acquired immediately and NOWAIT is set`
- MariaDB does not support the optional *init_vector* argument for AES_ENCRYPT and AES_DECRYPT or the block_encryption_mode variable - MDEV-9069
- MariaDB does not support the `--initialize` option. Use mariadb-install-db instead. - MDEV-19010
- MariaDB 10.11 does not support the ngram and MeCab full-text parser plugins - MDEV-10267, MDEV-10268.
- MariaDB 10.11 does not support the MySQL X plugin.
- MariaDB 10.11 before MariaDB 10.11.5 does not support MySQL 8's “native” InnoDB partitioning handler - MDEV-29253
- MariaDB 10.11 does not support CREATE TABLESPACE for InnoDB.

- The MySQL 8.0 and MariaDB 10.11 INFORMATION_SCHEMA.COLUMNS table contain slightly different fields.
- MariaDB 10.11 client executables allow the connection protocol to be forced by specifying only connection properties on the command-line. See mariadb Command-line client
- The MySQL binary log includes the thread_id, while MariaDB's binary log does not - MDEV-7850
- The MariaDB syntax supporting Spatial Reference System IDs for spatial data type columns with REF_SYSTEM_ID is not supported by MySQL. MySQL 8 introduced `CREATE SPATIAL REFERENCE SYSTEM`, which is not supported by MariaDB - MDEV-29953
- MariaDB 10.11 does not support the RESTART statement - MDEV-30813
- MariaDB 10.11 does not support the SELECT FOR UPDATE and FOR SHARE locks - MDEV-17514

2.1.14.1.7 Incompatibilities and Feature Differences Between MariaDB 10.10 and MySQL 8.0

Contents

MariaDB maintains high levels of compatibility with MySQL, and most applications that use MySQL will work seamlessly with MariaDB. However, take note of the following incompatibilities and feature differences between MariaDB 10.10 and MySQL 8.0. It is based on the versions MySQL 8.0.32 and MariaDB 10.10.3. Note that MySQL 8 is an 'evergreen' release, so features may be added or removed in later releases.

Storage Engines

In addition to the standard InnoDB, MyISAM, BLACKHOLE, CSV, MEMORY, ARCHIVE, and MERGE storage engines, the following are also available with MariaDB 10.10:

- ColumnStore utilizes a massively parallel distributed data architecture and is designed for big data scaling to process petabytes of data.
- MyRocks, a storage engine with great compression
- S3 storage engine allows one to archive MariaDB tables in Amazon S3, or any third-party public or private cloud that implements S3 API.
- Aria, MyISAM replacement with better caching.
- CONNECT
- SEQUENCE
- Spider
- SphinxSE
- FederatedX (drop-in replacement for Federated)
- OQGRAPH

Extensions and New Features

The most notable features available in MariaDB, but not in MySQL, are:

- Galera is a standard part of MariaDB Server.
- Temporal data tables in the form of:
  - System-versioned tables (allow you to query and operate on historic data).
  - Application-time periods (allow you to query and operate on a temporal range of data), including the WITHOUT OVERLAPS clause.
  - Bitemporal tables (which combine both system-versioning and application-time periods).
- DML-only flashback, allowing instances, databases or tables to be rolled back to an old snapshot.
- Oracle compatibility mode
- Sequences
- Table Value Constructors
- Dynamic Columns support
- Semi-sync plugin merged into the server
- OR REPLACE syntax for CREATE statements, such as CREATE OR REPLACE TABLE, CREATE OR REPLACE DATABASE, etc
- DELETE ... RETURNING, INSERT ... RETURNING, REPLACE ... RETURNING
- WAIT syntax for setting the lock wait timeout.
- UUID data type for storing UUIDs.

- INET6 and INET4 data types for storing IPv6 and IPv4 addresses.
- SUPER privileges made more granular.
- PROXY protocol support
- Multiple compression algorithms available as plugins
- Number of supported decimals in DECIMAL has increased from `30` to `38`
- Added catchall for list partitions
- Oracle-style EXECUTE IMMEDIATE statement
- Lots of new JSON functions
- Microsecond Precision in Processlist
- Table Elimination
- Virtual Columns
- Extended User Statistics
- KILL all queries for a user
- Storage-engine-specific CREATE TABLE
- MariaDB supports more collations (506) than MySQL (266).
- FLUSH SSL command to reload SSL certificates without server restart.
- `IF NOT EXISTS` clause added to INSTALL PLUGIN and `IF EXISTS` clause added to UNINSTALL PLUGIN and UNINSTALL SONAME
- Enhancements to INFORMATION SCHEMA.PLUGINS table
- Group commit for the binary log. This makes replication notably faster!
- The binary log in MariaDB can be compressed.
- BACKUP STAGE allows one to implement very efficient backups with minimal locking.
- Progress reporting for ALTER TABLE and LOAD DATA INFILE
- SHOW EXPLAIN gives the EXPLAIN plan for a query running in another thread. MySQL introduced the EXPLAIN FOR CONNECTION syntax to do the same thing.
- PCRE Regular Expressions (including REGEXP_REPLACE())
- HandlerSocket and faster HANDLER calls
- MySQL 8 does not support PROCEDURE ANALYSE
- MySQL 8 does not support the use of `\N` as an alias for NULL

Incompatibilities

When moving from MySQL 8.0 to MariaDB 10.10, please take note of the following incompatibilities:

- For a list of function differences, see Function Differences Between MariaDB 10.10 and MySQL 8.0
- For a list of system variable differences, see System Variable Differences Between MariaDB 10.10 and MySQL 8.0
- MariaDB does not support MySQL's SET PERSIST - MDEV-16228
- MariaDB's GTID is not compatible with MySQL's. Note that MariaDB and MySQL also have different GTID system variables, so these need to be adjusted when migrating.
- The unix_socket authentication plugin is now default on Unix-like systems, which is a major change to authentication in MariaDB. See Authentication from MariaDB 10.4 for an overview of the changes.
- All mysql* binaries are now named mariadb* (the previous mysql named is retained as a symlink for compatibility purposes)
- Not all character sets and collations are supported across both MySQL and MariaDB. As of 10.10, MariaDB supports 40 character sets and 506 collations . As of 8.0.32, MySQL supports 41 character sets (`gb18030` being the additional one - MDEV-7495) and 286 collations.
- MariaDB indicates collation pad status as part of the name (e.g. `utf8mb3_unicode_nopad_ci`), while MySQL indicates pad status by means of an extra column in SHOW COLLATION.
- To make CREATE TABLE ... SELECT work the same way in statement based and row based replication it's by default executed as CREATE OR REPLACE TABLE on the slave. One benefit of this is that if the slave dies in the middle of CREATE ... SELECT it will be able to continue.
  - One can use the slave-ddl-exec-mode variable to specify how `CREATE TABLE` and `DROP TABLE` is replicated.
- Users created with MySQL's SHA256 password algorithm cannot be used in MariaDB 10.10 - MDEV-9804.
- MariaDB 10.10 does not support Lateral Derived Tables - MDEV-19078.
- MariaDB 10.10 does not support CIDR notation for user accounts - MDEV-25515.
- MariaDB stores JSON as true text, not in binary format as MySQL. MariaDB's JSON functions are much faster than MySQL's so there is no need to store in binary format, which would add complexity when manipulating JSON objects.
- For the same reason, MariaDB's JSON data type is an alias for LONGTEXT. If you want to replicate JSON columns from MySQL to MariaDB, you should store JSON objects in MySQL in a TEXT or LONGTEXT column or use statement-based replication. If you are using JSON columns and want to upgrade to MariaDB, use the mysql_json plugin to automatically convert MySQL JSON to TEXT, or alternatively you need to either convert them to TEXT or use mysqldump to copy these tables to MariaDB.
- In MySQL, JSON is compared according to json values. In MariaDB JSON strings are normal strings and compared as strings.
- MariaDB 10.10 does not support MySQL's JSON operators (`->` and `->>`) - MDEV-13594
- MariaDB 10.10 supports the standard by producing null and a warning for JSON_SEARCH when given invalid data,

while MySQL produces an error.

- Roles
  - MariaDB never allows authentication via roles, while MySQL permits this.
  - MySQL permits activating multiple roles at the same time. MariaDB can achieve the same result by creating an intermediate aggregate role.
  - In the INFORMATION_SCHEMA.ENABLED_ROLES table, MySQL reports just the direct list of enabled roles, while MariaDB reports the enabled role, plus the effective inherited roles.
  - MySQL extends the INFORMATION_SCHEMA.APPLICABLE_ROLES table .
  - MySQL includes the tables INFORMATION_SCHEMA.ROLE_TABLE_GRANTS, INFORMATION_SCHEMA.ROLE_ROUTINE_GRANTS, INFORMATION_SCHEMA.ROLE_COLUMN_GRANTS, and INFORMATION_SCHEMA ADMINISTRABLE_ROLE_AUTHORIZATIONS.
- MySQL has the performance schema enabled by default. For performance reasons MariaDB 10.10 has it disabled by default. You can enable it by starting `mariadbd` with the option `--performance-schema` .
- In MariaDB 10.10, using FLUSH TABLES without any table list will only close tables not in use, and tables not locked by the FLUSH TABLES connection. If there are no locked tables, FLUSH TABLES will be instant and will not cause any waits, as it no longer waits for tables in use. When a table list is provided, the server will wait for the end of any transactions that are using the tables. In MySQL, FLUSH TABLES only waits for the statements to complete.
- MariaDB binaries (`mariadbd` , myisamchk etc.) give a warning if one uses a unique prefix of an option (such as `--big-table` instead of `--big-tables`). MySQL binaries require the full option name.
- MariaDB 10.10 implements InnoDB encryption in a different way to MySQL 8.0.
- MySQL's implementation of aborting statements that exceed a certain time to execute can only kill SELECTs, while MariaDB's can kill any queries (excluding stored procedures).
- MariaDB 10.10 does not support MySQL's `SELECT /*+ MAX_EXECUTION_TIME(n) */ ...` - see Aborting Statements that Exceed a Certain Time to Execute.
- MySQL 8.0 does not support the Query Cache.
- MariaDB 10.10 does not support the MySQL Memcached plugin (which has been deprecated in MySQL 8.0). However, data stored using memcached can be retrieved because the data is stored as InnoDB tables. MariaDB is able to start successfully with an error message of not being able to find libmemcached.so library.
- In MySQL, `X'HHHH'` , the standard SQL syntax for binary string literals, erroneously works in the same way as `0xHHHH` , which could work as a number or string depending on the context. In MariaDB, this has been fixed to behave as a string in all contexts (and never as a number). See CAST and Hexadecimal Literals for more details and examples.
- In MariaDB 10.10, SHOW CREATE TABLE does not quote the DEFAULT value of an integer. MariaDB 10.2 and earlier, and MySQL, do. Since MariaDB can support defaults for BLOB and TEXT fields, while MySQL does not, SHOW CREATE TABLE will also append `DEFAULT NULL` where no default is explicitly provided to nullable BLOB or TEXT fields in MariaDB.
- As a result of implementing Table Value Constructors, the VALUES function has been renamed to VALUE().
- MariaDB's NOWAIT supports SELECT statements, LOCK TABLES and various DDL statements, while MySQL's NOWAIT only supports SELECT.
- MariaDB's NOWAIT cannot be added on views and stored procedures while MySQL's can - MDEV-25247
- MariaDB does not support RENAME table while it is write-locked - MDEV-30814
- MariaDB returns an `ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction` when unable to lock within the time, while MySQL returns `ERROR 3572 (HY000): Statement aborted because lock(s) could not be acquired immediately and NOWAIT is set`
- MariaDB does not support the optional *init_vector* argument for AES_ENCRYPT and AES_DECRYPT or the block_encryption_mode variable - MDEV-9069
- MariaDB does not support the `--initialize` option. Use mysql_install_db instead. - MDEV-19010
- MariaDB 10.10 does not support the ngram and MeCab full-text parser plugins - MDEV-10267, MDEV-10268.
- MariaDB 10.10 does not support the MySQL X plugin.
- MariaDB 10.10 before MariaDB 10.10.6 does not support MySQL 8's “native” InnoDB partitioning handler - MDEV-29253
- MariaDB 10.10 does not support CREATE TABLESPACE for InnoDB.
- The MySQL 8.0 and MariaDB 10.10 INFORMATION_SCHEMA.COLUMNS table contain slightly different fields.
- MariaDB 10.10 client executables allow the connection protocol to be forced by specifying only connection properties on the command-line. See mariadb Command-line client
- The MySQL binary log includes the thread_id, while MariaDB's binary log does not - MDEV-7850
- The MariaDB syntax supporting Spatial Reference System IDs for spatial data type columns with REF_SYSTEM_ID is not supported by MySQL. MySQL 8 introduced `CREATE SPATIAL REFERENCE SYSTEM` , which is not supported by MariaDB - MDEV-29953
- MariaDB 10.10 does not support the RESTART statement - MDEV-30813
- MariaDB 10.10 does not support the SELECT FOR UPDATE and FOR SHARE locks - MDEV-17514

2.1.14.1.8 Incompatibilities and Feature

Differences Between MariaDB 10.6 and MySQL 8.0

Contents

MariaDB maintains high levels of compatibility with MySQL, and most applications that use MySQL will work seamlessly with MariaDB. However, take note of the following incompatibilities and feature differences between MariaDB 10.6 and MySQL 8.0. It is based on the versions MySQL 8.0.25 and MariaDB 10.6.0. Note that MySQL 8 is an 'evergreen' release, so features may be added or removed in later releases.

Storage Engines

In addition to the standard InnoDB, MyISAM, BLACKHOLE, CSV, MEMORY, ARCHIVE, and MERGE storage engines, the following are also available with MariaDB 10.6:

- ColumnStore utilizes a massively parallel distributed data architecture and is designed for big data scaling to process petabytes of data.
- MyRocks, a storage engine with great compression
- S3 storage engine allows one to archive MariaDB tables in Amazon S3, or any third-party public or private cloud that implements S3 API.
- Aria, MyISAM replacement with better caching.
- CONNECT
- SEQUENCE
- Spider
- SphinxSE
- FederatedX (drop-in replacement for Federated)
- OQGRAPH

Extensions and New Features

The most notable features available in MariaDB, but not in MySQL, are:

- Galera is a standard part of MariaDB Server.
- Temporal data tables in the form of:
  - System-versioned tables (allow you to query and operate on historic data).
  - Application-time periods (allow you to query and operate on a temporal range of data), including the WITHOUT OVERLAPS clause.
  - Bitemporal tables (which combine both system-versioning and application-time periods).
- DML-only flashback, allowing instances, databases or tables to be rolled back to an old snapshot.
- Oracle compatibility mode
- Sequences
- Invisible Columns
- Table Value Constructors
- Dynamic Columns support
- Semi-sync plugin merged into the server
- INTERSECT/INTERSECT ALL and EXCEPT/EXCEPT ALL
- OR REPLACE syntax for CREATE statements, such as CREATE OR REPLACE TABLE, CREATE OR REPLACE DATABASE, etc
- DELETE ... RETURNING, INSERT ... RETURNING, REPLACE ... RETURNING
- WAIT syntax for setting the lock wait timeout.
- INET6 data type for storing IPv6 addresses.
- SUPER privileges made more granular.
- PROXY protocol support
- Number of supported decimals in DECIMAL has increased from `30` to `38`
- Added catchall for list partitions
- Oracle-style EXECUTE IMMEDIATE statement
- Lots of new JSON functions
- Microsecond Precision in Processlist
- Table Elimination
- Virtual Columns
- Extended User Statistics
- KILL all queries for a user

- Storage-engine-specific CREATE TABLE
- MariaDB supports more collations than MySQL, including `NO PAD` collations.
- FLUSH SSL command to reload SSL certificates without server restart.
- `IF NOT EXISTS` clause added to INSTALL PLUGIN and `IF EXISTS` clause added to UNINSTALL PLUGIN and UNINSTALL SONAME
- Enhancements to INFORMATION SCHEMA.PLUGINS table
- Group commit for the binary log. This makes replication notably faster!
- The binary log in MariaDB can be compressed.
- BACKUP STAGE allows one to implement very efficient backups with minimal locking.
- Progress reporting for ALTER TABLE and LOAD DATA INFILE
- SHOW EXPLAIN gives the EXPLAIN plan for a query running in another thread. MySQL introduced the EXPLAIN FOR CONNECTION syntax to do the same thing.
- PCRE Regular Expressions (including REGEXP_REPLACE())
- HandlerSocket and faster HANDLER calls
- MySQL 8 does not support PROCEDURE ANALYSE
- MySQL 8 does not support the use of `\N` as an alias for NULL

Incompatibilities

When moving from MySQL 8.0 to MariaDB 10.6, please take note of the following incompatibilities:

- For a list of function differences, see Function Differences Between MariaDB 10.6 and MySQL 8.0
- For a list of system variable differences, see System Variable Differences Between MariaDB 10.6 and MySQL 8.0
- MariaDB does not support MySQL's SET PERSIST - MDEV-16228
- MariaDB's GTID is not compatible with MySQL's. Note that MariaDB and MySQL also have different GTID system variables, so these need to be adjusted when migrating.
- The unix_socket authentication plugin is now default on Unix-like systems, which is a major change to authentication in MariaDB. See Authentication from MariaDB 10.4 for an overview of the changes.
- All mysql* binaries are now named mariadb* (the previous mysql named is retained as a symlink for compatibility purposes)
- Not all character sets and collations are supported across both MySQL and MariaDB. As of 10.6.0, MariaDB supports 40 character sets and 322 collations (armscii8_general_nopad_ci, armscii8_nopad_bin, ascbig5_chinese_nopad_ci, big5_nopad_bin, iicp1250_general_nopad_ci, cp1250_nopad_bin, cp1250_general_nopad_ci, cp1250_nopad_bin, cp1251_general_nopad_ci, cp1251_nopad_bin, cp1256_general_nopad_ci, cp1256_nopad_bin, cp1257_general_nopad_ci, cp1257_nopad_bin, cp850_general_nopad_ci, cp850_nopad_bin, cp852_general_nopad_ci, cp852_nopad_bin, cp866_general_nopad_ci, cp866_nopad_bin, cp932_japanese_nopad_ci, cp932_nopad_bin, dec8_nopad_bin, dec8_swedish_nopad_ci, eucjpms_japanese_nopad_ci, eucjpms_nopad_bin, eucjpms_japanese_nopad_ci, eucjpms_nopad_bin, euckr_korean_nopad_ci, euckr_nopad_bin, gb2312_chinese_nopad_ci, gb2312_nopad_bin, gbk_chinese_nopad_ci, gbk_nopad_bin, geostd8_general_nopad_ci, geostd8_nopad_bin, greek_general_nopad_ci, greek_nopad_bin, hebrew_general_nopad_ci, hebrew_nopad_bin, hp8_english_nopad_ci, hp8_nopad_bin, keybcs2_general_nopad_ci, keybcs2_nopad_bin, koi8r_general_nopad_ci, koi8r_nopad_bin, koi8u_general_nopad_ci, koi8u_nopad_bin, latin1_nopad_bin, latin1_swedish_nopad_ci, latin2_general_nopad_ci, latin2_nopad_bin, latin5_nopad_bin, latin5_turkish_ci, latin5_turkish_nopad_ci, latin7_general_nopad_ci, latin7_nopad_bin, macce_general_nopad_ci, macce_nopad_bin, macroman_general_nopad_ci, macroman_nopad_bin, sjis_japanese_nopad_ci, sjis_nopad_bin, swe7_nopad_bin, tis620_thai_nopad_ci, tis620_nopad_bin, ucs2_croatian_mysql561_ci, ucs2_general_mysql500_ci, ucs2_general_nopad_ci, ucs2_myanmar_ci, ucs2_nopad_bin, ucs2_swedish_ci, ucs2_thai_520_w2, ucs2_unicode_ci, ucs2_unicode_nopad_ci, ujis_japanese_nopad_ci, ujis_nopad_bin, utf16le_general_nopad_ci, utf16le_nopad_bin, utf16_croatian_mysql561_ci, utf16_general_nopad_ci, utf16_myanmar_ci, utf16_nopad_bin, utf16_thai_520_w2, utf16_unicode_520_nopad_ci, utf16_unicode_nopad_ci, utf32_croatian_mysql561_ci, utf32_general_nopad_ci, utf32_myanmar_ci, utf32_nopad_bin, utf32_thai_520_w2, utf32_unicode_520_nopad_ci, utf32_unicode_nopad_ci, utf8mb4_general_nopad_ci, utf8mb4_myanmar_ci, utf8mb4_nopad_bin, utf8mb4_thai_520_w2, utf8mb4_unicode_520_nopad_ci, utf8mb4_unicode_nopad_ci, utf8_croatian_mysql561_ci, utf8_general_nopad_ci, utf8_myanmar_ci, utf8_nopad_bin, utf8_thai_520_w2, utf8_unicode_520_nopad_ci, utf8_unicode_ci and utf8_unicode_nopad_ci being the additional ones).

  As of 8.0.25, MySQL supports 41 character sets (`gb18030` being the additional one - MDEV-7495) and 272 collations (gb18030_bin, gb18030_chinese_ci, gb18030_unicode_520_ci, utf8mb4_0900_ai_ci, utf8mb4_0900_as_ci, utf8mb4_0900_as_cs, utf8mb4_0900_bin, utf8mb4_cs_0900_ai_ci, utf8mb4_cs_0900_as_cs, utf8mb4_da_0900_ai_ci, utf8mb4_da_0900_as_cs, utf8mb4_de_pb_0900_ai_ci, utf8mb4_de_pb_0900_as_cs, utf8mb4_eo_0900_ai_ci, utf8mb4_eo_0900_as_cs, utf8mb4_es_0900_ai_ci, utf8mb4_es_0900_as_cs, utf8mb4_es_trad_0900_ai_ci, utf8mb4_es_trad_0900_as_cs, utf8mb4_et_0900_ai_ci, utf8mb4_et_0900_as_cs, utf8mb4_hr_0900_ai_ci, utf8mb4_hr_0900_as_cs, utf8mb4_hu_0900_ai_ci, utf8mb4_hu_0900_as_cs, utf8mb4_is_0900_ai_ci, utf8mb4_is_0900_as_cs, utf8mb4_ja_0900_as_cs, utf8mb4_ja_0900_as_cs_ks, utf8mb4_la_0900_ai_ci, utf8mb4_la_0900_as_cs, utf8mb4_lt_0900_ai_ci, utf8mb4_lt_0900_as_cs, utf8mb4_lv_0900_ai_ci, utf8mb4_lv_0900_as_cs, utf8mb4_pl_0900_ai_ci, utf8mb4_pl_0900_as_cs, utf8mb4_ro_0900_ai_ci, utf8mb4_ro_0900_as_cs, utf8mb4_ru_0900_ai_ci, utf8mb4_ru_0900_as_cs,

utf8mb4_sk_0900_ai_ci, utf8mb4_sk_0900_as_cs, utf8mb4_sl_0900_ai_ci, utf8mb4_sl_0900_as_cs, utf8mb4_sv_0900_ai_ci, utf8mb4_sv_0900_as_cs, utf8mb4_tr_0900_ai_ci, utf8mb4_vi_0900_ai_ci, utf8mb4_vi_0900_as_cs, utf8mb4_zh_0900_as_cs being the additional ones) - MDEV-20912.

- To make CREATE TABLE ... SELECT work the same way in statement based and row based replication it's by default executed as CREATE OR REPLACE TABLE on the slave. One benefit of this is that if the slave dies in the middle of CREATE ... SELECT it will be able to continue.
  - One can use the slave-ddl-exec-mode variable to specify how `CREATE TABLE` and `DROP TABLE` is replicated.
- Users created with MySQL's SHA256 password algorithm cannot be used in MariaDB 10.6 - MDEV-9804.
- MariaDB 10.6 does not support CIDR notation for user accounts - MDEV-25515.
- MariaDB stores JSON as true text, not in binary format as MySQL. MariaDB's JSON functions are much faster than MySQL's so there is no need to store in binary format, which would add complexity when manipulating JSON objects.
- For the same reason, MariaDB's JSON data type is an alias for LONGTEXT. If you want to replicate JSON columns from MySQL to MariaDB, you should store JSON objects in MySQL in a TEXT or LONGTEXT column or use statement-based replication. If you are using JSON columns and want to upgrade to MariaDB, use the mysql_json plugin to automatically convert MySQL JSON to TEXT, or alternatively you need to either convert them to TEXT or use mysqldump to copy these tables to MariaDB.
- In MySQL, JSON is compared according to json values. In MariaDB JSON strings are normal strings and compared as strings.
- MariaDB 10.6 does not support MySQL's JSON operators (`->` and `->>`) - MDEV-13594
- MariaDB 10.6 supports the standard by producing null and a warning for JSON_SEARCH when given invalid data, while MySQL produces an error.
- Roles
  - MariaDB never allows authentication via roles, while MySQL permits this.
  - MySQL permits activating multiple roles at the same time. MariaDB can achieve the same result by creating an intermediate aggregate role.
  - In the INFORMATION_SCHEMA.ENABLED_ROLES table, MySQL reports just the direct list of enabled roles, while MariaDB reports the enabled role, plus the effective inherited roles.
  - MySQL extends the INFORMATION_SCHEMA.APPLICABLE_ROLES table .
  - MySQL includes the tables INFORMATION_SCHEMA.ROLE_TABLE_GRANTS, INFORMATION_SCHEMA.ROLE_ROUTINE_GRANTS, INFORMATION_SCHEMA.ROLE_COLUMN_GRANTS, and INFORMATION_SCHEMA ADMINISTRABLE_ROLE_AUTHORIZATIONS.
- MySQL has the performance schema enabled by default. For performance reasons MariaDB 10.6 has it disabled by default. You can enable it by starting `mariadbd` with the option `--performance-schema` .
- In MariaDB 10.6, using FLUSH TABLES without any table list will only close tables not in use, and tables not locked by the FLUSH TABLES connection. If there are no locked tables, FLUSH TABLES will be instant and will not cause any waits, as it no longer waits for tables in use. When a table list is provided, the server will wait for the end of any transactions that are using the tables. In MySQL, FLUSH TABLES only waits for the statements to complete.
- MariaDB binaries (`mariadbd` , myisamchk etc.) give a warning if one uses a unique prefix of an option (such as `--big-table` instead of `--big-tables`). MySQL binaries require the full option name.
- MariaDB 10.6 implements InnoDB encryption in a different way to MySQL 8.0.
- MySQL's implementation of aborting statements that exceed a certain time to execute can only kill SELECTs, while MariaDB's can kill any queries (excluding stored procedures).
- MariaDB 10.6 does not support MySQL's `SELECT /*+ MAX_EXECUTION_TIME(n) */ ...` - see Aborting Statements that Exceed a Certain Time to Execute.
- MySQL 8.0 does not support the Query Cache.
- MariaDB 10.6 does not support the MySQL Memcached plugin (which has been deprecated in MySQL 8.0). However, data stored using memcached can be retrieved because the data is stored as InnoDB tables. MariaDB is able to start successfully with an error message of not being able to find libmemcached.so library.
- In MySQL, `X'HHHH'` , the standard SQL syntax for binary string literals, erroneously works in the same way as `0xHHHH` , which could work as a number or string depending on the context. In MariaDB, this has been fixed to behave as a string in all contexts (and never as a number). See CAST and Hexadecimal Literals for more details and examples.
- In MariaDB 10.6, SHOW CREATE TABLE does not quote the DEFAULT value of an integer. MariaDB 10.2 and earlier, and MySQL, do. Since MariaDB can support defaults for BLOB and TEXT fields, while MySQL does not, SHOW CREATE TABLE will also append `DEFAULT NULL` where no default is explicitly provided to nullable BLOB or TEXT fields in MariaDB.
- Since MariaDB supports INTERSECT and EXCEPT, these are both reserved words and can't be used as an identifier without being quoted.
- As a result of implementing Table Value Constructors, the VALUES function has been renamed to VALUE().
- MariaDB's NOWAIT supports SELECT statements, LOCK TABLES and various DDL statements, while MySQL's NOWAIT only supports SELECT.
- MariaDB's NOWAIT cannot be added on views and stored procedures while MySQL's can - MDEV-25247
- MariaDB returns an `ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction` when unable to lock within the time, while MySQL returns `ERROR 3572 (HY000): Statement aborted because lock(s) could not be acquired immediately and NOWAIT is set`

- MariaDB does not support RENAME table while it is write-locked - MDEV-30814
- MariaDB 10.6 does not support Lateral Derived Tables - MDEV-19078.
- MariaDB does not support the optional *init_vector* argument for AES_ENCRYPT and AES_DECRYPT or the block_encryption_mode variable - MDEV-9069
- MariaDB does not support the `--initialize` option. Use mysql_install_db instead. - MDEV-19010
- MariaDB 10.6 does not support the ngram and MeCab full-text parser plugins - MDEV-10267, MDEV-10268.
- MariaDB 10.6 does not support the MySQL X plugin.
- MariaDB 10.6 before MariaDB 10.6.15 does not support MySQL 8's “native” InnoDB partitioning handler - MDEV-29253
- MariaDB 10.6 does not support CREATE TABLESPACE for InnoDB.
- The MySQL 8.0 and MariaDB 10.6 INFORMATION_SCHEMA.COLUMNS table contain slightly different fields.
- MariaDB 10.6 client executables allow the connection protocol to be forced by specifying only connection properties on the command-line. See mariadb Command-line client
- The MySQL binary log includes the thread_id, while MariaDB's binary log does not - MDEV-7850
- The MariaDB syntax supporting Spatial Reference System IDs for spatial data type columns with REF_SYSTEM_ID is not supported by MySQL. MySQL 8 introduced `CREATE SPATIAL REFERENCE SYSTEM`, which is not supported by MariaDB - MDEV-29953
- MariaDB 10.6 does not support the RESTART statement - MDEV-30813
- MariaDB 10.6 does not support the SELECT FOR UPDATE and FOR SHARE locks - MDEV-17514

2.1.14.1.9 Incompatibilities and Feature Differences Between MariaDB 10.5 and MySQL 8.0

Contents

MariaDB maintains high levels of compatibility with MySQL, and most applications that use MySQL will work seamlessly with MariaDB. However, take note of the following incompatibilities and feature differences between MariaDB 10.5 and MySQL 8.0. It is based on the stable versions MySQL 8.0.22 and MariaDB 10.5.7. Note that MySQL 8 is an 'evergreen' release, so features may be added or removed in later releases.

Storage Engines

In addition to the standard InnoDB, MyISAM, BLACKHOLE, CSV, MEMORY, ARCHIVE, and MERGE storage engines, the following are also available with MariaDB 10.5:

- ColumnStore utilizes a massively parallel distributed data architecture and is designed for big data scaling to process petabytes of data.
- MyRocks, a storage engine with great compression
- S3 storage engine allows one to archive MariaDB tables in Amazon S3, or any third-party public or private cloud that implements S3 API.
- Aria, MyISAM replacement with better caching.
- CONNECT
- SEQUENCE
- Spider
- SphinxSE
- FederatedX (drop-in replacement for Federated)
- OQGRAPH

Extensions and New Features

The most notable features available in MariaDB, but not in MySQL, are:

- Galera is a standard part of MariaDB Server.
- Temporal data tables in the form of:
  - System-versioned tables (allow you to query and operate on historic data).
  - Application-time periods (allow you to query and operate on a temporal range of data), including the WITHOUT OVERLAPS clause added in 10.5.
  - Bitemporal tables (which combine both system-versioning and application-time periods).
- DML-only flashback, allowing instances, databases or tables to be rolled back to an old snapshot.

- Oracle compatibility mode
- Sequences
- Invisible Columns
- Table Value Constructors
- Dynamic Columns support
- Semi-sync plugin merged into the server
- INTERSECT/INTERSECT ALL and EXCEPT/EXCEPT ALL
- OR REPLACE syntax for CREATE statements, such as CREATE OR REPLACE TABLE, CREATE OR REPLACE DATABASE, etc
- DELETE ... RETURNING, INSERT ... RETURNING, REPLACE ... RETURNING
- WAIT syntax for setting the lock wait timeout.
- INET6 data type for storing IPv6 addresses.
- SUPER privileges made more granular.
- PROXY protocol support
- Number of supported decimals in DECIMAL has increased from `30` to `38`
- Added catchall for list partitions
- Oracle-style EXECUTE IMMEDIATE statement
- Lots of new JSON functions
- Microsecond Precision in Processlist
- Table Elimination
- Virtual Columns
- Extended User Statistics
- KILL all queries for a user
- Storage-engine-specific CREATE TABLE
- MariaDB supports more collations than MySQL, including `NO PAD` collations.
- FLUSH SSL command to reload SSL certificates without server restart.
- `IF NOT EXISTS` clause added to INSTALL PLUGIN and `IF EXISTS` clause added to UNINSTALL PLUGIN and UNINSTALL SONAME
- Enhancements to INFORMATION SCHEMA.PLUGINS table
- Group commit for the binary log. This makes replication notably faster!
- The binary log in MariaDB can be compressed.
- BACKUP STAGE allows one to implement very efficient backups with minimal locking.
- Progress reporting for ALTER TABLE and LOAD DATA INFILE
- SHOW EXPLAIN gives the EXPLAIN plan for a query running in another thread. MySQL introduced the EXPLAIN FOR CONNECTION syntax to do the same thing.
- PCRE Regular Expressions (including REGEXP_REPLACE())
- HandlerSocket and faster HANDLER calls
- MySQL 8 does not support PROCEDURE ANALYSE
- MySQL 8 does not support the use of `\N` as an alias for NULL

Incompatibilities

When moving from MySQL 8.0 to MariaDB 10.5, please take note of the following incompatibilities:

- For a list of function differences, see Function Differences Between MariaDB 10.5 and MySQL 8.0
- For a list of system variable differences, see System Variable Differences Between MariaDB 10.5 and MySQL 8.0
- MariaDB does not support MySQL's SET PERSIST - MDEV-16228
- MariaDB's GTID is not compatible with MySQL's. Note that MariaDB and MySQL also have different GTID system variables, so these need to be adjusted when migrating.
- The unix_socket authentication plugin is now default on Unix-like systems, which is a major change to authentication in MariaDB. See Authentication from MariaDB 10.4 for an overview of the changes.
- All mysql* binaries are now named mariadb* (the previous mysql named is retained as a symlink for compatibility purposes)
- Not all character sets and collations are supported across both MySQL and MariaDB. As of 10.5.4, MariaDB supports 40 character sets and 322 collations (armscii8_general_nopad_ci, armscii8_nopad_bin, ascbig5_chinese_nopad_ci, big5_nopad_bin, iicp1250_general_nopad_ci, cp1250_nopad_bin, cp1250_general_nopad_ci, cp1250_nopad_bin, cp1251_general_nopad_ci, cp1251_nopad_bin, cp1256_general_nopad_ci, cp1256_nopad_bin, cp1257_general_nopad_ci, cp1257_nopad_bin, cp850_general_nopad_ci, cp850_nopad_bin, cp852_general_nopad_ci, cp852_nopad_bin, cp866_general_nopad_ci, cp866_nopad_bin, cp932_japanese_nopad_ci, cp932_nopad_bin, dec8_nopad_bin, dec8_swedish_nopad_ci, eucjpms_japanese_nopad_ci, eucjpms_nopad_bin, eucjpms_japanese_nopad_ci, eucjpms_nopad_bin, euckr_korean_nopad_ci, euckr_nopad_bin, gb2312_chinese_nopad_ci, gb2312_nopad_bin, gbk_chinese_nopad_ci, gbk_nopad_bin, geostd8_general_nopad_ci, geostd8_nopad_bin, greek_general_nopad_ci, greek_nopad_bin, hebrew_general_nopad_ci, hebrew_nopad_bin, hp8_english_nopad_ci, hp8_nopad_bin, keybcs2_general_nopad_ci, keybcs2_nopad_bin, koi8r_general_nopad_ci, koi8r_nopad_bin, koi8u_general_nopad_ci, koi8u_nopad_bin, latin1_nopad_bin, latin1_swedish_nopad_ci, latin2_general_nopad_ci, latin2_nopad_bin, latin5_nopad_bin, latin5_turkish_ci, latin5_turkish_nopad_ci, latin7_general_nopad_ci, latin7_nopad_bin, macce_general_nopad_ci,

macce_nopad_bin, macroman_general_nopad_ci, macroman_nopad_bin, sjis_japanese_nopad_ci, sjis_nopad_bin, swe7_nopad_bin, tis620_thai_nopad_ci, tis620_nopad_bin, ucs2_croatian_mysql561_ci, ucs2_general_mysql500_ci, ucs2_general_nopad_ci, ucs2_myanmar_ci, ucs2_nopad_bin, ucs2_swedish_ci, ucs2_thai_520_w2, ucs2_unicode_ci, ucs2_unicode_nopad_ci, ujis_japanese_nopad_ci, ujis_nopad_bin, utf16le_general_nopad_ci, utf16le_nopad_bin, utf16_croatian_mysql561_ci, utf16_general_nopad_ci, utf16_myanmar_ci, utf16_nopad_bin, utf16_thai_520_w2, utf16_unicode_520_nopad_ci, utf16_unicode_nopad_ci, utf32_croatian_mysql561_ci, utf32_general_nopad_ci, utf32_myanmar_ci, utf32_nopad_bin, utf32_thai_520_w2, utf32_unicode_520_nopad_ci, utf32_unicode_nopad_ci, utf8mb4_general_nopad_ci, utf8mb4_myanmar_ci, utf8mb4_nopad_bin, utf8mb4_thai_520_w2, utf8mb4_unicode_520_nopad_ci, utf8mb4_unicode_nopad_ci, utf8_croatian_mysql561_ci, utf8_general_nopad_ci, utf8_myanmar_ci, utf8_nopad_bin, utf8_thai_520_w2, utf8_unicode_520_nopad_ci, utf8_unicode_ci and utf8_unicode_nopad_ci being the additional ones).

As of 8.0.21, MySQL supports 41 character sets (`gb18030` being the additional one) and 272 collations (gb18030_bin, gb18030_chinese_ci, gb18030_unicode_520_ci, utf8mb4_0900_ai_ci, utf8mb4_0900_as_ci, utf8mb4_0900_as_cs, utf8mb4_0900_bin, utf8mb4_cs_0900_ai_ci, utf8mb4_cs_0900_as_cs, utf8mb4_da_0900_ai_ci, utf8mb4_da_0900_as_cs, utf8mb4_de_pb_0900_ai_ci, utf8mb4_de_pb_0900_as_cs, utf8mb4_eo_0900_ai_ci, utf8mb4_eo_0900_as_cs, utf8mb4_es_0900_ai_ci, utf8mb4_es_0900_as_cs, utf8mb4_es_trad_0900_ai_ci, utf8mb4_es_trad_0900_as_cs, utf8mb4_et_0900_ai_ci, utf8mb4_et_0900_as_cs, utf8mb4_hr_0900_ai_ci, utf8mb4_hr_0900_as_cs, utf8mb4_hu_0900_ai_ci, utf8mb4_hu_0900_as_cs, utf8mb4_is_0900_ai_ci, utf8mb4_is_0900_as_cs, utf8mb4_ja_0900_as_cs, utf8mb4_ja_0900_as_cs_ks, utf8mb4_la_0900_ai_ci, utf8mb4_la_0900_as_cs, utf8mb4_lt_0900_ai_ci, utf8mb4_lt_0900_as_cs, utf8mb4_lv_0900_ai_ci, utf8mb4_lv_0900_as_cs, utf8mb4_pl_0900_ai_ci, utf8mb4_pl_0900_as_cs, utf8mb4_ro_0900_ai_ci, utf8mb4_ro_0900_as_cs, utf8mb4_ru_0900_ai_ci, utf8mb4_ru_0900_as_cs, utf8mb4_sk_0900_ai_ci, utf8mb4_sk_0900_as_cs, utf8mb4_sl_0900_ai_ci, utf8mb4_sl_0900_as_cs, utf8mb4_sv_0900_ai_ci, utf8mb4_sv_0900_as_cs, utf8mb4_tr_0900_ai_ci, utf8mb4_vi_0900_ai_ci, utf8mb4_vi_0900_as_cs, utf8mb4_zh_0900_as_cs being the additional ones).

- To make CREATE TABLE ... SELECT work the same way in statement based and row based replication it's by default executed as CREATE OR REPLACE TABLE on the slave. One benefit of this is that if the slave dies in the middle of CREATE ... SELECT it will be able to continue.
  - One can use the slave-ddl-exec-mode variable to specify how `CREATE TABLE` and `DROP TABLE` is replicated.
- Users created with MySQL's SHA256 password algorithm cannot be used in MariaDB 10.5 - MDEV-9804.
- MariaDB stores JSON as true text, not in binary format as MySQL. MariaDB's JSON functions are much faster than MySQL's so there is no need to store in binary format, which would add complexity when manipulating JSON objects.
- For the same reason, MariaDB's JSON data type is an alias for LONGTEXT. If you want to replicate JSON columns from MySQL to MariaDB, you should store JSON objects in MySQL in a TEXT or LONGTEXT column or use statement-based replication. If you are using JSON columns and want to upgrade to MariaDB, use the mysql_json plugin to automatically convert MySQL JSON to TEXT, or alternatively you need to either convert them to TEXT or use mysqldump to copy these tables to MariaDB.
- In MySQL, JSON is compared according to json values. In MariaDB JSON strings are normal strings and compared as strings.
- MariaDB 10.5 does not support MySQL's JSON operators (`->` and `->>`) - MDEV-13594
- MariaDB 10.5 supports the standard by producing null and a warning for JSON_SEARCH when given invalid data, while MySQL produces an error.
- Roles
  - MariaDB never allows authentication via roles, while MySQL permits this.
  - MySQL permits activating multiple roles at the same time. MariaDB can achieve the same result by creating an intermediate aggregate role.
  - In the INFORMATION_SCHEMA.ENABLED_ROLES table, MySQL reports just the direct list of enabled roles, while MariaDB reports the enabled role, plus the effective inherited roles.
  - MySQL extends the INFORMATION_SCHEMA.APPLICABLE_ROLES table .
  - MySQL includes the tables INFORMATION_SCHEMA.ROLE_TABLE_GRANTS, INFORMATION_SCHEMA.ROLE_ROUTINE_GRANTS, INFORMATION_SCHEMA.ROLE_COLUMN_GRANTS, and INFORMATION_SCHEMA ADMINISTRABLE_ROLE_AUTHORIZATIONS.
- MySQL has the performance schema enabled by default. For performance reasons MariaDB 10.5 has it disabled by default. You can enable it by starting `mariadbd` with the option `--performance-schema` .
- In MariaDB 10.5, using FLUSH TABLES without any table list will only close tables not in use, and tables not locked by the FLUSH TABLES connection. If there are no locked tables, FLUSH TABLES will be instant and will not cause any waits, as it no longer waits for tables in use. When a table list is provided, the server will wait for the end of any transactions that are using the tables. In MySQL, FLUSH TABLES only waits for the statements to complete.
- MariaDB binaries (`mariadbd` , myisamchk etc.) give a warning if one uses a unique prefix of an option (such as `--big-table` instead of `--big-tables`). MySQL binaries require the full option name.
- MariaDB 10.5 implements InnoDB encryption in a different way to MySQL 8.0.
- MySQL's implementation of aborting statements that exceed a certain time to execute can only kill SELECTs, while MariaDB's can kill any queries (excluding stored procedures).
- MariaDB 10.5 does not support MySQL's `SELECT /*+ MAX_EXECUTION_TIME(n) */ ...` - see Aborting Statements that Exceed a Certain Time to Execute.

- MySQL 8.0 does not support the Query Cache.
- MariaDB 10.5 does not support the MySQL Memcached plugin (which has been deprecated in MySQL 8.0). However, data stored using memcached can be retrieved because the data is stored as InnoDB tables. MariaDB is able to start successfully with an error message of not being able to find libmemcached.so library.
- In MySQL, `X'HHHH'`, the standard SQL syntax for binary string literals, erroneously works in the same way as `0xHHHH`, which could work as a number or string depending on the context. In MariaDB, this has been fixed to behave as a string in all contexts (and never as a number). See CAST and Hexadecimal Literals for more details and examples.
- In MariaDB 10.5, SHOW CREATE TABLE does not quote the DEFAULT value of an integer. MariaDB 10.2 and earlier, and MySQL, do. Since MariaDB can support defaults for BLOB and TEXT fields, while MySQL does not, SHOW CREATE TABLE will also append `DEFAULT NULL` where no default is explicitly provided to nullable BLOB or TEXT fields in MariaDB.
- Since MariaDB supports INTERSECT and EXCEPT, these are both reserved words and can't be used as an identifier without being quoted.
- As a result of implementing Table Value Constructors, the VALUES function has been renamed to VALUE().
- MariaDB's NOWAIT supports SELECT statements, LOCK TABLES and various DDL statements, while MySQL's NOWAIT only supports SELECT.
- MariaDB's NOWAIT cannot be added on views and stored procedures while MySQL's can - MDEV-25247
- MariaDB returns an `ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction` when unable to lock within the time, while MySQL returns `ERROR 3572 (HY000): Statement aborted because lock(s) could not be acquired immediately and NOWAIT is set`
- MariaDB does not support RENAME table while it is write-locked - MDEV-30814
- MariaDB 10.5 does not support Lateral Derived Tables - MDEV-19078.
- MariaDB does not support the optional *init_vector* argument for AES_ENCRYPT and AES_DECRYPT or the block_encryption_mode variable - MDEV-9069
- MariaDB does not support the `--initialize` option. Use mysql_install_db instead. - MDEV-19010
- MariaDB 10.5 does not support the ngram and MeCab full-text parser plugins - MDEV-10267, MDEV-10268.
- MariaDB 10.5 does not support the MySQL X plugin.
- MariaDB 10.5 does not support MySQL 8's “native” InnoDB partitioning handler - MDEV-29253
- MariaDB 10.5 does not support CREATE TABLESPACE for InnoDB.
- The MySQL 8.0 and MariaDB 10.5 INFORMATION_SCHEMA.COLUMNS table contain slightly different fields.
- The MySQL binary log includes the thread_id, while MariaDB's binary log does not - MDEV-7850
- The MariaDB 10.1 syntax supporting Spatial Reference System IDs for spatial data type columns with REF_SYSTEM_ID is not supported by MySQL. MySQL 8 introduced `CREATE SPATIAL REFERENCE SYSTEM`, which is not supported by MariaDB - MDEV-29953
- MariaDB 10.5 does not support the RESTART statement - MDEV-30813
- MariaDB 10.5 does not support the SELECT FOR UPDATE and FOR SHARE locks - MDEV-17514

- Also see Incompatibilities between MariaDB 10.4 and MySQL 8.0 and Incompatibilities between MariaDB 10.3 and MySQL 5.7.

2.1.14.1.10 Incompatibilities and Feature Differences Between MariaDB 10.4 and MySQL 8.0

Contents

MariaDB maintains high levels of compatibility with MySQL, and most applications that use MySQL will work seamlessly with MariaDB. However, take note of the following incompatibilities and feature differences between MariaDB 10.4 and MySQL 8.0. It is based on the stable versions MySQL 8.0.22 and MariaDB 10.4.15. Note that MySQL 8 is an 'evergreen' release, so features may be added or removed in later releases.

Storage Engines

In addition to the standard InnoDB, MyISAM, BLACKHOLE, CSV, MEMORY, ARCHIVE, and MERGE storage engines, the following are also available with MariaDB 10.4:

- MyRocks, a storage engine with great compression
- Aria, MyISAM replacement with better caching.
- CONNECT
- SEQUENCE

- Spider
- SphinxSE
- TokuDB
- FederatedX (drop-in replacement for Federated)
- OQGRAPH

Extensions and New Features

The most notable features available in MariaDB, but not in MySQL, are:

- Galera is a standard part of MariaDB Server.
- Temporal data tables in the form of:
  - System-versioned tables (allow you to query and operate on historic data).
  - Application-time periods (allow you to query and operate on a temporal range of data).
  - Bitemporal tables (which combine both system-versioning and application-time periods).
- DML-only flashback, allowing instances, databases or tables to be rolled back to an old snapshot.
- Oracle compatibility mode
- Sequences
- Invisible Columns
- Table Value Constructors
- Semi-sync plugin merged into the server
- INTERSECT and EXCEPT
- OR REPLACE syntax for CREATE statements, such as CREATE OR REPLACE TABLE, CREATE OR REPLACE DATABASE, etc
- DELETE ... RETURNING
- WAIT syntax for setting the lock wait timeout.
- PROXY protocol support
- Number of supported decimals in DECIMAL has increased from `30` to `38`
- Added catchall for list partitions
- Oracle-style EXECUTE IMMEDIATE statement
- Lots of new JSON functions
- Microsecond Precision in Processlist
- Table Elimination
- Virtual Columns
- Extended User Statistics
- KILL all queries for a user
- Storage-engine-specific CREATE TABLE
- MariaDB supports more collations than MySQL, including `NO PAD` collations.
- FLUSH SSL command to reload SSL certificates without server restart.
- `IF NOT EXISTS` clause added to INSTALL PLUGIN and `IF EXISTS` clause added to UNINSTALL PLUGIN and UNINSTALL SONAME
- Enhancements to INFORMATION SCHEMA.PLUGINS table
- Group commit for the binary log. This makes replication notably faster!
- The binary log in MariaDB can be compressed.
- New server command, SHUTDOWN WAIT FOR ALL SLAVES, and a new mysqladmin shutdown --wait-for-all-slaves option, are added to instruct the server to wait for the last binlog event to be sent to all connected slaves before shutting down.
- BACKUP STAGE allows one to implement very efficient backups with minimal locking.
- Progress reporting for ALTER TABLE and LOAD DATA INFILE
- SHOW EXPLAIN gives the EXPLAIN plan for a query running in another thread. MySQL introduced the EXPLAIN FOR CONNECTION syntax to do the same thing.
- PCRE Regular Expressions (including REGEXP_REPLACE())
- HandlerSocket and faster HANDLER calls
- MySQL 8 does not support PROCEDURE ANALYSE
- MySQL 8 does not support the use of `\N` as an alias for NULL

Incompatibilities

When moving from MySQL 8.0 to MariaDB 10.4, please take note of the following incompatibilities:

- For a list of function differences, see Function Differences Between MariaDB 10.4 and MySQL 8.0
- For a list of system variable differences, see System Variable Differences Between MariaDB 10.4 and MySQL 8.0
- MariaDB does not support MySQL's SET PERSIST - MDEV-16228
- MariaDB's GTID is not compatible with MySQL's. Note that MariaDB and MySQL also have different GTID system variables, so these need to be adjusted when migrating.
- The unix_socket authentication plugin is now default on Unix-like systems, which is a major change to authentication

in MariaDB. See Authentication from MariaDB 10.4 for an overview of the changes.

- Not all character sets and collations are supported across both MySQL and MariaDB. As of 10.4.14, MariaDB supports 40 character sets and 322 collations (armscii8_general_nopad_ci, armscii8_nopad_bin, ascbig5_chinese_nopad_ci, big5_nopad_bin, iicp1250_general_nopad_ci, cp1250_nopad_bin, cp1250_general_nopad_ci, cp1250_nopad_bin, cp1251_general_nopad_ci, cp1251_nopad_bin, cp1256_general_nopad_ci, cp1256_nopad_bin, cp1257_general_nopad_ci, cp1257_nopad_bin, cp850_general_nopad_ci, cp850_nopad_bin, cp852_general_nopad_ci, cp852_nopad_bin, cp866_general_nopad_ci, cp866_nopad_bin, cp932_japanese_nopad_ci, cp932_nopad_bin, dec8_nopad_bin, dec8_swedish_nopad_ci, eucjpms_japanese_nopad_ci, eucjpms_nopad_bin, eucjpms_japanese_nopad_ci, eucjpms_nopad_bin, euckr_korean_nopad_ci, euckr_nopad_bin, gb2312_chinese_nopad_ci, gb2312_nopad_bin, gbk_chinese_nopad_ci, gbk_nopad_bin, geostd8_general_nopad_ci, geostd8_nopad_bin, greek_general_nopad_ci, greek_nopad_bin, hebrew_general_nopad_ci, hebrew_nopad_bin, hp8_english_nopad_ci, hp8_nopad_bin, keybcs2_general_nopad_ci, keybcs2_nopad_bin, koi8r_general_nopad_ci, koi8r_nopad_bin, koi8u_general_nopad_ci, koi8u_nopad_bin, latin1_nopad_bin, latin1_swedish_nopad_ci, latin2_general_nopad_ci, latin2_nopad_bin, latin5_nopad_bin, latin5_turkish_ci, latin5_turkish_nopad_ci, latin7_general_nopad_ci, latin7_nopad_bin, macce_general_nopad_ci, macce_nopad_bin, macroman_general_nopad_ci, macroman_nopad_bin, sjis_japanese_nopad_ci, sjis_nopad_bin, swe7_nopad_bin, tis620_thai_nopad_ci, tis620_nopad_bin, ucs2_croatian_mysql561_ci, ucs2_general_mysql500_ci, ucs2_general_nopad_ci, ucs2_myanmar_ci, ucs2_nopad_bin, ucs2_swedish_ci, ucs2_thai_520_w2, ucs2_unicode_ci, ucs2_unicode_nopad_ci, ujis_japanese_nopad_ci, ujis_nopad_bin, utf16le_general_nopad_ci, utf16le_nopad_bin, utf16_croatian_mysql561_ci, utf16_general_nopad_ci, utf16_myanmar_ci, utf16_nopad_bin, utf16_thai_520_w2, utf16_unicode_520_nopad_ci, utf16_unicode_nopad_ci, utf32_croatian_mysql561_ci, utf32_general_nopad_ci, utf32_myanmar_ci, utf32_nopad_bin, utf32_thai_520_w2, utf32_unicode_520_nopad_ci, utf32_unicode_nopad_ci, utf8mb4_general_nopad_ci, utf8mb4_myanmar_ci, utf8mb4_nopad_bin, utf8mb4_thai_520_w2, utf8mb4_unicode_520_nopad_ci, utf8mb4_unicode_nopad_ci, utf8_croatian_mysql561_ci, utf8_general_nopad_ci, utf8_myanmar_ci, utf8_nopad_bin, utf8_thai_520_w2, utf8_unicode_520_nopad_ci, utf8_unicode_ci and utf8_unicode_nopad_ci being the additional ones).

  As of 8.0.21, MySQL supports 41 character sets (gb18030 being the additional one) and 272 collations (gb18030_bin, gb18030_chinese_ci, gb18030_unicode_520_ci, utf8mb4_0900_ai_ci, utf8mb4_0900_as_ci, utf8mb4_0900_as_cs, utf8mb4_0900_bin, utf8mb4_cs_0900_ai_ci, utf8mb4_cs_0900_as_cs, utf8mb4_da_0900_ai_ci, utf8mb4_da_0900_as_cs, utf8mb4_de_pb_0900_ai_ci, utf8mb4_de_pb_0900_as_cs, utf8mb4_eo_0900_ai_ci, utf8mb4_eo_0900_as_cs, utf8mb4_es_0900_ai_ci, utf8mb4_es_0900_as_cs, utf8mb4_es_trad_0900_ai_ci, utf8mb4_es_trad_0900_as_cs, utf8mb4_et_0900_ai_ci, utf8mb4_et_0900_as_cs, utf8mb4_hr_0900_ai_ci, utf8mb4_hr_0900_as_cs, utf8mb4_hu_0900_ai_ci, utf8mb4_hu_0900_as_cs, utf8mb4_is_0900_ai_ci, utf8mb4_is_0900_as_cs, utf8mb4_ja_0900_as_cs, utf8mb4_ja_0900_as_cs_ks, utf8mb4_la_0900_ai_ci, utf8mb4_la_0900_as_cs, utf8mb4_lt_0900_ai_ci, utf8mb4_lt_0900_as_cs, utf8mb4_lv_0900_ai_ci, utf8mb4_lv_0900_as_cs, utf8mb4_pl_0900_ai_ci, utf8mb4_pl_0900_as_cs, utf8mb4_ro_0900_ai_ci, utf8mb4_ro_0900_as_cs, utf8mb4_ru_0900_ai_ci, utf8mb4_ru_0900_as_cs, utf8mb4_sk_0900_ai_ci, utf8mb4_sk_0900_as_cs, utf8mb4_sl_0900_ai_ci, utf8mb4_sl_0900_as_cs, utf8mb4_sv_0900_ai_ci, utf8mb4_sv_0900_as_cs, utf8mb4_tr_0900_ai_ci, utf8mb4_vi_0900_ai_ci, utf8mb4_vi_0900_as_cs, utf8mb4_zh_0900_as_cs being the additional ones).
- To make CREATE TABLE ... SELECT work the same way in statement based and row based replication it's by default executed as CREATE OR REPLACE TABLE on the slave. One benefit of this is that if the slave dies in the middle of CREATE ... SELECT it will be able to continue.
  - One can use the slave-ddl-exec-mode variable to specify how `CREATE TABLE` and `DROP TABLE` is replicated.
- Users created with MySQL's SHA256 password algorithm cannot be used in MariaDB 10.4 - MDEV-9804.
- MariaDB stores JSON as true text, not in binary format as MySQL. MariaDB's JSON functions are much faster than MySQL's so there is no need to store in binary format, which would add complexity when manipulating JSON objects.
- For the same reason, MariaDB's JSON data type is an alias for LONGTEXT. If you want to replicate JSON columns from MySQL to MariaDB, you should store JSON objects in MySQL in a TEXT or LONGTEXT column or use statement based replication. If you are using JSON columns and want to upgrade to MariaDB, you need to either convert them to TEXT or use mysqldump to copy these tables to MariaDB.
- In MySQL, JSON is compared according to json values. In MariaDB JSON strings are normal strings and compared as strings.
- MariaDB 10.4 does not support MySQL's JSON operators (`->` and `->>`) - MDEV-13594
- MariaDB 10.4 supports the standard by producing null and a warning for JSON_SEARCH when given invalid data, while MySQL produces an error.
- Roles
  - MariaDB never allows authentication via roles, while MySQL permits this.
  - MySQL permits activating multiple roles at the same time. MariaDB can achieve the same result by creating an intermediate aggregate role.
  - In the INFORMATION_SCHEMA.ENABLED_ROLES table, MySQL reports just the direct list of enabled roles, while MariaDB reports the enabled role, plus the effective inherited roles.
  - MySQL extends the INFORMATION_SCHEMA.APPLICABLE_ROLES table .
  - MySQL includes the tables INFORMATION_SCHEMA.ROLE_TABLE_GRANTS, INFORMATION_SCHEMA.ROLE_ROUTINE_GRANTS,

INFORMATION_SCHEMA.ROLE_COLUMN_GRANTS, and INFORMATION_SCHEMA ADMINISTRABLE_ROLE_AUTHORIZATIONS.

- MySQL has the performance schema enabled by default. For performance reasons MariaDB 10.4 has it disabled by default. You can enable it by starting `mysqld` with the option `--performance-schema`.
- MySQL features a new implementation of the performance_schema and a sys schema wrapper. These are only supported in MariaDB 10.5.
- In MariaDB 10.4, using FLUSH TABLES without any table list will only close tables not in use, and tables not locked by the FLUSH TABLES connection. If there are no locked tables, FLUSH TABLES will be instant and will not cause any waits, as it no longer waits for tables in use. When a table list is provided, the server will wait for the end of any transactions that are using the tables. In MySQL, FLUSH TABLES only waits for the statements to complete.
- MariaDB binaries (`mysqld`, myisamchk etc.) give a warning if one uses a unique prefix of an option (such as `--big-table` instead of `--big-tables`). MySQL binaries require the full option name.
- MariaDB 10.4 implements InnoDB encryption in a different way to MySQL 8.0.
- MySQL's implementation of aborting statements that exceed a certain time to execute can only kill SELECTs, while MariaDB's can kill any queries (excluding stored procedures).
- MariaDB 10.4 does not support MySQL's `SELECT /*+ MAX_EXECUTION_TIME(n) */ ...` - see Aborting Statements that Exceed a Certain Time to Execute.
- MySQL 8.0 does not support the Query Cache.
- MariaDB 10.4 does not support the MySQL Memcached plugin (which has been deprecated in MySQL 8.0). However, data stored using memcached can be retrieved because the data is stored as InnoDB tables. MariaDB is able to start successfully with an error message of not being able to find libmemcached.so library.
- MariaDB 10.4 does not support MySQL 8.0's ALTER TABLE...RENAME INDEX statements (supported in MariaDB 10.5).
- In MySQL, `X'HHHH'`, the standard SQL syntax for binary string literals, erroneously works in the same way as `0xHHHH`, which could work as a number or string depending on the context. In MariaDB, this has been fixed to behave as a string in all contexts (and never as a number). See CAST and Hexadecimal Literals for more details and examples.
- In MariaDB 10.4, SHOW CREATE TABLE does not quote the DEFAULT value of an integer. MariaDB 10.2 and earlier, and MySQL, do. Since MariaDB can support defaults for BLOB and TEXT fields, while MySQL does not, SHOW CREATE TABLE will also append `DEFAULT NULL` where no default is explicitly provided to nullable BLOB or TEXT fields in MariaDB.
- Since MariaDB supports INTERSECT and EXCEPT, these are both reserved words and can't be used as an identifier without being quoted.
- As a result of implementing Table Value Constructors, the VALUES function has been renamed to VALUE().
- MariaDB's NOWAIT supports SELECT statements, LOCK TABLES and various DDL statements, while MySQL's NOWAIT only supports SELECT.
- MariaDB's NOWAIT cannot be added on views and stored procedures while MySQL's can - MDEV-25247
- MariaDB returns an `ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction` when unable to lock within the time, while MySQL returns `ERROR 3572 (HY000): Statement aborted because lock(s) could not be acquired immediately and NOWAIT is set`
- MariaDB does not support RENAME table while it is write-locked - MDEV-30814
- MariaDB 10.4 does not support Lateral Derived Tables - MDEV-19078.
- MariaDB does not support the optional *init_vector* argument for AES_ENCRYPT and AES_DECRYPT or the block_encryption_mode variable - MDEV-9069
- MySQL supports `SKIP LOCKED`, while MariaDB doesn't.
- MariaDB does not support the `--initialize` option. Use mysql_install_db instead. - MDEV-19010
- MariaDB 10.4 does not support the ngram and MeCab full-text parser plugins - MDEV-10267, MDEV-10268.
- MariaDB 10.4 does not support the MySQL X plugin.
- MariaDB 10.4 does not support MySQL 8's “native” InnoDB partitioning handler - MDEV-29253
- MariaDB 10.4 does not support CREATE TABLESPACE for InnoDB.
- The MySQL 8.0 and MariaDB 10.4 INFORMATION_SCHEMA.COLUMNS table contain slightly different fields.
- The MySQL binary log includes the thread_id, while MariaDB's binary log does not - MDEV-7850
- The MariaDB 10.1 syntax supporting Spatial Reference System IDs for spatial data type columns with REF_SYSTEM_ID is not supported by MySQL. MySQL 8 introduced `CREATE SPATIAL REFERENCE SYSTEM`, which is not supported by MariaDB - MDEV-29953
- MariaDB 10.4 does not support the RESTART statement - MDEV-30813
- MariaDB 10.4 does not support the SELECT FOR UPDATE and FOR SHARE locks - MDEV-17514

- Also see Incompatibilities between MariaDB 10.3 and MySQL 5.7 and Incompatibilities between MariaDB 10.2 and MySQL 5.7.

2.1.14.1.11 Function Differences Between MariaDB and MySQL

Functions in MariaDB that are not present in MySQL, or vice-versa.

Function Differences Between MariaDB 11.1 and MySQL 8.0
Functions present in MariaDB 11.1 and not present in MySQL 8.0 and vice-versa.

Function Differences Between MariaDB 11.0 and MySQL 8.0
Functions present in MariaDB 11.0 and not present in MySQL 8.0 and vice-versa.

Function Differences Between MariaDB 10.11 and MySQL 8.0
Functions present in MariaDB 10.11 and not present in MySQL 8.0 and vice-versa.

Function Differences Between MariaDB 10.10 and MySQL 8.0
Functions present in MariaDB 10.10 and not present in MySQL 8.0 and vice-versa.

Function Differences Between MariaDB 10.9 and MySQL 8.0
Functions present in MariaDB 10.9 and not present in MySQL 8.0 and vice-versa.

Function Differences Between MariaDB 10.8 and MySQL 8.0
Functions present in MariaDB 10.8 and not present in MySQL 8.0 and vice-versa.

Function Differences Between MariaDB 10.6 and MySQL 8.0
Functions present in MariaDB 10.6 and not present in MySQL 8.0 and vice-versa.

Function Differences Between MariaDB 10.5 and MySQL 8.0
Functions present in MariaDB 10.5 and not present in MySQL 8.0 and vice-versa.

Function Differences Between MariaDB 10.4 and MySQL 8.0
Functions present in MariaDB 10.4 and not present in MySQL 8.0 and vice-versa.

Function Differences Between MariaDB and MySQL - Unmaintained Series
Comparison of function differences between major unmaintained series of MariaDB and MySQL.

2.1.14.1.11.1 Function Differences Between MariaDB 11.1 and MySQL 8.0

Contents

The following is a list of all function differences between MariaDB 11.1 and MySQL 8.0. It is based on functions available in the MySQL 8.0.34 and the MariaDB 11.0.1 releases. For a more complete list of differences, see Incompatibilities and Feature Differences Between MariaDB 11.1 and MySQL 8.0

Present in MariaDB Only

Dynamic Columns

- COLUMN_ADD
- COLUMN_CHECK
- COLUMN_CREATE
- COLUMN_DELETE

- COLUMN_EXISTS
- COLUMN_GET
- COLUMN_JSON
- COLUMN_LIST

Galera

- WSREP_LAST_SEEN_GTID
- WSREP_LAST_WRITTEN_GTID
- WSREP_SYNC_WAIT_UPTO_GTID

General

- ADD_MONTHS
- CHR
- DECODE_ORACLE
- DES_DECRYPT
- DES_ENCRYPT
- LENGTHB
- NATURAL_SORT_KEY
- NVL (Synonym for IFNULL)
- NVL2
- SFORMAT
- SYS_GUID
- TO_CHAR
- TRIM_ORACLE
- VALUE - the VALUES() function was renamed after MariaDB introduced Table Value Constructors.

Geographic

MySQL has removed the following functions in MySQL 8.0.

- AREA
- AsBinary
- AsText
- AsWKB
- AsWKT
- Buffer
- Centroid
- Contains
- ConvexHull
- Crosses
- Dimension
- Disjoint
- EndPoint
- Envelope
- Equals
- ExteriorRing
- GeomCollFromText
- GeomCollFromWKB
- GeomFromText
- GeomFromWKB
- GeometryCollectionFromText
- GeometryCollectionFromWKB
- GeometryFromText
- GeometryFromWKB
- GeometryN
- GeometryType
- GLENGTH
- InteriorRingN
- Intersects
- IsClosed
- IsEmpty
- IsSimple
- LineFromText
- LineFromWKB

- LineStringFromText
- LineStringFromWKB
- MLineFromText
- MLineFromWKB
- MPointFromText
- MPointFromWKB
- MPolyFromText
- MPolyFromWKB
- MultiLineStringFromText
- MultiLineStringFromWKB
- MultiPointFromText
- MultiPointFromWKB
- MultiPolygonFromText
- MultiPolygonFromWKB
- NumGeometries
- NumInteriorRings
- NumPoints
- Overlaps
- PointFromText
- PointFromWKB
- PointN
- PolyFromText
- PolyFromWKB
- PolygonFromText
- PolygonFromWKB
- SRID
- StartPoint
- Touches
- Within
- X
- Y

JSON

- JSON_COMPACT
- JSON_DETAILED
- JSON_EQUALS
- JSON_EXISTS
- JSON_LOOSE
- JSON_NORMALIZE
- JSON_QUERY

Sequences

- LASTVAL
- NEXTVAL
- SETVAL

Window Functions

- MEDIAN
- PERCENTILE_CONT
- PERCENTILE_DISC

Present in MySQL Only

GTID

MariaDB and MySQL have differing GTID implementations.

- GTID_SUBSET
- GTID_SUBTRACT
- WAIT_FOR_EXECUTED_GTID_SET
- WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS

Geographic

- MBRCOVEREDBY
- ST_BUFFER_STRATEGY
- ST_Collect
- ST_FrechetDistance
- ST_GeoHash
- ST_HausdorffDistance
- ST_IsValid
- ST_LatFromGeoHash
- ST_LATITUDE
- ST_LineInterpolatePoint
- ST_LineInterpolatePoints
- ST_LongFromGeoHash
- ST_LONGITUDE
- ST_PointAtDistance
- ST_PointFromGeoHash
- ST_SIMPLIFY
- ST_VALIDATE (MDEV-17398)

JSON

- JSON_SCHEMA_VALIDATION_REPORT
- JSON_STORAGE_FREE
- JSON_STORAGE_SIZE (MDEV-17397)
- MEMBER_OF operator

Regular Expressions

- REGEXP_LIKE (MDEV-16599)

UUID

- BIN_TO_UUID
- IS_UUID
- UUID_TO_BIN (MDEV-15854)

Miscellaneous

- ANY_VALUE (MDEV-10426)
- ASYNCHRONOUS_CONNECTION_FAILOVER_ADD_SOURCE
- ASYNCHRONOUS_CONNECTION_FAILOVER_DELETE_SOURCE
- FORMAT_BYTES (MDEV-19629)
- GROUPING (MDEV-32789)
- PS_THREAD_ID (MDEV-19629)
- PS_CURRENT_THREAD_ID
- SOURCE_POS_WAIT
- VALIDATE_PASSWORD_STRENGTH (MDEV-25703)

2.1.14.1.11.2 Function Differences Between MariaDB 11.0 and MySQL 8.0

Contents

The following is a list of all function differences between MariaDB 11.0 and MySQL 8.0. It is based on functions available in the MySQL 8.0.34 and the MariaDB 11.0.2 releases. For a more complete list of differences, see Incompatibilities and Feature Differences Between MariaDB 11.0 and MySQL 8.0

Present in MariaDB Only

Dynamic Columns

- COLUMN_ADD
- COLUMN_CHECK
- COLUMN_CREATE
- COLUMN_DELETE
- COLUMN_EXISTS
- COLUMN_GET
- COLUMN_JSON
- COLUMN_LIST

Galera

- WSREP_LAST_SEEN_GTID
- WSREP_LAST_WRITTEN_GTID
- WSREP_SYNC_WAIT_UPTO_GTID

General

- ADD_MONTHS
- CHR
- DECODE_ORACLE
- DES_DECRYPT
- DES_ENCRYPT
- LENGTHB
- NATURAL_SORT_KEY
- NVL (Synonym for IFNULL)
- NVL2
- SFORMAT
- SYS_GUID
- TO_CHAR
- TRIM_ORACLE
- VALUE - the VALUES() function was renamed after MariaDB introduced Table Value Constructors.

Geographic

MySQL has removed the following functions in MySQL 8.0.

- AREA
- AsBinary
- AsText
- AsWKB

- AsWKT
- Buffer
- Centroid
- Contains
- ConvexHull
- Crosses
- Dimension
- Disjoint
- EndPoint
- Envelope
- Equals
- ExteriorRing
- GeomCollFromText
- GeomCollFromWKB
- GeomFromText
- GeomFromWKB
- GeometryCollectionFromText
- GeometryCollectionFromWKB
- GeometryFromText
- GeometryFromWKB
- GeometryN
- GeometryType
- GLENGTH
- InteriorRingN
- Intersects
- IsClosed
- IsEmpty
- IsSimple
- LineFromText
- LineFromWKB
- LineStringFromText
- LineStringFromWKB
- MLineFromText
- MLineFromWKB
- MPointFromText
- MPointFromWKB
- MPolyFromText
- MPolyFromWKB
- MultiLineStringFromText
- MultiLineStringFromWKB
- MultiPointFromText
- MultiPointFromWKB
- MultiPolygonFromText
- MultiPolygonFromWKB
- NumGeometries
- NumInteriorRings
- NumPoints
- Overlaps
- PointFromText
- PointFromWKB
- PointN
- PolyFromText
- PolyFromWKB
- PolygonFromText
- PolygonFromWKB
- SRID
- StartPoint
- Touches
- Within
- X
- Y

JSON

- JSON_COMPACT
- JSON_DETAILED
- JSON_EQUALS

- JSON_EXISTS
- JSON_LOOSE
- JSON_NORMALIZE
- JSON_QUERY

Sequences

- LASTVAL
- NEXTVAL
- SETVAL

Window Functions

- MEDIAN
- PERCENTILE_CONT
- PERCENTILE_DISC

Present in MySQL Only

GTID

MariaDB and MySQL have differing GTID implementations.

- GTID_SUBSET
- GTID_SUBTRACT
- WAIT_FOR_EXECUTED_GTID_SET
- WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS

Geographic

- MBRCOVEREDBY
- ST_BUFFER_STRATEGY
- ST_Collect
- ST_FrechetDistance
- ST_GeoHash
- ST_HausdorffDistance
- ST_IsValid
- ST_LatFromGeoHash
- ST_LATITUDE
- ST_LineInterpolatePoint
- ST_LineInterpolatePoints
- ST_LongFromGeoHash
- ST_LONGITUDE
- ST_PointAtDistance
- ST_PointFromGeoHash
- ST_SIMPLIFY
- ST_VALIDATE (MDEV-17398)

JSON

- JSON_SCHEMA_VALID (MDEV-27128)
- JSON_SCHEMA_VALIDATION_REPORT
- JSON_STORAGE_FREE
- JSON_STORAGE_SIZE (MDEV-17397)
- MEMBER_OF operator

Regular Expressions

- REGEXP_LIKE (MDEV-16599)

UUID

- BIN_TO_UUID
- IS_UUID

- UUID_TO_BIN (MDEV-15854)

Miscellaneous

- ANY_VALUE (MDEV-10426)
- ASYNCHRONOUS_CONNECTION_FAILOVER_ADD_SOURCE
- ASYNCHRONOUS_CONNECTION_FAILOVER_DELETE_SOURCE
- FORMAT_BYTES (MDEV-19629)
- GROUPING (MDEV-32789)
- PS_THREAD_ID (MDEV-19629)
- PS_CURRENT_THREAD_ID
- SOURCE_POS_WAIT
- VALIDATE_PASSWORD_STRENGTH (MDEV-25703)

2.1.14.1.11.3 Function Differences Between MariaDB 10.11 and MySQL 8.0

Contents

The following is a list of all function differences between MariaDB 10.11 and MySQL 8.0. It is based on functions available in the MySQL 8.0.32 and the MariaDB 10.11.2 releases. For a more complete list of differences, see Incompatibilities and Feature Differences Between MariaDB 10.11 and MySQL 8.0

Present in MariaDB Only

Dynamic Columns

- COLUMN_ADD
- COLUMN_CHECK
- COLUMN_CREATE
- COLUMN_DELETE
- COLUMN_EXISTS
- COLUMN_GET
- COLUMN_JSON
- COLUMN_LIST

Galera

- WSREP_LAST_SEEN_GTID
- WSREP_LAST_WRITTEN_GTID
- WSREP_SYNC_WAIT_UPTO_GTID

General

- ADD_MONTHS
- CHR
- DECODE_ORACLE

- DES_DECRYPT
- DES_ENCRYPT
- LENGTHB
- NATURAL_SORT_KEY
- NVL (Synonym for IFNULL)
- NVL2
- SFORMAT
- SYS_GUID
- TO_CHAR
- TRIM_ORACLE
- VALUE - the VALUES() function was renamed after MariaDB introduced Table Value Constructors.

Geographic

MySQL has removed the following functions in MySQL 8.0.

- AREA
- AsBinary
- AsText
- AsWKB
- AsWKT
- Buffer
- Centroid
- Contains
- ConvexHull
- Crosses
- Dimension
- Disjoint
- EndPoint
- Envelope
- Equals
- ExteriorRing
- GeomCollFromText
- GeomCollFromWKB
- GeomFromText
- GeomFromWKB
- GeometryCollectionFromText
- GeometryCollectionFromWKB
- GeometryFromText
- GeometryFromWKB
- GeometryN
- GeometryType
- GLENGTH
- InteriorRingN
- Intersects
- IsClosed
- IsEmpty
- IsSimple
- LineFromText
- LineFromWKB
- LineStringFromText
- LineStringFromWKB
- MLineFromText
- MLineFromWKB
- MPointFromText
- MPointFromWKB
- MPolyFromText
- MPolyFromWKB
- MultiLineStringFromText
- MultiLineStringFromWKB
- MultiPointFromText
- MultiPointFromWKB
- MultiPolygonFromText
- MultiPolygonFromWKB
- NumGeometries
- NumInteriorRings
- NumPoints

- Overlaps
- PointFromText
- PointFromWKB
- PointN
- PolyFromText
- PolyFromWKB
- PolygonFromText
- PolygonFromWKB
- SRID
- StartPoint
- Touches
- Within
- X
- Y

JSON

- JSON_COMPACT
- JSON_DETAILED
- JSON_EQUALS
- JSON_EXISTS
- JSON_LOOSE
- JSON_NORMALIZE
- JSON_QUERY

Sequences

- LASTVAL
- NEXTVAL
- SETVAL

Window Functions

- MEDIAN
- PERCENTILE_CONT
- PERCENTILE_DISC

Present in MySQL Only

GTID

MariaDB and MySQL have differing GTID implementations.

- GTID_SUBSET
- GTID_SUBTRACT
- WAIT_FOR_EXECUTED_GTID_SET
- WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS

Geographic

- MBRCOVEREDBY
- ST_BUFFER_STRATEGY
- ST_Collect
- ST_FrechetDistance
- ST_GeoHash
- ST_HausdorffDistance
- ST_IsValid
- ST_LatFromGeoHash
- ST_LATITUDE
- ST_LineInterpolatePoint
- ST_LineInterpolatePoints
- ST_LongFromGeoHash
- ST_LONGITUDE
- ST_PointAtDistance
- ST_PointFromGeoHash

- ST_SIMPLIFY
- ST_VALIDATE (MDEV-17398)

JSON

- JSON_SCHEMA_VALID (MDEV-27128)
- JSON_SCHEMA_VALIDATION_REPORT
- JSON_STORAGE_FREE
- JSON_STORAGE_SIZE (MDEV-17397)
- MEMBER_OF operator

Regular Expressions

- REGEXP_LIKE (MDEV-16599)

UUID

- BIN_TO_UUID
- IS_UUID
- UUID_TO_BIN (MDEV-15854)

Miscellaneous

- ANY_VALUE (MDEV-10426)
- ASYNCHRONOUS_CONNECTION_FAILOVER_ADD_SOURCE
- ASYNCHRONOUS_CONNECTION_FAILOVER_DELETE_SOURCE
- FORMAT_BYTES (MDEV-19629)
- FORMAT_PICO_TIME (MDEV-19629)
- GROUPING (MDEV-32789)
- PS_THREAD_ID (MDEV-19629)
- PS_CURRENT_THREAD_ID
- SOURCE_POS_WAIT
- VALIDATE_PASSWORD_STRENGTH (MDEV-25703)

2.1.14.1.11.4 Function Differences Between MariaDB 10.10 and MySQL 8.0

Contents

The following is a list of all function differences between MariaDB 10.10 and MySQL 8.0. It is based on functions available in the MySQL 8.0.32 and the MariaDB 10.10.3 releases. For a more complete list of differences, see Incompatibilities and Feature Differences Between MariaDB 10.10 and MySQL 8.0

Present in MariaDB Only

Dynamic Columns

- COLUMN_ADD
- COLUMN_CHECK
- COLUMN_CREATE
- COLUMN_DELETE
- COLUMN_EXISTS
- COLUMN_GET
- COLUMN_JSON
- COLUMN_LIST

Galera

- WSREP_LAST_SEEN_GTID
- WSREP_LAST_WRITTEN_GTID
- WSREP_SYNC_WAIT_UPTO_GTID

General

- ADD_MONTHS
- CHR
- DECODE_ORACLE
- DES_DECRYPT
- DES_ENCRYPT
- LENGTHB
- NATURAL_SORT_KEY
- NVL (Synonym for IFNULL)
- SFORMAT
- NVL2
- SFORMAT
- SYS_GUID
- TO_CHAR
- TRIM_ORACLE
- VALUE - the VALUES() function was renamed after MariaDB introduced Table Value Constructors.

Geographic

MySQL has removed the following functions in MySQL 8.0.

- AREA
- AsBinary
- AsText
- AsWKB
- AsWKT
- Buffer
- Centroid
- Contains
- ConvexHull
- Crosses
- Dimension
- Disjoint
- EndPoint
- Envelope
- Equals
- ExteriorRing
- GeomCollFromText
- GeomCollFromWKB
- GeomFromText
- GeomFromWKB
- GeometryCollectionFromText
- GeometryCollectionFromWKB
- GeometryFromText
- GeometryFromWKB
- GeometryN
- GeometryType
- GLENGTH
- InteriorRingN
- Intersects

- IsClosed
- IsEmpty
- IsSimple
- LineFromText
- LineFromWKB
- LineStringFromText
- LineStringFromWKB
- MLineFromText
- MLineFromWKB
- MPointFromText
- MPointFromWKB
- MPolyFromText
- MPolyFromWKB
- MultiLineStringFromText
- MultiLineStringFromWKB
- MultiPointFromText
- MultiPointFromWKB
- MultiPolygonFromText
- MultiPolygonFromWKB
- NumGeometries
- NumInteriorRings
- NumPoints
- Overlaps
- PointFromText
- PointFromWKB
- PointN
- PolyFromText
- PolyFromWKB
- PolygonFromText
- PolygonFromWKB
- SRID
- StartPoint
- Touches
- Within
- X
- Y

JSON

- JSON_COMPACT
- JSON_DETAILED
- JSON_EQUALS
- JSON_EXISTS
- JSON_LOOSE
- JSON_NORMALIZE
- JSON_QUERY

Sequences

- LASTVAL
- NEXTVAL
- SETVAL

Window Functions

- MEDIAN
- PERCENTILE_CONT
- PERCENTILE_DISC

Present in MySQL Only

GTID

MariaDB and MySQL have differing GTID implementations.

- GTID_SUBSET
- GTID_SUBTRACT
- WAIT_FOR_EXECUTED_GTID_SET
- WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS

Geographic

- MBRCOVEREDBY
- ST_BUFFER_STRATEGY
- ST_Collect
- ST_FrechetDistance
- ST_GeoHash
- ST_HausdorffDistance
- ST_IsValid
- ST_LatFromGeoHash
- ST_LATITUDE
- ST_LineInterpolatePoint
- ST_LineInterpolatePoints
- ST_LongFromGeoHash
- ST_LONGITUDE
- ST_PointAtDistance
- ST_PointFromGeoHash
- ST_SIMPLIFY
- ST_TRANSFORM
- ST_VALIDATE (MDEV-17398)

JSON

- JSON_SCHEMA_VALID (MDEV-27128)
- JSON_SCHEMA_VALIDATION_REPORT
- JSON_STORAGE_FREE
- JSON_STORAGE_SIZE (MDEV-17397)
- MEMBER_OF operator

Regular Expressions

- REGEXP_LIKE (MDEV-16599)

UUID

- BIN_TO_UUID
- IS_UUID
- UUID_TO_BIN (MDEV-15854)

Miscellaneous

- ANY_VALUE (MDEV-10426)
- ASYNCHRONOUS_CONNECTION_FAILOVER_ADD_SOURCE
- ASYNCHRONOUS_CONNECTION_FAILOVER_DELETE_SOURCE
- FORMAT_BYTES (MDEV-19629)
- FORMAT_PICO_TIME (MDEV-19629)
- GROUPING (MDEV-32789)
- PS_THREAD_ID (MDEV-19629)
- PS_CURRENT_THREAD_ID
- VALIDATE_PASSWORD_STRENGTH (MDEV-25703)

2.1.14.1.11.5 Function Differences Between MariaDB 10.9 and MySQL 8.0

Contents

The following is a list of all function differences between MariaDB 10.9 and MySQL 8.0. It is based on functions available in the MySQL 8.0.24 and the MariaDB 10.9.5 releases. For a more complete list of differences, see Incompatibilities and Feature Differences Between MariaDB 10.9 and MySQL 8.0

Present in MariaDB Only

Dynamic Columns

- COLUMN_ADD
- COLUMN_CHECK
- COLUMN_CREATE
- COLUMN_DELETE
- COLUMN_EXISTS
- COLUMN_GET
- COLUMN_JSON
- COLUMN_LIST

Galera

- WSREP_LAST_SEEN_GTID
- WSREP_LAST_WRITTEN_GTID
- WSREP_SYNC_WAIT_UPTO_GTID

General

- ADD_MONTHS
- CHR
- DECODE_ORACLE
- DES_DECRYPT
- DES_ENCRYPT
- LENGTHB
- NATURAL_SORT_KEY
- NVL (Synonym for IFNULL)
- NVL2
- SFORMAT
- SYS_GUID
- TO_CHAR
- TRIM_ORACLE
- VALUE - the VALUES() function was renamed after MariaDB introduced Table Value Constructors.

Geographic

MySQL has removed the following functions in MySQL 8.0.

- AREA
- AsBinary
- AsText
- AsWKB

- AsWKT
- Buffer
- Centroid
- Contains
- ConvexHull
- Crosses
- Dimension
- Disjoint
- EndPoint
- Envelope
- Equals
- ExteriorRing
- GeomCollFromText
- GeomCollFromWKB
- GeomFromText
- GeomFromWKB
- GeometryCollectionFromText
- GeometryCollectionFromWKB
- GeometryFromText
- GeometryFromWKB
- GeometryN
- GeometryType
- GLENGTH
- InteriorRingN
- Intersects
- IsClosed
- IsEmpty
- IsSimple
- LineFromText
- LineFromWKB
- LineStringFromText
- LineStringFromWKB
- MLineFromText
- MLineFromWKB
- MPointFromText
- MPointFromWKB
- MPolyFromText
- MPolyFromWKB
- MultiLineStringFromText
- MultiLineStringFromWKB
- MultiPointFromText
- MultiPointFromWKB
- MultiPolygonFromText
- MultiPolygonFromWKB
- NumGeometries
- NumInteriorRings
- NumPoints
- Overlaps
- PointFromText
- PointFromWKB
- PointN
- PolyFromText
- PolyFromWKB
- PolygonFromText
- PolygonFromWKB
- SRID
- StartPoint
- Touches
- Within
- X
- Y

JSON

- JSON_COMPACT
- JSON_DETAILED
- JSON_EQUALS

- JSON_EXISTS
- JSON_LOOSE
- JSON_NORMALIZE
- JSON_QUERY

Sequences

- LASTVAL
- NEXTVAL
- SETVAL

Window Functions

- MEDIAN
- PERCENTILE_CONT
- PERCENTILE_DISC

Present in MySQL Only

GTID

MariaDB and MySQL have differing GTID implementations.

- GTID_SUBSET
- GTID_SUBTRACT
- WAIT_FOR_EXECUTED_GTID_SET
- WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS

Geographic

- MBRCOVEREDBY
- ST_BUFFER_STRATEGY
- ST_Collect
- ST_FrechetDistance
- ST_GeoHash
- ST_HausdorffDistance
- ST_IsValid
- ST_LatFromGeoHash
- ST_LATITUDE
- ST_LineInterpolatePoint
- ST_LineInterpolatePoints
- ST_LongFromGeoHash
- ST_LONGITUDE
- ST_PointAtDistance
- ST_PointFromGeoHash
- ST_SIMPLIFY
- ST_TRANSFORM
- ST_VALIDATE (MDEV-17398)

JSON

- JSON_SCHEMA_VALID (MDEV-27128)
- JSON_SCHEMA_VALIDATION_REPORT
- JSON_STORAGE_FREE
- JSON_STORAGE_SIZE (MDEV-17397)
- MEMBER_OF operator

Regular Expressions

- REGEXP_LIKE (MDEV-16599)

UUID

- BIN_TO_UUID

- IS_UUID
- UUID_TO_BIN (MDEV-15854)

Miscellaneous

- ANY_VALUE (MDEV-10426)
- ASYNCHRONOUS_CONNECTION_FAILOVER_ADD_SOURCE
- ASYNCHRONOUS_CONNECTION_FAILOVER_DELETE_SOURCE
- FORMAT_BYTES (MDEV-19629)
- FORMAT_PICO_TIME (MDEV-19629)
- GROUPING (MDEV-32789)
- PS_THREAD_ID (MDEV-19629)
- PS_CURRENT_THREAD_ID
- RANDOM_BYTES (MDEV-25704)
- VALIDATE_PASSWORD_STRENGTH (MDEV-25703)

2.1.14.1.11.6 Function Differences Between MariaDB 10.6 and MySQL 8.0

Contents

The following is a list of all function differences between MariaDB 10.6 and MySQL 8.0. It is based on functions available in the MySQL 8.0.23 and the MariaDB 10.6.12 releases. For a more complete list of differences, see Incompatibilities and Feature Differences Between MariaDB 10.6 and MySQL 8.0

Present in MariaDB Only

Dynamic Columns

- COLUMN_ADD
- COLUMN_CHECK
- COLUMN_CREATE
- COLUMN_DELETE
- COLUMN_EXISTS
- COLUMN_GET
- COLUMN_JSON
- COLUMN_LIST

Galera

- WSREP_LAST_SEEN_GTID
- WSREP_LAST_WRITTEN_GTID
- WSREP_SYNC_WAIT_UPTO_GTID

General

- ADD_MONTHS

- CHR
- DECODE_ORACLE
- DES_DECRYPT
- DES_ENCRYPT
- LENGTHB
- NVL (Synonym for IFNULL)
- NVL2
- SYS_GUID
- TO_CHAR
- TRIM_ORACLE
- VALUE - the VALUES() function was renamed after MariaDB introduced Table Value Constructors.

Geographic

MySQL has removed the following functions in MySQL 8.0.

- AREA
- AsBinary
- AsText
- AsWKB
- AsWKT
- Buffer
- Centroid
- Contains
- ConvexHull
- Crosses
- Dimension
- Disjoint
- EndPoint
- Envelope
- Equals
- ExteriorRing
- GeomCollFromText
- GeomCollFromWKB
- GeomFromText
- GeomFromWKB
- GeometryCollectionFromText
- GeometryCollectionFromWKB
- GeometryFromText
- GeometryFromWKB
- GeometryN
- GeometryType
- GLENGTH
- InteriorRingN
- Intersects
- IsClosed
- IsEmpty
- IsSimple
- LineFromText
- LineFromWKB
- LineStringFromText
- LineStringFromWKB
- MLineFromText
- MLineFromWKB
- MPointFromText
- MPointFromWKB
- MPolyFromText
- MPolyFromWKB
- MultiLineStringFromText
- MultiLineStringFromWKB
- MultiPointFromText
- MultiPointFromWKB
- MultiPolygonFromText
- MultiPolygonFromWKB
- NumGeometries
- NumInteriorRings
- NumPoints

- Overlaps
- PointFromText
- PointFromWKB
- PointN
- PolyFromText
- PolyFromWKB
- PolygonFromText
- PolygonFromWKB
- SRID
- StartPoint
- Touches
- Within
- X
- Y

JSON

- JSON_COMPACT
- JSON_DETAILED
- JSON_EXISTS
- JSON_LOOSE
- JSON_QUERY
- JSON_VALUE

Sequences

- LASTVAL
- NEXTVAL
- SETVAL

Window Functions

- MEDIAN
- PERCENTILE_CONT
- PERCENTILE_DISC

Present in MySQL Only

GTID

MariaDB and MySQL have differing GTID implementations.

- GTID_SUBSET
- GTID_SUBTRACT
- WAIT_FOR_EXECUTED_GTID_SET
- WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS

Geographic

- MBRCOVEREDBY
- ST_BUFFER_STRATEGY
- ST_FrechetDistance
- ST_GeoHash
- ST_HausdorffDistance
- ST_IsValid
- ST_LatFromGeoHash
- ST_LATITUDE
- ST_LongFromGeoHash
- ST_LONGITUDE
- ST_PointFromGeoHash
- ST_SIMPLIFY
- ST_TRANSFORM
- ST_VALIDATE (MDEV-17398)

JSON

- JSON_OVERLAPS (MDEV-27677)
- JSON_SCHEMA_VALID (MDEV-27128)
- JSON_SCHEMA_VALIDATION_REPORT
- JSON_STORAGE_FREE
- JSON_STORAGE_SIZE (MDEV-17397)
- MEMBER_OF operator

Regular Expressions

- REGEXP_LIKE (MDEV-16599)

UUID

- BIN_TO_UUID
- IS_UUID
- UUID_TO_BIN (MDEV-15854)

Miscellaneous

- ANY_VALUE (MDEV-10426)
- ASYNCHRONOUS_CONNECTION_FAILOVER_ADD_SOURCE
- ASYNCHRONOUS_CONNECTION_FAILOVER_DELETE_SOURCE
- FORMAT_BYTES (MDEV-19629)
- FORMAT_PICO_TIME (MDEV-19629)
- GROUPING (MDEV-32789)
- PS_THREAD_ID (MDEV-19629)
- PS_CURRENT_THREAD_ID
- RANDOM_BYTES (MDEV-25704)
- VALIDATE_PASSWORD_STRENGTH (MDEV-25703)

2.1.14.1.11.7 Function Differences Between MariaDB 10.5 and MySQL 8.0

Contents

The following is a list of all function differences between MariaDB 10.5 and MySQL 8.0. It is based on functions available in the MySQL 8.0.17 and the MariaDB 10.5.19 releases. For a more complete list of differences, see Incompatibilities and Feature Differences Between MariaDB 10.5 and MySQL 8.0

Present in MariaDB Only

Dynamic Columns

- COLUMN_ADD
- COLUMN_CHECK

- COLUMN_CREATE
- COLUMN_DELETE
- COLUMN_EXISTS
- COLUMN_GET
- COLUMN_JSON
- COLUMN_LIST

Galera

- WSREP_LAST_SEEN_GTID
- WSREP_LAST_WRITTEN_GTID
- WSREP_SYNC_WAIT_UPTO_GTID

General

- CHR
- DECODE_ORACLE
- DES_DECRYPT
- DES_ENCRYPT
- LENGTHB
- NVL (Synonym for IFNULL)
- NVL2
- TRIM_ORACLE
- VALUE - the VALUES() function was renamed after MariaDB introduced Table Value Constructors.

Geographical

MySQL has removed the following functions in MySQL 8.0.

- AREA
- AsBinary
- AsText
- AsWKB
- AsWKT
- Buffer
- Centroid
- Contains
- ConvexHull
- Crosses
- Dimension
- Disjoint
- EndPoint
- Envelope
- Equals
- ExteriorRing
- GeomCollFromText
- GeomCollFromWKB
- GeomFromText
- GeomFromWKB
- GeometryCollectionFromText
- GeometryCollectionFromWKB
- GeometryFromText
- GeometryFromWKB
- GeometryN
- GeometryType
- GLENGTH
- InteriorRingN
- Intersects
- IsClosed
- IsEmpty
- IsSimple
- LineFromText
- LineFromWKB
- LineStringFromText
- LineStringFromWKB
- MLineFromText

- MLineFromWKB
- MPointFromText
- MPointFromWKB
- MPolyFromText
- MPolyFromWKB
- MultiLineStringFromText
- MultiLineStringFromWKB
- MultiPointFromText
- MultiPointFromWKB
- MultiPolygonFromText
- MultiPolygonFromWKB
- NumGeometries
- NumInteriorRings
- NumPoints
- Overlaps
- PointFromText
- PointFromWKB
- PointN
- PolyFromText
- PolyFromWKB
- PolygonFromText
- PolygonFromWKB
- SRID
- StartPoint
- Touches
- Within
- X
- Y

JSON

- JSON_COMPACT
- JSON_DETAILED
- JSON_EXISTS
- JSON_LOOSE
- JSON_QUERY
- JSON_VALUE

Sequences

- LASTVAL
- NEXTVAL
- SETVAL

Window Functions

- MEDIAN
- PERCENTILE_CONT
- PERCENTILE_DISC

Present in MySQL Only

GTID

MariaDB and MySQL have differing GTID implementations.

- GTID_SUBSET
- GTID_SUBTRACT
- WAIT_FOR_EXECUTED_GTID_SET
- WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS()

Geographic

- MBRCOVEREDBY
- ST_BUFFER_STRATEGY

- ST_GeoHash
- ST_IsValid
- ST_LatFromGeoHash
- ST_LATITUDE
- ST_LongFromGeoHash
- ST_LONGITUDE
- ST_PointFromGeoHash
- ST_SIMPLIFY
- ST_TRANSFORM
- ST_VALIDATE

JSON

- JSON_OVERLAPS
- JSON_SCHEMA_VALID (MDEV-27128)
- JSON_SCHEMA_VALIDATION_REPORT
- JSON_STORAGE_FREE
- JSON_STORAGE_SIZE (MDEV-17397)
- JSON_TABLE
- MEMBER_OF operator

Regular Expressions

- REGEXP_LIKE (MDEV-16599)

UUID

- BIN_TO_UUID
- IS_UUID
- UUID_TO_BIN

Miscellaneous

- ANY_VALUE
- FORMAT_BYTES (MDEV-19629)
- FORMAT_PICO_TIME (MDEV-19629)
- GROUPING (MDEV-32789)
- PS_THREAD_ID (MDEV-19629)
- PS_CURRENT_THREAD_ID
- RANDOM_BYTES
- VALIDATE_PASSWORD_STRENGTH

2.1.14.1.11.8 Function Differences Between MariaDB 10.4 and MySQL 8.0

Contents

The following is a list of all function differences between MariaDB 10.4 and MySQL 8.0. It is based on functions available in

the stable MySQL 8.0.17 and MariaDB 10.4.28 releases. For a more complete list of differences, see Incompatibilities and Feature Differences Between MariaDB 10.4 and MySQL 8.0.

Present in MariaDB Only

Dynamic Columns

- COLUMN_ADD
- COLUMN_CHECK
- COLUMN_CREATE
- COLUMN_DELETE
- COLUMN_EXISTS
- COLUMN_GET
- COLUMN_JSON
- COLUMN_LIST

Galera

- WSREP_LAST_SEEN_GTID
- WSREP_LAST_WRITTEN_GTID
- WSREP_SYNC_WAIT_UPTO_GTID

General

- CHR
- DECODE_ORACLE
- DES_DECRYPT
- DES_ENCRYPT
- LENGTHB
- NVL (Synonym for IFNULL)
- NVL2
- TRIM_ORACLE
- VALUE - the VALUES() function was renamed after MariaDB introduced Table Value Constructors.

Geographical

MySQL has removed the following functions in MySQL 8.0.

- AREA
- AsBinary
- AsText
- AsWKB
- AsWKT
- Buffer
- Centroid
- Contains
- ConvexHull
- Crosses
- Dimension
- Disjoint
- EndPoint
- Envelope
- Equals
- ExteriorRing
- GeomCollFromText
- GeomCollFromWKB
- GeomFromText
- GeomFromWKB
- GeometryCollectionFromText
- GeometryCollectionFromWKB
- GeometryFromText
- GeometryFromWKB
- GeometryN
- GeometryType
- GLENGTH

- InteriorRingN
- Intersects
- IsClosed
- IsEmpty
- IsSimple
- LineFromText
- LineFromWKB
- LineStringFromText
- LineStringFromWKB
- MLineFromText
- MLineFromWKB
- MPointFromText
- MPointFromWKB
- MPolyFromText
- MPolyFromWKB
- MultiLineStringFromText
- MultiLineStringFromWKB
- MultiPointFromText
- MultiPointFromWKB
- MultiPolygonFromText
- MultiPolygonFromWKB
- NumGeometries
- NumInteriorRings
- NumPoints
- Overlaps
- PointFromText
- PointFromWKB
- PointN
- PolyFromText
- PolyFromWKB
- PolygonFromText
- PolygonFromWKB
- SRID
- StartPoint
- Touches
- Within
- X
- Y

JSON

- JSON_COMPACT
- JSON_DETAILED
- JSON_EXISTS
- JSON_LOOSE
- JSON_QUERY
- JSON_VALUE

Sequences

- LASTVAL
- NEXTVAL
- SETVAL

Window Functions

- MEDIAN
- PERCENTILE_CONT
- PERCENTILE_DISC

Present in MySQL Only

GTID

MariaDB and MySQL have differing GTID implementations.

- GTID_SUBSET
- GTID_SUBTRACT
- WAIT_FOR_EXECUTED_GTID_SET
- WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS()

Geographic

- MBRCOVEREDBY
- ST_BUFFER_STRATEGY
- ST_GeoHash
- ST_IsValid
- ST_LatFromGeoHash
- ST_LATITUDE
- ST_LongFromGeoHash
- ST_LONGITUDE
- ST_PointFromGeoHash
- ST_SIMPLIFY
- ST_TRANSFORM
- ST_VALIDATE

JSON

- JSON_ARRAYAGG
- JSON_OBJECTAGG
- JSON_OVERLAPS
- JSON_SCHEMA_VALID (MDEV-27128)
- JSON_SCHEMA_VALIDATION_REPORT
- JSON_STORAGE_FREE
- JSON_STORAGE_SIZE (MDEV-17397)
- JSON_TABLE
- MEMBER_OF operator

Regular Expressions

- REGEXP_LIKE (MDEV-16599)

UUID

- BIN_TO_UUID
- IS_UUID
- UUID_TO_BIN

Miscellaneous

- ANY_VALUE
- FORMAT_BYTES (MDEV-19629)
- FORMAT_PICO_TIME (MDEV-19629)
- GROUPING (MDEV-32789)
- PS_THREAD_ID (MDEV-19629)
- PS_CURRENT_THREAD_ID
- RANDOM_BYTES
- RELEASE_ALL_LOCKS
- VALIDATE_PASSWORD_STRENGTH

2.1.14.1.12 System Variable Differences between MariaDB and MySQL

The following articles list the differences between the system variables available in MariaDB and in MySQL for each of the major releases.

System Variable Differences Between MariaDB 11.1 and MySQL 8.0

Comparison of MariaDB 11.1 and MySQL 8.0 system variables.

System Variable Differences Between MariaDB 11.0 and MySQL 8.0
Comparison of MariaDB 11.0 and MySQL 8.0 system variables.

System Variable Differences Between MariaDB 10.11 and MySQL 8.0
Comparison of MariaDB 10.11 and MySQL 8.0 system variables.

System Variable Differences Between MariaDB 10.10 and MySQL 8.0
Comparison of MariaDB 10.10 and MySQL 8.0 system variables.

System Variable Differences Between MariaDB 10.6 and MySQL 8.0
Comparison of MariaDB 10.6 and MySQL 8.0 system variables.

System Variable Differences Between MariaDB 10.5 and MySQL 8.0
Comparison of MariaDB 10.5 and MySQL 8.0 system variables.

System Variable Differences Between MariaDB 10.4 and MySQL 8.0
Comparison of MariaDB 10.4 and MySQL 8.0 system variables.

System Variable Differences Between MariaDB and MySQL - Unmaintained Series
Comparison of variable differences between major series of MariaDB and MySQL.

2.1.14.1.12.1 System Variable Differences Between MariaDB 11.1 and MySQL 8.0

Contents

The following is a comparison of variables that either appear only in MariaDB 11.1 or MySQL 8.0, or have different default settings in MariaDB 11.1, and MySQL 8.0. The releases MariaDB 11.1.2 and MySQL 8.0.34, with only default plugins enabled, were used for the comparison. Note that MySQL 8 is an 'evergreen' release, so features may be added or removed in later releases.

For a more complete list of differences, see Incompatibilities and Feature Differences Between MariaDB 11.1 and MySQL 8.0 and Function Differences Between MariaDB 11.1 and MySQL 8.0

Comparison Table

Variable	MariaDB 11.1 Default	MySQL 8.0 Default	Notes
activate_all_roles_on_login	-	OFF	Determines whether to automatically activate roles on login.
admin*	-	*	MySQL admin connections.
allow_suspicious_udfs	0	-	Only available as an option in MySQL.
alter_algorithm	DEFAULT	-	MariaDB 10.3 introduced new ALTER TABLE ALGORITHM clauses to avoid slow copies in certain instances. This variable allows setting this if no `ALGORITHM` clause is specified.
analyze_sample_percentage	100.0000	-	Percentage of rows from the table ANALYZE TABLE will sample to collect table statistics.
aria_*	*	-	The Aria storage engine is only available in MariaDB.
authentication_policy	-	*,,	MySQL authentication policy.
auto_generate_certs	-	ON	Whether to automatically generate SSL key and certificate files.
avoid_temporal_upgrade	-	OFF	Determines whether ALTER TABLE implicitly upgrades temporal columns.
back_log	Autosized	Autosized	MariaDB and MySQL have different autosizing algorithms.
binlog_alter_two_phase	OFF	-	When set, split ALTER at binary logging into two statements: START ALTER and COMMIT/ROLLBACK ALTER.
binlog_annotate_row_events	ON	-	Introduced in MariaDB 5.3 for replicating between MariaDB 5.3 and MySQL/MariaDB 5.1.

binlog_commit_wait_count	0	-	For use in MariaDB's parallel replication.
binlog_commit_wait_usec	100000	-	For use in MariaDB's parallel replication.
binlog_encryption	-	OFF	MySQL name for encrypt_binlog.
binlog_error_action		ABORT_SERVER	MySQL-only variable for controlling what happens when the server cannot write to the binary log.
binlog_expire_logs_auto_purge	-	ON	Enables or disables automatic purging of binary log files.
binlog_expire_logs_seconds	0	2592000	Sets the binary log expiration period in seconds
binlog_file_cache_size	16184	-	For setting the size of the file cache for the binary log.
binlog_format	MIXED	ROW	MariaDB and MySQL have differing binary log formats.
binlog_group_commit_sync_delay		0	MySQL-only variable for controlling the wait time before synchronizing the binary log file to disk.
binlog_group_commit_sync_no_delay_count		0	MySQL-only variable for setting the maximum number of transactions to wait for before aborting the current `binlog_group_commit_sync_delay` delay.
binlog_gtid_simple_recovery	-	ON	MySQL-only GTID variable. MariaDB's GTID implementation is different.
binlog_max_flush_queue_time	-	0	Specifies a timeout for reading transactions from the flush queue before continuing with group commit and syncing log to disk.
binlog_optimize_thread_scheduling	ON	-	For optimized kernel thread scheduling.
binlog_order_commits	-	ON	Determines whether transactions may be committed in parallel.
binlog_rotate_encryption_master_key_at_startup	-	OFF	Specifically for use with MySQL binary key encryption.
binlog_row_event_max_size	-	8192	Only available as a system variable in MariaDB 11.2.
binlog_row_metadata	NO_LOG	MINIMAL	Determines the amount of table metadata added to the binary log with row-based logging.
binlog_row_value_options	-	(empty)	Permits an alternative binlog format for JSON document updates.
binlog_rows_query_log_events	-	OFF	MySQL-only variable for logging extra information in row-based logging.
binlog_transaction_compression*	-	*	MySQL variables relating to binary log compression.
binlog_transaction_dependency_history_size	-	25000	Maximum number of row hashes kept for looking up transactions that last modified a given row.
binlog_transaction_dependency_tracking	-	COMMIT_ORDER	For determining how to best use the slave's multithreaded applier.
block_encryption_mode	-	aes-128-ecb	MySQL-only variable for controlling the block encryption mode for block-based algorithms.
caching_sha2_password*	-	*	For use with MySQL's SHA-256 authentication with caching.
character_set_*	latin1 or utf8	utf8mb4	MySQL 8.0 defaults to the utf8mb4 character set.
check_constraint_checks	ON	-	Permits disabling constraint checks, for example when loading a table that violates some constraints that you plan to fix later.
check_proxy_users		OFF	MySQL-only variable for controlling whether the server performs proxy user mapping for authentication plugins.
collation_*	latin1_swedish_ci or utf8_general_ci	utf8mb4_0900_ai_ci	MySQL 8.0 defaults to the utf8mb4 character set.
column_compression_threshold	100	-	MariaDB supports Storage-engine Independent Column Compression.
column_compression_zlib_level	6	-	MariaDB supports Storage-engine Independent Column Compression.
column_compression_zlib_strategy	DEFAULT_STRATEGY	-	MariaDB supports Storage-engine Independent Column Compression.
column_compression_zlib_wrap	OFF	-	MariaDB supports Storage-engine Independent Column Compression.

connection_memory_chunk_size	-	8192	Chunk size for updates to the Global_connection_memory counter.
connection_memory_limit	-	18446744073709551615	Maximum memory for a single user connection.
create_admin_listener_thread	-	OFF	MySQL-only variable for whether to use a dedicated listening thread for admin network interface connections.
cte_max_recursion_depth	-	1000	When MySQL 8.0 introduced common table expressions they used a different name. MariaDB's variable is called max_recursive_iterations.
date_format	%Y-%m-%d	-	Unused variable removed in MySQL 8.0
datetime_format	%Y-%m-%d	-	Unused variable removed in MySQL 8.0
deadlock_search_depth_long	15	-	The Aria storage engine is only available in MariaDB.
deadlock_search_depth_short	4	-	The Aria storage engine is only available in MariaDB.
deadlock_timeout_long	50000000	-	The Aria storage engine is only available in MariaDB.
deadlock_timeout_short	10000	-	The Aria storage engine is only available in MariaDB.
debug_no_thread_alarm	OFF	-	Disable system thread alarm calls, for debugging or testing.
default_authentication_plugin	-	caching_sha2_password	MySQL 8 introduced a new authentication plugin.
default_collation_for_utf8mb4	-	utf8mb4_0900_ai_ci	For internal use in MySQL 8 replication.
default_master_connection	empty	-	For use with MariaDB's multi-source replication.
default_password_lifetime	0	360	MariaDB defaults to password expiration off.
default_regex_flags	empty	-	For handling incompatibilities between MariaDB's PCRE and the old regex library.
default_tmp_storage_engine	empty	InnoDB	Default storage engine used for tables created with CREATE TEMPORARY TABLE.
disabled_storage_engines		empty	MySQL-only variable for disabling specific storage engines.
disconnect_on_expired_password	OFF	ON	MariaDB password expiration is off by default, and by default does not disconnect a client when a password has expired.
encrypt_binlog	OFF	-	MariaDB enables table and tablespace encryption.
encrypt_tmp_files	OFF	-	MariaDB enables table and tablespace encryption.
encrypt_tmp_disk_tables	OFF	-	MariaDB enables table and tablespace encryption.
end_markers_in_json	-	OFF	MySQL-only variable for adding end markers to JSON output.
enforce_gtid_consistency	-	OFF	MariaDB and MySQL have different GTID implementations.
enforce_storage_engine	none		Forces the use of a particular storage engine for new tables.
eq_range_index_dive_limit	0	200	Variable for tuning when the optimizer should switch from using index dives to index statistics for qualifying rows estimation.
event_scheduler	OFF	ON	MySQL enables the event scheduler by default.
expensive_subquery_limit	100	-	Used for determining expensive queries for optimization.
extra_max_connections	1	-	Introduced in the MariaDB 5.1 threadpool.
extra_port	0	-	Introduced in the MariaDB 5.1 threadpool.
group_concat_max_len	1048576	1024	MariaDB increases the maximum length for a GROUP_CONCAT() result from 1K to 1M.
gtid_binlog_pos	empty	-	MariaDB and MySQL have different GTID implementations.
gtid_binlog_state	empty	-	MariaDB and MySQL have different GTID implementations.

gtid_cleanup_batch_size	64	-	MariaDB and MySQL have different GTID implementations.
gtid_current_pos	empty	-	MariaDB and MySQL have different GTID implementations.
gtid_domain_id	0	-	MariaDB and MySQL have different GTID implementations.
gtid_executed	-	empty	MariaDB and MySQL have different GTID implementations.
gtid_executed_compression_period	-	1000	MariaDB and MySQL have different GTID implementations.
gtid_ignore_duplicates	OFF	-	MariaDB and MySQL have different GTID implementations.
gtid_mode	-	OFF	MariaDB and MySQL have different GTID implementations.
gtid_next	-	AUTOMATIC	MariaDB and MySQL have different GTID implementations.
gtid_owned	-	empty	MariaDB and MySQL have different GTID implementations.
gtid_pos_auto_engines	empty	-	MariaDB and MySQL have different GTID implementations.
gtid_purged	-	empty	MariaDB and MySQL have different GTID implementations.
gtid_seq_no	0	-	MariaDB and MySQL have different GTID implementations.
gtid_slave_pos	empty	-	MariaDB and MySQL have different GTID implementations.
gtid_strict_mode	OFF	-	MariaDB and MySQL have different GTID implementations.
have_crypt	YES	-	MySQL has removed the ENCRYPT function.
have_openssl			MariaDB's version indicates whether YaSSL or openssl was used. MySQL's is a synonym for `have_ssl`.
have_query_cache	YES	-	MySQL has removed the query cache.
have_statement_timeout	-		Whether MySQL's statement execution timeout feature is available.
have_symlink	YES	DISABLED	MySQL has removed symlink support.
histogram_generation_max_mem_size	-	20000000	Added when MySQL 8 introduced Histogram-based Statistics. MariaDB uses histogram_size
histogram_size	0	-	MariaDB introduced Histogram-based Statistics.
histogram_type	JSON_HB	-	MariaDB introduced Histogram-based Statistics.
idle_readonly_transaction_timeout	0	-	Time in seconds that the server waits for idle read-only transactions.
idle_transaction_timeout	0	-	Time in seconds that the server waits for idle transactions.
idle_write_transaction_timeout	0	-	Time in seconds that the server waits for idle write transactions.
ignore_builtin_innodb	OFF	-	Ignored and removed in MySQL 8.
in_predicate_conversion_threshold	1000	-	Controls the Conversion of Big IN Predicates Into Subqueries optimization.
in_transaction	0	-	Set to `1` if you are in a transaction, and `0` if not.
information_schema_stats_expiry	-	86400	Time until MySQL Information Schema cached statistics expire.
innodb_adaptive_flushing_lwm	10.000000	10	Adaptive flushing is enabled when this this low water mark percentage of the redo log capacity is reached. MariaDB's variable is a double, MySQL's an integer.
innodb_adaptive_hash_index	OFF	ON	Defaulting to OFF is a performance improvement especially for DROP TABLE, TRUNCATE TABLE, ALTER TABLE, or DROP INDEX operations
innodb_adaptive_max_sleep_delay	-	150000	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_api_*	-	*	Specific to MySQL's memcached, removed in MariaDB 10.2.

innodb_autoinc_lock_mode	1	2	MariaDB has an extra mode, `3`, for skipping the rollback of connected transactions. MySQL defaults to row-based replication, so can safely use `2`.
innodb_buffer_pool_instances	-	1	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6 since the original reasons for introducing no longer apply.
innodb_change*		*	The InnoDB Change Buffer was removed in MariaDB 11.0.
innodb_checksum_algorithm	full_crc32	crc32	`fullcrc32` permits encryption to be supported over a SPATIAL INDEX, which crc32 does not support.
innodb_checksums	ON	-	Deprecated option removed in MySQL.
innodb_commit_concurrency	-	0	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_compression_*	*	-	Introduced with MariaDB's InnoDB compression.
innodb_concurrency_tickets	-	5000	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_deadlock_report	Full	-	How to report deadlocks.
innodb_dedicated_server	-	OFF	MySQL option that automatically configures various settings if the server is a dedicated InnoDB database server.
innodb_default_encryption_key_id	1	-	Default encryption key id used for table encryption. See Data at Rest Encryption.
innodb_directories	-	(empty)	Used to search for tablespace files when moving or restoring a new location.
innodb_disallow_writes	OFF	-	Tell InnoDB to stop any writes to disk.
innodb_encrypt_*	1	-	See MariaDB's Data at Rest Encryption.
innodb_fatal_semaphore_wait_threshold	600	-	MariaDB's fatal semaphore timeout is configurable.
innodb_flush_method	O_DIRECT	fsync	MariaDB InnoDB flushing method by default on Unix systems bypasses the file system cache for improved performance in most cases.
innodb_flush_neighbors	1	0	MySQL 8 by default now assumes the use of an SSD device.
innodb_force_primary_key	OFF	-	If set to 1 in MariaDB (0 is default) CREATE TABLEs without a primary or unique key where all keyparts are NOT NULL will not be accepted, and will return an error.
innodb_idle_flush_pct	100	-	Up to what percentage of dirty pages in MariaDB should be flushed when InnoDB finds it has spare resources to do so.
innodb_immediate_scrub_data_uncompressed	OFF	-	MariaDB has support for data scrubbing.
innodb_instant_alter_column_allowed	add_drop_reorder	-	See Instant ADD COLUMN for InnoDB.
innodb_locks_unsafe_for_binlog	OFF	-	Deprecated option in MariaDB for disabling gap locking for searches and index scans. Deprecated in MariaDB, use READ COMMITTED transaction isolation instead.
innodb_log_checksums	-	ON	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6, as there is no reason to allow checksums to be disabled on the redo log.
innodb_log_compressed_pages	-	ON	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6 as part of the InnoDB redo log performance improvements.
innodb_log_files_in_group	-	2	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6 as part of the InnoDB redo log performance improvements.
innodb_log_spin_*	-	*	MySQL variables for constraining CPU usage while waiting for flushed redo.
innodb_log_wait_for_flush_spin_hwm	-	*	MySQL variable for constraining CPU usage while waiting for flushed redo.
innodb_max_dirty_pages_pct	75	90	MySQL 8 increased the default to 90.
innodb_max_dirty_pages_pct_lwm	0	10	MySQL 8 increased the default to 10.
innodb_max_undo_log_size	10485760	1073741824	MariaDB 10.2 reduced the limit for when an undo tablespace is marked for truncation.

innodb_open_files	Autosized (2000)	Autosized (4000)	In most systems, autosized based on the table_open_cache setting, which differs between MariaDB and MySQL.
innodb_page_cleaners	-	1	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6 as the original reasons for for splitting the buffer pool have mostly gone away.
innodb_prefix_index_cluster_optimization	OFF	-	MariaDB includes the Facebook prefix index queries optimization.
innodb_print_ddl_logs	-	OFF	MySQL option for writing DDL logs to stderr.
innodb_read_only_compressed	ON	-	Whether to set ROW_FORMAT=COMPRESSED tables to read-only.
innodb_redo_log_encrypt	-	OFF	MySQL 8 has also now introduced redo log encryption, but used a different name. The equivalent option in MariaDB is innodb_encrypt_log.
innodb_replication_delay	-	0	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_spin_wait_delay	4	6	MariaDB changed the default from 6 to 4 based on extensive benchmarking.
innodb_stats_modified_counter	0	-	MariaDB option to control the calculation of new statistics.
innodb_stats_sample_pages	8	-	Deprecated MariaDB option for control over index distribution statistics.
innodb_stats_traditional	ON	-	Enabling gives a larger sample of pages for larger tables for the purposes of index statistics calculation.
innodb_sync_array_size	-	1	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_thread_concurrency	-	0	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_thread_sleep_delay	-	10000	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_undo_log_encrypt	-	OFF	MySQL option for encrypting undo logs residing in separate undo tablespaces.
innodb_undo_log_truncate	OFF	ON	MySQL 8 changes the default to ON, marking larger undo logs for truncation.
innodb_undo_tablespaces	3	2	Number of tablespace files used for dividing up the undo logs. MySQL 8 has deprecated this setting, and increased the default (and minimum) to 2.
innodb_use_atomic_writes	ON	-	Atomic writes are a faster alternative to innodb_doublewrite and MariaDB automatically detects when supporting SSD cards are used.
internal_tmp_disk_storage_engine	-	INNODB	MySQL uses this variable to set the storage engine for on-disk internal temporary tables.
internal_tmp_mem_storage_engine	-	TEMPTABLE	MySQL and MariaDB use different formats for temporary tables. In MariaDB, the aria_used_for_temp_tables performs a similar function.
join_buffer_space_limit	2097152	-	Maximum size in bytes of the query buffer. See block-based join algorithms.
join_cache_level	2	-	For determining the join algorithms. See block-based join algorithms
key_buffer_size	134217728	8388608	Size of the buffer for the index blocks used by MyISAM tables and shared for all threads.
key_cache_file_hash_size	512	-	Number of hash buckets for open and changed files.
key_cache_segments	0	-	The number of segments in a key cache. See Segmented Key Cache.
keyring_operations	-	ON	Whether MySQL 8's keyring operations are enabled.
last_gtid	-	empty	MariaDB and MySQL have different GTID implementations.
local_infile	ON	OFF	MySQL no longer supports LOAD DATA LOCAL by default.
lock_wait_timeout	86400	31536000	MariaDB has reduced the timeout for acquiring metadata locks.
log_bin	OFF	ON	MySQL 8 enables the binary log by default.

log_bin_compress	OFF	-	MariaDB setting for whether or not the binary log can be compressed.
log_bin_compress_min_len	256	-	Minimum length of sql statement (in statement mode) or record (in row mode) that can be compressed. See Compressing Events to Reduce Size of the Binary Log.
log_bin_use_v1_row_events	-	OFF	MySQL-only variable showing whether or not MySQL's version 2 binary logging format is being used.
log_disabled_statements	sp	-	Disable logging of certain statements to the general log.
log_error_services	-	log_filter_internal; log_sink_internal	Components to enable for MySQL error logging.
log_error_verbosity	-	3	MySQL variable for setting verbosity of error, warning, and note messages in the error log.
log_slave_updates	OFF	ON	MySQL 8 has by default enabled binary logging of updates a slave receives from a master.
log_slow_admin_statements	ON	OFF	MariaDB logs slow admin statements to the slow query log by default.
log_slow_disabled_statements	admin,call,slave,sp	-	Disable logging of certain statements to the slow query log.
log_slow_filter	admin, filesort, filesort_on_disk, full_join, full_scan, query_cache, query_cache_miss, tmp_table, tmp_table_on_disk	-	For slow query log filtering.
log_slow_rate_limit	1	-	Limits the number of queries logged to the slow query log.
log_slow_min_examined_row_limit	0	-	Previously named min_examined_row_limit (still an alias).
log_slow_query	0	-	Previously named log_slow_query (still an alias).
log_slow_query_file	host_name-slow.log	-	Previously named slow_query_log_file (still an alias).
log_slow_query_time	10.000000	-	Previously named long_query_time (still an alias).
log_slow_slave_statements	ON	OFF	MariaDB logs slow slave statements to the slow query log by default.
log_slow_verbosity	empty	-	Controls information to be added to the slow query log. See also Slow Query Log Extended Statistics.
log_statements_unsafe_for_binlog	-	ON	MySQL setting for controlling whether binlog warnings are written to the error log.
log_syslog*	platform-dependent	-	MySQL variables with settings for writing to syslog.
log_tc_size	24576	-	Size in bytes of the transaction coordinator log, defined in multiples of 4096.
log_throttle_queries_not_using_indexes	-	0	MySQL-only variable for limiting the number of statements without indexes written to the slow query log.
log_timestamps	-	UTC	MySQL-only variable controlling the timezone for certain logging conditions.
log_warnings	2	-	MySQL 8 has replaced with `log_error_verbosity`.
mandatory_roles	-	(empty)	MySQL variable for assigning roles to all users.
master_info_repository	-	TABLE	Whether slave logs master status and connection info to a table or a file.
max_allowed_packet	16M	64M	
max_error_count	64	1024	Specifies the maximum number of messages stored for display by SHOW ERRORS and SHOW WARNINGS statements.
max_execution_time	-	0	MySQL renamed the `max_statement_time` variable.

max_length_for_sort_data	64	1024	Used to decide which algorithm to choose when sorting rows. If the total size of the column data, not including columns that are part of the sort, is less than max_length_for_sort_data, then these are added to the sort key. This can speed up the sort as there's no need to re-read the same row again later. Setting the value too high can slow things down as there will be a higher disk activity for doing the sort.
max_long_data_size	16777216	-	Maximum size for parameter values sent with mysql_stmt_send_long_data(). Removed in MySQL 5.6.
max_password_errors	4294967295	-	Maximum number of failed connections attempts before no more are permitted.
max_points_in_geometry	-	65536	Maximum points_per_circle for MySQL's ST_Buffer_Strategy() function.
max_recursive_iterations	4294967295	-	Maximum number of iterations when executing recursive queries.
max_relay_log_size	1073741824	0	Can be set by session in MariaDB.
max_seeks_for_key	4294967295	4294967295 (32-bit) or 18446744073709547520 (64-bit)	The most key seeks required when searching with an index, regardless of the actual index cardinality. If this value is set lower than its default and maximum, indexes will tend to be preferred over table scans.
max_session_mem_used	9223372036854775807	-	Amount of memory a single user session is allowed to allocate.
max_statement_time	0	-	Maximum time in seconds that a query can execute before being aborted. MySQL used to have a variable of this name before renaming it `max_execution_time`.
max_tmp_tables	32	-	Unused variable removed in MySQL.
max_write_lock_count	4294967295	4294967295 (32-bit) or 18446744073709547520 (64-bit)	Read lock requests will be permitted for processing after this many write locks.
mrr_buffer_size	262144	-	Size of buffer to use when using multi-range read with range access. See Multi Range Read optimization.
myisam_block_size	1024	-	Block size used for MyISAM index pages.
myisam_recover_options	BACKUP,QUICK	OFF	MyISAM recovery mode.
myisam_sort_buffer_size	134216704	8388608	Size in bytes of the buffer allocated when creating or sorting indexes on a MyISAM table.
mysql_native_password_proxy_users	-	OFF	Whether MySQL's authentication plugin supports proxy users. I
mysql56_temporal_format	ON		Causes MariaDB to use the MySQL-5.6 low level formats for TIME, DATETIME and TIMESTAMP instead of the MariaDB 5.3+ version.
new	-	OFF	Used for backward-compatibility with MySQL 4.1, not present in MariaDB.
mysqlx+*	-	*	MySQL's X plugin related variables.
ngram_token_size	-	2	Sets the n-gram token size for MySQL's n-gram full-text parser.
offline_mode	-	OFF	MySQL settting for specifying whether the server should run in offline mode.
old_alter_table	DEFAULT	OFF	An alias for alter_algorithm.
old_mode	Empty string	-	Used for getting MariaDB to emulate behavior from an old version of MySQL or MariaDB. See OLD Mode.
old_passwords	OFF	-	MySQL 8 is no longer compatible with the old pre-MySQL 4.1 form of password hashing.
optimizer_extra_pruning_depth	8	-	If the optimizer needs to enumerate a join prefix of this size or larger, then it will try aggressively prune away the search space.
optimizer_selectivity_sampling_limit	100	-	Controls number of record samples to check condition selectivity.
optimizer_switch	See details		A series of flags for controlling the query optimizer. MariaDB has introduced a number of new settings.
optimizer_trace_*	-	*	MySQL has more settings for optimizer tracing.

optimizer_use_condition_selectivity	4	-	Controls which statistics can be used by the optimizer when looking for the best query execution plan.
original_commit_timestamp	-	*	Used by MySQL 8 for delaying replication.
parser_max_mem_size	-	4294967295 (32-bit) or 18446744073709547520 (64-bit)	MySQL variable for limiting memory available to the parser.
password_*	-	*	Controls reuse of previous passwords in MySQL.
performance_schema	OFF	ON	The Performance Schema is off by default in MariaDB.
performance_schema_*			Many performance schema variables are autoset in MySQL, and MySQL has a different version, with additional variables.
plugin_maturity	One less than the server maturity	-	Minimum acceptable plugin maturity.
progress_report_time	5	-	Time in seconds between sending progress reports to the client for time-consuming statements.
proxy_protocol_networks	(empty)	-	Enable proxy protocol for these source networks.
query_alloc_block_size	16384	8192	Size in bytes of the extra blocks allocated during query parsing and execution (after query_prealloc_size is used up).
query_cache_*	*	-	MySQL has removed the query cache.
query_prealloc_size	24576	8192	Size in bytes of the persistent buffer for query parsing and execution, allocated on connect and freed on disconnect.
range_optimizer_max_mem_size	-	8388608	MySQL-only variable setting a limit on the range optimizer's memory usage.
rbr_exec_mode	-	STRICT	MySQL-only variable for determining the handling of certain key errors.
read_binlog_speed_limit	0	-	Permits restricting the speed at which the slave reads the binlog from the master.
regexp_*	-	*	Memory and time limits for regular expression matching operations.
relay_log_info_repository	-	TABLE	MySQL-only variable determining whether the slave's position in the relay logs is written to a file or table.
replicate_annotate_row_events	ON	-	Tells the slave to reproduce annotate_rows_events received from the master in its own binary log.
replicate_do_db	empty string	-	See Dynamic Replication Variables.
replicate_do_table	empty string	-	See Dynamic Replication Variables.
replicate_events_marked_for_skip	replicate	-	See Selectively skipping replication of binlog events.
replicate_ignore_db	empty string	-	See Dynamic Replication Variables.
replicate_ignore_table	empty string	-	See Dynamic Replication Variables.
replicate_rewrite_db	empty string	-	Only available as an option in MySQL.
replicate_wild_do_table	empty string	-	See Dynamic Replication Variables.
replicate_wild_ignore_table	empty string	-	See Dynamic Replication Variables.
result_metadata	-	FULL	Determine whether the server returns result set metadata for connections where this is optional.
rowid_merge_buff_size	8388608	-	See Non-semi-join subquery optimizations.
rpl_read_size	-	8192	Minimum data in bytes read from the binary and relay log files.
rpl_semi_sync_*	-	-	MariaDB includes semisynchronous replication without the need to install a plugin.
rpl_stop_slave_timeout	-	31536000	Controls the time that STOP SLAVE waits before timing out.
s3_*	*	-	The S3 storage engine is only available in MariaDB.
schema_definition_cache	-	256	Limits the number of schema definition objects kept in the dictionary object cache.
secure_auth	ON	-	Removed in MySQL.

secure_timestamp	NO	-	MariaDB-only option permitting the restricting of direct setting of a session timestamp..
server_id_bits	-	server_id	MySQL-only variable for use in MySQL Cluster.
server_uuid	-	UUID	MySQL-only variable containing the UUID.
session_track_gtids	-	OFF	MySQL-only variables for tracking gtid changes. MariaDB and MySQL's gtid implementation is different.
sha256_password_proxy_users	-	OFF	MySQL-only variable determining whether the sha256_password plugin supports proxy users.
show_create_table_verbosity	-	OFF	Option to cause SHOW CREATE TABLE to display ROW_FORMAT in all cases.
show_old_temporals	-	OFF	MySQL-only variable for determining whether SHOW CREATE TABLE output should include comments for old format temporal columns.
skip_parallel_replication	OFF	-	See parallel replication.
skip_replication	OFF	-	See Selectively skipping replication of binlog events.
slave_allow_batching	-	OFF	MySQL-only replication variable.
slave_checkpoint_group	-	512	MySQL-only replication variable.
slave_checkpoint_period	-	300	MySQL-only replication variable.
slave_ddl_exec_mode	IDEMPOTENT	-	Modes for how replication of DDL events should be executed.
slave_domain_parallel_threads	0	-	For configuring parallel replication.
slave_net_timeout	3600	60	MySQL reduced the timeout to 60s.
slave_parallel_max_queued	131072	-	For configuring parallel replication.
skip_grant_tables	0	-	Only available as an option in MySQL.
slave_max_statement_time	0.000000	-	MariaDB setting to abort a query that has taken more than this in seconds to run on the replica.
slave_parallel_mode	optimistic	-	Controls what transactions are applied in parallel when using parallel_replication.
slave_parallel_threads	0	-	For configuring parallel replication.
slave_parallel_type	-	DATABASE	MySQL-only replication variable.
slave_pending_jobs_size_max	-	16777216	MySQL-only replication variable.
slave_preserve_commit_order	-	OFF	MySQL-only replication variable.
slave_rows_search_algorithms	-	INDEX_SCAN, HASH_SCAN	MySQL-only replication variable.
slave_run_triggers_for_rbr	NO		See Running triggers on the slave for Row-based events for a description and use-case for this setting.
slave_transaction_retry_errors	1213,1205	-	When an error occurs during a transaction on the slave, replication usually halts. By default, transactions that caused a deadlock or elapsed lock wait timeout will be retried. One can add other errors to the the list of errors that should be retried by adding a comma-separated list of error numbers to this variable.
slave_transaction_retry_interval	0	-	Interval in seconds for the slave SQL thread to retry a failed transaction due to a deadlock, elapsed lock wait timeout or an error listed in slave_transaction_retry_errors.
sort_buffer_size	2097152	262144	The default sort buffer allocated has been reduced in MySQL.
sql_if_exists	OFF	-	Adds an implicit IF EXISTS to ALTER, RENAME and DROP of TABLES, VIEWS, FUNCTIONS and PACKAGES
sql_mode	STRICT_TRANS_TABLES, ERROR_FOR_DIVISION_BY_ZERO, NO_AUTO_CREATE_USER, NO_ENGINE_SUBSTITUTION	ONLY_FULL_GROUP_BY, STRICT_TRANS_TABLES, NO_ZERO_IN_DATE, NO_ZERO_DATE, ERROR_FOR_DIVISION_BY_ZERO, NO_ENGINE_SUBSTITUTION	See SQL Mode.
ssl_fips_mode	-	OFF	Whether FIPS mode is enabled on the server side.
standard_compliant_cte	ON	-	See Common Table Expressions.
storage_engine	InnoDB	-	Alias for default_storage_engine, removed in MySQL.

strict_password_validation	ON	-	In MariaDB, when password validation plugins are enabled, reject passwords that cannot be validated (passwords specified as a hash).
stored_program_definition_cache	-	256	Limits the number of stored program definition objects kept in the dictionary object cache.
super_read_only	-	OFF	MySQL variable for prohibiting client updates from users with the SUPER privilege.
sync_binlog	0	1	MySQL synchronizes all actions to the binary log before they are committed.
sync_frm	1	-	.frm files have been removed in MySQL.
system_versioning_alter_history	ERROR	-	MariaDB has System-Versioned Tables
system_versioning_asof	DEFAULT	-	MariaDB has System-Versioned Tables
system_versioning_insert_history	OFF	-	MariaDB has System-Versioned Tables
table_definition_cache	400	-1 (autosized)	Number of table definitions that can be cached.
table_open_cache_instances	8	16	Maximum number of table cache instances.
tablespace_definition_cache	-	256	Limits the number of tablespace definition objects kept in the dictionary object cache.
tcp_keepalive_interval	0	-	Interval, in seconds, between when successive keep-alive packets are sent if no acknowledgement is received.
tcp_keepalive_probes	0	-	Number of unacknowledged probes to send before considering the connection dead and notifying the application layer.
tcp_keepalive_time	0	-	Set the TCP_NODELAY option (disable Nagle's algorithm) on socket.
tcp_nodelay	1	-	Timeout, in milliseconds, with no activity until the first TCP keep-alive packet is sent.
temptable_max_ram	-	1GB	Limits the RAM used by MySQL's TempTable storage engine.
thread_cache_size	Autosized	-1 (autosized)	MariaDB uses an improved thread pool.
thread_concurrency	10	-	Removed in MySQL 5.7.
thread_pool_dedicated_listener	0	-	Better precision for the data in the Information Schema THREADPOOL_QUEUES Table.
thread_pool_exact_stats	0	-	Better precision for the data in the Information Schema THREADPOOL_QUEUES Table.
thread_pool_idle_timeout	60	-	See Using the Thread Pool .
thread_pool_max_threads	65536	-	See Using the Thread Pool .
thread_pool_min_threads	1	-	Windows-only. See Using the Thread Pool.
thread_pool_oversubscribe	3	-	See Using the Thread Pool .
thread_pool_prio_kickup	auto	-	See Using the Thread Pool .
thread_pool_priority	auto	-	See Using the Thread Pool .
thread_pool_size	Number of processors	16*	See Using the Thread Pool . *Only available in MySQL with a commercial plugin.
thread_pool_stall_limit	500	6*	See Using the Thread Pool . *Only available in MySQL with a commercial plugin.
thread_stack	299008	Varies	See Using the Thread Pool .
time_format	%H:%i:%s	-	Removed in MySQL.
timed_mutexes	OFF	-	Removed in MySQL.
tmp_disk_table_size	18446744073709551615	-	Max size for data for an internal temporary on-disk MyISAM or Aria table.
tmp_memory_table_size	16777216	-	Alias for tmp_table_size.
transaction_allow_batching	-	OFF	Variable for enabling batching of statements within the same transaction in MySQL Cluster.
transaction_write_set_extraction	-	OFF	Unused MySQL-only variable.
tx_isolation	REPEATABLE-READ	-	A MariaDB-only variable, replaced with transaction_isolation.
tx_read_only	OFF	-	A MariaDB-only variable, replaced with transaction_read_only.

use_stat_tables	preferably_for_queries	-	Controls the use of engine-independent table statistics.
userstat	OFF	-	Whether to activate MariaDB's User Statistics implementation, not available in MySQL.
version_compile_zlib	-	*	Version of the zlib library compiled in.
version_malloc_library	*	-	Version of the used malloc library.
version_source_revision	Varies	-	Permits seeing exactly which version of the source was used for a build.
version_ssl_library	*	-	Version of the used TLS library.
windowing_high_use_precision	-	*	MySQL option allowing safety to be sacrificed for speed in window function calculations.
wsrep_*	*	-	Galera cluster is only available in MariaDB.
Variable	**MariaDB 11.1**	**MySQL 8.0**	**Notes**

2.1.14.1.12.2 System Variable Differences Between MariaDB 11.0 and MySQL 8.0

Contents

The following is a comparison of variables that either appear only in MariaDB 11.0 or MySQL 8.0, or have different default settings in MariaDB 11.0, and MySQL 8.0. The releases MariaDB 11.0.2 and MySQL 8.0.34, with only default plugins enabled, were used for the comparison. Note that MySQL 8 is an 'evergreen' release, so features may be added or removed in later releases.

For a more complete list of differences, see Incompatibilities and Feature Differences Between MariaDB 11.0 and MySQL 8.0 and Function Differences Between MariaDB 11.0 and MySQL 8.0

Comparison Table

Variable	MariaDB 11.0 Default	MySQL 8.0 Default	Notes
activate_all_roles_on_login	-	OFF	Determines whether to automatically activate roles on login.
admin*	-	*	MySQL admin connections.
allow_suspicious_udfs	0	-	Only available as an option in MySQL.
alter_algorithm	DEFAULT	-	MariaDB 10.3 introduced new ALTER TABLE ALGORITHM clauses to avoid slow copies in certain instances. This variable allows setting this if no `ALGORITHM` clause is specified.
analyze_sample_percentage	100.0000	-	Percentage of rows from the table ANALYZE TABLE will sample to collect table statistics.
aria_*	*	-	The Aria storage engine is only available in MariaDB.
authentication_policy	-	*,,	MySQL authentication policy.
auto_generate_certs	-	ON	Whether to automatically generate SSL key and certificate files.
avoid_temporal_upgrade	-	OFF	Determines whether ALTER TABLE implicitly upgrades temporal columns.
back_log	Autosized	Autosized	MariaDB and MySQL have different autosizing algorithms.
binlog_alter_two_phase	OFF	-	When set, split ALTER at binary logging into two statements: START ALTER and COMMIT/ROLLBACK ALTER.
binlog_annotate_row_events	ON	-	Introduced in MariaDB 5.3 for replicating between MariaDB 5.3 and MySQL/MariaDB 5.1.
binlog_commit_wait_count	0	-	For use in MariaDB's parallel replication.
binlog_commit_wait_usec	100000	-	For use in MariaDB's parallel replication.
binlog_encryption	-	OFF	MySQL name for encrypt_binlog.
binlog_error_action		ABORT_SERVER	MySQL-only variable for controlling what happens when the server cannot write to the binary log.

binlog_expire_logs_auto_purge	-	ON	Enables or disables automatic purging of binary log files.
binlog_expire_logs_seconds	0	2592000	Sets the binary log expiration period in seconds
binlog_file_cache_size	16184	-	For setting the size of the file cache for the binary log.
binlog_format	MIXED	ROW	MariaDB and MySQL have differing binary log formats.
binlog_group_commit_sync_delay		0	MySQL-only variable for controlling the wait time before synchronizing the binary log file to disk.
binlog_group_commit_sync_no_delay_count		0	MySQL-only variable for setting the maximum number of transactions to wait for before aborting the current `binlog_group_commit_sync_delay` delay.
binlog_gtid_simple_recovery	-	ON	MySQL-only GTID variable. MariaDB's GTID implementation is different.
binlog_max_flush_queue_time	-	0	Specifies a timeout for reading transactions from the flush queue before continuing with group commit and syncing log to disk.
binlog_optimize_thread_scheduling	ON	-	For optimized kernel thread scheduling.
binlog_order_commits	-	ON	Determines whether transactions may be committed in parallel.
binlog_rotate_encryption_master_key_at_startup	-	OFF	Specifically for use with MySQL binary key encryption.
binlog_row_event_max_size	-	8192	Only available as a system variable in MariaDB 11.2.
binlog_row_metadata	NO_LOG	MINIMAL	Determines the amount of table metadata added to the binary log with row-based logging.
binlog_row_value_options	-	(empty)	Permits an alternative binlog format for JSON document updates.
binlog_rows_query_log_events	-	OFF	MySQL-only variable for logging extra information in row-based logging.
binlog_transaction_compression*	-	*	MySQL variables relating to binary log compression.
binlog_transaction_dependency_history_size	-	25000	Maximum number of row hashes kept for looking up transactions that last modified a given row.
binlog_transaction_dependency_tracking	-	COMMIT_ORDER	For determining how to best use the slave's multithreaded applier.
block_encryption_mode	-	aes-128-ecb	MySQL-only variable for controlling the block encryption mode for block-based algorithms.
caching_sha2_password*	-	*	For use with MySQL's SHA-256 authentication with caching.
character_set_*	latin1 or utf8	utf8mb4	MySQL 8.0 defaults to the utf8mb4 character set.
check_constraint_checks	ON	-	Permits disabling constraint checks, for example when loading a table that violates some constraints that you plan to fix later.
check_proxy_users		OFF	MySQL-only variable for controlling whether the server performs proxy user mapping for authentication plugins.
collation_*	latin1_swedish_ci or utf8_general_ci	utf8mb4_0900_ai_ci	MySQL 8.0 defaults to the utf8mb4 character set.
column_compression_threshold	100	-	MariaDB supports Storage-engine Independent Column Compression.
column_compression_zlib_level	6	-	MariaDB supports Storage-engine Independent Column Compression.
column_compression_zlib_strategy	DEFAULT_STRATEGY	-	MariaDB supports Storage-engine Independent Column Compression.
column_compression_zlib_wrap	OFF	-	MariaDB supports Storage-engine Independent Column Compression.
connection_memory_chunk_size	-	8192	Chunk size for updates to the Global_connection_memory counter.
connection_memory_limit	-	18446744073709551615	Maximum memory for a single user connection.
create_admin_listener_thread	-	OFF	MySQL-only variable for whether to use a dedicated listening thread for admin network interface connections.

cte_max_recursion_depth	-	1000	When MySQL 8.0 introduced common table expressions they used a different name. MariaDB's variable is called max_recursive_iterations.
date_format	%Y-%m-%d	-	Unused variable removed in MySQL 8.0
datetime_format	%Y-%m-%d	-	Unused variable removed in MySQL 8.0
deadlock_search_depth_long	15	-	The Aria storage engine is only available in MariaDB.
deadlock_search_depth_short	4	-	The Aria storage engine is only available in MariaDB.
deadlock_timeout_long	50000000	-	The Aria storage engine is only available in MariaDB.
deadlock_timeout_short	10000	-	The Aria storage engine is only available in MariaDB.
debug_no_thread_alarm	OFF	-	Disable system thread alarm calls, for debugging or testing.
default_authentication_plugin	-	caching_sha2_password	MySQL 8 introduced a new authentication plugin.
default_collation_for_utf8mb4	-	utf8mb4_0900_ai_ci	For internal use in MySQL 8 replication.
default_master_connection	empty	-	For use with MariaDB's multi-source replication.
default_password_lifetime	0	360	MariaDB defaults to password expiration off.
default_regex_flags	empty	-	For handling incompatibilities between MariaDB's PCRE and the old regex library.
default_tmp_storage_engine	empty	InnoDB	Default storage engine used for tables created with CREATE TEMPORARY TABLE.
disabled_storage_engines		empty	MySQL-only variable for disabling specific storage engines.
disconnect_on_expired_password	OFF	ON	MariaDB password expiration is off by default, and by default does not disconnect a client when a password has expired.
encrypt_binlog	OFF	-	MariaDB enables table and tablespace encryption.
encrypt_tmp_files	OFF	-	MariaDB enables table and tablespace encryption.
encrypt_tmp_disk_tables	OFF	-	MariaDB enables table and tablespace encryption.
end_markers_in_json	-	OFF	MySQL-only variable for adding end markers to JSON output.
enforce_gtid_consistency	-	OFF	MariaDB and MySQL have different GTID implementations.
enforce_storage_engine	none		Forces the use of a particular storage engine for new tables.
eq_range_index_dive_limit	0	200	Variable for tuning when the optimizer should switch from using index dives to index statistics for qualifying rows estimation.
event_scheduler	OFF	ON	MySQL enables the event scheduler by default.
expensive_subquery_limit	100	-	Used for determining expensive queries for optimization.
extra_max_connections	1	-	Introduced in the MariaDB 5.1 threadpool.
extra_port	0	-	Introduced in the MariaDB 5.1 threadpool.
group_concat_max_len	1048576	1024	MariaDB increases the maximum length for a GROUP_CONCAT() result from 1K to 1M.
gtid_binlog_pos	empty	-	MariaDB and MySQL have different GTID implementations.
gtid_binlog_state	empty	-	MariaDB and MySQL have different GTID implementations.
gtid_cleanup_batch_size	64	-	MariaDB and MySQL have different GTID implementations.
gtid_current_pos	empty	-	MariaDB and MySQL have different GTID implementations.
gtid_domain_id	0	-	MariaDB and MySQL have different GTID implementations.

gtid_executed	-	empty	MariaDB and MySQL have different GTID implementations.
gtid_executed_compression_period	-	1000	MariaDB and MySQL have different GTID implementations.
gtid_ignore_duplicates	OFF	-	MariaDB and MySQL have different GTID implementations.
gtid_mode	-	OFF	MariaDB and MySQL have different GTID implementations.
gtid_next	-	AUTOMATIC	MariaDB and MySQL have different GTID implementations.
gtid_owned	-	empty	MariaDB and MySQL have different GTID implementations.
gtid_pos_auto_engines	empty	-	MariaDB and MySQL have different GTID implementations.
gtid_purged	-	empty	MariaDB and MySQL have different GTID implementations.
gtid_seq_no	0	-	MariaDB and MySQL have different GTID implementations.
gtid_slave_pos	empty	-	MariaDB and MySQL have different GTID implementations.
gtid_strict_mode	OFF	-	MariaDB and MySQL have different GTID implementations.
have_crypt	YES	-	MySQL has removed the ENCRYPT function.
have_openssl			MariaDB's version indicates whether YaSSL or openssl was used. MySQL's is a synonym for `have_ssl`.
have_query_cache	YES	-	MySQL has removed the query cache.
have_statement_timeout	-		Whether MySQL's statement execution timeout feature is available.
have_symlink	YES	DISABLED	MySQL has removed symlink support.
histogram_generation_max_mem_size	-	20000000	Added when MySQL 8 introduced Histogram-based Statistics. MariaDB uses histogram_size
histogram_size	0	-	MariaDB introduced Histogram-based Statistics.
histogram_type	JSON_HB	-	MariaDB introduced Histogram-based Statistics.
idle_readonly_transaction_timeout	0	-	Time in seconds that the server waits for idle read-only transactions.
idle_transaction_timeout	0	-	Time in seconds that the server waits for idle transactions.
idle_write_transaction_timeout	0	-	Time in seconds that the server waits for idle write transactions.
ignore_builtin_innodb	OFF	-	Ignored and removed in MySQL 8.
in_predicate_conversion_threshold	1000	-	Controls the Conversion of Big IN Predicates Into Subqueries optimization.
in_transaction	0	-	Set to `1` if you are in a transaction, and `0` if not.
information_schema_stats_expiry	-	86400	Time until MySQL Information Schema cached statistics expire.
innodb_adaptive_flushing_lwm	10.000000	10	Adaptive flushing is enabled when this this low water mark percentage of the redo log capacity is reached. MariaDB's variable is a double, MySQL's an integer.
innodb_adaptive_hash_index	OFF	ON	Defaulting to OFF is a performance improvement especially for DROP TABLE, TRUNCATE TABLE, ALTER TABLE, or DROP INDEX operations
innodb_adaptive_max_sleep_delay	-	150000	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_api_*	-	*	Specific to MySQL's memcached, removed in MariaDB 10.2.
innodb_autoinc_lock_mode	1	2	MariaDB has an extra mode, `3`, for skipping the rollback of connected transactions. MySQL defaults to row-based replication, so can safely use `2`.
innodb_buffer_pool_instances	-	1	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6 since the original reasons for introducing no longer apply.
innodb_change*		*	The InnoDB Change Buffer was removed in MariaDB 11.0.

innodb_checksum_algorithm	full_crc32	crc32	`fullcrc32` permits encryption to be supported over a SPATIAL INDEX, which crc32 does not support.
innodb_checksums	ON	-	Deprecated option removed in MySQL.
innodb_commit_concurrency	-	0	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_compression_*	*	-	Introduced with MariaDB's InnoDB compression.
innodb_concurrency_tickets	-	5000	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_deadlock_report	Full	-	How to report deadlocks.
innodb_dedicated_server	-	OFF	MySQL option that automatically configures various settings if the server is a dedicated InnoDB database server.
innodb_default_encryption_key_id	1	-	Default encryption key id used for table encryption. See Data at Rest Encryption.
innodb_defragment	*	-	MariaDB can defragment InnoDB tablespaces.
innodb_directories	-	(empty)	Used to search for tablespace files when moving or restoring a new location.
innodb_disallow_writes	OFF	-	Tell InnoDB to stop any writes to disk.
innodb_encrypt_*	1	-	See MariaDB's Data at Rest Encryption.
innodb_fatal_semaphore_wait_threshold	600	-	MariaDB's fatal semaphore timeout is configurable.
innodb_flush_method	O_DIRECT	fsync	MariaDB InnoDB flushing method by default on Unix systems bypasses the file system cache for improved performance in most cases.
innodb_flush_neighbors	1	0	MySQL 8 by default now assumes the use of an SSD device.
innodb_force_primary_key	OFF	-	If set to 1 in MariaDB (0 is default) CREATE TABLEs without a primary or unique key where all keyparts are NOT NULL will not be accepted, and will return an error.
innodb_idle_flush_pct	100	-	Up to what percentage of dirty pages in MariaDB should be flushed when InnoDB finds it has spare resources to do so.
innodb_immediate_scrub_data_uncompressed	OFF	-	MariaDB has support for data scrubbing.
innodb_instant_alter_column_allowed	add_drop_reorder	-	See Instant ADD COLUMN for InnoDB.
innodb_locks_unsafe_for_binlog	OFF	-	Deprecated option in MariaDB for disabling gap locking for searches and index scans. Deprecated in MariaDB, use READ COMMITTED transaction isolation instead.
innodb_log_checksums	-	ON	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6, as there is no reason to allow checksums to be disabled on the redo log.
innodb_log_compressed_pages	-	ON	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6 as part of the InnoDB redo log performance improvements.
innodb_log_files_in_group	-	2	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6 as part of the InnoDB redo log performance improvements.
innodb_log_spin_*	-	*	MySQL variables for constraining CPU usage while waiting for flushed redo.
innodb_log_wait_for_flush_spin_hwm	-	*	MySQL variable for constraining CPU usage while waiting for flushed redo.
innodb_max_dirty_pages_pct	75	90	MySQL 8 increased the default to 90.
innodb_max_dirty_pages_pct_lwm	0	10	MySQL 8 increased the default to 10.
innodb_max_undo_log_size	10485760	1073741824	MariaDB 10.2 reduced the limit for when an undo tablespace is marked for truncation.
innodb_open_files	Autosized (2000)	Autosized (4000)	In most systems, autosized based on the table_open_cache setting, which differs between MariaDB and MySQL.
innodb_page_cleaners	-	1	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6 as the original reasons for for splitting the buffer pool have mostly gone away.

innodb_prefix_index_cluster_optimization	OFF	-	MariaDB includes the Facebook prefix index queries optimization.
innodb_print_ddl_logs	-	OFF	MySQL option for writing DDL logs to stderr.
innodb_read_only_compressed	ON	-	Whether to set ROW_FORMAT=COMPRESSED tables to read-only.
innodb_redo_log_encrypt	-	OFF	MySQL 8 has also now introduced redo log encryption, but used a different name. The equivalent option in MariaDB is innodb_encrypt_log.
innodb_replication_delay	-	0	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_spin_wait_delay	4	6	MariaDB changed the default from 6 to 4 based on extensive benchmarking.
innodb_stats_modified_counter	0	-	MariaDB option to control the calculation of new statistics.
innodb_stats_sample_pages	8	-	Deprecated MariaDB option for control over index distribution statistics.
innodb_stats_traditional	ON	-	Enabling gives a larger sample of pages for larger tables for the purposes of index statistics calculation.
innodb_sync_array_size	-	1	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_thread_concurrency	-	0	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_thread_sleep_delay	-	10000	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_undo_log_encrypt	-	OFF	MySQL option for encrypting undo logs residing in separate undo tablespaces.
innodb_undo_log_truncate	OFF	ON	MySQL 8 changes the default to ON, marking larger undo logs for truncation.
innodb_undo_tablespaces	3	2	Number of tablespace files used for dividing up the undo logs. MySQL 8 has deprecated this setting, and increased the default (and minimum) to 2.
innodb_use_atomic_writes	ON	-	Atomic writes are a faster alternative to innodb_doublewrite and MariaDB automatically detects when supporting SSD cards are used.
internal_tmp_disk_storage_engine	-	INNODB	MySQL uses this variable to set the storage engine for on-disk internal temporary tables.
internal_tmp_mem_storage_engine	-	TEMPTABLE	MySQL and MariaDB use different formats for temporary tables. In MariaDB, the aria_used_for_temp_tables performs a similar function.
join_buffer_space_limit	2097152	-	Maximum size in bytes of the query buffer. See block-based join algorithms.
join_cache_level	2	-	For determining the join algorithms. See block-based join algorithms
key_buffer_size	134217728	8388608	Size of the buffer for the index blocks used by MyISAM tables and shared for all threads.
key_cache_file_hash_size	512	-	Number of hash buckets for open and changed files.
key_cache_segments	0	-	The number of segments in a key cache. See Segmented Key Cache.
keyring_operations	-	ON	Whether MySQL 8's keyring operations are enabled.
last_gtid	-	empty	MariaDB and MySQL have different GTID implementations.
local_infile	ON	OFF	MySQL no longer supports LOAD DATA LOCAL by default.
lock_wait_timeout	86400	31536000	MariaDB has reduced the timeout for acquiring metadata locks.
log_bin	OFF	ON	MySQL 8 enables the binary log by default.
log_bin_compress	OFF	-	MariaDB setting for whether or not the binary log can be compressed.
log_bin_compress_min_len	256	-	Minimum length of sql statement (in statement mode) or record (in row mode) that can be compressed. See Compressing Events to Reduce Size of the Binary Log.

log_bin_use_v1_row_events	-	OFF	MySQL-only variable showing whether or not MySQL's version 2 binary logging format is being used.
log_disabled_statements	sp	-	Disable logging of certain statements to the general log.
log_error_services	-	log_filter_internal; log_sink_internal	Components to enable for MySQL error logging.
log_error_verbosity	-	3	MySQL variable for setting verbosity of error, warning, and note messages in the error log.
log_slave_updates	OFF	ON	MySQL 8 has by default enabled binary logging of updates a slave receives from a master.
log_slow_admin_statements	ON	OFF	MariaDB logs slow admin statements to the slow query log by default.
log_slow_disabled_statements	admin,call,slave,sp	-	Disable logging of certain statements to the slow query log.
log_slow_filter	admin, filesort, filesort_on_disk, full_join, full_scan, query_cache, query_cache_miss, tmp_table, tmp_table_on_disk	-	For slow query log filtering.
log_slow_rate_limit	1	-	Limits the number of queries logged to the slow query log.
log_slow_min_examined_row_limit	0	-	Previously named min_examined_row_limit (still an alias).
log_slow_query	0	-	Previously named log_slow_query (still an alias).
log_slow_query_file	host_name-slow.log	-	Previously named slow_query_log_file (still an alias).
log_slow_query_time	10.000000	-	Previously named long_query_time (still an alias).
log_slow_slave_statements	ON	OFF	MariaDB logs slow slave statements to the slow query log by default.
log_slow_verbosity	empty	-	Controls information to be added to the slow query log. See also Slow Query Log Extended Statistics.
log_statements_unsafe_for_binlog	-	ON	MySQL setting for controlling whether binlog warnings are written to the error log.
log_syslog*	platform-dependent	-	MySQL variables with settings for writing to syslog.
log_tc_size	24576	-	Size in bytes of the transaction coordinator log, defined in multiples of 4096.
log_throttle_queries_not_using_indexes	-	0	MySQL-only variable for limiting the number of statements without indexes written to the slow query log.
log_timestamps	-	UTC	MySQL-only variable controlling the timezone for certain logging conditions.
log_warnings	2	-	MySQL 8 has replaced with `log_error_verbosity`.
mandatory_roles	-	(empty)	MySQL variable for assigning roles to all users.
master_info_repository	-	TABLE	Whether slave logs master status and connection info to a table or a file.
max_allowed_packet	16M	64M	
max_error_count	64	1024	Specifies the maximum number of messages stored for display by SHOW ERRORS and SHOW WARNINGS statements.
max_execution_time	-	0	MySQL renamed the `max_statement_time` variable.
max_length_for_sort_data	64	1024	Used to decide which algorithm to choose when sorting rows. If the total size of the column data, not including columns that are part of the sort, is less than max_length_for_sort_data, then these are added to the sort key. This can speed up the sort as there's no need to re-read the same row again later. Setting the value too high can slow things down as there will be a higher disk activity for doing the sort.
max_long_data_size	16777216	-	Maximum size for parameter values sent with mysql_stmt_send_long_data(). Removed in MySQL 5.6.

max_password_errors	4294967295	-	Maximum number of failed connections attempts before no more are permitted.
max_points_in_geometry	-	65536	Maximum points_per_circle for MySQL's ST_Buffer_Strategy() function.
max_recursive_iterations	4294967295	-	Maximum number of iterations when executing recursive queries.
max_relay_log_size	1073741824	0	Can be set by session in MariaDB.
max_seeks_for_key	4294967295	4294967295 (32-bit) or 18446744073709547520 (64-bit)	The most key seeks required when searching with an index, regardless of the actual index cardinality. If this value is set lower than its default and maximum, indexes will tend to be preferred over table scans.
max_session_mem_used	9223372036854775807	-	Amount of memory a single user session is allowed to allocate.
max_statement_time	0	-	Maximum time in seconds that a query can execute before being aborted. MySQL used to have a variable of this name before renaming it `max_execution_time`.
max_tmp_tables	32	-	Unused variable removed in MySQL.
max_write_lock_count	4294967295	4294967295 (32-bit) or 18446744073709547520 (64-bit)	Read lock requests will be permitted for processing after this many write locks.
mrr_buffer_size	262144	-	Size of buffer to use when using multi-range read with range access. See Multi Range Read optimization.
myisam_block_size	1024	-	Block size used for MyISAM index pages.
myisam_recover_options	BACKUP,QUICK	OFF	MyISAM recovery mode.
myisam_sort_buffer_size	134216704	8388608	Size in bytes of the buffer allocated when creating or sorting indexes on a MyISAM table.
mysql_native_password_proxy_users	-	OFF	Whether MySQL's authentication plugin supports proxy users. I
mysql56_temporal_format	ON		Causes MariaDB to use the MySQL-5.6 low level formats for TIME, DATETIME and TIMESTAMP instead of the MariaDB 5.3+ version.
new	-	OFF	Used for backward-compatibility with MySQL 4.1, not present in MariaDB.
mysqlx+*	-	*	MySQL's X plugin related variables.
ngram_token_size	-	2	Sets the n-gram token size for MySQL's n-gram full-text parser.
offline_mode	-	OFF	MySQL settting for specifying whether the server should run in offline mode.
old_alter_table	DEFAULT	OFF	An alias for alter_algorithm.
old_mode	Empty string	-	Used for getting MariaDB to emulate behavior from an old version of MySQL or MariaDB. See OLD Mode.
old_passwords	OFF	-	MySQL 8 is no longer compatible with the old pre-MySQL 4.1 form of password hashing.
optimizer_extra_pruning_depth	8	-	If the optimizer needs to enumerate a join prefix of this size or larger, then it will try aggressively prune away the search space.
optimizer_selectivity_sampling_limit	100	-	Controls number of record samples to check condition selectivity.
optimizer_switch	See details		A series of flags for controlling the query optimizer. MariaDB has introduced a number of new settings.
optimizer_trace_*	-	*	MySQL has more settings for optimizer tracing.
optimizer_use_condition_selectivity	4	-	Controls which statistics can be used by the optimizer when looking for the best query execution plan.
original_commit_timestamp	-	*	Used by MySQL 8 for delaying replication.
parser_max_mem_size	-	4294967295 (32-bit) or 18446744073709547520 (64-bit)	MySQL variable for limiting memory available to the parser.
password_*	-	*	Controls reuse of previous passwords in MySQL.
performance_schema	OFF	ON	The Performance Schema is off by default in MariaDB.

performance_schema_*			Many performance schema variables are autoset in MySQL, and MySQL has a different version, with additional variables.
plugin_maturity	One less than the server maturity	-	Minimum acceptable plugin maturity.
progress_report_time	5	-	Time in seconds between sending progress reports to the client for time-consuming statements.
proxy_protocol_networks	(empty)	-	Enable proxy protocol for these source networks.
query_alloc_block_size	16384	8192	Size in bytes of the extra blocks allocated during query parsing and execution (after query_prealloc_size is used up).
query_cache_*	*	-	MySQL has removed the query cache.
query_prealloc_size	24576	8192	Size in bytes of the persistent buffer for query parsing and execution, allocated on connect and freed on disconnect.
range_optimizer_max_mem_size	-	8388608	MySQL-only variable setting a limit on the range optimizer's memory usage.
rbr_exec_mode	-	STRICT	MySQL-only variable for determining the handling of certain key errors.
read_binlog_speed_limit	0	-	Permits restricting the speed at which the slave reads the binlog from the master.
regexp_*	-	*	Memory and time limits for regular expression matching operations.
relay_log_info_repository	-	TABLE	MySQL-only variable determining whether the slave's position in the relay logs is written to a file or table.
replicate_annotate_row_events	ON	-	Tells the slave to reproduce annotate_rows_events received from the master in its own binary log.
replicate_do_db	empty string	-	See Dynamic Replication Variables.
replicate_do_table	empty string	-	See Dynamic Replication Variables.
replicate_events_marked_for_skip	replicate	-	See Selectively skipping replication of binlog events.
replicate_ignore_db	empty string	-	See Dynamic Replication Variables.
replicate_ignore_table	empty string	-	See Dynamic Replication Variables.
replicate_rewrite_db	empty string	-	Only available as an option in MySQL.
replicate_wild_do_table	empty string	-	See Dynamic Replication Variables.
replicate_wild_ignore_table	empty string	-	See Dynamic Replication Variables.
result_metadata	-	FULL	Determine whether the server returns result set metadata for connections where this is optional.
rowid_merge_buff_size	8388608	-	See Non-semi-join subquery optimizations.
rpl_read_size	-	8192	Minimum data in bytes read from the binary and relay log files.
rpl_semi_sync_*	-	-	MariaDB includes semisynchronous replication without the need to install a plugin.
rpl_stop_slave_timeout	-	31536000	Controls the time that STOP SLAVE waits before timing out.
s3_*	*	-	The S3 storage engine is only available in MariaDB.
schema_definition_cache	-	256	Limits the number of schema definition objects kept in the dictionary object cache.
secure_auth	ON	-	Removed in MySQL.
secure_timestamp	NO	-	MariaDB-only option permitting the restricting of direct setting of a session timestamp..
server_id_bits	-	server_id	MySQL-only variable for use in MySQL Cluster.
server_uuid	-	UUID	MySQL-only variable containing the UUID.
session_track_gtids	-	OFF	MySQL-only variables for tracking gtid changes. MariaDB and MySQL's gtid implementation is different.
sha256_password_proxy_users	-	OFF	MySQL-only variable determining whether the sha256_password plugin supports proxy users.

show_create_table_verbosity	-	OFF	Option to cause SHOW CREATE TABLE to display ROW_FORMAT in all cases.
show_old_temporals	-	OFF	MySQL-only variable for determining whether SHOW CREATE TABLE output should include comments for old format temporal columns.
skip_parallel_replication	OFF	-	See parallel replication.
skip_replication	OFF	-	See Selectively skipping replication of binlog events.
slave_allow_batching	-	OFF	MySQL-only replication variable.
slave_checkpoint_group	-	512	MySQL-only replication variable.
slave_checkpoint_period	-	300	MySQL-only replication variable.
slave_ddl_exec_mode	IDEMPOTENT	-	Modes for how replication of DDL events should be executed.
slave_domain_parallel_threads	0	-	For configuring parallel replication.
slave_net_timeout	3600	60	MySQL reduced the timeout to 60s.
slave_parallel_max_queued	131072	-	For configuring parallel replication.
skip_grant_tables	0	-	Only available as an option in MySQL.
slave_max_statement_time	0.000000	-	MariaDB setting to abort a query that has taken more than this in seconds to run on the replica.
slave_parallel_mode	optimistic	-	Controls what transactions are applied in parallel when using parallel_replication.
slave_parallel_threads	0	-	For configuring parallel replication.
slave_parallel_type	-	DATABASE	MySQL-only replication variable.
slave_pending_jobs_size_max	-	16777216	MySQL-only replication variable.
slave_preserve_commit_order	-	OFF	MySQL-only replication variable.
slave_rows_search_algorithms	-	INDEX_SCAN, HASH_SCAN	MySQL-only replication variable.
slave_run_triggers_for_rbr	NO		See Running triggers on the slave for Row-based events for a description and use-case for this setting.
slave_transaction_retry_errors	1213,1205	-	When an error occurs during a transaction on the slave, replication usually halts. By default, transactions that caused a deadlock or elapsed lock wait timeout will be retried. One can add other errors to the the list of errors that should be retried by adding a comma-separated list of error numbers to this variable.
slave_transaction_retry_interval	0	-	Interval in seconds for the slave SQL thread to retry a failed transaction due to a deadlock, elapsed lock wait timeout or an error listed in slave_transaction_retry_errors.
sort_buffer_size	2097152	262144	The default sort buffer allocated has been reduced in MySQL.
sql_if_exists	OFF	-	Adds an implicit IF EXISTS to ALTER, RENAME and DROP of TABLES, VIEWS, FUNCTIONS and PACKAGES
sql_mode	STRICT_TRANS_TABLES, ERROR_FOR_DIVISION_BY_ZERO, NO_AUTO_CREATE_USER, NO_ENGINE_SUBSTITUTION	ONLY_FULL_GROUP_BY, STRICT_TRANS_TABLES, NO_ZERO_IN_DATE, NO_ZERO_DATE, ERROR_FOR_DIVISION_BY_ZERO, NO_ENGINE_SUBSTITUTION	See SQL Mode.
ssl_fips_mode	-	OFF	Whether FIPS mode is enabled on the server side.
standard_compliant_cte	ON	-	See Common Table Expressions.
storage_engine	InnoDB	-	Alias for default_storage_engine, removed in MySQL.
strict_password_validation	ON	-	In MariaDB, when password validation plugins are enabled, reject passwords that cannot be validated (passwords specified as a hash).
stored_program_definition_cache	-	256	Limits the number of stored program definition objects kept in the dictionary object cache.
super_read_only	-	OFF	MySQL variable for prohibiting client updates from users with the SUPER privilege.
sync_binlog	0	1	MySQL synchronizes all actions to the binary log before they are committed.
sync_frm	1	-	.frm files have been removed in MySQL.

system_versioning_alter_history	ERROR	-	MariaDB has System-Versioned Tables
system_versioning_asof	DEFAULT	-	MariaDB has System-Versioned Tables
system_versioning_insert_history	OFF	-	MariaDB has System-Versioned Tables
table_definition_cache	400	-1 (autosized)	Number of table definitions that can be cached.
table_open_cache_instances	8	16	Maximum number of table cache instances.
tablespace_definition_cache	-	256	Limits the number of tablespace definition objects kept in the dictionary object cache.
tcp_keepalive_interval	0	-	Interval, in seconds, between when successive keep-alive packets are sent if no acknowledgement is received.
tcp_keepalive_probes	0	-	Number of unacknowledged probes to send before considering the connection dead and notifying the application layer.
tcp_keepalive_time	0	-	Set the TCP_NODELAY option (disable Nagle's algorithm) on socket.
tcp_nodelay	1	-	Timeout, in milliseconds, with no activity until the first TCP keep-alive packet is sent.
temptable_max_ram	-	1GB	Limits the RAM used by MySQL's TempTable storage engine.
thread_cache_size	Autosized	-1 (autosized)	MariaDB uses an improved thread pool.
thread_concurrency	10	-	Removed in MySQL 5.7.
thread_pool_dedicated_listener	0	-	Better precision for the data in the Information Schema THREADPOOL_QUEUES Table.
thread_pool_exact_stats	0	-	Better precision for the data in the Information Schema THREADPOOL_QUEUES Table.
thread_pool_idle_timeout	60	-	See Using the Thread Pool.
thread_pool_max_threads	65536	-	See Using the Thread Pool.
thread_pool_min_threads	1	-	Windows-only. See Using the Thread Pool.
thread_pool_oversubscribe	3	-	See Using the Thread Pool.
thread_pool_prio_kickup	auto	-	See Using the Thread Pool.
thread_pool_priority	auto	-	See Using the Thread Pool.
thread_pool_size	Number of processors	16*	See Using the Thread Pool. *Only available in MySQL with a commercial plugin.
thread_pool_stall_limit	500	6*	See Using the Thread Pool. *Only available in MySQL with a commercial plugin.
thread_stack	299008	Varies	See Using the Thread Pool.
time_format	%H:%i:%s	-	Removed in MySQL.
timed_mutexes	OFF	-	Removed in MySQL.
tmp_disk_table_size	18446744073709551615	-	Max size for data for an internal temporary on-disk MyISAM or Aria table.
tmp_memory_table_size	16777216	-	Alias for tmp_table_size.
transaction_allow_batching	-	OFF	Variable for enabling batching of statements within the same transaction in MySQL Cluster.
transaction_isolation	-	REPEATABLE-READ	The MariaDB equivalent is tx_isolation.
transaction_read_only	-	OFF	The MariaDB equivalent is tx_read_only.
transaction_write_set_extraction	-	OFF	Unused MySQL-only variable.
tx_isolation	REPEATABLE-READ	-	The MySQL equivalent is transaction_isolation.
tx_read_only	OFF	-	The MySQL equivalent is transaction_read_only.
use_stat_tables	preferably_for_queries	-	Controls the use of engine-independent table statistics.
userstat	OFF	-	Whether to activate MariaDB's User Statistics implementation, not available in MySQL.
version_compile_zlib	-	*	Version of the zlib library compiled in.
version_malloc_library	*	-	Version of the used malloc library.
version_source_revision	Varies	-	Permits seeing exactly which version of the source was used for a build.
version_ssl_library	*	-	Version of the used TLS library.

windowing_high_use_precision	-	*	MySQL option allowing safety to be sacrificed for speed in window function calculations.
wsrep_*	*	-	Galera cluster is only available in MariaDB.
Variable	**MariaDB 11.01**	**MySQL 8.0**	**Notes**

2.1.14.1.12.3 System Variable Differences Between MariaDB 10.11 and MySQL 8.0

Contents

The following is a comparison of variables that either appear only in MariaDB 10.11 or MySQL 8.0, or have different default settings in MariaDB 10.11, and MySQL 8.0. The releases MariaDB 10.11.2 and MySQL 8.0.34, with only default plugins enabled, were used for the comparison. Note that MySQL 8 is an 'evergreen' release, so features may be added or removed in later releases.

For a more complete list of differences, see Incompatibilities and Feature Differences Between MariaDB 10.11 and MySQL 8.0 and Function Differences Between MariaDB 10.11 and MySQL 8.0

Comparison Table

Variable	MariaDB 10.11 Default	MySQL 8.0 Default	Notes
activate_all_roles_on_login	-	OFF	Determines whether to automatically activate roles on login.
admin*	-	*	MySQL admin connections.
allow_suspicious_udfs	0	-	Only available as an option in MySQL.
alter_algorithm	DEFAULT	-	MariaDB 10.3 introduced new ALTER TABLE ALGORITHM clauses to avoid slow copies in certain instances. This variable allows setting this if no `ALGORITHM` clause is specified.
analyze_sample_percentage	100.0000	-	Percentage of rows from the table ANALYZE TABLE will sample to collect table statistics.
aria_*	*	-	The Aria storage engine is only available in MariaDB.
authentication_policy	-	*,,	MySQL authentication policy.
auto_generate_certs	-	ON	Whether to automatically generate SSL key and certificate files.
avoid_temporal_upgrade	-	OFF	Determines whether ALTER TABLE implicitly upgrades temporal columns.
back_log	Autosized	Autosized	MariaDB and MySQL have different autosizing algorithms.
binlog_alter_two_phase	OFF	-	When set, split ALTER at binary logging into two statements: START ALTER and COMMIT/ROLLBACK ALTER.
binlog_annotate_row_events	ON	-	Introduced in MariaDB 5.3 for replicating between MariaDB 5.3 and MySQL/MariaDB 5.1.
binlog_commit_wait_count	0	-	For use in MariaDB's parallel replication.
binlog_commit_wait_usec	100000	-	For use in MariaDB's parallel replication.
binlog_encryption	-	OFF	MySQL name for encrypt_binlog.
binlog_error_action		ABORT_SERVER	MySQL-only variable for controlling what happens when the server cannot write to the binary log.
binlog_expire_logs_auto_purge	-	ON	Enables or disables automatic purging of binary log files.
binlog_expire_logs_seconds	0	2592000	Sets the binary log expiration period in seconds
binlog_file_cache_size	16184	-	For setting the size of the file cache for the binary log.
binlog_format	MIXED	ROW	MariaDB and MySQL have differing binary log formats.
binlog_group_commit_sync_delay		0	MySQL-only variable for controlling the wait time before synchronizing the binary log file to disk.

binlog_group_commit_sync_no_delay_count		0	MySQL-only variable for setting the maximum number of transactions to wait for before aborting the current `binlog_group_commit_sync_delay` delay.
binlog_gtid_simple_recovery	-	ON	MySQL-only GTID variable. MariaDB's GTID implementation is different.
binlog_max_flush_queue_time	-	0	Specifies a timeout for reading transactions from the flush queue before continuing with group commit and syncing log to disk.
binlog_optimize_thread_scheduling	ON	-	For optimized kernel thread scheduling.
binlog_order_commits	-	ON	Determines whether transactions may be committed in parallel.
binlog_rotate_encryption_master_key_at_startup	-	OFF	Specifically for use with MySQL binary key encryption.
binlog_row_event_max_size	-	8192	Only available as a system variable in MariaDB 11.2.
binlog_row_metadata	NO_LOG	MINIMAL	Determines the amount of table metadata added to the binary log with row-based logging.
binlog_row_value_options	-	(empty)	Permits an alternative binlog format for JSON document updates.
binlog_rows_query_log_events	-	OFF	MySQL-only variable for logging extra information in row-based logging.
binlog_transaction_compression*	-	*	MySQL variables relating to binary log compression.
binlog_transaction_dependency_history_size	-	25000	Maximum number of row hashes kept for looking up transactions that last modified a given row.
binlog_transaction_dependency_tracking	-	COMMIT_ORDER	For determining how to best use the slave's multithreaded applier.
block_encryption_mode	-	aes-128-ecb	MySQL-only variable for controlling the block encryption mode for block-based algorithms.
caching_sha2_password*	-	*	For use with MySQL's SHA-256 authentication with caching.
character_set_*	latin1 or utf8	utf8mb4	MySQL 8.0 defaults to the utf8mb4 character set.
check_constraint_checks	ON	-	Permits disabling constraint checks, for example when loading a table that violates some constraints that you plan to fix later.
check_proxy_users		OFF	MySQL-only variable for controlling whether the server performs proxy user mapping for authentication plugins.
collation_*	latin1_swedish_ci or utf8_general_ci	utf8mb4_0900_ai_ci	MySQL 8.0 defaults to the utf8mb4 character set.
column_compression_threshold	100	-	MariaDB supports Storage-engine Independent Column Compression.
column_compression_zlib_level	6	-	MariaDB supports Storage-engine Independent Column Compression.
column_compression_zlib_strategy	DEFAULT_STRATEGY	-	MariaDB supports Storage-engine Independent Column Compression.
column_compression_zlib_wrap	OFF	-	MariaDB supports Storage-engine Independent Column Compression.
connection_memory_chunk_size	-	8192	Chunk size for updates to the Global_connection_memory counter.
connection_memory_limit	-	18446744073709551615	Maximum memory for a single user connection.
create_admin_listener_thread	-	OFF	MySQL-only variable for whether to use a dedicated listening thread for admin network interface connections.
cte_max_recursion_depth	-	1000	When MySQL 8.0 introduced common table expressions they used a different name. MariaDB's variable is called max_recursive_iterations.
date_format	%Y-%m-%d	-	Unused variable removed in MySQL 8.0
datetime_format	%Y-%m-%d	-	Unused variable removed in MySQL 8.0
deadlock_search_depth_long	15	-	The Aria storage engine is only available in MariaDB.
deadlock_search_depth_short	4	-	The Aria storage engine is only available in MariaDB.

deadlock_timeout_long	50000000	-	The Aria storage engine is only available in MariaDB.
deadlock_timeout_short	10000	-	The Aria storage engine is only available in MariaDB.
debug_no_thread_alarm	OFF	-	Disable system thread alarm calls, for debugging or testing.
default_authentication_plugin	-	caching_sha2_password	MySQL 8 introduced a new authentication plugin.
default_collation_for_utf8mb4	-	utf8mb4_0900_ai_ci	For internal use in MySQL 8 replication.
default_master_connection	empty	-	For use with MariaDB's multi-source replication.
default_password_lifetime	0	360	MariaDB defaults to password expiration off.
default_regex_flags	empty	-	For handling incompatibilities between MariaDB's PCRE and the old regex library.
default_tmp_storage_engine	empty	InnoDB	Default storage engine used for tables created with CREATE TEMPORARY TABLE.
disabled_storage_engines		empty	MySQL-only variable for disabling specific storage engines.
disconnect_on_expired_password	OFF	ON	MariaDB password expiration is off by default, and by default does not disconnect a client when a password has expired.
encrypt_binlog	OFF	-	MariaDB enables table and tablespace encryption.
encrypt_tmp_files	OFF	-	MariaDB enables table and tablespace encryption.
encrypt_tmp_disk_tables	OFF	-	MariaDB enables table and tablespace encryption.
end_markers_in_json	-	OFF	MySQL-only variable for adding end markers to JSON output.
enforce_gtid_consistency	-	OFF	MariaDB and MySQL have different GTID implementations.
enforce_storage_engine	none		Forces the use of a particular storage engine for new tables.
eq_range_index_dive_limit	0	200	Variable for tuning when the optimizer should switch from using index dives to index statistics for qualifying rows estimation.
event_scheduler	OFF	ON	MySQL enables the event scheduler by default.
expensive_subquery_limit	100	-	Used for determining expensive queries for optimization.
extra_max_connections	1	-	Introduced in the MariaDB 5.1 threadpool.
extra_port	0	-	Introduced in the MariaDB 5.1 threadpool.
group_concat_max_len	1048576	1024	MariaDB increases the maximum length for a GROUP_CONCAT() result from 1K to 1M.
gtid_binlog_pos	empty	-	MariaDB and MySQL have different GTID implementations.
gtid_binlog_state	empty	-	MariaDB and MySQL have different GTID implementations.
gtid_cleanup_batch_size	64	-	MariaDB and MySQL have different GTID implementations.
gtid_current_pos	empty	-	MariaDB and MySQL have different GTID implementations.
gtid_domain_id	0	-	MariaDB and MySQL have different GTID implementations.
gtid_executed	-	empty	MariaDB and MySQL have different GTID implementations.
gtid_executed_compression_period	-	1000	MariaDB and MySQL have different GTID implementations.
gtid_ignore_duplicates	OFF	-	MariaDB and MySQL have different GTID implementations.
gtid_mode	-	OFF	MariaDB and MySQL have different GTID implementations.
gtid_next	-	AUTOMATIC	MariaDB and MySQL have different GTID implementations.
gtid_owned	-	empty	MariaDB and MySQL have different GTID implementations.

gtid_pos_auto_engines	empty	-	MariaDB and MySQL have different GTID implementations.
gtid_purged	-	empty	MariaDB and MySQL have different GTID implementations.
gtid_seq_no	0	-	MariaDB and MySQL have different GTID implementations.
gtid_slave_pos	empty	-	MariaDB and MySQL have different GTID implementations.
gtid_strict_mode	OFF	-	MariaDB and MySQL have different GTID implementations.
have_crypt	YES	-	MySQL has removed the ENCRYPT function.
have_openssl			MariaDB's version indicates whether YaSSL or openssl was used. MySQL's is a synonym for `have_ssl`.
have_query_cache	YES	-	MySQL has removed the query cache.
have_statement_timeout	-		Whether MySQL's statement execution timeout feature is available.
have_symlink	YES	DISABLED	MySQL has removed symlink support.
histogram_generation_max_mem_size	-	20000000	Added when MySQL 8 introduced Histogram-based Statistics. MariaDB uses histogram_size
histogram_size	0	-	MariaDB introduced Histogram-based Statistics.
histogram_type	DOUBLE_PREC_HB	-	MariaDB introduced Histogram-based Statistics.
idle_readonly_transaction_timeout	0	-	Time in seconds that the server waits for idle read-only transactions.
idle_transaction_timeout	0	-	Time in seconds that the server waits for idle transactions.
idle_write_transaction_timeout	0	-	Time in seconds that the server waits for idle write transactions.
ignore_builtin_innodb	OFF	-	Ignored and removed in MySQL 8.
in_predicate_conversion_threshold	1000	-	Controls the Conversion of Big IN Predicates Into Subqueries optimization.
in_transaction	0	-	Set to `1` if you are in a transaction, and `0` if not.
information_schema_stats_expiry	-	86400	Time until MySQL Information Schema cached statistics expire.
innodb_adaptive_flushing_lwm	10.000000	10	Adaptive flushing is enabled when this this low water mark percentage of the redo log capacity is reached. MariaDB's variable is a double, MySQL's an integer.
innodb_adaptive_hash_index	OFF	ON	Defaulting to OFF is a performance improvement especially for DROP TABLE, TRUNCATE TABLE, ALTER TABLE, or DROP INDEX operations
innodb_adaptive_max_sleep_delay	-	150000	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_api_*	-	*	Specific to MySQL's memcached, removed in MariaDB 10.2.
innodb_autoinc_lock_mode	1	2	MariaDB has an extra mode, `3`, for skipping the rollback of connected transactions. MySQL defaults to row-based replication, so can safely use `2`.
innodb_buffer_pool_instances	-	1	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6 since the original reasons for introducing no longer apply.
innodb_checksum_algorithm	full_crc32	crc32	`fullcrc32` permits encryption to be supported over a SPATIAL INDEX, which crc32 does not support.
innodb_checksums	ON	-	Deprecated option removed in MySQL.
innodb_commit_concurrency	-	0	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_compression_*	*	-	Introduced with MariaDB's InnoDB compression.
innodb_concurrency_tickets	-	5000	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_deadlock_report	Full	-	How to report deadlocks.
innodb_dedicated_server	-	OFF	MySQL option that automatically configures various settings if the server is a dedicated InnoDB database server.

innodb_default_encryption_key_id	1	-	Default encryption key id used for table encryption. See Data at Rest Encryption.
innodb_defragment	*	-	MariaDB can defragment InnoDB tablespaces.
innodb_directories	-	(empty)	Used to search for tablespace files when moving or restoring a new location.
innodb_disallow_writes	OFF	-	Tell InnoDB to stop any writes to disk.
innodb_encrypt_*	1	-	See MariaDB's Data at Rest Encryption.
innodb_fatal_semaphore_wait_threshold	600	-	MariaDB's fatal semaphore timeout is configurable.
innodb_flush_method	O_DIRECT	fsync	MariaDB InnoDB flushing method by default on Unix systems bypasses the file system cache for improved performance in most cases.
innodb_flush_neighbors	1	0	MySQL 8 by default now assumes the use of an SSD device.
innodb_force_primary_key	OFF	-	If set to 1 in MariaDB (0 is default) CREATE TABLEs without a primary or unique key where all keyparts are NOT NULL will not be accepted, and will return an error.
innodb_idle_flush_pct	100	-	Up to what percentage of dirty pages in MariaDB should be flushed when InnoDB finds it has spare resources to do so.
innodb_immediate_scrub_data_uncompressed	OFF	-	MariaDB has support for data scrubbing.
innodb_instant_alter_column_allowed	add_drop_reorder	-	See Instant ADD COLUMN for InnoDB.
innodb_locks_unsafe_for_binlog	OFF	-	Deprecated option in MariaDB for disabling gap locking for searches and index scans. Deprecated in MariaDB, use READ COMMITTED transaction isolation instead.
innodb_log_checksums	-	ON	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6, as there is no reason to allow checksums to be disabled on the redo log.
innodb_log_compressed_pages	-	ON	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6 as part of the InnoDB redo log performance improvements.
innodb_log_files_in_group	-	2	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6 as part of the InnoDB redo log performance improvements.
innodb_log_spin_*	-	*	MySQL variables for constraining CPU usage while waiting for flushed redo.
innodb_log_wait_for_flush_spin_hwm	-	*	MySQL variable for constraining CPU usage while waiting for flushed redo.
innodb_max_dirty_pages_pct	75	90	MySQL 8 increased the default to 90.
innodb_max_dirty_pages_pct_lwm	0	10	MySQL 8 increased the default to 10.
innodb_max_undo_log_size	10485760	1073741824	MariaDB 10.2 reduced the limit for when an undo tablespace is marked for truncation.
innodb_open_files	Autosized (2000)	Autosized (4000)	In most systems, autosized based on the table_open_cache setting, which differs between MariaDB and MySQL.
innodb_page_cleaners	-	1	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6 as the original reasons for for splitting the buffer pool have mostly gone away.
innodb_prefix_index_cluster_optimization	OFF	-	MariaDB includes the Facebook prefix index queries optimization.
innodb_print_ddl_logs	-	OFF	MySQL option for writing DDL logs to stderr.
innodb_read_only_compressed	ON	-	Whether to set ROW_FORMAT=COMPRESSED tables to read-only.
innodb_redo_log_encrypt	-	OFF	MySQL 8 has also now introduced redo log encryption, but used a different name. The equivalent option in MariaDB is innodb_encrypt_log.
innodb_replication_delay	-	0	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_spin_wait_delay	4	6	MariaDB changed the default from 6 to 4 based on extensive benchmarking.

innodb_stats_modified_counter	0	-	MariaDB option to control the calculation of new statistics.
innodb_stats_sample_pages	8	-	Deprecated MariaDB option for control over index distribution statistics.
innodb_stats_traditional	ON	-	Enabling gives a larger sample of pages for larger tables for the purposes of index statistics calculation.
innodb_sync_array_size	-	1	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_thread_concurrency	-	0	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_thread_sleep_delay	-	10000	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_undo_log_encrypt	-	OFF	MySQL option for encrypting undo logs residing in separate undo tablespaces.
innodb_undo_log_truncate	OFF	ON	MySQL 8 changes the default to ON, marking larger undo logs for truncation.
innodb_undo_tablespaces	0	2	Number of tablespace files used for dividing up the undo logs. MySQL 8 has deprecated this setting, and increased the default (and minimum) to 2.
innodb_use_atomic_writes	ON	-	Atomic writes are a faster alternative to innodb_doublewrite and MariaDB automatically detects when supporting SSD cards are used.
internal_tmp_disk_storage_engine	-	INNODB	MySQL uses this variable to set the storage engine for on-disk internal temporary tables.
internal_tmp_mem_storage_engine	-	TEMPTABLE	MySQL and MariaDB use different formats for temporary tables. In MariaDB, the aria_used_for_temp_tables performs a similar function.
join_buffer_space_limit	2097152	-	Maximum size in bytes of the query buffer. See block-based join algorithms.
join_cache_level	2	-	For determining the join algorithms. See block-based join algorithms
key_buffer_size	134217728	8388608	Size of the buffer for the index blocks used by MyISAM tables and shared for all threads.
key_cache_file_hash_size	512	-	Number of hash buckets for open and changed files.
key_cache_segments	0	-	The number of segments in a key cache. See Segmented Key Cache.
keyring_operations	-	ON	Whether MySQL 8's keyring operations are enabled.
last_gtid	-	empty	MariaDB and MySQL have different GTID implementations.
local_infile	ON	OFF	MySQL no longer supports LOAD DATA LOCAL by default.
lock_wait_timeout	86400	31536000	MariaDB has reduced the timeout for acquiring metadata locks.
log_bin	OFF	ON	MySQL 8 enables the binary log by default.
log_bin_compress	OFF	-	MariaDB setting for whether or not the binary log can be compressed.
log_bin_compress_min_len	256	-	Minimum length of sql statement (in statement mode) or record (in row mode) that can be compressed. See Compressing Events to Reduce Size of the Binary Log.
log_bin_use_v1_row_events	-	OFF	MySQL-only variable showing whether or not MySQL's version 2 binary logging format is being used.
log_disabled_statements	sp	-	Disable logging of certain statements to the general log.
log_error_services	-	log_filter_internal; log_sink_internal	Components to enable for MySQL error logging.
log_error_verbosity	-	3	MySQL variable for setting verbosity of error, warning, and note messages in the error log.
log_slave_updates	OFF	ON	MySQL 8 has by default enabled binary logging of updates a slave receives from a master.
log_slow_admin_statements	ON	OFF	MariaDB logs slow admin statements to the slow query log by default.

log_slow_disabled_statements	admin,call,slave,sp	-	Disable logging of certain statements to the slow query log.
log_slow_filter	admin, filesort, filesort_on_disk, full_join, full_scan, query_cache, query_cache_miss, tmp_table, tmp_table_on_disk	-	For slow query log filtering.
log_slow_rate_limit	1	-	Limits the number of queries logged to the slow query log.
log_slow_min_examined_row_limit	0	-	Previously named min_examined_row_limit (still an alias).
log_slow_query	0	-	Previously named log_slow_query (still an alias).
log_slow_query_file	host_name-slow.log	-	Previously named slow_query_log_file (still an alias).
log_slow_query_time	10.000000	-	Previously named long_query_time (still an alias).
log_slow_slave_statements	ON	OFF	MariaDB logs slow slave statements to the slow query log by default.
log_slow_verbosity	empty	-	Controls information to be added to the slow query log. See also Slow Query Log Extended Statistics.
log_statements_unsafe_for_binlog	-	ON	MySQL setting for controlling whether binlog warnings are written to the error log.
log_syslog*	platform-dependent	-	MySQL variables with settings for writing to syslog.
log_tc_size	24576	-	Size in bytes of the transaction coordinator log, defined in multiples of 4096.
log_throttle_queries_not_using_indexes	-	0	MySQL-only variable for limiting the number of statements without indexes written to the slow query log.
log_timestamps	-	UTC	MySQL-only variable controlling the timezone for certain logging conditions.
log_warnings	2	-	MySQL 8 has replaced with `log_error_verbosity`.
mandatory_roles	-	(empty)	MySQL variable for assigning roles to all users.
master_info_repository	-	TABLE	Whether slave logs master status and connection info to a table or a file.
max_allowed_packet	16M	64M	
max_error_count	64	1024	Specifies the maximum number of messages stored for display by SHOW ERRORS and SHOW WARNINGS statements.
max_execution_time	-	0	MySQL renamed the `max_statement_time` variable.
max_length_for_sort_data	64	1024	Used to decide which algorithm to choose when sorting rows. If the total size of the column data, not including columns that are part of the sort, is less than max_length_for_sort_data, then these are added to the sort key. This can speed up the sort as there's no need to re-read the same row again later. Setting the value too high can slow things down as there will be a higher disk activity for doing the sort.
max_long_data_size	16777216	-	Maximum size for parameter values sent with mysql_stmt_send_long_data(). Removed in MySQL 5.6.
max_password_errors	4294967295	-	Maximum number of failed connections attempts before no more are permitted.
max_points_in_geometry	-	65536	Maximum points_per_circle for MySQL's ST_Buffer_Strategy() function.
max_recursive_iterations	4294967295	-	Maximum number of iterations when executing recursive queries.
max_relay_log_size	1073741824	0	Can be set by session in MariaDB.
max_seeks_for_key	4294967295	4294967295 (32-bit) or 18446744073709547520 (64-bit)	The most key seeks required when searching with an index, regardless of the actual index cardinality. If this value is set lower than its default and maximum, indexes will tend to be preferred over table scans.
max_session_mem_used	9223372036854775807	-	Amount of memory a single user session is allowed to allocate.

max_statement_time	0	-	Maximum time in seconds that a query can execute before being aborted. MySQL used to have a variable of this name before renaming it `max_execution_time`.
max_tmp_tables	32	-	Unused variable removed in MySQL.
max_write_lock_count	4294967295	4294967295 (32-bit) or 18446744073709547520 (64-bit)	Read lock requests will be permitted for processing after this many write locks.
mrr_buffer_size	262144	-	Size of buffer to use when using multi-range read with range access. See Multi Range Read optimization.
myisam_block_size	1024	-	Block size used for MyISAM index pages.
myisam_recover_options	BACKUP,QUICK	OFF	MyISAM recovery mode.
myisam_sort_buffer_size	134216704	8388608	Size in bytes of the buffer allocated when creating or sorting indexes on a MyISAM table.
mysql_native_password_proxy_users	-	OFF	Whether MySQL's authentication plugin supports proxy users. I
mysql56_temporal_format	ON		Causes MariaDB to use the MySQL-5.6 low level formats for TIME, DATETIME and TIMESTAMP instead of the MariaDB 5.3+ version.
new	-	OFF	Used for backward-compatibility with MySQL 4.1, not present in MariaDB.
mysqlx+*	-	*	MySQL's X plugin related variables.
ngram_token_size	-	2	Sets the n-gram token size for MySQL's n-gram full-text parser.
offline_mode	-	OFF	MySQL setttting for specifying whether the server should run in offline mode.
old_alter_table	DEFAULT	OFF	An alias for alter_algorithm.
old_mode	Empty string	-	Used for getting MariaDB to emulate behavior from an old version of MySQL or MariaDB. See OLD Mode.
old_passwords	OFF	-	MySQL 8 is no longer compatible with the old pre-MySQL 4.1 form of password hashing.
optimizer_extra_pruning_depth	8	-	If the optimizer needs to enumerate a join prefix of this size or larger, then it will try aggressively prune away the search space.
optimizer_selectivity_sampling_limit	100	-	Controls number of record samples to check condition selectivity.
optimizer_switch	See details		A series of flags for controlling the query optimizer. MariaDB has introduced a number of new settings.
optimizer_trace_*	-	*	MySQL has more settings for optimizer tracing.
optimizer_use_condition_selectivity	4	-	Controls which statistics can be used by the optimizer when looking for the best query execution plan.
original_commit_timestamp	-	*	Used by MySQL 8 for delaying replication.
parser_max_mem_size	-	4294967295 (32-bit) or 18446744073709547520 (64-bit)	MySQL variable for limiting memory available to the parser.
password_*	-	*	Controls reuse of previous passwords in MySQL.
performance_schema	OFF	ON	The Performance Schema is off by default in MariaDB.
performance_schema_*			Many performance schema variables are autoset in MySQL, and MySQL has a different version, with additional variables.
plugin_maturity	One less than the server maturity	-	Minimum acceptable plugin maturity.
progress_report_time	5	-	Time in seconds between sending progress reports to the client for time-consuming statements.
proxy_protocol_networks	(empty)	-	Enable proxy protocol for these source networks.
query_alloc_block_size	16384	8192	Size in bytes of the extra blocks allocated during query parsing and execution (after query_prealloc_size is used up).
query_cache_*	*	-	MySQL has removed the query cache.

query_prealloc_size	24576	8192	Size in bytes of the persistent buffer for query parsing and execution, allocated on connect and freed on disconnect.
range_optimizer_max_mem_size	-	8388608	MySQL-only variable setting a limit on the range optimizer's memory usage.
rbr_exec_mode	-	STRICT	MySQL-only variable for determining the handling of certain key errors.
read_binlog_speed_limit	0	-	Permits restricting the speed at which the slave reads the binlog from the master.
regexp_*	-	*	Memory and time limits for regular expression matching operations.
relay_log_info_repository	-	TABLE	MySQL-only variable determining whether the slave's position in the relay logs is written to a file or table.
replicate_annotate_row_events	ON	-	Tells the slave to reproduce annotate_rows_events received from the master in its own binary log.
replicate_do_db	empty string	-	See Dynamic Replication Variables.
replicate_do_table	empty string	-	See Dynamic Replication Variables.
replicate_events_marked_for_skip	replicate	-	See Selectively skipping replication of binlog events.
replicate_ignore_db	empty string	-	See Dynamic Replication Variables.
replicate_ignore_table	empty string	-	See Dynamic Replication Variables.
replicate_rewrite_db	empty string	-	Only available as an option in MySQL.
replicate_wild_do_table	empty string	-	See Dynamic Replication Variables.
replicate_wild_ignore_table	empty string	-	See Dynamic Replication Variables.
result_metadata	-	FULL	Determine whether the server returns result set metadata for connections where this is optional.
rowid_merge_buff_size	8388608	-	See Non-semi-join subquery optimizations.
rpl_read_size	-	8192	Minimum data in bytes read from the binary and relay log files.
rpl_semi_sync_*	-	-	MariaDB includes semisynchronous replication without the need to install a plugin.
rpl_stop_slave_timeout	-	31536000	Controls the time that STOP SLAVE waits before timing out.
s3_*	*	-	The S3 storage engine is only available in MariaDB.
schema_definition_cache	-	256	Limits the number of schema definition objects kept in the dictionary object cache.
secure_auth	ON	-	Removed in MySQL.
secure_timestamp	NO	-	MariaDB-only option permitting the restricting of direct setting of a session timestamp..
server_id_bits	-	server_id	MySQL-only variable for use in MySQL Cluster.
server_uuid	-	UUID	MySQL-only variable containing the UUID.
session_track_gtids	-	OFF	MySQL-only variables for tracking gtid changes. MariaDB and MySQL's gtid implementation is different.
sha256_password_proxy_users	-	OFF	MySQL-only variable determining whether the sha256_password plugin supports proxy users.
show_create_table_verbosity	-	OFF	Option to cause SHOW CREATE TABLE to display ROW_FORMAT in all cases.
show_old_temporals	-	OFF	MySQL-only variable for determining whether SHOW CREATE TABLE output should include comments for old format temporal columns.
skip_parallel_replication	OFF	-	See parallel replication.
skip_replication	OFF	-	See Selectively skipping replication of binlog events.
slave_allow_batching	-	OFF	MySQL-only replication variable.
slave_checkpoint_group	-	512	MySQL-only replication variable.
slave_checkpoint_period	-	300	MySQL-only replication variable.
slave_ddl_exec_mode	IDEMPOTENT	-	Modes for how replication of DDL events should be executed.

slave_domain_parallel_threads	0	-	For configuring parallel replication.
slave_max_statement_time	0.000000	-	MariaDB setting to abort a query that has taken more than this in seconds to run on the replica.
slave_net_timeout	3600	60	MySQL reduced the timeout to 60s.
slave_parallel_max_queued	131072	-	For configuring parallel replication.
skip_grant_tables	0	-	Only available as an option in MySQL.
slave_parallel_mode	optimistic	-	Controls what transactions are applied in parallel when using parallel_replication.
slave_parallel_threads	0	-	For configuring parallel replication.
slave_parallel_type	-	DATABASE	MySQL-only replication variable.
slave_pending_jobs_size_max	-	16777216	MySQL-only replication variable.
slave_preserve_commit_order	-	OFF	MySQL-only replication variable.
slave_rows_search_algorithms	-	INDEX_SCAN, HASH_SCAN	MySQL-only replication variable.
slave_run_triggers_for_rbr	NO		See Running triggers on the slave for Row-based events for a description and use-case for this setting.
slave_transaction_retry_errors	1213,1205	-	When an error occurs during a transaction on the slave, replication usually halts. By default, transactions that caused a deadlock or elapsed lock wait timeout will be retried. One can add other errors to the the list of errors that should be retried by adding a comma-separated list of error numbers to this variable.
slave_transaction_retry_interval	0	-	Interval in seconds for the slave SQL thread to retry a failed transaction due to a deadlock, elapsed lock wait timeout or an error listed in slave_transaction_retry_errors.
sort_buffer_size	2097152	262144	The default sort buffer allocated has been reduced in MySQL.
sql_if_exists	OFF	-	Adds an implicit IF EXISTS to ALTER, RENAME and DROP of TABLES, VIEWS, FUNCTIONS and PACKAGES
sql_mode	STRICT_TRANS_TABLES, ERROR_FOR_DIVISION_BY_ZERO, NO_AUTO_CREATE_USER, NO_ENGINE_SUBSTITUTION	ONLY_FULL_GROUP_BY, STRICT_TRANS_TABLES, NO_ZERO_IN_DATE, NO_ZERO_DATE, ERROR_FOR_DIVISION_BY_ZERO, NO_ENGINE_SUBSTITUTION	See SQL Mode.
ssl_fips_mode	-	OFF	Whether FIPS mode is enabled on the server side.
standard_compliant_cte	ON	-	See Common Table Expressions.
storage_engine	InnoDB	-	Alias for default_storage_engine, removed in MySQL.
strict_password_validation	ON	-	In MariaDB, when password validation plugins are enabled, reject passwords that cannot be validated (passwords specified as a hash).
stored_program_definition_cache	-	256	Limits the number of stored program definition objects kept in the dictionary object cache.
super_read_only	-	OFF	MySQL variable for prohibiting client updates from users with the SUPER privilege.
sync_binlog	0	1	MySQL synchronizes all actions to the binary log before they are committed.
sync_frm	1	-	.frm files have been removed in MySQL.
system_versioning_alter_history	ERROR	-	MariaDB has System-Versioned Tables
system_versioning_asof	DEFAULT	-	MariaDB has System-Versioned Tables
system_versioning_insert_history	OFF	-	MariaDB has System-Versioned Tables
table_definition_cache	400	-1 (autosized)	Number of table definitions that can be cached.
table_open_cache_instances	8	16	Maximum number of table cache instances.
tablespace_definition_cache	-	256	Limits the number of tablespace definition objects kept in the dictionary object cache.
tcp_keepalive_interval	0	-	Interval, in seconds, between when successive keep-alive packets are sent if no acknowledgement is received.

tcp_keepalive_probes	0	-	Number of unacknowledged probes to send before considering the connection dead and notifying the application layer.
tcp_keepalive_time	0	-	Set the TCP_NODELAY option (disable Nagle's algorithm) on socket.
tcp_nodelay	1	-	Timeout, in milliseconds, with no activity until the first TCP keep-alive packet is sent.
temptable_max_ram	-	1GB	Limits the RAM used by MySQL's TempTable storage engine.
thread_cache_size	Autosized	-1 (autosized)	MariaDB uses an improved thread pool.
thread_concurrency	10	-	Removed in MySQL 5.7.
thread_pool_dedicated_listener	0	-	Better precision for the data in the Information Schema THREADPOOL_QUEUES Table.
thread_pool_exact_stats	0	-	Better precision for the data in the Information Schema THREADPOOL_QUEUES Table.
thread_pool_idle_timeout	60	-	See Using the Thread Pool.
thread_pool_max_threads	65536	-	See Using the Thread Pool.
thread_pool_min_threads	1	-	Windows-only. See Using the Thread Pool.
thread_pool_oversubscribe	3	-	See Using the Thread Pool.
thread_pool_prio_kickup	auto	-	See Using the Thread Pool.
thread_pool_priority	auto	-	See Using the Thread Pool.
thread_pool_size	Number of processors	16*	See Using the Thread Pool. *Only available in MySQL with a commercial plugin.
thread_pool_stall_limit	500	6*	See Using the Thread Pool. *Only available in MySQL with a commercial plugin.
thread_stack	299008	Varies	See Using the Thread Pool.
time_format	%H:%i:%s	-	Removed in MySQL.
timed_mutexes	OFF	-	Removed in MySQL.
tmp_disk_table_size	18446744073709551615	-	Max size for data for an internal temporary on-disk MyISAM or Aria table.
tmp_memory_table_size	16777216	-	Alias for tmp_table_size.
transaction_allow_batching	-	OFF	Variable for enabling batching of statements within the same transaction in MySQL Cluster.
transaction_isolation	-	REPEATABLE-READ	The MariaDB equivalent is tx_isolation.
transaction_read_only	-	OFF	The MariaDB equivalent is tx_read_only.
transaction_write_set_extraction	-	OFF	Unused MySQL-only variable.
tx_isolation	REPEATABLE-READ	-	The MySQL equivalent is transaction_isolation.
tx_read_only	OFF	-	The MySQL equivalent is transaction_read_only.
use_stat_tables	preferably_for_queries	-	Controls the use of engine-independent table statistics.
userstat	OFF	-	Whether to activate MariaDB's User Statistics implementation, not available in MySQL.
version_compile_zlib	-	*	Version of the zlib library compiled in.
version_malloc_library	*	-	Version of the used malloc library.
version_source_revision	Varies	-	Permits seeing exactly which version of the source was used for a build.
version_ssl_library	*	-	Version of the used TLS library.
windowing_high_use_precision	-	*	MySQL option allowing safety to be sacrificed for speed in window function calculations.
wsrep_*	*	-	Galera cluster is only available in MariaDB.
Variable	**MariaDB 10.11**	**MySQL 8.0**	**Notes**

2.1.14.1.12.4 System Variable Differences Between MariaDB 10.10 and MySQL 8.0

Contents

The following is a comparison of variables that either appear only in MariaDB 10.10 or MySQL 8.0, or have different default settings in MariaDB 10.10, and MySQL 8.0. The releases MariaDB 10.10.3 and MySQL 8.0.11, with only default plugins enabled, were used for the comparison. Note that MySQL 8 is an 'evergreen' release, so features may be added or removed in later releases.

For a more complete list of differences, see Incompatibilities and Feature Differences Between MariaDB 10.10 and MySQL 8.0 and Function Differences Between MariaDB 10.10 and MySQL 8.0

Comparison Table

Variable	MariaDB 10.10 Default	MySQL 8.0 Default	Notes
activate_all_roles_on_login	-	OFF	Determines whether to automatically activate roles on login.
allow_suspicious_udfs	0	-	Only available as an option in MySQL.
alter_algorithm	DEFAULT	-	MariaDB 10.3 introduced new ALTER TABLE ALGORITHM clauses to avoid slow copies in certain instances. This variable allows setting this if no `ALGORITHM` clause is specified.
analyze_sample_percentage	100.0000	-	Percentage of rows from the table ANALYZE TABLE will sample to collect table statistics.
aria_*	*	-	The Aria storage engine is only available in MariaDB.
auto_generate_certs	-	ON	Whether to automatically generate SSL key and certificate files.
avoid_temporal_upgrade	-	OFF	Determines whether ALTER TABLE implicitly upgrades temporal columns.
back_log	Autosized	Autosized	MariaDB and MySQL have different autosizing algorithms.
binlog_alter_two_phase	OFF	-	When set, split ALTER at binary logging into two statements: START ALTER and COMMIT/ROLLBACK ALTER.
binlog_annotate_row_events	ON	-	Introduced in MariaDB 5.3 for replicating between MariaDB 5.3 and MySQL/MariaDB 5.1.
binlog_commit_wait_count	0	-	For use in MariaDB's parallel replication.
binlog_commit_wait_usec	100000	-	For use in MariaDB's parallel replication.
binlog_error_action		ABORT_SERVER	MySQL-only variable for controlling what happens when the server cannot write to the binary log.
binlog_expire_logs_seconds	0	2592000	Sets the binary log expiration period in seconds
binlog_file_cache_size	16184	-	For setting the size of the file cache for the binary log.
binlog_format	MIXED	ROW	MariaDB and MySQL have differing binary log formats.
binlog_group_commit_sync_delay		0	MySQL-only variable for controlling the wait time before synchronizing the binary log file to disk.
binlog_group_commit_sync_no_delay_count		0	MySQL-only variable for setting the maximum number of transactions to wait for before aborting the current `binlog_group_commit_sync_delay` delay.
binlog_gtid_simple_recovery	-	ON	MySQL-only GTID variable. MariaDB's GTID implementation is different.
binlog_max_flush_queue_time	-	0	Specifies a timeout for reading transactions from the flush queue before continuing with group commit and syncing log to disk.
binlog_optimize_thread_scheduling	ON	-	For optimized kernel thread scheduling.
binlog_order_commits	-	ON	Determines whether transactions may be committed in parallel.
binlog_row_metadata	NO_LOG	MINIMAL	Determines the amount of table metadata added to the binary log with row-based logging.
binlog_row_value_options	-	(empty)	Permits an alternative binlog format for JSON document updates.

binlog_rows_query_log_events	-	OFF	MySQL-only variable for logging extra information in row-based logging.
binlog_transaction_dependency_history_size	-	25000	Maximum number of row hashes kept for looking up transactions that last modified a given row.
binlog_transaction_dependency_tracking	-	COMMIT_ORDER	For determining how to best use the slave's multithreaded applier.
block_encryption_mode	-	aes-128-ecb	MySQL-only variable for controlling the block encryption mode for block-based algorithms.
caching_sha2_password*	-	*	For use with MySQL's SHA-256 authentication with caching.
character_set_*	latin1 or utf8	utf8mb4	MySQL 8.0 defaults to the utf8mb4 character set.
check_constraint_checks	ON	-	Permits disabling constraint checks, for example when loading a table that violates some constraints that you plan to fix later.
check_proxy_users		OFF	MySQL-only variable for controlling whether the server performs proxy user mapping for authentication plugins.
collation_*	latin1_swedish_ci or utf8_general_ci	utf8mb4_0900_ai_ci	MySQL 8.0 defaults to the utf8mb4 character set.
column_compression_threshold	100	-	MariaDB supports Storage-engine Independent Column Compression.
column_compression_zlib_level	6	-	MariaDB supports Storage-engine Independent Column Compression.
column_compression_zlib_strategy	DEFAULT_STRATEGY	-	MariaDB supports Storage-engine Independent Column Compression.
column_compression_zlib_wrap	OFF	-	MariaDB supports Storage-engine Independent Column Compression.
cte_max_recursion_depth	-	1000	When MySQL 8.0 introduced common table expressions they used a different name. MariaDB's variable is called max_recursive_iterations.
date_format	%Y-%m-%d	-	Unused variable removed in MySQL 8.0
datetime_format	%Y-%m-%d	-	Unused variable removed in MySQL 8.0
deadlock_search_depth_long	15	-	The Aria storage engine is only available in MariaDB.
deadlock_search_depth_short	4	-	The Aria storage engine is only available in MariaDB.
deadlock_timeout_long	50000000	-	The Aria storage engine is only available in MariaDB.
deadlock_timeout_short	10000	-	The Aria storage engine is only available in MariaDB.
debug_no_thread_alarm	OFF	-	Disable system thread alarm calls, for debugging or testing.
default_authentication_plugin	-	caching_sha2_password	MySQL 8 introduced a new authentication plugin.
default_collation_for_utf8mb4	-	utf8mb4_0900_ai_ci	For internal use in MySQL 8 replication.
default_master_connection	empty	-	For use with MariaDB's multi-source replication.
default_password_lifetime	0	360	MariaDB defaults to password expiration off.
default_regex_flags	empty	-	For handling incompatibilities between MariaDB's PCRE and the old regex library.
default_tmp_storage_engine	empty	InnoDB	Default storage engine used for tables created with CREATE TEMPORARY TABLE.
disabled_storage_engines		empty	MySQL-only variable for disabling specific storage engines.
disconnect_on_expired_password	OFF	ON	MariaDB password expiration is off by default, and by default does not disconnect a client when a password has expired.
encrypt_binlog	OFF	-	MariaDB enables table and tablespace encryption.
encrypt_tmp_files	OFF	-	MariaDB enables table and tablespace encryption.
encrypt_tmp_disk_tables	OFF	-	MariaDB enables table and tablespace encryption.
end_markers_in_json	-	OFF	MySQL-only variable for adding end markers to JSON output.

enforce_gtid_consistency	-	OFF	MariaDB and MySQL have different GTID implementations.
enforce_storage_engine	none		Forces the use of a particular storage engine for new tables.
eq_range_index_dive_limit	0	200	Variable for tuning when the optimizer should switch from using index dives to index statistics for qualifying rows estimation.
event_scheduler	OFF	ON	MySQL enables the event scheduler by default.
expensive_subquery_limit	100	-	Used for determining expensive queries for optimization.
extra_max_connections	1	-	Introduced in the MariaDB 5.1 threadpool.
extra_port	0	-	Introduced in the MariaDB 5.1 threadpool.
group_concat_max_len	1048576	1024	MariaDB increases the maximum length for a GROUP_CONCAT() result from 1K to 1M.
gtid_binlog_pos	empty	-	MariaDB and MySQL have different GTID implementations.
gtid_binlog_state	empty	-	MariaDB and MySQL have different GTID implementations.
gtid_cleanup_batch_size	64	-	MariaDB and MySQL have different GTID implementations.
gtid_current_pos	empty	-	MariaDB and MySQL have different GTID implementations.
gtid_domain_id	0	-	MariaDB and MySQL have different GTID implementations.
gtid_executed	-	empty	MariaDB and MySQL have different GTID implementations.
gtid_executed_compression_period	-	1000	MariaDB and MySQL have different GTID implementations.
gtid_ignore_duplicates	OFF	-	MariaDB and MySQL have different GTID implementations.
gtid_mode	-	OFF	MariaDB and MySQL have different GTID implementations.
gtid_next	-	AUTOMATIC	MariaDB and MySQL have different GTID implementations.
gtid_owned	-	empty	MariaDB and MySQL have different GTID implementations.
gtid_pos_auto_engines	empty	-	MariaDB and MySQL have different GTID implementations.
gtid_purged	-	empty	MariaDB and MySQL have different GTID implementations.
gtid_seq_no	0	-	MariaDB and MySQL have different GTID implementations.
gtid_slave_pos	empty	-	MariaDB and MySQL have different GTID implementations.
gtid_strict_mode	OFF	-	MariaDB and MySQL have different GTID implementations.
have_crypt	YES	-	MySQL has removed the ENCRYPT function.
have_openssl			MariaDB's version indicates whether YaSSL or openssl was used. MySQL's is a synonym for `have_ssl`.
have_query_cache	YES	-	MySQL has removed the query cache.
have_statement_timeout	-		Whether MySQL's statement execution timeout feature is available.
have_symlink	YES	DISABLED	MySQL has removed symlink support.
histogram_generation_max_mem_size	-	20000000	Added when MySQL 8 introduced Histogram-based Statistics. MariaDB uses histogram_size
histogram_size	0	-	MariaDB introduced Histogram-based Statistics.
histogram_type	SINGLE_PREC_HB	-	MariaDB introduced Histogram-based Statistics.
idle_readonly_transaction_timeout	0	-	Time in seconds that the server waits for idle read-only transactions.
idle_transaction_timeout	0	-	Time in seconds that the server waits for idle transactions.
idle_write_transaction_timeout	0	-	Time in seconds that the server waits for idle write transactions.
ignore_builtin_innodb	OFF	-	Ignored and removed in MySQL 8.

in_predicate_conversion_threshold	1000	-	Controls the Conversion of Big IN Predicates Into Subqueries optimization.
in_transaction	0	-	Set to `1` if you are in a transaction, and `0` if not.
information_schema_stats_expiry	-	86400	Time until MySQL Information Schema cached statistics expire.
innodb_adaptive_flushing_lwm	10.000000	10	Adaptive flushing is enabled when this this low water mark percentage of the redo log capacity is reached. MariaDB's variable is a double, MySQL's an integer.
innodb_adaptive_hash_index	OFF	ON	Defaulting to OFF is a performance improvement especially for DROP TABLE, TRUNCATE TABLE, ALTER TABLE, or DROP INDEX operations
innodb_adaptive_max_sleep_delay	-	150000	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_api_*	-	*	Specific to MySQL's memcached, removed in MariaDB 10.2.
innodb_autoinc_lock_mode	1	2	MariaDB has an extra mode, `3`, for skipping the rollback of connected transactions. MySQL defaults to row-based replication, so can safely use `2`.
innodb_buffer_pool_instances	-	1	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6 since the original reasons for introducing no longer apply.
innodb_checksum_algorithm	full_crc32	crc32	`fullcrc32` permits encryption to be supported over a SPATIAL INDEX, which crc32 does not support.
innodb_checksums	ON	-	Deprecated option removed in MySQL.
innodb_commit_concurrency	-	0	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_compression_*	*	-	Introduced with MariaDB's InnoDB compression.
innodb_concurrency_tickets	-	5000	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_deadlock_report	Full	-	How to report deadlocks.
innodb_dedicated_server	-	OFF	MySQL option that automatically configures various settings if the server is a dedicated InnoDB database server.
innodb_default_encryption_key_id	1	-	Default encryption key id used for table encryption. See Data at Rest Encryption.
innodb_defragment	*	-	MariaDB can defragment InnoDB tablespaces.
innodb_directories	-	(empty)	Used to search for tablespace files when moving or restoring a new location.
innodb_disallow_writes	OFF	-	Tell InnoDB to stop any writes to disk.
innodb_encrypt_*	1	-	See MariaDB's Data at Rest Encryption.
innodb_fatal_semaphore_wait_threshold	600	-	MariaDB's fatal semaphore timeout is configurable.
innodb_flush_method	O_DIRECT	fsync	MariaDB InnoDB flushing method by default on Unix systems bypasses the file system cache for improved performance in most cases.
innodb_flush_neighbors	1	0	MySQL 8 by default now assumes the use of an SSD device.
innodb_force_primary_key	OFF	-	If set to 1 in MariaDB (0 is default) CREATE TABLEs without a primary or unique key where all keyparts are NOT NULL will not be accepted, and will return an error.
innodb_idle_flush_pct	100	-	Up to what percentage of dirty pages in MariaDB should be flushed when InnoDB finds it has spare resources to do so.
innodb_immediate_scrub_data_uncompressed	OFF	-	MariaDB has support for data scrubbing.
innodb_instant_alter_column_allowed	add_drop_reorder	-	See Instant ADD COLUMN for InnoDB.
innodb_locks_unsafe_for_binlog	OFF	-	Deprecated option in MariaDB for disabling gap locking for searches and index scans. Deprecated in MariaDB, use READ COMMITTED transaction isolation instead.
innodb_log_checksums	-	ON	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6, as there is no reason to allow checksums to be disabled on the redo log.

innodb_log_compressed_pages	-	ON	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6 as part of the InnoDB redo log performance improvements.
innodb_log_files_in_group	-	2	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6 as part of the InnoDB redo log performance improvements.
innodb_log_spin_*	-	*	MySQL variables for constraining CPU usage while waiting for flushed redo.
innodb_log_wait_for_flush_spin_hwm	-	*	MySQL variable for constraining CPU usage while waiting for flushed redo.
innodb_max_dirty_pages_pct	75	90	MySQL 8 increased the default to 90.
innodb_max_dirty_pages_pct_lwm	0	10	MySQL 8 increased the default to 10.
innodb_max_undo_log_size	10485760	1073741824	MariaDB 10.2 reduced the limit for when an undo tablespace is marked for truncation.
innodb_open_files	Autosized (2000)	Autosized (4000)	In most systems, autosized based on the table_open_cache setting, which differs between MariaDB and MySQL.
innodb_page_cleaners	-	1	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6 as the original reasons for for splitting the buffer pool have mostly gone away.
innodb_prefix_index_cluster_optimization	OFF	-	MariaDB includes the Facebook prefix index queries optimization.
innodb_print_ddl_logs	-	OFF	MySQL option for writing DDL logs to stderr.
innodb_read_only_compressed	ON	-	Whether to set ROW_FORMAT=COMPRESSED tables to read-only.
innodb_redo_log_encrypt	-	OFF	MySQL 8 has also now introduced redo log encryption, but used a different name. The equivalent option in MariaDB is innodb_encrypt_log.
innodb_replication_delay	-	0	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_spin_wait_delay	4	6	MariaDB changed the default from 6 to 4 based on extensive benchmarking.
innodb_stats_modified_counter	0	-	MariaDB option to control the calculation of new statistics.
innodb_stats_sample_pages	8	-	Deprecated MariaDB option for control over index distribution statistics.
innodb_stats_traditional	ON	-	Enabling gives a larger sample of pages for larger tables for the purposes of index statistics calculation.
innodb_sync_array_size	-	1	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_thread_concurrency	-	0	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_thread_sleep_delay	-	10000	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_undo_log_encrypt	-	OFF	MySQL option for encrypting undo logs residing in separate undo tablespaces.
innodb_undo_log_truncate	OFF	ON	MySQL 8 changes the default to ON, marking larger undo logs for truncation.
innodb_undo_tablespaces	0	2	Number of tablespace files used for dividing up the undo logs. MySQL 8 has deprecated this setting, and increased the default (and minimum) to 2.
innodb_use_atomic_writes	ON	-	Atomic writes are a faster alternative to innodb_doublewrite and MariaDB automatically detects when supporting SSD cards are used.
internal_tmp_disk_storage_engine	-	INNODB	MySQL uses this variable to set the storage engine for on-disk internal temporary tables.
internal_tmp_mem_storage_engine	-	TEMPTABLE	MySQL and MariaDB use different formats for temporary tables. In MariaDB, the aria_used_for_temp_tables performs a similar function.
join_buffer_space_limit	2097152	-	Maximum size in bytes of the query buffer. See block-based join algorithms.
join_cache_level	2	-	For determining the join algorithms. See block-based join algorithms

key_buffer_size	134217728	8388608	Size of the buffer for the index blocks used by MyISAM tables and shared for all threads.
key_cache_file_hash_size	512	-	Number of hash buckets for open and changed files.
key_cache_segments	0	-	The number of segments in a key cache. See Segmented Key Cache.
keyring_operations	-	ON	Whether MySQL 8's keyring operations are enabled.
last_gtid	-	empty	MariaDB and MySQL have different GTID implementations.
local_infile	ON	OFF	MySQL no longer supports LOAD DATA LOCAL by default.
lock_wait_timeout	86400	31536000	MariaDB has reduced the timeout for acquiring metadata locks.
log_bin	OFF	ON	MySQL 8 enables the binary log by default.
log_bin_compress	OFF	-	MariaDB setting for whether or not the binary log can be compressed.
log_bin_compress_min_len	256	-	Minimum length of sql statement (in statement mode) or record (in row mode) that can be compressed. See Compressing Events to Reduce Size of the Binary Log.
log_bin_use_v1_row_events	-	OFF	MySQL-only variable showing whether or not MySQL's version 2 binary logging format is being used.
log_disabled_statements	sp	-	Disable logging of certain statements to the general log.
log_error_services	-	log_filter_internal; log_sink_internal	Components to enable for MySQL error logging.
log_error_verbosity	-	3	MySQL variable for setting verbosity of error, warning, and note messages in the error log.
log_slave_updates	OFF	ON	MySQL 8 has by default enabled binary logging of updates a slave receives from a master.
log_slow_admin_statements	ON	OFF	MariaDB logs slow admin statements to the slow query log by default.
log_slow_disabled_statements	admin,call,slave,sp	-	Disable logging of certain statements to the slow query log.
log_slow_filter	admin, filesort, filesort_on_disk, full_join, full_scan, query_cache, query_cache_miss, tmp_table, tmp_table_on_disk	-	For slow query log filtering.
log_slow_rate_limit	1	-	Limits the number of queries logged to the slow query log.
log_slow_slave_statements	ON	OFF	MariaDB logs slow slave statements to the slow query log by default.
log_slow_verbosity	empty	-	Controls information to be added to the slow query log. See also Slow Query Log Extended Statistics.
log_statements_unsafe_for_binlog	-	ON	MySQL setting for controlling whether binlog warnings are written to the error log.
log_syslog*	platform-dependent	-	MySQL variables with settings for writing to syslog.
log_tc_size	24576	-	Size in bytes of the transaction coordinator log, defined in multiples of 4096.
log_throttle_queries_not_using_indexes	-	0	MySQL-only variable for limiting the number of statements without indexes written to the slow query log.
log_timestamps	-	UTC	MySQL-only variable controlling the timezone for certain logging conditions.
log_warnings	2	-	MySQL 8 has replaced with `log_error_verbosity`.
mandatory_roles	-	(empty)	MySQL variable for assigning roles to all users.
master_info_repository	-	TABLE	Whether slave logs master status and connection info to a table or a file.
max_allowed_packet	16M	64M	
max_error_count	64	1024	Specifies the maximum number of messages stored for display by SHOW ERRORS and SHOW WARNINGS statements.

max_execution_time	-	0	MySQL renamed the `max_statement_time` variable.
max_length_for_sort_data	64	1024	Used to decide which algorithm to choose when sorting rows. If the total size of the column data, not including columns that are part of the sort, is less than max_length_for_sort_data, then these are added to the sort key. This can speed up the sort as there's no need to re-read the same row again later. Setting the value too high can slow things down as there will be a higher disk activity for doing the sort.
max_long_data_size	16777216	-	Maximum size for parameter values sent with mysql_stmt_send_long_data(). Removed in MySQL 5.6.
max_password_errors	4294967295	-	Maximum number of failed connections attempts before no more are permitted.
max_points_in_geometry	-	65536	Maximum points_per_circle for MySQL's ST_Buffer_Strategy() function.
max_recursive_iterations	4294967295	-	Maximum number of iterations when executing recursive queries.
max_relay_log_size	1073741824	0	Can be set by session in MariaDB.
max_seeks_for_key	4294967295	4294967295 (32-bit) or 18446744073709547520 (64-bit)	The most key seeks required when searching with an index, regardless of the actual index cardinality. If this value is set lower than its default and maximum, indexes will tend to be preferred over table scans.
max_session_mem_used	9223372036854775807	-	Amount of memory a single user session is allowed to allocate.
max_statement_time	0	-	Maximum time in seconds that a query can execute before being aborted. MySQL used to have a variable of this name before renaming it `max_execution_time`.
max_tmp_tables	32	-	Unused variable removed in MySQL.
max_write_lock_count	4294967295	4294967295 (32-bit) or 18446744073709547520 (64-bit)	Read lock requests will be permitted for processing after this many write locks.
mrr_buffer_size	262144	-	Size of buffer to use when using multi-range read with range access. See Multi Range Read optimization.
myisam_block_size	1024	-	Block size used for MyISAM index pages.
myisam_recover_options	BACKUP,QUICK	OFF	MyISAM recovery mode.
myisam_sort_buffer_size	134216704	8388608	Size in bytes of the buffer allocated when creating or sorting indexes on a MyISAM table.
mysql_native_password_proxy_users	-	OFF	Whether MySQL's authentication plugin supports proxy users. I
mysql56_temporal_format	ON		Causes MariaDB to use the MySQL-5.6 low level formats for TIME, DATETIME and TIMESTAMP instead of the MariaDB 5.3+ version.
new	-	OFF	Used for backward-compatibility with MySQL 4.1, not present in MariaDB.
mysqlx+*	-	*	MySQL's X plugin related variables.
ngram_token_size	-	2	Sets the n-gram token size for MySQL's n-gram full-text parser.
offline_mode	-	OFF	MySQL settting for specifying whether the server should run in offline mode.
old_alter_table	DEFAULT	OFF	An alias for alter_algorithm.
old_mode	Empty string	-	Used for getting MariaDB to emulate behavior from an old version of MySQL or MariaDB. See OLD Mode.
old_passwords	OFF	-	MySQL 8 is no longer compatible with the old pre-MySQL 4.1 form of password hashing.
optimizer_extra_pruning_depth	8	-	If the optimizer needs to enumerate a join prefix of this size or larger, then it will try aggressively prune away the search space.
optimizer_selectivity_sampling_limit	100	-	Controls number of record samples to check condition selectivity.
optimizer_switch	See details		A series of flags for controlling the query optimizer. MariaDB has introduced a number of new settings.

optimizer_trace_*	-	*	MySQL has more settings for optimizer tracing.
optimizer_use_condition_selectivity	4	-	Controls which statistics can be used by the optimizer when looking for the best query execution plan.
original_commit_timestamp	-	*	Used by MySQL 8 for delaying replication.
parser_max_mem_size	-	4294967295 (32-bit) or 18446744073709547520 (64-bit)	MySQL variable for limiting memory available to the parser.
password_*	-	*	Controls reuse of previous passwords in MySQL.
performance_schema	OFF	ON	The Performance Schema is off by default in MariaDB.
performance_schema_*			Many performance schema variables are autoset in MySQL, and MySQL has a different version, with additional variables.
plugin_maturity	One less than the server maturity	-	Minimum acceptable plugin maturity.
progress_report_time	5	-	Time in seconds between sending progress reports to the client for time-consuming statements.
proxy_protocol_networks	(empty)	-	Enable proxy protocol for these source networks.
query_alloc_block_size	16384	8192	Size in bytes of the extra blocks allocated during query parsing and execution (after query_prealloc_size is used up).
query_cache_*	*	-	MySQL has removed the query cache.
query_prealloc_size	24576	8192	Size in bytes of the persistent buffer for query parsing and execution, allocated on connect and freed on disconnect.
range_optimizer_max_mem_size	-	8388608	MySQL-only variable setting a limit on the range optimizer's memory usage.
rbr_exec_mode	-	STRICT	MySQL-only variable for determining the handling of certain key errors.
read_binlog_speed_limit	0	-	Permits restricting the speed at which the slave reads the binlog from the master.
regexp_*	-	*	Memory and time limits for regular expression matching operations.
relay_log_info_repository	-	TABLE	MySQL-only variable determining whether the slave's position in the relay logs is written to a file or table.
replicate_annotate_row_events	ON	-	Tells the slave to reproduce annotate_rows_events received from the master in its own binary log.
replicate_do_db	empty string	-	See Dynamic Replication Variables.
replicate_do_table	empty string	-	See Dynamic Replication Variables.
replicate_events_marked_for_skip	replicate	-	See Selectively skipping replication of binlog events.
replicate_ignore_db	empty string	-	See Dynamic Replication Variables.
replicate_ignore_table	empty string	-	See Dynamic Replication Variables.
replicate_wild_do_table	empty string	-	See Dynamic Replication Variables.
replicate_wild_ignore_table	empty string	-	See Dynamic Replication Variables.
result_metadata	-	FULL	Determine whether the server returns result set metadata for connections where this is optional.
rowid_merge_buff_size	8388608	-	See Non-semi-join subquery optimizations.
rpl_read_size	-	8192	Minimum data in bytes read from the binary and relay log files.
rpl_semi_sync_*	-	-	MariaDB includes semisynchronous replication without the need to install a plugin.
rpl_stop_slave_timeout	-	31536000	Controls the time that STOP SLAVE waits before timing out.
s3_*	*	-	The S3 storage engine is only available in MariaDB.
schema_definition_cache	-	256	Limits the number of schema definition objects kept in the dictionary object cache.
secure_auth	ON	-	Removed in MySQL.
secure_timestamp	NO	-	MariaDB-only option permitting the restricting of direct setting of a session timestamp..
server_id_bits	-	server_id	MySQL-only variable for use in MySQL Cluster.

server_uuid	-	UUID	MySQL-only variable containing the UUID.
session_track_gtids	-	OFF	MySQL-only variables for tracking gtid changes. MariaDB and MySQL's gtid implementation is different.
sha256_password_proxy_users	-	OFF	MySQL-only variable determining whether the sha256_password plugin supports proxy users.
show_create_table_verbosity	-	OFF	Option to cause SHOW CREATE TABLE to display ROW_FORMAT in all cases.
show_old_temporals	-	OFF	MySQL-only variable for determining whether SHOW CREATE TABLE output should include comments for old format temporal columns.
skip_parallel_replication	OFF	-	See parallel replication.
skip_replication	OFF	-	See Selectively skipping replication of binlog events.
slave_allow_batching	-	OFF	MySQL-only replication variable.
slave_checkpoint_group	-	512	MySQL-only replication variable.
slave_checkpoint_period	-	300	MySQL-only replication variable.
slave_ddl_exec_mode	IDEMPOTENT	-	Modes for how replication of DDL events should be executed.
slave_domain_parallel_threads	0	-	For configuring parallel replication.
slave_net_timeout	3600	60	MySQL reduced the timeout to 60s.
slave_parallel_max_queued	131072	-	For configuring parallel replication.
skip_grant_tables	0	-	Only available as an option in MySQL.
slave_max_statement_time	0.000000	-	MariaDB setting to abort a query that has taken more than this in seconds to run on the replica.
slave_parallel_mode	optimistic	-	Controls what transactions are applied in parallel when using parallel_replication.
slave_parallel_threads	0	-	For configuring parallel replication.
slave_parallel_type	-	DATABASE	MySQL-only replication variable.
slave_pending_jobs_size_max	-	16777216	MySQL-only replication variable.
slave_preserve_commit_order	-	OFF	MySQL-only replication variable.
slave_rows_search_algorithms	-	INDEX_SCAN, HASH_SCAN	MySQL-only replication variable.
slave_run_triggers_for_rbr	NO		See Running triggers on the slave for Row-based events for a description and use-case for this setting.
slave_transaction_retry_errors	1213,1205	-	When an error occurs during a transaction on the slave, replication usually halts. By default, transactions that caused a deadlock or elapsed lock wait timeout will be retried. One can add other errors to the the list of errors that should be retried by adding a comma-separated list of error numbers to this variable.
slave_transaction_retry_interval	0	-	Interval in seconds for the slave SQL thread to retry a failed transaction due to a deadlock, elapsed lock wait timeout or an error listed in slave_transaction_retry_errors.
sort_buffer_size	2097152	262144	The default sort buffer allocated has been reduced in MySQL.
sql_if_exists	OFF	-	Adds an implicit IF EXISTS to ALTER, RENAME and DROP of TABLES, VIEWS, FUNCTIONS and PACKAGES
sql_mode	STRICT_TRANS_TABLES, ERROR_FOR_DIVISION_BY_ZERO, NO_AUTO_CREATE_USER, NO_ENGINE_SUBSTITUTION	ONLY_FULL_GROUP_BY, STRICT_TRANS_TABLES, NO_ZERO_IN_DATE, NO_ZERO_DATE, ERROR_FOR_DIVISION_BY_ZERO, NO_ENGINE_SUBSTITUTION	See SQL Mode.
ssl_fips_mode	-	OFF	Whether FIPS mode is enabled on the server side.
standard_compliant_cte	ON	-	See Common Table Expressions.
storage_engine	InnoDB	-	Alias for default_storage_engine, removed in MySQL.
strict_password_validation	ON	-	In MariaDB, when password validation plugins are enabled, reject passwords that cannot be validated (passwords specified as a hash).
stored_program_definition_cache	-	256	Limits the number of stored program definition objects kept in the dictionary object cache.

super_read_only	-	OFF	MySQL variable for prohibiting client updates from users with the SUPER privilege.
sync_binlog	0	1	MySQL synchronizes all actions to the binary log before they are committed.
sync_frm	1	-	.frm files have been removed in MySQL.
system_versioning_alter_history	ERROR	-	MariaDB has System-Versioned Tables
system_versioning_asof	DEFAULT	-	MariaDB has System-Versioned Tables
table_definition_cache	400	-1 (autosized)	Number of table definitions that can be cached.
table_open_cache_instances	8	16	Maximum number of table cache instances.
tablespace_definition_cache	-	256	Limits the number of tablespace definition objects kept in the dictionary object cache.
tcp_keepalive_interval	0	-	Interval, in seconds, between when successive keep-alive packets are sent if no acknowledgement is received.
tcp_keepalive_probes	0	-	Number of unacknowledged probes to send before considering the connection dead and notifying the application layer.
tcp_keepalive_time	0	-	Set the TCP_NODELAY option (disable Nagle's algorithm) on socket.
tcp_nodelay	1	-	Timeout, in milliseconds, with no activity until the first TCP keep-alive packet is sent.
temptable_max_ram	-	1GB	Limits the RAM used by MySQL's TempTable storage engine.
thread_cache_size	Autosized	-1 (autosized)	MariaDB uses an improved thread pool.
thread_concurrency	10	-	Removed in MySQL 5.7.
thread_pool_dedicated_listener	0	-	Better precision for the data in the Information Schema THREADPOOL_QUEUES Table.
thread_pool_exact_stats	0	-	Better precision for the data in the Information Schema THREADPOOL_QUEUES Table.
thread_pool_idle_timeout	60	-	See Using the Thread Pool.
thread_pool_max_threads	65536	-	See Using the Thread Pool.
thread_pool_min_threads	1	-	Windows-only. See Using the Thread Pool.
thread_pool_oversubscribe	3	-	See Using the Thread Pool.
thread_pool_prio_kickup	auto	-	See Using the Thread Pool.
thread_pool_priority	auto	-	See Using the Thread Pool.
thread_pool_size	Number of processors	16*	See Using the Thread Pool. *Only available in MySQL with a commercial plugin.
thread_pool_stall_limit	500	6*	See Using the Thread Pool. *Only available in MySQL with a commercial plugin.
thread_stack	299008	Varies	See Using the Thread Pool.
time_format	%H:%i:%s	-	Removed in MySQL.
timed_mutexes	OFF	-	Removed in MySQL.
tmp_disk_table_size	18446744073709551615	-	Max size for data for an internal temporary on-disk MyISAM or Aria table.
tmp_memory_table_size	16777216	-	Alias for tmp_table_size.
transaction_allow_batching	-	OFF	Variable for enabling batching of statements within the same transaction in MySQL Cluster.
transaction_isolation	-	REPEATABLE-READ	The MariaDB equivalent is tx_isolation.
transaction_read_only	-	OFF	The MariaDB equivalent is tx_read_only.
transaction_write_set_extraction	-	OFF	Unused MySQL-only variable.
tx_isolation	REPEATABLE-READ	-	The MySQL equivalent is transaction_isolation.
tx_read_only	OFF	-	The MySQL equivalent is transaction_read_only.
use_stat_tables	preferably_for_queries	-	Controls the use of engine-independent table statistics.
userstat	OFF	-	Whether to activate MariaDB's User Statistics implementation, not available in MySQL.
version_compile_zlib	-	*	Version of the zlib library compiled in.

version_malloc_library	*	-	Version of the used malloc library.
version_source_revision	Varies	-	Permits seeing exactly which version of the source was used for a build.
version_ssl_library	*	-	Version of the used TLS library.
windowing_high_use_precision	-	*	MySQL option allowing safety to be sacrificed for speed in window function calculations.
wsrep_*	*	-	Galera cluster is only available in MariaDB.
Variable	**MariaDB 10.10**	**MySQL 8.0**	**Notes**

2.1.14.1.12.5 System Variable Differences Between MariaDB 10.6 and MySQL 8.0

Contents

The following is a comparison of variables that either appear only in MariaDB 10.6 or MySQL 8.0, or have different default settings in MariaDB 10.6, and MySQL 8.0. The stable releases MariaDB 10.6.4 and MySQL 8.0.11, with only default plugins enabled, were used for the comparison. Note that MySQL 8 is an 'evergreen' release, so features may be added or removed in later releases.

For a more complete list of differences, see Incompatibilities and Feature Differences Between MariaDB 10.6 and MySQL 8.0 and Function Differences Between MariaDB 10.6 and MySQL 8.0

The most notable differences are that MariaDB includes, by default, the Aria storage engine (resulting in extra memory allocation), Galera Cluster, and has a different thread pool implementation. For this reason, a default implementation of MariaDB 10.6 will use more memory than MySQL 8.0. MariaDB 10.6 and MySQL 8.0 also have different GTID implementations.

MariaDB's extra memory usage can be handled with the following rules of thumb:

- If you are not using MyISAM and don't plan to use Aria:
  - Set key_buffer_size to something very low (16K) as it's not used.
  - Set aria_pagecache_buffer_size to what you think you need for handling internal tmp tables that didn't fit in memory.
  - Normally this is what before you had set for key_buffer_size (at least 1M).
- If you are using MyISAM and not planning to use Aria:
  - Set aria_pagecache_buffer_size to what you think you need for handling internal tmp tables that didn't fit in memory.
- If you are planning to use Aria, you should set aria_pagecache_buffer_size to something that fits a big part of your normal data + overflow temporary tables.

Comparison Table

Variable	MariaDB 10.6 Default	MySQL 8.0 Default	Notes
activate_all_roles_on_login	-	OFF	Determines whether to automatically activate roles on login.
alter_algorithm	DEFAULT	-	MariaDB 10.3 introduced new ALTER TABLE ALGORITHM clauses to avoid slow copies in certain instances. This variable allows setting this if no `ALGORITHM` clause is specified.
analyze_sample_percentage	100.0000	-	Percentage of rows from the table ANALYZE TABLE will sample to collect table statistics.
aria_*	*	-	The Aria storage engine is only available in MariaDB.
auto_generate_certs	-	ON	Whether to automatically generate SSL key and certificate files.
avoid_temporal_upgrade	-	OFF	Determines whether ALTER TABLE implicitly upgrades temporal columns.
back_log	Autosized	Autosized	MariaDB and MySQL have different autosizing algorithms.
binlog-annotate-row-events	ON	-	Introduced in MariaDB 5.3 for replicating between MariaDB 5.3 and MySQL/MariaDB 5.1.
binlog_commit_wait_count	0	-	For use in MariaDB's parallel replication.

binlog_commit_wait_usec	100000	-	For use in MariaDB's parallel replication.
binlog_error_action		ABORT_SERVER	MySQL-only variable for controlling what happens when the server cannot write to the binary log.
binlog_expire_logs_seconds	0	2592000	Sets the binary log expiration period in seconds
binlog_file_cache_size	16184	-	For setting the size of the file cache for the binary log.
binlog_format	MIXED	ROW	MariaDB and MySQL have differing binary log formats.
binlog_group_commit_sync_delay		0	MySQL-only variable for controlling the wait time before synchronizing the binary log file to disk.
binlog_group_commit_sync_no_delay_count		0	MySQL-only variable for setting the maximum number of transactions to wait for before aborting the current `binlog_group_commit_sync_delay` delay.
binlog_gtid_simple_recovery	-	ON	MySQL-only GTID variable. MariaDB's GTID implementation is different.
binlog_max_flush_queue_time	-	0	Specifies a timeout for reading transactions from the flush queue before continuing with group commit and syncing log to disk.
binlog_optimize_thread_scheduling	ON	-	For optimized kernel thread scheduling.
binlog_order_commits	-	ON	Determines whether transactions may be committed in parallel.
binlog_row_metadata	NO_LOG	MINIMAL	Determines the amount of table metadata added to the binary log with row-based logging.
binlog_row_value_options	-	(empty)	Permits an alternative binlog format for JSON document updates.
binlog_rows_query_log_events	-	OFF	MySQL-only variable for logging extra information in row-based logging.
binlog_transaction_dependency_history_size	-	25000	Maximum number of row hashes kept for looking up transactions that last modified a given row.
binlog_transaction_dependency_tracking	-	COMMIT_ORDER	For determining how to best use the slave's multithreaded applier.
block_encryption_mode	-	aes-128-ecb	MySQL-only variable for controlling the block encryption mode for block-based algorithms.
caching_sha2_password*	-	*	For use with MySQL's SHA-256 authentication with caching.
character_set_*	latin1 or utf8	utf8mb4	MySQL 8.0 defaults to the utf8mb4 character set.
check_constraint_checks	ON	-	Permits disabling constraint checks, for example when loading a table that violates some constraints that you plan to fix later.
check_proxy_users		OFF	MySQL-only variable for controlling whether the server performs proxy user mapping for authentication plugins.
collation_*	latin1_swedish_ci or utf8_general_ci	utf8mb4_0900_ai_ci	MySQL 8.0 defaults to the utf8mb4 character set.
column_compression_threshold	100	-	MariaDB supports Storage-engine Independent Column Compression.
column_compression_zlib_level	6	-	MariaDB supports Storage-engine Independent Column Compression.
column_compression_zlib_strategy	DEFAULT_STRATEGY	-	MariaDB supports Storage-engine Independent Column Compression.
column_compression_zlib_wrap	OFF	-	MariaDB supports Storage-engine Independent Column Compression.
cte_max_recursion_depth	-	1000	When MySQL 8.0 introduced common table expressions they used a different name. MariaDB's variable is called max_recursive_iterations.
date_format	%Y-%m-%d	-	Unused variable removed in MySQL 8.0
datetime_format	%Y-%m-%d	-	Unused variable removed in MySQL 8.0
deadlock_search_depth_long	15	-	The Aria storage engine is only available in MariaDB.
deadlock_search_depth_short	4	-	The Aria storage engine is only available in MariaDB.
deadlock_timeout_long	50000000	-	The Aria storage engine is only available in MariaDB.

deadlock_timeout_short	10000	-	The Aria storage engine is only available in MariaDB.
debug_no_thread_alarm	OFF	-	Disable system thread alarm calls, for debugging or testing.
default_authentication_plugin	-	caching_sha2_password	MySQL 8 introduced a new authentication plugin.
default_collation_for_utf8mb4	-	utf8mb4_0900_ai_ci	For internal use in MySQL 8 replication.
default_master_connection	empty	-	For use with MariaDB's multi-source replication.
default_password_lifetime	0	360	MariaDB defaults to password expiration off.
default_regex_flags	empty	-	For handling incompatibilities between MariaDB's PCRE and the old regex library.
default_tmp_storage_engine	empty	InnoDB	Default storage engine used for tables created with CREATE TEMPORARY TABLE.
disabled_storage_engines		empty	MySQL-only variable for disabling specific storage engines.
disconnect_on_expired_password	OFF	ON	MariaDB password expiration is off by default, and by default does not disconnect a client when a password has expired.
encrypt_binlog	OFF	-	MariaDB enables table and tablespace encryption.
encrypt_tmp_files	OFF	-	MariaDB enables table and tablespace encryption.
encrypt_tmp_disk_tables	OFF	-	MariaDB enables table and tablespace encryption.
end_markers_in_json	-	OFF	MySQL-only variable for adding end markers to JSON output.
enforce_gtid_consistency	-	OFF	MariaDB and MySQL have different GTID implementations.
enforce_storage_engine	none		Forces the use of a particular storage engine for new tables.
eq_range_index_dive_limit	0	200	Variable for tuning when the optimizer should switch from using index dives to index statistics for qualifying rows estimation.
event_scheduler	OFF	ON	MySQL enables the event scheduler by default.
expensive_subquery_limit	100	-	Used for determining expensive queries for optimization.
explicit_defaults_for_timestamp	OFF	ON	MySQL 8 disables the old timestamp behavior.
extra_max_connections	1	-	Introduced in the MariaDB 5.1 threadpool.
extra_port	0	-	Introduced in the MariaDB 5.1 threadpool.
group_concat_max_len	1048576	1024	MariaDB increases the maximum length for a GROUP_CONCAT() result from 1K to 1M.
gtid_binlog_pos	empty	-	MariaDB and MySQL have different GTID implementations.
gtid_binlog_state	empty	-	MariaDB and MySQL have different GTID implementations.
gtid_cleanup_batch_size	64	-	MariaDB and MySQL have different GTID implementations.
gtid_current_pos	empty	-	MariaDB and MySQL have different GTID implementations.
gtid_domain_id	0	-	MariaDB and MySQL have different GTID implementations.
gtid_executed	-	empty	MariaDB and MySQL have different GTID implementations.
gtid_executed_compression_period	-	1000	MariaDB and MySQL have different GTID implementations.
gtid_ignore_duplicates	OFF	-	MariaDB and MySQL have different GTID implementations.
gtid_mode	-	OFF	MariaDB and MySQL have different GTID implementations.
gtid_next	-	AUTOMATIC	MariaDB and MySQL have different GTID implementations.
gtid_owned	-	empty	MariaDB and MySQL have different GTID implementations.
gtid_pos_auto_engines	empty	-	MariaDB and MySQL have different GTID implementations.

gtid_purged	-	empty	MariaDB and MySQL have different GTID implementations.
gtid_seq_no	0	-	MariaDB and MySQL have different GTID implementations.
gtid_slave_pos	empty	-	MariaDB and MySQL have different GTID implementations.
gtid_strict_mode	OFF	-	MariaDB and MySQL have different GTID implementations.
have_crypt	YES	-	MySQL has removed the ENCRYPT function.
have_openssl			MariaDB's version indicates whether YaSSL or openssl was used. MySQL's is a synonym for `have_ssl`.
have_query_cache	YES	-	MySQL has removed the query cache.
have_statement_timeout	-		Whether MySQL's statement execution timeout feature is available.
have_symlink	YES	DISABLED	MySQL has removed symlink support.
histogram_generation_max_mem_size	-	20000000	Added when MySQL 8 introduced Histogram-based Statistics. MariaDB uses histogram_size
histogram_size	0	-	MariaDB introduced Histogram-based Statistics.
histogram_type	DOUBLE_PREC_HB	-	MariaDB introduced Histogram-based Statistics.
idle_readonly_transaction_timeout	0	-	Time in seconds that the server waits for idle read-only transactions.
idle_transaction_timeout	0	-	Time in seconds that the server waits for idle transactions.
idle_write_transaction_timeout	0	-	Time in seconds that the server waits for idle write transactions.
ignore_builtin_innodb	OFF	-	Ignored and removed in MySQL 8.
in_predicate_conversion_threshold	1000	-	Controls the Conversion of Big IN Predicates Into Subqueries optimization.
in_transaction	0	-	Set to `1` if you are in a transaction, and `0` if not.
information_schema_stats_expiry	-	86400	Time until MySQL Information Schema cached statistics expire.
innodb_adaptive_flushing_lwm	10.000000	10	Adaptive flushing is enabled when this this low water mark percentage of the redo log capacity is reached. MariaDB's variable is a double, MySQL's an integer.
innodb_adaptive_hash_index	OFF	ON	Defaulting to OFF is a performance improvement especially for DROP TABLE, TRUNCATE TABLE, ALTER TABLE, or DROP INDEX operations
innodb_adaptive_max_sleep_delay	-	150000	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_api_*	-	*	Specific to MySQL's memcached, removed in MariaDB 10.2.
innodb_autoinc_lock_mode	1	2	MariaDB has an extra mode, `3`, for skipping the rollback of connected transactions. MySQL defaults to row-based replication, so can safely use `2`.
innodb_buffer_pool_instances	-	1	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6 since the original reasons for introducing no longer apply.
innodb_checksum_algorithm	full_crc32	crc32	`fullcrc32` permits encryption to be supported over a SPATIAL INDEX, which crc32 does not support.
innodb_checksums	ON	-	Deprecated option removed in MySQL.
innodb_commit_concurrency	-	0	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_compression_*	*	-	Introduced with MariaDB's InnoDB compression.
innodb_concurrency_tickets	-	5000	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_deadlock_report	Full	-	How to report deadlocks.
innodb_dedicated_server	-	OFF	MySQL option that automatically configures various settings if the server is a dedicated InnoDB database server.
innodb_default_encryption_key_id	1	-	Default encryption key id used for table encryption. See Data at Rest Encryption.

innodb_defragment	*	-	MariaDB can defragment InnoDB tablespaces.
innodb_directories	-	(empty)	Used to search for tablespace files when moving or restoring a new location.
innodb_disallow_writes	OFF	-	Tell InnoDB to stop any writes to disk.
innodb_encrypt_*	1	-	See MariaDB's Data at Rest Encryption.
innodb_fatal_semaphore_wait_threshold	600	-	MariaDB's fatal semaphore timeout is configurable.
innodb_flush_method	O_DIRECT	fsync	MariaDB 10.6 InnoDB flushing method by default on Unix systems bypasses the file system cache for improved performance in most cases.
innodb_flush_neighbors	1	0	MySQL 8 by default now assumes the use of an SSD device.
innodb_force_primary_key	OFF	-	If set to 1 in MariaDB (0 is default) CREATE TABLEs without a primary or unique key where all keyparts are NOT NULL will not be accepted, and will return an error.
innodb_idle_flush_pct	100	-	Up to what percentage of dirty pages in MariaDB should be flushed when InnoDB finds it has spare resources to do so.
innodb_immediate_scrub_data_uncompressed	OFF	-	MariaDB has support for data scrubbing.
innodb_instant_alter_column_allowed	add_drop_reorder	-	See Instant ADD COLUMN for InnoDB.
innodb_locks_unsafe_for_binlog	OFF	-	Deprecated option in MariaDB for disabling gap locking for searches and index scans. Deprecated in MariaDB, use READ COMMITTED transaction isolation instead.
innodb_log_checksums	-	ON	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6, as there is no reason to allow checksums to be disabled on the redo log.
innodb_log_compressed_pages	-	ON	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6 as part of the InnoDB redo log performance improvements.
innodb_log_files_in_group	-	2	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6 as part of the InnoDB redo log performance improvements.
innodb_log_spin_*	-	*	MySQL variables for constraining CPU usage while waiting for flushed redo.
innodb_log_wait_for_flush_spin_hwm	-	*	MySQL variable for constraining CPU usage while waiting for flushed redo.
innodb_max_dirty_pages_pct	75	90	MySQL 8 increased the default to 90.
innodb_max_dirty_pages_pct_lwm	0	10	MySQL 8 increased the default to 10.
innodb_max_undo_log_size	10485760	1073741824	MariaDB 10.2 reduced the limit for when an undo tablespace is marked for truncation.
innodb_open_files	Autosized (2000)	Autosized (4000)	In most systems, autosized based on the table_open_cache setting, which differs between MariaDB and MySQL.
innodb_page_cleaners	-	1	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6 as the original reasons for for splitting the buffer pool have mostly gone away.
innodb_prefix_index_cluster_optimization	OFF	-	MariaDB includes the Facebook prefix index queries optimization.
innodb_print_ddl_logs	-	OFF	MySQL option for writing DDL logs to stderr.
innodb_read_only_compressed	ON	-	Whether to set ROW_FORMAT=COMPRESSED tables to read-only.
innodb_redo_log_encrypt	-	OFF	MySQL 8 has also now introduced redo log encryption, but used a different name. The equivalent option in MariaDB is innodb_encrypt_log.
innodb_replication_delay	-	0	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_spin_wait_delay	4	6	MariaDB changed the default from 6 to 4 based on extensive benchmarking.
innodb_stats_modified_counter	0	-	MariaDB option to control the calculation of new statistics.
innodb_stats_sample_pages	8	-	Deprecated MariaDB option for control over index distribution statistics.

innodb_stats_traditional	ON	-	Enabling gives a larger sample of pages for larger tables for the purposes of index statistics calculation.
innodb_sync_array_size	-	1	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_thread_concurrency	-	0	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_thread_sleep_delay	-	10000	Deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6.
innodb_undo_log_encrypt	-	OFF	MySQL option for encrypting undo logs residing in separate undo tablespaces.
innodb_undo_log_truncate	OFF	ON	MySQL 8 changes the default to ON, marking larger undo logs for truncation.
innodb_undo_tablespaces	0	2	Number of tablespace files used for dividing up the undo logs. MySQL 8 has deprecated this setting, and increased the default (and minimum) to 2.
innodb_use_atomic_writes	ON	-	Atomic writes are a faster alternative to innodb_doublewrite and MariaDB automatically detects when supporting SSD cards are used.
internal_tmp_disk_storage_engine	-	INNODB	MySQL uses this variable to set the storage engine for on-disk internal temporary tables.
internal_tmp_mem_storage_engine	-	TEMPTABLE	MySQL and MariaDB use different formats for temporary tables. In MariaDB, the aria_used_for_temp_tables performs a similar function.
join_buffer_space_limit	2097152	-	Maximum size in bytes of the query buffer. See block-based join algorithms.
join_cache_level	2	-	For determining the join algorithms. See block-based join algorithms
key_buffer_size	134217728	8388608	Size of the buffer for the index blocks used by MyISAM tables and shared for all threads.
key_cache_file_hash_size	512	-	Number of hash buckets for open and changed files.
key_cache_segments	0	-	The number of segments in a key cache. See Segmented Key Cache.
keyring_operations	-	ON	Whether MySQL 8's keyring operations are enabled.
last_gtid	-	empty	MariaDB and MySQL have different GTID implementations.
local_infile	ON	OFF	MySQL no longer supports LOAD DATA LOCAL by default.
lock_wait_timeout	86400	31536000	MariaDB has reduced the timeout for acquiring metadata locks.
log_bin	OFF	ON	MySQL 8 enables the binary log by default.
log_bin_compress	OFF	-	MariaDB setting for whether or not the binary log can be compressed.
log_bin_compress_min_len	256	-	Minimum length of sql statement (in statement mode) or record (in row mode) that can be compressed. See Compressing Events to Reduce Size of the Binary Log.
log_bin_use_v1_row_events	-	OFF	MySQL-only variable showing whether or not MySQL's version 2 binary logging format is being used.
log_disabled_statements	sp	-	Disable logging of certain statements to the general log.
log_error_services	-	log_filter_internal; log_sink_internal	Components to enable for MySQL error logging.
log_error_verbosity	-	3	MySQL variable for setting verbosity of error, warning, and note messages in the error log.
log_slave_updates	OFF	ON	MySQL 8 has by default enabled binary logging of updates a slave receives from a master.
log_slow_admin_statements	ON	OFF	MariaDB logs slow admin statements to the slow query log by default.
log_slow_disabled_statements	admin,call,slave,sp	-	Disable logging of certain statements to the slow query log.

log_slow_filter	admin, filesort, filesort_on_disk, full_join, full_scan, query_cache, query_cache_miss, tmp_table, tmp_table_on_disk	-	For slow query log filtering.
log_slow_rate_limit	1	-	Limits the number of queries logged to the slow query log.
log_slow_slave_statements	ON	OFF	MariaDB logs slow slave statements to the slow query log by default.
log_slow_verbosity	empty	-	Controls information to be added to the slow query log. See also Slow Query Log Extended Statistics.
log_statements_unsafe_for_binlog	-	ON	MySQL setting for controlling whether binlog warnings are written to the error log.
log_syslog*	platform-dependent	-	MySQL variables with settings for writing to syslog.
log_tc_size	24576	-	Size in bytes of the transaction coordinator log, defined in multiples of 4096.
log_throttle_queries_not_using_indexes	-	0	MySQL-only variable for limiting the number of statements without indexes written to the slow query log.
log_timestamps	-	UTC	MySQL-only variable controlling the timezone for certain logging conditions.
log_warnings	2	-	MySQL 8 has replaced with `log_error_verbosity`.
mandatory_roles	-	(empty)	MySQL variable for assigning roles to all users.
master_info_repository	-	TABLE	Whether slave logs master status and connection info to a table or a file.
max_allowed_packet	16M	64M	
max_error_count	64	1024	Specifies the maximum number of messages stored for display by SHOW ERRORS and SHOW WARNINGS statements.
max_execution_time	-	0	MySQL renamed the `max_statement_time` variable.
max_length_for_sort_data	64	1024	Used to decide which algorithm to choose when sorting rows. If the total size of the column data, not including columns that are part of the sort, is less than max_length_for_sort_data, then these are added to the sort key. This can speed up the sort as there's no need to re-read the same row again later. Setting the value too high can slow things down as there will be a higher disk activity for doing the sort.
max_long_data_size	16777216	-	Maximum size for parameter values sent with mysql_stmt_send_long_data(). Removed in MySQL 5.6.
max_password_errors	4294967295	-	Maximum number of failed connections attempts before no more are permitted.
max_points_in_geometry	-	65536	Maximum points_per_circle for MySQL's ST_Buffer_Strategy() function.
max_recursive_iterations	4294967295	-	Maximum number of iterations when executing recursive queries.
max_relay_log_size	1073741824	0	Can be set by session in MariaDB.
max_seeks_for_key	4294967295	4294967295 (32-bit) or 18446744073709547520 (64-bit)	The most key seeks required when searching with an index, regardless of the actual index cardinality. If this value is set lower than its default and maximum, indexes will tend to be preferred over table scans.
max_session_mem_used	9223372036854775807	-	Amount of memory a single user session is allowed to allocate.
max_statement_time	0	-	Maximum time in seconds that a query can execute before being aborted. MySQL used to have a variable of this name before renaming it `max_execution_time`.
max_tmp_tables	32	-	Unused variable removed in MySQL.
max_write_lock_count	4294967295	4294967295 (32-bit) or 18446744073709547520 (64-bit)	Read lock requests will be permitted for processing after this many write locks.
mrr_buffer_size	262144	-	Size of buffer to use when using multi-range read with range access. See Multi Range Read optimization.
myisam_block_size	1024	-	Block size used for MyISAM index pages.

myisam_recover_options	BACKUP,QUICK	OFF	MyISAM recovery mode.
myisam_sort_buffer_size	134216704	8388608	Size in bytes of the buffer allocated when creating or sorting indexes on a MyISAM table.
mysql_native_password_proxy_users	-	OFF	Whether MySQL's authentication plugin supports proxy users. I
mysql56_temporal_format	ON		Causes MariaDB to use the MySQL-5.6 low level formats for TIME, DATETIME and TIMESTAMP instead of the MariaDB 5.3+ version.
new	-	OFF	Used for backward-compatibility with MySQL 4.1, not present in MariaDB.
mysqlx+*	-	*	MySQL's X plugin related variables.
ngram_token_size	-	2	Sets the n-gram token size for MySQL's n-gram full-text parser.
offline_mode	-	OFF	MySQL settting for specifying whether the server should run in offline mode.
old_alter_table	DEFAULT	OFF	An alias for alter_algorithm.
old_mode	Empty string	-	Used for getting MariaDB to emulate behavior from an old version of MySQL or MariaDB. See OLD Mode.
old_passwords	OFF	-	MySQL 8 is no longer compatible with the old pre-MySQL 4.1 form of password hashing.
optimizer_selectivity_sampling_limit	100	-	Controls number of record samples to check condition selectivity.
optimizer_switch	See details		A series of flags for controlling the query optimizer. MariaDB has introduced a number of new settings.
optimizer_trace_*	-	*	MySQL has more settings for optimizer tracing.
optimizer_use_condition_selectivity	4	-	Controls which statistics can be used by the optimizer when looking for the best query execution plan.
original_commit_timestamp	-	*	Used by MySQL 8 for delaying replication.
parser_max_mem_size	-	4294967295 (32-bit) or 18446744073709547520 (64-bit)	MySQL variable for limiting memory available to the parser.
password_*	-	*	Controls reuse of previous passwords in MySQL.
performance_schema	OFF	ON	The Performance Schema is off by default in MariaDB.
performance_schema_*			Many performance schema variables are autoset in MySQL, and MySQL has a different version, with additional variables.
plugin_maturity	One less than the server maturity	-	Minimum acceptable plugin maturity.
progress_report_time	5	-	Time in seconds between sending progress reports to the client for time-consuming statements.
proxy_protocol_networks	(empty)	-	Enable proxy protocol for these source networks.
query_alloc_block_size	16384	8192	Size in bytes of the extra blocks allocated during query parsing and execution (after query_prealloc_size is used up).
query_cache_*	*	-	MySQL has removed the query cache.
query_prealloc_size	24576	8192	Size in bytes of the persistent buffer for query parsing and execution, allocated on connect and freed on disconnect.
range_optimizer_max_mem_size	-	8388608	MySQL-only variable setting a limit on the range optimizer's memory usage.
rbr_exec_mode	-	STRICT	MySQL-only variable for determining the handling of certain key errors.
read_binlog_speed_limit	0	-	Permits restricting the speed at which the slave reads the binlog from the master.
regexp_*	-	*	Memory and time limits for regular expression matching operations.
relay_log_info_repository	-	TABLE	MySQL-only variable determining whether the slave's position in the relay logs is written to a file or table.
replicate_annotate_row_events	ON	-	Tells the slave to reproduce annotate_rows_events received from the master in its own binary log.
replicate_do_db	empty string	-	See Dynamic Replication Variables.

replicate_do_table	empty string	-	See Dynamic Replication Variables.
replicate_events_marked_for_skip	replicate	-	See Selectively skipping replication of binlog events.
replicate_ignore_db	empty string	-	See Dynamic Replication Variables.
replicate_ignore_table	empty string	-	See Dynamic Replication Variables.
replicate_wild_do_table	empty string	-	See Dynamic Replication Variables.
replicate_wild_ignore_table	empty string	-	See Dynamic Replication Variables.
result_metadata	-	FULL	Determine whether the server returns result set metadata for connections where this is optional.
rowid_merge_buff_size	8388608	-	See Non-semi-join subquery optimizations.
rpl_read_size	-	8192	Minimum data in bytes read from the binary and relay log files.
rpl_semi_sync_*	-	-	MariaDB includes semisynchronous replication without the need to install a plugin.
rpl_stop_slave_timeout	-	31536000	Controls the time that STOP SLAVE waits before timing out.
s3_*	*	-	The S3 storage engine is only available in MariaDB.
schema_definition_cache	-	256	Limits the number of schema definition objects kept in the dictionary object cache.
secure_auth	ON	-	Removed in MySQL.
secure_timestamp	NO	-	MariaDB-only option permitting the restricting of direct setting of a session timestamp..
server_id_bits	-	server_id	MySQL-only variable for use in MySQL Cluster.
server_uuid	-	UUID	MySQL-only variable containing the UUID.
session_track_gtids	-	OFF	MySQL-only variables for tracking gtid changes. MariaDB and MySQL's gtid implementation is different.
sha256_password_proxy_users	-	OFF	MySQL-only variable determining whether the sha256_password plugin supports proxy users.
show_create_table_verbosity	-	OFF	Option to cause SHOW CREATE TABLE to display ROW_FORMAT in all cases.
show_old_temporals	-	OFF	MySQL-only variable for determining whether SHOW CREATE TABLE output should include comments for old format temporal columns.
skip_parallel_replication	OFF	-	See parallel replication.
skip_replication	OFF	-	See Selectively skipping replication of binlog events.
slave_allow_batching	-	OFF	MySQL-only replication variable.
slave_checkpoint_group	-	512	MySQL-only replication variable.
slave_checkpoint_period	-	300	MySQL-only replication variable.
slave_ddl_exec_mode	IDEMPOTENT	-	Modes for how replication of DDL events should be executed.
slave_domain_parallel_threads	0	-	For configuring parallel replication.
slave_net_timeout	3600	60	MySQL reduced the timeout to 60s.
slave_parallel_max_queued	131072	-	For configuring parallel replication.
slave_parallel_mode	optimistic	-	Controls what transactions are applied in parallel when using parallel_replication.
slave_parallel_threads	0	-	For configuring parallel replication.
slave_parallel_type	-	DATABASE	MySQL-only replication variable.
slave_pending_jobs_size_max	-	16777216	MySQL-only replication variable.
slave_preserve_commit_order	-	OFF	MySQL-only replication variable.
slave_rows_search_algorithms	-	INDEX_SCAN, HASH_SCAN	MySQL-only replication variable.
slave_run_triggers_for_rbr	NO		See Running triggers on the slave for Row-based events for a description and use-case for this setting.
slave_transaction_retry_errors	1213,1205	-	When an error occurs during a transaction on the slave, replication usually halts. By default, transactions that caused a deadlock or elapsed lock wait timeout will be retried. One can add other errors to the the list of errors that should be retried by adding a comma-separated list of error numbers to this variable.

slave_transaction_retry_interval	0	-	Interval in seconds for the slave SQL thread to retry a failed transaction due to a deadlock, elapsed lock wait timeout or an error listed in slave_transaction_retry_errors.
sort_buffer_size	2097152	262144	The default sort buffer allocated has been reduced in MySQL.
sql_if_exists	OFF	-	Adds an implicit IF EXISTS to ALTER, RENAME and DROP of TABLES, VIEWS, FUNCTIONS and PACKAGES
sql_mode	STRICT_TRANS_TABLES, ERROR_FOR_DIVISION_BY_ZERO, NO_AUTO_CREATE_USER, NO_ENGINE_SUBSTITUTION	ONLY_FULL_GROUP_BY, STRICT_TRANS_TABLES, NO_ZERO_IN_DATE, NO_ZERO_DATE, ERROR_FOR_DIVISION_BY_ZERO, NO_ENGINE_SUBSTITUTION	See SQL Mode.
ssl_fips_mode	-	OFF	Whether FIPS mode is enabled on the server side.
standard_compliant_cte	ON	-	See Common Table Expressions.
storage_engine	InnoDB	-	Alias for default_storage_engine, removed in MySQL.
strict_password_validation	ON	-	In MariaDB, when password validation plugins are enabled, reject passwords that cannot be validated (passwords specified as a hash).
stored_program_definition_cache	-	256	Limits the number of stored program definition objects kept in the dictionary object cache.
super_read_only	-	OFF	MySQL variable for prohibiting client updates from users with the SUPER privilege.
sync_binlog	0	1	MySQL synchronizes all actions to the binary log before they are committed.
sync_frm	1	-	.frm files have been removed in MySQL.
system_versioning_alter_history	ERROR	-	MariaDB has System-Versioned Tables
system_versioning_asof	DEFAULT	-	MariaDB has System-Versioned Tables
table_definition_cache	400	-1 (autosized)	Number of table definitions that can be cached.
table_open_cache_instances	8	16	Maximum number of table cache instances.
tablespace_definition_cache	-	256	Limits the number of tablespace definition objects kept in the dictionary object cache.
tcp_keepalive_interval	0	-	Interval, in seconds, between when successive keep-alive packets are sent if no acknowledgement is received.
tcp_keepalive_probes	0	-	Number of unacknowledged probes to send before considering the connection dead and notifying the application layer.
tcp_keepalive_time	0	-	Set the TCP_NODELAY option (disable Nagle's algorithm) on socket.
tcp_nodelay	1	-	Timeout, in milliseconds, with no activity until the first TCP keep-alive packet is sent.
temptable_max_ram	-	1GB	Limits the RAM used by MySQL's TempTable storage engine.
thread_cache_size	Autosized	-1 (autosized)	MariaDB uses an improved thread pool.
thread_concurrency	10	-	Removed in MySQL 5.7.
thread_pool_dedicated_listener	0	-	Better precision for the data in the Information Schema THREADPOOL_QUEUES Table.
thread_pool_exact_stats	0	-	Better precision for the data in the Information Schema THREADPOOL_QUEUES Table.
thread_pool_idle_timeout	60	-	See Using the Thread Pool.
thread_pool_max_threads	65536	-	See Using the Thread Pool.
thread_pool_min_threads	1	-	Windows-only. See Using the Thread Pool.
thread_pool_oversubscribe	3	-	See Using the Thread Pool.
thread_pool_prio_kickup	auto	-	See Using the Thread Pool.
thread_pool_priority	auto	-	See Using the Thread Pool.
thread_pool_size	Number of processors	16*	See Using the Thread Pool. *Only available in MySQL with a commercial plugin.

thread_pool_stall_limit	500	6*	See Using the Thread Pool. *Only available in MySQL with a commercial plugin.
thread_stack	299008	Varies	See Using the Thread Pool.
time_format	%H:%i:%s	-	Removed in MySQL.
timed_mutexes	OFF	-	Removed in MySQL.
tmp_disk_table_size	18446744073709551615	-	Max size for data for an internal temporary on-disk MyISAM or Aria table.
tmp_memory_table_size	16777216	-	Alias for tmp_table_size.
transaction_allow_batching	-	OFF	Variable for enabling batching of statements within the same transaction in MySQL Cluster.
transaction_isolation	-	REPEATABLE-READ	The MariaDB equivalent is tx_isolation.
transaction_read_only	-	OFF	The MariaDB equivalent is tx_read_only.
transaction_write_set_extraction	-	OFF	Unused MySQL-only variable.
tx_isolation	REPEATABLE-READ	-	The MySQL equivalent is transaction_isolation.
tx_read_only	OFF	-	The MySQL equivalent is transaction_read_only.
use_stat_tables	preferably_for_queries	-	Controls the use of engine-independent table statistics.
userstat	OFF	-	Whether to activate MariaDB's User Statistics implementation, not available in MySQL.
version_compile_zlib	-	*	Version of the zlib library compiled in.
version_malloc_library	*	-	Version of the used malloc library.
version_source_revision	Varies	-	Permits seeing exactly which version of the source was used for a build.
version_ssl_library	*	-	Version of the used TLS library.
windowing_high_use_precision	-	*	MySQL option allowing safety to be sacrificed for speed in window function calculations.
wsrep_*	*	-	Galera cluster is only available in MariaDB.
Variable	**MariaDB 10.6**	**MySQL 8.0**	**Notes**

2.1.14.1.12.6 System Variable Differences Between MariaDB 10.5 and MySQL 8.0

Contents

The following is a comparison of variables that either appear only in MariaDB 10.5 or MySQL 8.0, or have different default settings in MariaDB 10.5, and MySQL 8.0. The RC release MariaDB 10.5.3 and the stable MySQL 8.0.11, with only default plugins enabled, were used for the comparison. Note that MySQL 8 is an 'evergreen' release, so features may be added or removed in later releases.

For a more complete list of differences, see Incompatibilities and Feature Differences Between MariaDB 10.5 and MySQL 8.0 and Function Differences Between MariaDB 10.5 and MySQL 8.0

The most notable differences are that MariaDB includes, by default, the Aria storage engine (resulting in extra memory allocation), Galera Cluster, and has a different thread pool implementation. For this reason, a default implementation of MariaDB 10.5 will use more memory than MySQL 8.0. MariaDB 10.5 and MySQL 8.0 also have different GTID implementations.

MariaDB's extra memory usage can be handled with the following rules of thumb:

- If you are not using MyISAM and don't plan to use Aria:
  - Set key_buffer_size to something very low (16K) as it's not used.
  - Set aria_pagecache_buffer_size to what you think you need for handling internal tmp tables that didn't fit in memory.
  - Normally this is what before you had set for key_buffer_size (at least 1M).
- If you are using MyISAM and not planning to use Aria:
  - Set aria_pagecache_buffer_size to what you think you need for handling internal tmp tables that didn't fit in memory.
- If you are planning to use Aria, you should set aria_pagecache_buffer_size to something that fits a big part of your normal data + overflow temporary tables.

Comparison Table

Variable	MariaDB 10.5 Default	MySQL 8.0 Default	Notes
activate_all_roles_on_login	-	OFF	Determines whether to automatically activate roles on login.
alter_algorithm	DEFAULT	-	MariaDB 10.3 introduced new ALTER TABLE ALGORITHM clauses to avoid slow copies in certain instances. This variable allows setting this if no `ALGORITHM` clause is specified.
analyze_sample_percentage	100.0000	-	Percentage of rows from the table ANALYZE TABLE will sample to collect table statistics.
aria_*	*	-	The Aria storage engine is only available in MariaDB.
auto_generate_certs	-	ON	Whether to automatically generate SSL key and certificate files.
avoid_temporal_upgrade	-	OFF	Determines whether ALTER TABLE implicitly upgrades temporal columns.
back_log	Autosized	Autosized	MariaDB and MySQL have different autosizing algorithms.
binlog-annotate-row-events	ON	-	Introduced in MariaDB 5.3 for replicating between MariaDB 5.3 and MySQL/MariaDB 5.1.
binlog_commit_wait_count	0	-	For use in MariaDB's parallel replication.
binlog_commit_wait_usec	100000	-	For use in MariaDB's parallel replication.
binlog_error_action		ABORT_SERVER	MySQL-only variable for controlling what happens when the server cannot write to the binary log.
binlog_expire_logs_seconds	-	2592000	Sets the binary log expiration period in seconds
binlog_file_cache_size	16184	-	For setting the size of the file cache for the binary log.
binlog_format	MIXED	ROW	MariaDB and MySQL have differing binary log formats.
binlog_group_commit_sync_delay		0	MySQL-only variable for controlling the wait time before synchronizing the binary log file to disk.
binlog_group_commit_sync_no_delay_count		0	MySQL-only variable for setting the maximum number of transactions to wait for before aborting the current `binlog_group_commit_sync_delay` delay.
binlog_gtid_simple_recovery	-	ON	MySQL-only GTID variable. MariaDB's GTID implementation is different.
binlog_max_flush_queue_time	-	0	Specifies a timeout for reading transactions from the flush queue before continuing with group commit and syncing log to disk.
binlog_optimize_thread_scheduling	ON	-	For optimized kernel thread scheduling.
binlog_order_commits	-	ON	Determines whether transactions may be committed in parallel.
binlog_row_metadata	NO_LOG	MINIMAL	Determines the amount of table metadata added to the binary log with row-based logging.
binlog_row_value_options	-	(empty)	Permits an alternative binlog format for JSON document updates.
binlog_rows_query_log_events	-	OFF	MySQL-only variable for logging extra information in row-based logging.
binlog_transaction_dependency_history_size	-	25000	Maximum number of row hashes kept for looking up transactions that last modified a given row.
binlog_transaction_dependency_tracking	-	COMMIT_ORDER	For determining how to best use the slave's multithreaded applier.
block_encryption_mode	-	aes-128-ecb	MySQL-only variable for controlling the block encryption mode for block-based algorithms.
caching_sha2_password*	-	*	For use with MySQL's SHA-256 authentication with caching.
character_set_*	latin1 or utf8	utf8mb4	MySQL 8.0 defaults to the utf8mb4 character set.
check_constraint_checks	ON	-	Permits disabling constraint checks, for example when loading a table that violates some constraints that you plan to fix later.

check_proxy_users		OFF	MySQL-only variable for controlling whether the server performs proxy user mapping for authentication plugins.
collation_*	latin1_swedish_ci or utf8_general_ci	utf8mb4_0900_ai_ci	MySQL 8.0 defaults to the utf8mb4 character set.
column_compression_threshold	100	-	MariaDB supports Storage-engine Independent Column Compression.
column_compression_zlib_level	6	-	MariaDB supports Storage-engine Independent Column Compression.
column_compression_zlib_strategy	DEFAULT_STRATEGY	-	MariaDB supports Storage-engine Independent Column Compression.
column_compression_zlib_wrap	OFF	-	MariaDB supports Storage-engine Independent Column Compression.
cte_max_recursion_depth	-	1000	When MySQL 8.0 introduced common table expressions they used a different name. MariaDB's variable is called max_recursive_iterations.
date_format	%Y-%m-%d	-	Unused variable removed in MySQL 8.0
datetime_format	%Y-%m-%d	-	Unused variable removed in MySQL 8.0
deadlock_search_depth_long	15	-	The Aria storage engine is only available in MariaDB.
deadlock_search_depth_short	4	-	The Aria storage engine is only available in MariaDB.
deadlock_timeout_long	50000000	-	The Aria storage engine is only available in MariaDB.
deadlock_timeout_short	10000	-	The Aria storage engine is only available in MariaDB.
debug_no_thread_alarm	OFF	-	Disable system thread alarm calls, for debugging or testing.
default_authentication_plugin	-	caching_sha2_password	MySQL 8 introduced a new authentication plugin.
default_collation_for_utf8mb4	-	utf8mb4_0900_ai_ci	For internal use in MySQL 8 replication.
default_master_connection	empty	-	For use with MariaDB's multi-source replication.
default_password_lifetime	0	360	MariaDB defaults to password expiration off.
default_regex_flags	empty	-	For handling incompatibilities between MariaDB's PCRE and the old regex library.
default_tmp_storage_engine	empty	InnoDB	Default storage engine used for tables created with CREATE TEMPORARY TABLE.
disabled_storage_engines		empty	MySQL-only variable for disabling specific storage engines.
disconnect_on_expired_password	OFF	ON	MariaDB password expiration is off by default, and by default does not disconnect a client when a password has expired.
encrypt_binlog	OFF	-	MariaDB enables table and tablespace encryption.
encrypt_tmp_files	OFF	-	MariaDB enables table and tablespace encryption.
encrypt_tmp_disk_tables	OFF	-	MariaDB enables table and tablespace encryption.
end_markers_in_json	-	OFF	MySQL-only variable for adding end markers to JSON output.
enforce_gtid_consistency	-	OFF	MariaDB and MySQL have different GTID implementations.
enforce_storage_engine	none		Forces the use of a particular storage engine for new tables.
eq_range_index_dive_limit	0	200	Variable for tuning when the optimizer should switch from using index dives to index statistics for qualifying rows estimation.
event_scheduler	OFF	ON	MySQL enables the event scheduler by default.
expensive_subquery_limit	100	-	Used for determining expensive queries for optimization.
explicit_defaults_for_timestamp	OFF	ON	MySQL 8 disables the old timestamp behavior.
extra_max_connections	1	-	Introduced in the MariaDB 5.1 threadpool.
extra_port	0	-	Introduced in the MariaDB 5.1 threadpool.

group_concat_max_len	1048576	1024	MariaDB increases the maximum length for a GROUP_CONCAT() result from 1K to 1M.
gtid_binlog_pos	empty	-	MariaDB and MySQL have different GTID implementations.
gtid_binlog_state	empty	-	MariaDB and MySQL have different GTID implementations.
gtid_cleanup_batch_size	64	-	MariaDB and MySQL have different GTID implementations.
gtid_current_pos	empty	-	MariaDB and MySQL have different GTID implementations.
gtid_domain_id	0	-	MariaDB and MySQL have different GTID implementations.
gtid_executed	-	empty	MariaDB and MySQL have different GTID implementations.
gtid_executed_compression_period	-	1000	MariaDB and MySQL have different GTID implementations.
gtid_ignore_duplicates	OFF	-	MariaDB and MySQL have different GTID implementations.
gtid_mode	-	OFF	MariaDB and MySQL have different GTID implementations.
gtid_next	-	AUTOMATIC	MariaDB and MySQL have different GTID implementations.
gtid_owned	-	empty	MariaDB and MySQL have different GTID implementations.
gtid_pos_auto_engines	empty	-	MariaDB and MySQL have different GTID implementations.
gtid_purged	-	empty	MariaDB and MySQL have different GTID implementations.
gtid_seq_no	0	-	MariaDB and MySQL have different GTID implementations.
gtid_slave_pos	empty	-	MariaDB and MySQL have different GTID implementations.
gtid_strict_mode	OFF	-	MariaDB and MySQL have different GTID implementations.
have_crypt	YES	-	MySQL has removed the ENCRYPT function.
have_openssl			MariaDB's version indicates whether YaSSL or openssl was used. MySQL's is a synonym for `have_ssl`.
have_query_cache	YES	-	MySQL has removed the query cache.
have_statement_timeout	-		Whether MySQL's statement execution timeout feature is available.
have_symlink	YES	DISABLED	MySQL has removed symlink support.
histogram_generation_max_mem_size	-	20000000	Added when MySQL 8 introduced Histogram-based Statistics. MariaDB uses histogram_size
histogram_size	0	-	MariaDB introduced Histogram-based Statistics.
histogram_type	DOUBLE_PREC_HB	-	MariaDB introduced Histogram-based Statistics.
idle_readonly_transaction_timeout	0	-	Time in seconds that the server waits for idle read-only transactions.
idle_transaction_timeout	0	-	Time in seconds that the server waits for idle transactions.
idle_write_transaction_timeout	0	-	Time in seconds that the server waits for idle write transactions.
ignore_builtin_innodb	OFF	-	Ignored and removed in MySQL 8.
in_predicate_conversion_threshold	1000	-	Controls the Conversion of Big IN Predicates Into Subqueries optimization.
in_transaction	0	-	Set to `1` if you are in a transaction, and `0` if not.
information_schema_stats_expiry	-	86400	Time until MySQL Information Schema cached statistics expire.
innodb_adaptive_flushing_lwm	10.000000	10	Adaptive flushing is enabled when this this low water mark percentage of the redo log capacity is reached. MariaDB's variable is a double, MySQL's an integer.
innodb_adaptive_hash_index	OFF	ON	Defaulting to OFF is a performance improvement especially for DROP TABLE, TRUNCATE TABLE, ALTER TABLE, or DROP INDEX operations

innodb_api_*	-	*	Specific to MySQL's memcached, removed in MariaDB 10.2.
innodb_autoinc_lock_mode	1	2	MariaDB has an extra mode, `3`, for skipping the rollback of connected transactions. MySQL defaults to row-based replication, so can safely use `2`.
innodb_background_*	*	-	Earlier versions of MariaDB had support for background data scrubbing.
innodb_checksum_algorithm	full_crc32	crc32	`fullcrc32` permits encryption to be supported over a SPATIAL INDEX, which crc32 does not support.
innodb_checksums	ON	-	Deprecated option removed in MySQL.
innodb_compression_*	*	-	Introduced with MariaDB's InnoDB compression.
innodb_dedicated_server	-	OFF	MySQL option that automatically configures various settings if the server is a dedicated InnoDB database server.
innodb_default_encryption_key_id	1	-	Default encryption key id used for table encryption. See Data at Rest Encryption.
innodb_defragment	*	-	MariaDB can defragment InnoDB tablespaces.
innodb_directories	-	(empty)	Used to search for tablespace files when moving or restoring a new location.
innodb_disallow_writes	OFF	-	Tell InnoDB to stop any writes to disk.
innodb_encrypt_*	1	-	See MariaDB's Data at Rest Encryption.
innodb_fatal_semaphore_wait_threshold	600	-	MariaDB's fatal semaphore timeout is configurable.
innodb_file_format	(empty)	-	MariaDB 10.4 has restored this unused, deprecated variable for compatibility reasons.
innodb_flush_neighbors	1	0	MySQL 8 by default now assumes the use of an SSD device.
innodb_force_primary_key	OFF	-	If set to 1 in MariaDB (0 is default) CREATE TABLEs without a primary or unique key where all keyparts are NOT NULL will not be accepted, and will return an error.
innodb_idle_flush_pct	100	-	Up to what percentage of dirty pages in MariaDB should be flushed when InnoDB finds it has spare resources to do so.
innodb_immediate_scrub_data_uncompressed	OFF	-	MariaDB has support for data scrubbing.
innodb_instant_alter_column_allowed	add_drop_reorder	-	See Instant ADD COLUMN for InnoDB.
innodb_large_prefix	(empty)	-	MariaDB 10.4 has restored this unused, deprecated variable for compatibility reasons.
innodb_lock_schedule_algorithm	VATS	-	MariaDB has an improved algorithm for deciding which of the waiting transactions should be granted a lock once it has been released.
innodb_locks_unsafe_for_binlog	OFF	-	Deprecated option in MariaDB for disabling gap locking for searches and index scans. Deprecated in MariaDB, use READ COMMITTED transaction isolation instead.
innodb_log_optimize_ddl	OFF	-	Deprecated and ignored in MariaDB. Previously determined whether redo logging should be reduced when natively creating indexes or rebuilding tables.
innodb_log_spin_*	-	*	MySQL variables for constraining CPU usage while waiting for flushed redo.
innodb_log_wait_for_flush_spin_hwm	-	*	MySQL variable for constraining CPU usage while waiting for flushed redo.
innodb_max_dirty_pages_pct	75	90	MySQL 8 increased the default to 90.
innodb_max_dirty_pages_pct_lwm	0	10	MySQL 8 increased the default to 10.
innodb_max_undo_log_size	10485760	1073741824	MariaDB 10.2 reduced the limit for when an undo tablespace is marked for truncation.
innodb_open_files	Autosized (2000)	Autosized (4000)	In most systems, autosized based on the table_open_cache setting, which differs between MariaDB and MySQL.
innodb_prefix_index_cluster_optimization	OFF	-	MariaDB includes the Facebook prefix index queries optimization.
innodb_print_ddl_logs	-	OFF	MySQL option for writing DDL logs to stderr.

innodb_redo_log_encrypt	-	OFF	MySQL 8 has also now introduced redo log encryption, but used a different name. The equivalent option in MariaDB is innodb_encrypt_log.
innodb_scrub_*	*	-	Earlier version of MariaDB included options to scrub the redo log.
innodb_spin_wait_delay	4	6	MariaDB changed the default from 6 to 4 based on extensive benchmarking.
innodb_stats_modified_counter	0	-	MariaDB option to control the calculation of new statistics.
innodb_stats_sample_pages	8	-	Deprecated MariaDB option for control over index distribution statistics.
innodb_stats_traditional	ON	-	Enabling gives a larger sample of pages for larger tables for the purposes of index statistics calculation.
innodb_undo_log_encrypt	-	OFF	MySQL option for encrypting undo logs residing in separate undo tablespaces.
innodb_undo_log_truncate	OFF	ON	MySQL 8 changes the default to ON, marking larger undo logs for truncation.
innodb_undo_logs	128	-	Removed in MySQL 8.
innodb_undo_tablespaces	0	2	Number of tablespace files used for dividing up the undo logs. MySQL 8 has deprecated this setting, and increased the default (and minimum) to 2.
innodb_use_atomic_writes	ON	-	Atomic writes are a faster alternative to innodb_doublewrite and MariaDB automatically detects when supporting SSD cards are used.
internal_tmp_disk_storage_engine	-	INNODB	MySQL uses this variable to set the storage engine for on-disk internal temporary tables.
internal_tmp_mem_storage_engine	-	TEMPTABLE	MySQL and MariaDB use different formats for temporary tables. In MariaDB, the aria_used_for_temp_tables performs a similar function.
join_buffer_space_limit	2097152	-	Maximum size in bytes of the query buffer. See block-based join algorithms.
join_cache_level	2	-	For determining the join algorithms. See block-based join algorithms
key_buffer_size	134217728	8388608	Size of the buffer for the index blocks used by MyISAM tables and shared for all threads.
key_cache_file_hash_size	512	-	Number of hash buckets for open and changed files.
key_cache_segments	0	-	The number of segments in a key cache. See Segmented Key Cache.
keyring_operations	-	ON	Whether MySQL 8's keyring operations are enabled.
last_gtid	-	empty	MariaDB and MySQL have different GTID implementations.
local_infile	ON	OFF	MySQL no longer supports LOAD DATA LOCAL by default.
lock_wait_timeout	86400	31536000	MariaDB has reduced the timeout for acquiring metadata locks.
log_bin	OFF	ON	MySQL 8 enables the binary log by default.
log_bin_compress	OFF	-	MariaDB setting for whether or not the binary log can be compressed.
log_bin_compress_min_len	256	-	Minimum length of sql statement (in statement mode) or record (in row mode) that can be compressed. See Compressing Events to Reduce Size of the Binary Log.
log_bin_use_v1_row_events	-	OFF	MySQL-only variable showing whether or not MySQL's version 2 binary logging format is being used.
log_disabled_statements	sp	-	Disable logging of certain statements to the general log.
log_error_services	-	log_filter_internal; log_sink_internal	Components to enable for MySQL error logging.
log_error_verbosity	-	3	MySQL variable for setting verbosity of error, warning, and note messages in the error log.
log_slave_updates	OFF	ON	MySQL 8 has by default enabled binary logging of updates a slave receives from a master.

log_slow_admin_statements	ON	OFF	MariaDB logs slow admin statements to the slow query log by default.
log_slow_disabled_statements	admin,call,slave,sp	-	Disable logging of certain statements to the slow query log.
log_slow_filter	admin, filesort, filesort_on_disk, full_join, full_scan, query_cache, query_cache_miss, tmp_table, tmp_table_on_disk	-	For slow query log filtering.
log_slow_rate_limit	1	-	Limits the number of queries logged to the slow query log.
log_slow_slave_statements	ON	OFF	MariaDB logs slow slave statements to the slow query log by default.
log_slow_verbosity	empty	-	Controls information to be added to the slow query log. See also Slow Query Log Extended Statistics.
log_statements_unsafe_for_binlog	-	ON	MySQL setting for controlling whether binlog warnings are written to the error log.
log_syslog*	platform-dependent	-	MySQL variables with settings for writing to syslog.
log_tc_size	24576	-	Size in bytes of the transaction coordinator log, defined in multiples of 4096.
log_throttle_queries_not_using_indexes	-	0	MySQL-only variable for limiting the number of statements without indexes written to the slow query log.
log_timestamps	-	UTC	MySQL-only variable controlling the timezone for certain logging conditions.
log_warnings	2	-	MySQL 8 has replaced with `log_error_verbosity`.
mandatory_roles	-	(empty)	MySQL variable for assigning roles to all users.
master_info_repository	-	TABLE	Whether slave logs master status and connection info to a table or a file.
max_allowed_packet	16M	64M	
max_error_count	64	1024	Specifies the maximum number of messages stored for display by SHOW ERRORS and SHOW WARNINGS statements.
max_execution_time	-	0	MySQL renamed the `max_statement_time` variable.
max_length_for_sort_data	64	1024	Used to decide which algorithm to choose when sorting rows. If the total size of the column data, not including columns that are part of the sort, is less than max_length_for_sort_data, then these are added to the sort key. This can speed up the sort as there's no need to re-read the same row again later. Setting the value too high can slow things down as there will be a higher disk activity for doing the sort.
max_long_data_size	16777216	-	Maximum size for parameter values sent with mysql_stmt_send_long_data(). Removed in MySQL 5.6.
max_password_errors	4294967295	-	Maximum number of failed connections attempts before no more are permitted.
max_points_in_geometry	-	65536	Maximum points_per_circle for MySQL's ST_Buffer_Strategy() function.
max_recursive_iterations	4294967295	-	Maximum number of iterations when executing recursive queries.
max_relay_log_size	1073741824	0	Can be set by session in MariaDB.
max_seeks_for_key	4294967295	4294967295 (32-bit) or 18446744073709547520 (64-bit)	The most key seeks required when searching with an index, regardless of the actual index cardinality. If this value is set lower than its default and maximum, indexes will tend to be preferred over table scans.
max_session_mem_used	9223372036854775807	-	Amount of memory a single user session is allowed to allocate.
max_statement_time	0	-	Maximum time in seconds that a query can execute before being aborted. MySQL used to have a variable of this name before renaming it `max_execution_time`.
max_tmp_tables	32	-	Unused variable removed in MySQL.
max_write_lock_count	4294967295	4294967295 (32-bit) or 18446744073709547520 (64-bit)	Read lock requests will be permitted for processing after this many write locks.

mrr_buffer_size	262144	-	Size of buffer to use when using multi-range read with range access. See Multi Range Read optimization.
myisam_block_size	1024	-	Block size used for MyISAM index pages.
myisam_recover_options	BACKUP,QUICK	OFF	MyISAM recovery mode.
myisam_sort_buffer_size	134216704	8388608	Size in bytes of the buffer allocated when creating or sorting indexes on a MyISAM table.
mysql_native_password_proxy_users	-	OFF	Whether MySQL's authentication plugin supports proxy users. I
mysql56_temporal_format	ON		Causes MariaDB to use the MySQL-5.6 low level formats for TIME, DATETIME and TIMESTAMP instead of the MariaDB 5.3+ version.
new	-	OFF	Used for backward-compatibility with MySQL 4.1, not present in MariaDB.
mysqlx+*	-	*	MySQL's X plugin related variables.
ngram_token_size	-	2	Sets the n-gram token size for MySQL's n-gram full-text parser.
offline_mode	-	OFF	MySQL settting for specifying whether the server should run in offline mode.
old_alter_table	DEFAULT	OFF	An alias for alter_algorithm.
old_mode	Empty string	-	Used for getting MariaDB to emulate behavior from an old version of MySQL or MariaDB. See OLD Mode.
old_passwords	OFF	-	MySQL 8 is no longer compatible with the old pre-MySQL 4.1 form of password hashing.
optimizer_selectivity_sampling_limit	100	-	Controls number of record samples to check condition selectivity.
optimizer_switch	See details		A series of flags for controlling the query optimizer. MariaDB has introduced a number of new settings.
optimizer_trace_*	-	*	MySQL has more settings for optimizer tracing.
optimizer_use_condition_selectivity	4	-	Controls which statistics can be used by the optimizer when looking for the best query execution plan.
original_commit_timestamp	-	*	Used by MySQL 8 for delaying replication.
parser_max_mem_size	-	4294967295 (32-bit) or 18446744073709547520 (64-bit)	MySQL variable for limiting memory available to the parser.
password_*	-	*	Controls reuse of previous passwords in MySQL.
performance_schema	OFF	ON	The Performance Schema is off by default in MariaDB.
performance_schema_*			Many performance schema variables are autoset in MySQL, and MySQL has a different version, with additional variables.
plugin_maturity	One less than the server maturity	-	Minimum acceptable plugin maturity.
progress_report_time	5	-	Time in seconds between sending progress reports to the client for time-consuming statements.
proxy_protocol_networks	(empty)	-	Enable proxy protocol for these source networks.
query_alloc_block_size	16384	8192	Size in bytes of the extra blocks allocated during query parsing and execution (after query_prealloc_size is used up).
query_cache_*	*	-	MySQL has removed the query cache.
query_prealloc_size	24576	8192	Size in bytes of the persistent buffer for query parsing and execution, allocated on connect and freed on disconnect.
range_optimizer_max_mem_size	-	8388608	MySQL-only variable setting a limit on the range optimizer's memory usage.
rbr_exec_mode	-	STRICT	MySQL-only variable for determining the handling of certain key errors.
read_binlog_speed_limit	0	-	Permits restricting the speed at which the slave reads the binlog from the master.
regexp_*	-	*	Memory and time limits for regular expression matching operations.
relay_log_info_repository	-	TABLE	MySQL-only variable determining whether the slave's position in the relay logs is written to a file or table.

replicate_annotate_row_events	ON	-	Tells the slave to reproduce annotate_rows_events received from the master in its own binary log.
replicate_do_db	empty string	-	See Dynamic Replication Variables.
replicate_do_table	empty string	-	See Dynamic Replication Variables.
replicate_events_marked_for_skip	replicate	-	See Selectively skipping replication of binlog events.
replicate_ignore_db	empty string	-	See Dynamic Replication Variables.
replicate_ignore_table	empty string	-	See Dynamic Replication Variables.
replicate_wild_do_table	empty string	-	See Dynamic Replication Variables.
replicate_wild_ignore_table	empty string	-	See Dynamic Replication Variables.
result_metadata	-	FULL	Determine whether the server returns result set metadata for connections where this is optional.
rowid_merge_buff_size	8388608	-	See Non-semi-join subquery optimizations.
rpl_read_size	-	8192	Minimum data in bytes read from the binary and relay log files.
rpl_semi_sync_*	-	-	MariaDB includes semisynchronous replication without the need to install a plugin.
rpl_stop_slave_timeout	-	31536000	Controls the time that STOP SLAVE waits before timing out.
s3_*	*	-	The S3 storage engine is only available in MariaDB.
schema_definition_cache	-	256	Limits the number of schema definition objects kept in the dictionary object cache.
secure_auth	ON	-	Removed in MySQL.
secure_timestamp	NO	-	MariaDB-only option permitting the restricting of direct setting of a session timestamp..
server_id_bits	-	server_id	MySQL-only variable for use in MySQL Cluster.
server_uuid	-	UUID	MySQL-only variable containing the UUID.
session_track_gtids	-	OFF	MySQL-only variables for tracking gtid changes. MariaDB and MySQL's gtid implementation is different.
sha256_password_proxy_users	-	OFF	MySQL-only variable determining whether the sha256_password plugin supports proxy users.
show_create_table_verbosity	-	OFF	Option to cause SHOW CREATE TABLE to display ROW_FORMAT in all cases.
show_old_temporals	-	OFF	MySQL-only variable for determining whether SHOW CREATE TABLE output should include comments for old format temporal columns.
skip_parallel_replication	OFF	-	See parallel replication.
skip_replication	OFF	-	See Selectively skipping replication of binlog events.
slave_allow_batching	-	OFF	MySQL-only replication variable.
slave_checkpoint_group	-	512	MySQL-only replication variable.
slave_checkpoint_period	-	300	MySQL-only replication variable.
slave_ddl_exec_mode	IDEMPOTENT	-	Modes for how replication of DDL events should be executed.
slave_domain_parallel_threads	0	-	For configuring parallel replication.
slave_net_timeout	3600	60	MySQL reduced the timeout to 60s.
slave_parallel_max_queued	131072	-	For configuring parallel replication.
slave_parallel_mode	optimistic	-	Controls what transactions are applied in parallel when using parallel_replication.
slave_parallel_threads	0	-	For configuring parallel replication.
slave_parallel_type	-	DATABASE	MySQL-only replication variable.
slave_pending_jobs_size_max	-	16777216	MySQL-only replication variable.
slave_preserve_commit_order	-	OFF	MySQL-only replication variable.
slave_rows_search_algorithms	-	INDEX_SCAN, HASH_SCAN	MySQL-only replication variable.
slave_run_triggers_for_rbr	NO		See Running triggers on the slave for Row-based events for a description and use-case for this setting.

slave_transaction_retry_errors	1213,1205	-	When an error occurs during a transaction on the slave, replication usually halts. By default, transactions that caused a deadlock or elapsed lock wait timeout will be retried. One can add other errors to the the list of errors that should be retried by adding a comma-separated list of error numbers to this variable.
slave_transaction_retry_interval	0	-	Interval in seconds for the slave SQL thread to retry a failed transaction due to a deadlock, elapsed lock wait timeout or an error listed in slave_transaction_retry_errors.
sort_buffer_size	2097152	262144	The default sort buffer allocated has been reduced in MySQL.
sql_if_exists	OFF	-	Adds an implicit IF EXISTS to ALTER, RENAME and DROP of TABLES, VIEWS, FUNCTIONS and PACKAGES
sql_mode	STRICT_TRANS_TABLES, ERROR_FOR_DIVISION_BY_ZERO, NO_AUTO_CREATE_USER, NO_ENGINE_SUBSTITUTION	ONLY_FULL_GROUP_BY, STRICT_TRANS_TABLES, NO_ZERO_IN_DATE, NO_ZERO_DATE, ERROR_FOR_DIVISION_BY_ZERO, NO_ENGINE_SUBSTITUTION	See SQL Mode.
ssl_fips_mode	-	OFF	Whether FIPS mode is enabled on the server side.
standard_compliant_cte	ON	-	See Common Table Expressions.
storage_engine	InnoDB	-	Alias for default_storage_engine, removed in MySQL.
strict_password_validation	ON	-	In MariaDB, when password validation plugins are enabled, reject passwords that cannot be validated (passwords specified as a hash).
stored_program_definition_cache	-	256	Limits the number of stored program definition objects kept in the dictionary object cache.
super_read_only	-	OFF	MySQL variable for prohibiting client updates from users with the SUPER privilege.
sync_binlog	0	1	MySQL synchronizes all actions to the binary log before they are committed.
sync_frm	1	-	.frm files have been removed in MySQL.
system_versioning_alter_history	ERROR	-	MariaDB has System-Versioned Tables
system_versioning_asof	DEFAULT	-	MariaDB has System-Versioned Tables
table_definition_cache	400	-1 (autosized)	Number of table definitions that can be cached.
table_open_cache_instances	8	16	Maximum number of table cache instances.
tablespace_definition_cache	-	256	Limits the number of tablespace definition objects kept in the dictionary object cache.
tcp_keepalive_interval	0	-	Interval, in seconds, between when successive keep-alive packets are sent if no acknowledgement is received.
tcp_keepalive_probes	0	-	Number of unacknowledged probes to send before considering the connection dead and notifying the application layer.
tcp_keepalive_time	0	-	Set the TCP_NODELAY option (disable Nagle's algorithm) on socket.
tcp_nodelay	1	-	Timeout, in milliseconds, with no activity until the first TCP keep-alive packet is sent.
temptable_max_ram	-	1GB	Limits the RAM used by MySQL's TempTable storage engine.
thread_cache_size	Autosized	-1 (autosized)	MariaDB uses an improved thread pool.
thread_concurrency	10	-	Removed in MySQL 5.7.
thread_pool_dedicated_listener	0	-	Better precision for the data in the Information Schema THREADPOOL_QUEUES Table.
thread_pool_exact_stats	0	-	Better precision for the data in the Information Schema THREADPOOL_QUEUES Table.
thread_pool_idle_timeout	60	-	See Using the Thread Pool.
thread_pool_max_threads	65536	-	See Using the Thread Pool.
thread_pool_min_threads	1	-	Windows-only. See Using the Thread Pool.

thread_pool_oversubscribe	3	-	See Using the Thread Pool.
thread_pool_prio_kickup	auto	-	See Using the Thread Pool.
thread_pool_priority	auto	-	See Using the Thread Pool.
thread_pool_size	Number of processors	16*	See Using the Thread Pool. *Only available in MySQL with a commercial plugin.
thread_pool_stall_limit	500	6*	See Using the Thread Pool. *Only available in MySQL with a commercial plugin.
thread_stack	299008	Varies	See Using the Thread Pool.
time_format	%H:%i:%s	-	Removed in MySQL.
timed_mutexes	OFF	-	Removed in MySQL.
tmp_disk_table_size	18446744073709551615	-	Max size for data for an internal temporary on-disk MyISAM or Aria table.
tmp_memory_table_size	16777216	-	Alias for tmp_table_size.
transaction_allow_batching	-	OFF	Variable for enabling batching of statements within the same transaction in MySQL Cluster.
transaction_isolation	-	REPEATABLE-READ	The MariaDB equivalent is tx_isolation.
transaction_read_only	-	OFF	The MariaDB equivalent is tx_read_only.
transaction_write_set_extraction	-	OFF	Unused MySQL-only variable.
tx_isolation	REPEATABLE-READ	-	The MySQL equivalent is transaction_isolation.
tx_read_only	OFF	-	The MySQL equivalent is transaction_read_only.
use_stat_tables	preferably_for_queries	-	Controls the use of engine-independent table statistics.
userstat	OFF	-	Whether to activate MariaDB's User Statistics implementation, not available in MySQL.
version_compile_zlib	-	*	Version of the zlib library compiled in.
version_malloc_library	*	-	Version of the used malloc library.
version_source_revision	Varies	-	Permits seeing exactly which version of the source was used for a build.
version_ssl_library	*	-	Version of the used TLS library.
windowing_high_use_precision	-	*	MySQL option allowing safety to be sacrificed for speed in window function calculations.
wsrep_*	*	-	Galera cluster is only available in MariaDB.
Variable	**MariaDB 10.5**	**MySQL 8.0**	**Notes**

2.1.14.1.12.7 System Variable Differences Between MariaDB 10.4 and MySQL 8.0

Contents

The following is a comparison of variables that either appear only in MariaDB 10.4 or MySQL 8.0, or have different default settings in MariaDB 10.4, and MySQL 8.0. The stable releases MariaDB 10.4.6 and MySQL 8.0.11, with only default plugins enabled, were used for the comparison. Note that MySQL 8 is an 'evergreen' release, so features may be added or removed in later releases.

For a more complete list of differences, see Incompatibilities and Feature Differences Between MariaDB 10.4 and MySQL 8.0 and Function Differences Between MariaDB 10.4 and MySQL 8.0

The most notable differences are that MariaDB includes, by default, the Aria storage engine (resulting in extra memory allocation), Galera Cluster, and has a different thread pool implementation. For this reason, a default implementation of MariaDB 10.4 will use more memory than MySQL 8.0. MariaDB 10.4 and MySQL 8.0 also have different GTID implementations.

MariaDB's extra memory usage can be handled with the following rules of thumb:

- If you are not using MyISAM and don't plan to use Aria:
  - Set key_buffer_size to something very low (16K) as it's not used.
  - Set aria_pagecache_buffer_size to what you think you need for handling internal tmp tables that didn't fit in memory.
  - Normally this is what before you had set for key_buffer_size (at least 1M).

- If you are using MyISAM and not planning to use Aria:
  - Set aria_pagecache_buffer_size to what you think you need for handling internal tmp tables that didn't fit in memory.
- If you are planning to use Aria, you should set aria_pagecache_buffer_size to something that fits a big part of your normal data + overflow temporary tables.

Comparison Table

Variable	MariaDB 10.4 Default	MySQL 8.0 Default	Notes
activate_all_roles_on_login	-	OFF	Determines whether to automatically activate roles on login.
alter_algorithm	DEFAULT	-	MariaDB 10.3 introduced new ALTER TABLE ALGORITHM clauses to avoid slow copies in certain instances. This variable allows setting this if no `ALGORITHM` clause is specified.
analyze_sample_percentage	100.0000	-	Percentage of rows from the table ANALYZE TABLE will sample to collect table statistics.
aria_*	*	-	The Aria storage engine is only available in MariaDB.
auto_generate_certs	-	ON	Whether to automatically generate SSL key and certificate files.
avoid_temporal_upgrade	-	OFF	Determines whether ALTER TABLE implicitly upgrades temporal columns.
back_log	Autosized	Autosized	MariaDB and MySQL have different autosizing algorithms.
binlog-annotate-row-events	ON	-	Introduced in MariaDB 5.3 for replicating between MariaDB 5.3 and MySQL/MariaDB 5.1.
binlog_commit_wait_count	0	-	For use in MariaDB's parallel replication.
binlog_commit_wait_usec	100000	-	For use in MariaDB's parallel replication.
binlog_error_action		ABORT_SERVER	MySQL-only variable for controlling what happens when the server cannot write to the binary log.
binlog_expire_logs_seconds	-	2592000	Sets the binary log expiration period in seconds
binlog_file_cache_size	16184	-	For setting the size of the file cache for the binary log.
binlog_format	MIXED	ROW	MariaDB and MySQL have differing binary log formats.
binlog_group_commit_sync_delay		0	MySQL-only variable for controlling the wait time before synchronizing the binary log file to disk.
binlog_group_commit_sync_no_delay_count		0	MySQL-only variable for setting the maximum number of transactions to wait for before aborting the current `binlog_group_commit_sync_delay` delay.
binlog_gtid_simple_recovery	-	ON	MySQL-only GTID variable. MariaDB's GTID implementation is different.
binlog_max_flush_queue_time	-	0	Specifies a timeout for reading transactions from the flush queue before continuing with group commit and syncing log to disk.
binlog_optimize_thread_scheduling	ON	-	For optimized kernel thread scheduling.
binlog_order_commits	-	ON	Determines whether transactions may be committed in parallel.
binlog_row_metadata	-	MINIMAL	Determines the amount of table metadata added to the binary log with row-based logging.
binlog_row_value_options	-	(empty)	Permits an alternative binlog format for JSON document updates.
binlog_rows_query_log_events	-	OFF	MySQL-only variable for logging extra information in row-based logging.
binlog_transaction_dependency_history_size	-	25000	Maximum number of row hashes kept for looking up transactions that last modified a given row.
binlog_transaction_dependency_tracking	-	COMMIT_ORDER	For determining how to best use the slave's multithreaded applier.
block_encryption_mode	-	aes-128-ecb	MySQL-only variable for controlling the block encryption mode for block-based algorithms.

caching_sha2_password*	-	*	For use with MySQL's SHA-256 authentication with caching.
character_set_*	latin1 or utf8	utf8mb4	MySQL 8.0 defaults to the utf8mb4 character set.
check_constraint_checks	ON	-	Permits disabling constraint checks, for example when loading a table that violates some constraints that you plan to fix later.
check_proxy_users		OFF	MySQL-only variable for controlling whether the server performs proxy user mapping for authentication plugins.
collation_*	latin1_swedish_ci or utf8_general_ci	utf8mb4_0900_ai_ci	MySQL 8.0 defaults to the utf8mb4 character set.
column_compression_threshold	100	-	MariaDB supports Storage-engine Independent Column Compression.
column_compression_zlib_level	6	-	MariaDB supports Storage-engine Independent Column Compression.
column_compression_zlib_strategy	DEFAULT_STRATEGY	-	MariaDB supports Storage-engine Independent Column Compression.
column_compression_zlib_wrap	OFF	-	MariaDB supports Storage-engine Independent Column Compression.
cte_max_recursion_depth	-	1000	When MySQL 8.0 introduced common table expressions they used a different name. MariaDB's variable is called max_recursive_iterations.
date_format	%Y-%m-%d	-	Unused variable removed in MySQL 8.0
datetime_format	%Y-%m-%d	-	Unused variable removed in MySQL 8.0
deadlock_search_depth_long	15	-	The Aria storage engine is only available in MariaDB.
deadlock_search_depth_short	4	-	The Aria storage engine is only available in MariaDB.
deadlock_timeout_long	50000000	-	The Aria storage engine is only available in MariaDB.
deadlock_timeout_short	10000	-	The Aria storage engine is only available in MariaDB.
debug_no_thread_alarm	OFF	-	Disable system thread alarm calls, for debugging or testing.
default_authentication_plugin	-	caching_sha2_password	MySQL 8 introduced a new authentication plugin.
default_collation_for_utf8mb4	-	utf8mb4_0900_ai_ci	For internal use in MySQL 8 replication.
default_master_connection	empty	-	For use with MariaDB's multi-source replication.
default_password_lifetime	0	360	MariaDB defaults to password expiration off.
default_regex_flags	empty	-	For handling incompatibilities between MariaDB's PCRE and the old regex library.
default_tmp_storage_engine	empty	InnoDB	Default storage engine used for tables created with CREATE TEMPORARY TABLE.
disabled_storage_engines		empty	MySQL-only variable for disabling specific storage engines.
disconnect_on_expired_password	OFF	ON	MariaDB password expiration is off by default, and by default does not disconnect a client when a password has expired.
encrypt_binlog	OFF	-	MariaDB enables table and tablespace encryption.
encrypt_tmp_files	OFF	-	MariaDB enables table and tablespace encryption.
encrypt_tmp_disk_tables	OFF	-	MariaDB enables table and tablespace encryption.
end_markers_in_json	-	OFF	MySQL-only variable for adding end markers to JSON output.
enforce_gtid_consistency	-	OFF	MariaDB and MySQL have different GTID implementations.
enforce_storage_engine	none		Forces the use of a particular storage engine for new tables.
eq_range_index_dive_limit	0	200	Variable for tuning when the optimizer should switch from using index dives to index statistics for qualifying rows estimation.
event_scheduler	OFF	ON	MySQL enables the event scheduler by default.

expensive_subquery_limit	100	-	Used for determining expensive queries for optimization.
explicit_defaults_for_timestamp	OFF	ON	MySQL 8 disables the old timestamp behavior.
extra_max_connections	1	-	Introduced in the MariaDB 5.1 threadpool.
extra_port	0	-	Introduced in the MariaDB 5.1 threadpool.
group_concat_max_len	1048576	1024	MariaDB increases the maximum length for a GROUP_CONCAT() result from 1K to 1M.
gtid_binlog_pos	empty	-	MariaDB and MySQL have different GTID implementations.
gtid_binlog_state	empty	-	MariaDB and MySQL have different GTID implementations.
gtid_cleanup_batch_size	64	-	MariaDB and MySQL have different GTID implementations.
gtid_current_pos	empty	-	MariaDB and MySQL have different GTID implementations.
gtid_domain_id	0	-	MariaDB and MySQL have different GTID implementations.
gtid_executed	-	empty	MariaDB and MySQL have different GTID implementations.
gtid_executed_compression_period	-	1000	MariaDB and MySQL have different GTID implementations.
gtid_ignore_duplicates	OFF	-	MariaDB and MySQL have different GTID implementations.
gtid_mode	-	OFF	MariaDB and MySQL have different GTID implementations.
gtid_next	-	AUTOMATIC	MariaDB and MySQL have different GTID implementations.
gtid_owned	-	empty	MariaDB and MySQL have different GTID implementations.
gtid_pos_auto_engines	empty	-	MariaDB and MySQL have different GTID implementations.
gtid_purged	-	empty	MariaDB and MySQL have different GTID implementations.
gtid_seq_no	0	-	MariaDB and MySQL have different GTID implementations.
gtid_slave_pos	empty	-	MariaDB and MySQL have different GTID implementations.
gtid_strict_mode	OFF	-	MariaDB and MySQL have different GTID implementations.
have_crypt	YES	-	MySQL has removed the ENCRYPT function.
have_openssl			MariaDB's version indicates whether YaSSL or openssl was used. MySQL's is a synonym for `have_ssl`.
have_query_cache	YES	-	MySQL has removed the query cache.
have_statement_timeout	-		Whether MySQL's statement execution timeout feature is available.
have_symlink	YES	DISABLED	MySQL has removed symlink support.
histogram_generation_max_mem_size	-	20000000	Added when MySQL 8 introduced Histogram-based Statistics. MariaDB uses histogram_size
histogram_size	0	-	MariaDB introduced Histogram-based Statistics.
histogram_type	SINGLE_PREC_HB	-	MariaDB introduced Histogram-based Statistics.
idle_readonly_transaction_timeout	0	-	Time in seconds that the server waits for idle read-only transactions.
idle_transaction_timeout	0	-	Time in seconds that the server waits for idle transactions.
idle_write_transaction_timeout	0	-	Time in seconds that the server waits for idle write transactions.
ignore_builtin_innodb	OFF	-	Ignored and removed in MySQL 8.
in_predicate_conversion_threshold	1000	-	Controls the Conversion of Big IN Predicates Into Subqueries optimization.
in_transaction	0	-	Set to `1` if you are in a transaction, and `0` if not.
information_schema_stats_expiry	-	86400	Time until MySQL Information Schema cached statistics expire.

innodb_adaptive_flushing_lwm	10.000000	10	Adaptive flushing is enabled when this this low water mark percentage of the redo log capacity is reached. MariaDB's variable is a double, MySQL's an integer.
innodb_api_*	-	*	Specific to MySQL's memcached, removed in MariaDB 10.2.
innodb_autoinc_lock_mode	1	2	MariaDB has an extra mode, `3`, for skipping the rollback of connected transactions. MySQL defaults to row-based replication, so can safely use `2`.
innodb_background_*	*	-	MariaDB has support for data scrubbing.
innodb_checksums	ON	-	Deprecated option removed in MySQL.
innodb_compression_*	*	-	Introduced with MariaDB's InnoDB compression.
innodb_dedicated_server	-	OFF	MySQL option that automatically configures various settings if the server is a dedicated InnoDB database server.
innodb_default_encryption_key_id	1	-	Default encryption key id used for table encryption. See Data at Rest Encryption.
innodb_defragment	*	-	MariaDB can defragment InnoDB tablespaces.
innodb_directories	-	(empty)	Used to search for tablespace files when moving or restoring a new location.
innodb_disallow_writes	OFF	-	Tell InnoDB to stop any writes to disk.
innodb_encrypt_*	1	-	See MariaDB's Data at Rest Encryption.
innodb_fatal_semaphore_wait_threshold	600	-	MariaDB's fatal semaphore timeout is configurable.
innodb_file_format	(empty)	-	MariaDB 10.4 has restored this unused, deprecated variable for compatibility reasons.
innodb_flush_neighbors	1	0	MySQL 8 by default now assumes the use of an SSD device.
innodb_force_primary_key	OFF	-	If set to 1 in MariaDB (0 is default) CREATE TABLEs without a primary or unique key where all keyparts are NOT NULL will not be accepted, and will return an error.
innodb_idle_flush_pct	100	-	Up to what percentage of dirty pages in MariaDB should be flushed when InnoDB finds it has spare resources to do so.
innodb_immediate_scrub_data_uncompressed	OFF	-	MariaDB has support for data scrubbing.
innodb_large_prefix	(empty)	-	MariaDB 10.4 has restored this unused, deprecated variable for compatibility reasons.
innodb_lock_schedule_algorithm	VATS	-	MariaDB has an improved algorithm for deciding which of the waiting transactions should be granted a lock once it has been released.
innodb_locks_unsafe_for_binlog	OFF	-	Deprecated option in MariaDB for disabling gap locking for searches and index scans. Deprecated in MariaDB, use READ COMMITTED transaction isolation instead.
innodb_log_optimize_ddl	ON	-	Whether redo logging should be reduced when natively creating indexes or rebuilding tables
innodb_log_spin_*	-	*	MySQL variables for constraining CPU usage while waiting for flushed redo.
innodb_log_wait_for_flush_spin_hwm	-	*	MySQL variable for constraining CPU usage while waiting for flushed redo.
innodb_max_dirty_pages_pct	75	90	MySQL 8 increased the default to 90.
innodb_max_dirty_pages_pct_lwm	0	10	MySQL 8 increased the default to 10.
innodb_max_undo_log_size	10485760	1073741824	MariaDB 10.2 reduced the limit for when an undo tablespace is marked for truncation.
innodb_open_files	Autosized (2000)	Autosized (4000)	In most systems, autosized based on the table_open_cache setting, which differs between MariaDB and MySQL.
innodb_prefix_index_cluster_optimization	OFF	-	MariaDB includes the Facebook prefix index queries optimization.
innodb_print_ddl_logs	-	OFF	MySQL option for writing DDL logs to stderr.

innodb_redo_log_encrypt	-	OFF	MySQL 8 has also now introduced redo log encryption, but used a different name. The equivalent option in MariaDB is innodb_encrypt_log.
innodb_scrub_*	*	-	MariaDB includes options to scrub the redo log.
innodb_spin_wait_delay	4	6	MariaDB changed the default from 6 to 4 based on extensive benchmarking.
innodb_stats_modified_counter	0	-	MariaDB option to control the calculation of new statistics.
innodb_stats_sample_pages	8	-	Deprecated MariaDB option for control over index distribution statistics.
innodb_stats_traditional	ON	-	Enabling gives a larger sample of pages for larger tables for the purposes of index statistics calculation.
innodb_undo_log_encrypt	-	OFF	MySQL option for encrypting undo logs residing in separate undo tablespaces.
innodb_undo_log_truncate	OFF	ON	MySQL 8 changes the default to ON, marking larger undo logs for truncation.
innodb_undo_logs	128	-	Removed in MySQL 8.
innodb_undo_tablespaces	0	2	Number of tablespace files used for dividing up the undo logs. MySQL 8 has deprecated this setting, and increased the default (and minimum) to 2.
innodb_use_atomic_writes	ON	-	Atomic writes are a faster alternative to innodb_doublewrite and MariaDB automatically detects when supporting SSD cards are used.
internal_tmp_disk_storage_engine	-	INNODB	MySQL uses this variable to set the storage engine for on-disk internal temporary tables.
internal_tmp_mem_storage_engine	-	TEMPTABLE	MySQL and MariaDB use different formats for temporary tables. In MariaDB, the aria_used_for_temp_tables performs a similar function.
join_buffer_space_limit	2097152	-	Maximum size in bytes of the query buffer. See block-based join algorithms.
join_cache_level	2	-	For determining the join algorithms. See block-based join algorithms
key_buffer_size	134217728	8388608	Size of the buffer for the index blocks used by MyISAM tables and shared for all threads.
key_cache_file_hash_size	512	-	Number of hash buckets for open and changed files.
key_cache_segments	0	-	The number of segments in a key cache. See Segmented Key Cache.
keyring_operations	-	ON	Whether MySQL 8's keyring operations are enabled.
last_gtid	-	empty	MariaDB and MySQL have different GTID implementations.
local_infile	ON	OFF	MySQL no longer supports LOAD DATA LOCAL by default.
lock_wait_timeout	86400	31536000	MariaDB has reduced the timeout for acquiring metadata locks.
log_bin	OFF	ON	MySQL 8 enables the binary log by default.
log_bin_compress	OFF	-	MariaDB setting for whether or not the binary log can be compressed.
log_bin_compress_min_len	256	-	Minimum length of sql statement (in statement mode) or record (in row mode) that can be compressed. See Compressing Events to Reduce Size of the Binary Log.
log_bin_use_v1_row_events	-	OFF	MySQL-only variable showing whether or not MySQL's version 2 binary logging format is being used.
log_disabled_statements	sp	-	Disable logging of certain statements to the general log.
log_error_services	-	log_filter_internal; log_sink_internal	Components to enable for MySQL error logging.
log_error_verbosity	-	3	MySQL variable for setting verbosity of error, warning, and note messages in the error log.
log_slave_updates	OFF	ON	MySQL 8 has by default enabled binary logging of updates a slave receives from a master.

log_slow_admin_statements	ON	OFF	MariaDB logs slow admin statements to the slow query log by default.
log_slow_disabled_statements	admin,call,slave,sp	-	Disable logging of certain statements to the slow query log.
log_slow_filter	admin, filesort, filesort_on_disk, full_join, full_scan, query_cache, query_cache_miss, tmp_table, tmp_table_on_disk	-	For slow query log filtering.
log_slow_rate_limit	1	-	Limits the number of queries logged to the slow query log.
log_slow_slave_statements	ON	OFF	MariaDB logs slow slave statements to the slow query log by default.
log_slow_verbosity	empty	-	Controls information to be added to the slow query log. See also Slow Query Log Extended Statistics.
log_statements_unsafe_for_binlog	-	ON	MySQL setting for controlling whether binlog warnings are written to the error log.
log_syslog*	platform-dependent	-	MySQL variables with settings for writing to syslog.
log_tc_size	24576	-	Size in bytes of the transaction coordinator log, defined in multiples of 4096.
log_throttle_queries_not_using_indexes	-	0	MySQL-only variable for limiting the number of statements without indexes written to the slow query log.
log_timestamps	-	UTC	MySQL-only variable controlling the timezone for certain logging conditions.
log_warnings	2	-	MySQL 8 has replaced with `log_error_verbosity`.
mandatory_roles	-	(empty)	MySQL variable for assigning roles to all users.
master_info_repository	-	TABLE	Whether slave logs master status and connection info to a table or a file.
max_allowed_packet	16M	64M	
max_error_count	64	1024	Specifies the maximum number of messages stored for display by SHOW ERRORS and SHOW WARNINGS statements.
max_execution_time	-	0	MySQL renamed the `max_statement_time` variable.
max_length_for_sort_data	64	1024	Used to decide which algorithm to choose when sorting rows. If the total size of the column data, not including columns that are part of the sort, is less than max_length_for_sort_data, then these are added to the sort key. This can speed up the sort as there's no need to re-read the same row again later. Setting the value too high can slow things down as there will be a higher disk activity for doing the sort.
max_long_data_size	16777216	-	Maximum size for parameter values sent with mysql_stmt_send_long_data(). Removed in MySQL 5.6.
max_password_errors	4294967295	-	Maximum number of failed connections attempts before no more are permitted.
max_points_in_geometry	-	65536	Maximum points_per_circle for MySQL's ST_Buffer_Strategy() function.
max_recursive_iterations	4294967295	-	Maximum number of iterations when executing recursive queries.
max_relay_log_size	1073741824	0	Can be set by session in MariaDB.
max_seeks_for_key	4294967295	4294967295 (32-bit) or 18446744073709547520 (64-bit)	The most key seeks required when searching with an index, regardless of the actual index cardinality. If this value is set lower than its default and maximum, indexes will tend to be preferred over table scans.
max_session_mem_used	9223372036854775807	-	Amount of memory a single user session is allowed to allocate.
max_statement_time	0	-	Maximum time in seconds that a query can execute before being aborted. MySQL used to have a variable of this name before renaming it `max_execution_time`.
max_tmp_tables	32	-	Unused variable removed in MySQL.

max_write_lock_count	4294967295	4294967295 (32-bit) or 18446744073709547520 (64-bit)	Read lock requests will be permitted for processing after this many write locks.
mrr_buffer_size	262144	-	Size of buffer to use when using multi-range read with range access. See Multi Range Read optimization.
multi_range_count	1024	-	Unused variable removed in MySQL.
myisam_block_size	1024	-	Block size used for MyISAM index pages.
myisam_recover_options	BACKUP,QUICK	OFF	MyISAM recovery mode.
myisam_sort_buffer_size	134216704	8388608	Size in bytes of the buffer allocated when creating or sorting indexes on a MyISAM table.
mysql_native_password_proxy_users	-	OFF	Whether MySQL's authentication plugin supports proxy users. I
mysql56_temporal_format	ON		Causes MariaDB to use the MySQL-5.6 low level formats for TIME, DATETIME and TIMESTAMP instead of the MariaDB 5.3+ version.
new	-	OFF	Used for backward-compatibility with MySQL 4.1, not present in MariaDB.
mysqlx+*	-	*	MySQL's X plugin related variables.
ngram_token_size	-	2	Sets the n-gram token size for MySQL's n-gram full-text parser.
offline_mode	-	OFF	MySQL settting for specifying whether the server should run in offline mode.
old_alter_table	DEFAULT	OFF	An alias for alter_algorithm.
old_mode	Empty string	-	Used for getting MariaDB to emulate behavior from an old version of MySQL or MariaDB. See OLD Mode.
old_passwords	OFF	-	MySQL 8 is no longer compatible with the old pre-MySQL 4.1 form of password hashing.
optimizer_selectivity_sampling_limit	100	-	Controls number of record samples to check condition selectivity.
optimizer_switch	See details		A series of flags for controlling the query optimizer. MariaDB has introduced a number of new settings.
optimizer_trace_*	-	*	MySQL has more settings for optimizer tracing.
optimizer_use_condition_selectivity	4	-	Controls which statistics can be used by the optimizer when looking for the best query execution plan.
original_commit_timestamp	-	*	Used by MySQL 8 for delaying replication.
parser_max_mem_size	-	4294967295 (32-bit) or 18446744073709547520 (64-bit)	MySQL variable for limiting memory available to the parser.
password_*	-	*	Controls reuse of previous passwords in MySQL.
performance_schema	OFF	ON	The Performance Schema is off by default in MariaDB.
performance_schema_*			Many performance schema variables are autoset in MySQL, and MySQL has a different version, with additional variables.
plugin_maturity	One less than the server maturity	-	Minimum acceptable plugin maturity.
progress_report_time	5	-	Time in seconds between sending progress reports to the client for time-consuming statements.
proxy_protocol_networks	(empty)	-	Enable proxy protocol for these source networks.
query_alloc_block_size	16384	8192	Size in bytes of the extra blocks allocated during query parsing and execution (after query_prealloc_size is used up).
query_cache_*	*	-	MySQL has removed the query cache.
query_prealloc_size	24576	8192	Size in bytes of the persistent buffer for query parsing and execution, allocated on connect and freed on disconnect.
range_optimizer_max_mem_size	-	8388608	MySQL-only variable setting a limit on the range optimizer's memory usage.
rbr_exec_mode	-	STRICT	MySQL-only variable for determining the handling of certain key errors.
read_binlog_speed_limit	0	-	Permits restricting the speed at which the slave reads the binlog from the master.
regexp_*	-	*	Memory and time limits for regular expression matching operations.

relay_log_info_repository	-	TABLE	MySQL-only variable determining whether the slave's position in the relay logs is written to a file or table.
replicate_annotate_row_events	ON	-	Tells the slave to reproduce annotate_rows_events received from the master in its own binary log.
replicate_do_db	empty string	-	See Dynamic Replication Variables.
replicate_do_table	empty string	-	See Dynamic Replication Variables.
replicate_events_marked_for_skip	replicate	-	See Selectively skipping replication of binlog events.
replicate_ignore_db	empty string	-	See Dynamic Replication Variables.
replicate_ignore_table	empty string	-	See Dynamic Replication Variables.
replicate_wild_do_table	empty string	-	See Dynamic Replication Variables.
replicate_wild_ignore_table	empty string	-	See Dynamic Replication Variables.
require_secure_transport	-	OFF	MySQL-only variable determining whether client to server connections need to be secure.
result_metadata	-	FULL	Determine whether the server returns result set metadata for connections where this is optional.
rowid_merge_buff_size	8388608	-	See Non-semi-join subquery optimizations.
rpl_read_size	-	8192	Minimum data in bytes read from the binary and relay log files.
rpl_semi_sync_*	-	-	MariaDB includes semisynchronous replication without the need to install a plugin.
rpl_stop_slave_timeout	-	31536000	Controls the time that STOP SLAVE waits before timing out.
schema_definition_cache	-	256	Limits the number of schema definition objects kept in the dictionary object cache.
secure_auth	ON	-	Removed in MySQL.
secure_timestamp	NO	-	MariaDB-only option permitting the restricting of direct setting of a session timestamp..
server_id_bits	-	server_id	MySQL-only variable for use in MySQL Cluster.
server_uuid	-	UUID	MySQL-only variable containing the UUID.
session_track_gtids	-	OFF	MySQL-only variables for tracking gtid changes. MariaDB and MySQL's gtid implementation is different.
sha256_password_proxy_users	-	OFF	MySQL-only variable determining whether the sha256_password plugin supports proxy users.
show_create_table_verbosity	-	OFF	Option to cause SHOW CREATE TABLE to display ROW_FORMAT in all cases.
show_old_temporals	-	OFF	MySQL-only variable for determining whether SHOW CREATE TABLE output should include comments for old format temporal columns.
skip_parallel_replication	OFF	-	See parallel replication.
skip_replication	OFF	-	See Selectively skipping replication of binlog events.
slave_allow_batching	-	OFF	MySQL-only replication variable.
slave_checkpoint_group	-	512	MySQL-only replication variable.
slave_checkpoint_period	-	300	MySQL-only replication variable.
slave_ddl_exec_mode	IDEMPOTENT	-	Modes for how replication of DDL events should be executed.
slave_domain_parallel_threads	0	-	For configuring parallel replication.
slave_net_timeout	3600	60	MySQL reduced the timeout to 60s.
slave_parallel_max_queued	131072	-	For configuring parallel replication.
slave_parallel_mode	conservative	-	Controls what transactions are applied in parallel when using parallel_replication.
slave_parallel_threads	0	-	For configuring parallel replication.
slave_parallel_type	-	DATABASE	MySQL-only replication variable.
slave_pending_jobs_size_max	-	16777216	MySQL-only replication variable.
slave_preserve_commit_order	-	OFF	MySQL-only replication variable.
slave_rows_search_algorithms	-	INDEX_SCAN, HASH_SCAN	MySQL-only replication variable.

slave_run_triggers_for_rbr	NO		See Running triggers on the slave for Row-based events for a description and use-case for this setting.
slave_transaction_retry_errors	1213,1205	-	When an error occurs during a transaction on the slave, replication usually halts. By default, transactions that caused a deadlock or elapsed lock wait timeout will be retried. One can add other errors to the the list of errors that should be retried by adding a comma-separated list of error numbers to this variable.
slave_transaction_retry_interval	0	-	Interval in seconds for the slave SQL thread to retry a failed transaction due to a deadlock, elapsed lock wait timeout or an error listed in slave_transaction_retry_errors.
sort_buffer_size	2097152	262144	The default sort buffer allocated has been reduced in MySQL.
sql_mode	STRICT_TRANS_TABLES, ERROR_FOR_DIVISION_BY_ZERO, NO_AUTO_CREATE_USER, NO_ENGINE_SUBSTITUTION	ONLY_FULL_GROUP_BY, STRICT_TRANS_TABLES, NO_ZERO_IN_DATE, NO_ZERO_DATE, ERROR_FOR_DIVISION_BY_ZERO, NO_ENGINE_SUBSTITUTION	See SQL Mode.
ssl_fips_mode	-	OFF	Whether FIPS mode is enabled on the server side.
standard_compliant_cte	ON	-	See Common Table Expressions.
storage_engine	InnoDB	-	Alias for default_storage_engine, removed in MySQL.
strict_password_validation	ON	-	In MariaDB, when password validation plugins are enabled, reject passwords that cannot be validated (passwords specified as a hash).
stored_program_definition_cache	-	256	Limits the number of stored program definition objects kept in the dictionary object cache.
super_read_only	-	OFF	MySQL variable for prohibiting client updates from users with the SUPER privilege.
sync_binlog	0	1	MySQL synchronizes all actions to the binary log before they are committed.
sync_frm	1	-	.frm files have been removed in MySQL.
system_versioning_alter_history	ERROR	-	MariaDB has System-Versioned Tables
system_versioning_asof	DEFAULT	-	MariaDB has System-Versioned Tables
table_definition_cache	400	-1 (autosized)	Number of table definitions that can be cached.
table_open_cache_instances	8	16	Maximum number of table cache instances.
tablespace_definition_cache	-	256	Limits the number of tablespace definition objects kept in the dictionary object cache.
tcp_keepalive_interval	0	-	Interval, in seconds, between when successive keep-alive packets are sent if no acknowledgement is received.
tcp_keepalive_probes	0	-	Number of unacknowledged probes to send before considering the connection dead and notifying the application layer.
tcp_keepalive_time	0	-	Set the TCP_NODELAY option (disable Nagle's algorithm) on socket.
tcp_nodelay	1	-	Timeout, in milliseconds, with no activity until the first TCP keep-alive packet is sent.
temptable_max_ram	-	1GB	Limits the RAM used by MySQL's TempTable storage engine.
thread_cache_size	Autosized	-1 (autosized)	MariaDB uses an improved thread pool.
thread_concurrency	10	-	Removed in MySQL 5.7.
thread_pool_idle_timeout	60	-	See Using the Thread Pool.
thread_pool_max_threads	65536	-	See Using the Thread Pool.
thread_pool_min_threads	1	-	Windows-only. See Using the Thread Pool.
thread_pool_oversubscribe	3	-	See Using the Thread Pool.
thread_pool_prio_kickup	auto	-	See Using the Thread Pool.
thread_pool_priority	auto	-	See Using the Thread Pool.
thread_pool_size	Number of processors	16*	See Using the Thread Pool. *Only available in MySQL with a commercial plugin.

thread_pool_stall_limit	500	6*	See Using the Thread Pool. *Only available in MySQL with a commercial plugin.
thread_stack	299008	Varies	See Using the Thread Pool.
time_format	%H:%i:%s	-	Removed in MySQL.
timed_mutexes	OFF	-	Removed in MySQL.
tmp_disk_table_size	18446744073709551615	-	Max size for data for an internal temporary on-disk MyISAM or Aria table.
tmp_memory_table_size	16777216	-	Alias for tmp_table_size.
transaction_allow_batching	-	OFF	Variable for enabling batching of statements within the same transaction in MySQL Cluster.
transaction_isolation	-	REPEATABLE-READ	The MariaDB equivalent is tx_isolation.
transaction_read_only	-	OFF	The MariaDB equivalent is tx_read_only.
transaction_write_set_extraction	-	OFF	Unused MySQL-only variable.
tx_isolation	REPEATABLE-READ	-	The MySQL equivalent is transaction_isolation.
tx_read_only	OFF	-	The MySQL equivalent is transaction_read_only.
use_stat_tables	preferably_for_queries	-	Controls the use of engine-independent table statistics.
userstat	OFF	-	Whether to activate MariaDB's User Statistics implementation, not available in MySQL.
version_compile_zlib	-	*	Version of the zlib library compiled in.
version_malloc_library	*	-	Version of the used malloc library.
version_source_revision	Varies	-	Permits seeing exactly which version of the source was used for a build.
version_ssl_library	*	-	Version of the used TLS library.
windowing_high_use_precision	-	*	MySQL option allowing safety to be sacrificed for speed in window function calculations.
wsrep_*	*	-	Galera cluster is only available in MariaDB.
Variable	**MariaDB 10.4**	**MySQL 8.0**	**Notes**

2.1.14.1.13 Upgrading from MySQL 5.7 to MariaDB 10.2

Following compatibility report was done on 10.2.4 and may get some fixing in next minor releases

- MySQL unix socket plugin can be different. MariaDB can get similar usage via INSTALL PLUGIN unix_socket SONAME 'auth_socket.so'; you may have to enable this plugin in config files via load plugin.
- When using data type JSON , one should convert type to TEXT, virtual generated column works the same after.
- When using InnoDB FULLTEXT index one should not use innodb_defragment
- MySQL re-implemented partitioning in 5.7, thus you cannot perform in-place upgrades for partitioned tables. They will require mysqldump/import to work correctly in MariaDB.

2.1.2.11 Installing MariaDB Alongside MySQL

2.1.3.12.2 Moving from MySQL to MariaDB in Debian 9

2.1.14.2 Migrating to MariaDB from SQL Server

This section is a guide to help you migrate from SQL Server to MariaDB. This includes a general understanding of MariaDB, information to help plan the migration, and differences in the configuration and syntax.

Understanding MariaDB Architecture

An overview of MariaDB server architecture, especially where it differs from SQL Server.

SQL Server Features Not Available in MariaDB

List of SQL Server features not available in MariaDB.

SQL Server Features Implemented Differently in MariaDB
List of SQL Server features that MariaDB implements in a different way.

MariaDB Features Not Available in SQL Server
List of MariaDB features not available in SQL Server.

Setting Up MariaDB for Testing for SQL Server Users
Hints for SQL Server users to setup MariaDB for testing.

Syntax Differences between MariaDB and SQL Server
MariaDB syntax hints for SQL Server users.

SQL Server and MariaDB Types Comparison
Comparison tables between SQL Server types and MariaDB types.

MariaDB Transactions and Isolation Levels for SQL Server Users
How MariaDB transactions and isolation levels differ from SQL Server.

MariaDB Authorization and Permissions for SQL Server Users
MariaDB handles users and permissions quite differently to SQL Server.

Repairing MariaDB Tables for SQL Server Users
MariaDB table repair explained to SQL Server users.

MariaDB Backups Overview for SQL Server Users
An overview of MariaDB backup tools and techniques for SQL Server users.

MariaDB Replication Overview for SQL Server Users
An overview of MariaDB replication types for SQL Server users.

Moving Data Between SQL Server and MariaDB
Information and some advice on how to import SQL Server data into MariaDB.

SQL_MODE=MSSQL
Microsoft SQL Server compatibility mode.

There are 2 related questions.

2.1.14.2.1 Understanding MariaDB Architecture

Contents

MariaDB architecture is partly different from the architecture of traditional DBMSs, like SQL Server. Here we will examine the main components that a new MariaDB DBA needs to know. We will also discuss a bit of history, because this may help understand MariaDB philosophy and certain design choices.

This section is an overview of the most important components. More information is included in specific sections of this migration guide, or in other pages of the MariaDB Knowledge Base (see the links scattered over the text).

Storage Engines

MariaDB was born from the source code of MySQL, in 2008. Therefore, its history begins with MySQL.

MySQL was born at the beginning of the 90s. Back in the days, if compared to its existing competitors, MySQL was lightweight, simple to install, easy to learn. While it had a very limited set of features, it was also fast in certain common operations. And it was open source. These characteristics made it suitable to back the simple websites that existed at that time.

The web evolved rapidly, and the same happened to MySQL. Being open source helped a lot in this respect, because the community needed functionalities that weren't supported at that time.

MySQL was probably the first database system to support a pluggable storage engine architecture. Basically, this means that MySQL knows very little about creating or populating a table, reading from it, building proper indexes and caches. It just delegated all these operations to a special plugin type called a storage engine.

One of the first plugins developed by third parties was InnoDB. It is very fast, and it adds two important features that are not otherwise supported: transactions and foreign keys.

Note that when MariaDB asks a storage engine to write or read a row, the storage engine could theoretically do anything. This led to the creation of very interesting alternative engines, like BLACKHOLE (which doesn't write or read any data, acting like the /dev/null file in Linux), or CONNECT (which can read and write to files written in many different formats, or remote DBMSs, or some other special data sources).

Nowadays InnoDB is the default MariaDB storage engine, and it is the best choice for most use cases. But for particular needs, sometimes using a different storage engine is desirable. In case of doubts about the best storage engine to use for a specific case, check the Choosing the Right Storage Engine page.

When we create a table, we specify its storage engine or use the default one. It is possible to convert an existing table to another storage engine, though this is a blocking operation which requires a complete table copy. Third-party storage engines can also be installed while MariaDB is running.

Note that it is perfectly possible to use tables with different storage engines in the same transaction (even if some engines are not transactional). It is even possible to use different engines in the same query, for example with JOINs and subqueries.

The default storage engine can be changed by changing the default_storage_engine variable. A different default can be specified for temporary tables by setting default_tmp_storage_engine. MariaDB uses Aria for system tables and temporary tables created internally to store the intermediate results of a query.

InnoDB

It is worth spending some more words here about InnoDB, the default storage engine.

Primary Key and Indexes

InnoDB primary keys are always the equivalent of SQL Server clustered indexes. In other words, an InnoDB table is always ordered by the primary key.

If an InnoDB table doesn't have a user-defined primary key, the first `UNIQUE` index whose columns are all `NOT NULL` is used as a primary key. If there is no such index, the table will have a *clustered index*. The terminology here can be a bit confusing for SQL Server and other DBMS users. A clustered index in InnoDB is a 6 bytes value that is added to the table. This index and its values are completely invisible to the users. It's important to note that clustered indexes are governed by a global mutex that greatly reduces their scalability.

Secondary indexes are ordered by the columns that are part of the index, and contain a reference to each entry's corresponding primary key value.

Some consequences of these design choices are the following:

- For performance reasons, a primary key value should be inserted in order. In other words, the last inserted value should be the highest. This order is normally followed when inserting values into an `AUTO_INCREMENT` primary key. The reason is that inserting values in the middle of an ordered data structure is slower, unless they fit into existing holes. If we insert primary key values randomly, InnoDB often has to rearrange pages to make some room for the new data.
- A big primary keys means that all secondary indexes are also big.
- A query by primary key will require a single search. A query on a secondary index that also reads columns not contained in the index will require one search on the index, plus one more search for each row that satisfies the index condition.
- We shouldn't explicitly include the primary key in a secondary index. If we do so, the primary key column will be duplicated in the index.

Tablespaces

For InnoDB, a *tablespace* is a file containing data (not a file group as in SQL Server). The types of tablespaces are:

- System tablespace.
- File-per-table tablespaces.
- Temporary tablespaces.

The system tablespace is stored in the file `ibdata`. It contains information used by InnoDB internally, like rollback segments, as well as some system tables. Historically, the system tablespace also contained all tables created by the user. In modern MariaDB versions, a table is created in the system tablespace only if the innodb_file_per_table system variable is set to 0 at the moment of the table creation. By default, innodb_file_per_table is 1.

Tables created while `innodb_file_per_table=1` are written into their own tablespace. These are `.ibd` files.

Starting from MariaDB 10.2, temporary tables are written into temporary tablespaces, which means `ibtmp*` files. Previously, they were created in the system tablespace or in file-per-table tablespaces according to the value of `innodb_file_per_table`, just like regular tables. Temporary tablespaces, if present, are deleted when MariaDB starts.

It is important to remember that tablespaces can never shrink. If a file-per-table tablespace grows too much, deleting data won't recover space. Instead, a new table must be created and data needs to be copied. Finally, the old table will be deleted. If the system tablespace grows too much, the only solution is to move data into a new MariaDB installation.

Transaction Logs

In SQL Server, the transaction log contains both the undo log and the redo log. Usually we have only one transaction log.

In MariaDB the undo log and the redo log are stored separately. By default, the redo log is written to two files, called `ib_logfile0` and `ib_logfile1`. The undo log by default is written to the *system tablespace*, which is in the `ibdata1` file. However, it is possible to write it in separate files in a specified directory.

MariaDB provides no way to inspect the contents of the transaction logs. However, it is possible to inspect the binary log.

InnoDB transaction logs are written in a circular fashion: their size is normally fixed, and when the end is reached, InnoDB continues to write from the beginning. However, if very long transactions are running, InnoDB cannot overwrite the oldest data, so it has to expand the log size instead.

InnoDB Buffer Pool

MariaDB doesn't have a central buffer pool. Each storage engine may or may not have a buffer pool. The InnoDB buffer

pool is typically assigned a big amount of memory. See MariaDB Memory Allocation.

MariaDB has no extension like the SQL Server buffer pool extension.

A part of the buffer pool is called the change buffer. It contains dirty pages that have been modified in memory and not yet flushed.

InnoDB Background Threads

InnoDB has background threads that take care of flushing dirty pages from the change buffer to the tablespaces. They don't directly affect the latency of queries, but they are very important for performance.

SHOW ENGINE InnoDB STATUS shows information about them in the `BACKGROUND THREAD` section. They can also be seen using the threads table, in the performance_schema.

InnoDB flushing is similar to *lazy writes* and *checkpoints* in SQL Server. It has no equivalent for *eager writing*.

For more information, see InnoDB Page Flushing and InnoDB Purge.

Checksums and Doublewrite Buffer

InnoDB pages have checksums. After writing pages to disk, InnoDB verifies that the checksums match. The checksum algorithm is determined by innodb_checksum_algorithm. Check the variable documentation for its consequences on performance, backward compatibility and encryption.

In case of a system crash, hardware failure or power outage, a page could be half-written on disk. For some pages, this causes a disaster. Therefore, InnoDB writes essential pages to disk twice. A backup copy of the new page version is written first. Then, the old page is overwritten. The backup copies are written into a file called the *doublewrite buffer*.

- If an event prevents the first page from being written, the old version of the page will still be available.
- If an event prevents the old page from being completely overwritten by its new version, the page can still be recovered using the doublewrite buffer.

The doublewrite buffer can disabled using the innodb_doublewrite variable, but this usually doesn't bring big performance benefits. The doublewrite buffer location can be changed with innodb_doublewrite_file.

Aria

Even if we only create InnoDB tables, we use Aria indirectly, in two ways:

- For system tables.
- For internal temporary tables.

Aria is a non-transactional storage engine. By default it is crash-safe, meaning that all changes to data are written and fsynced to a write-ahead log and can always be recovered in case of a crash.

Aria caches indexes into the pagecache. Data are not directly cached by Aria, so it's important that the underlying filesystem caches reads and writes.

The pagecache size is determined by the aria_pagecache_buffer_size system variable. To know if it is big enough we can check the proportion of free pages (the ratio between Aria_pagecache_blocks_used and Aria_pagecache_blocks_unused) and the proportion of cache misses (the ratio between Aria_pagecache_read_requests and Aria_pagecache_reads.

The proportion of dirty pages is the ratio between Aria_pagecache_blocks_used and Aria_pagecache_blocks_not_flushed tells us if the log file is big enough.

The size of Aria log is determined by aria_log_file_size.

Databases

MariaDB does not support the concept of schema. In MariaDB SQL, *schema* and *schemas* are synonyms for *database* and *databases*.

When a user connects to MariaDB, they don't connect to a specific database. Instead, they can access any table they have permissions for. There is however a concept of *default database*, see below.

A database is a container for database objects like tables and views. A database serves the following purposes:

- A database is a namespace.
- A database is a logical container to separate objects.
- A database has a default character set and collation, which are inherited by their tables.
- Permissions can be assigned on a whole database, to make permission maintenance simpler.
- Physical data files are stored in a directory which has the same name as the database to which they belong.

System Databases

MariaDB has the following system databases:

- mysql is for internal use only, and should not be read or written directly.
- information_schema contains all information that can be found in SQL Server's information_schema and more. However, while SQL Server's `information_schema` is a schema containing information about the local database, MariaDB's `information_schema` is a database that contains information about all databases.
- performance_schema contains information about MariaDB runtime. It is disabled by default. Enabling it requires setting the performance_schema system variable to 1 and restarting MariaDB.

Default Database

When a user connects to MariaDB, they can optionally specify a default database. A default database can also be specified or changed later, with the USE command.

Having a default database specified allows one to specify tables without specifying the name of the database where they are located. If no default database is specified, all table names must be fully qualified.

For example, the two following snippets are equivalent:

```
SELECT * FROM my_database.my_table;

-- is equivalent to:
USE my_database;
SELECT * FROM my_table;
```

Even if a default database is specified, tables from other databases can be accessed by specifying their fully qualified names:

```
-- this query joins my_database.my_table to your_database.your_table
USE my_database;
SELECT m.*
    FROM my_table m
    JOIN your_database.your_table y
        ON m.xyz = y.xyz;
```

MariaDB has the DATABASE() function to determine the current database:

```
SELECT DATABASE();
```

Stored procedures and triggers don't inherit a default database from the session, nor by a caller procedure. In that context, the default database is the database which contains the procedure. `USE` can be used to change it. The default database will only be valid for the rest of the procedure.

The Binary Log

Different tables can be built using different storage engines. It is important to note that not all engines are transactional, and that different engines implement the transaction logs in different ways. For this reason, MariaDB cannot replicate data from a primary to a replica using an equivalent of SQL Server transactional replication.

Instead, it needs a global mechanism to log the changes that are applied to data. This mechanism is the binary log, often abbreviated to binlog.

The binary log can be written in the following formats:

- STATEMENT logs SQL statements that modify data;
- ROW logs a reference to the rows that have been modified, if any (usually it's the primary key), and the new values that have been added or modified, in a binary format.
- MIXED is a combination of the above formats. It means that ROW is used for statements that can safely be logged in this way (see below), and STATEMENT is used in other cases. This is the default format from MariaDB 10.2.

In most cases, STATEMENT is slower because the SQL statement needs to be re-executed by the replica, and because certain statements may produce a different result in the replica (think about queries that use LIMIT without ORDER BY, or the CURRENT_TIMESTAMP() function). But there are exceptions, and besides, DDL statements are always logged as STATEMENT to avoid flooding the binary log. Therefore, the binary log may well contain both ROW and STATEMENT entries.

See Binary Log Formats.

The binary log allows:

- replication, if enabled on the primary;
- promoting a replica to a primary, if enabled on that replica;
- incremental backups;
- seeing data as they were in a point of time in the past (flashback);
- restoring a backup and re-appling the binary log, with the exception of a data change which caused problems (human mistake, application bug, SQL injection);
- Capture Data Changes (CDC), by streaming the binary log to technologies like Apache Kafka.

If you don't plan to use any of these features on a server, it is possible to disable the binary log to slightly improve the performance.

The binary log can be inspected using the mariadb-binlog utility, which comes with MariaDB. Enabling or disabling the binary log requires restarting MariaDB.

See also MariaDB Replication Overview for SQL Server Users and MariaDB Backups Overview for SQL Server Users for a better understanding of how the binary log is used.

Plugins

Storage engines are a special type of plugin. But others exist. For example, plugins can add authentication methods, new features, SQL syntax, functions, informative tables, and more.

A plugin may add some server variables and some status variables. Server variables can be used to configure the plugin, and status variables can be used to monitor its activities and status. These variables generally use the plugin's name as a prefix. For example InnoDB has a server variable called innodb_buffer_pool_size to configure the size of its buffer pool, and a status variable called Innodb_pages_read which indicates the number of memory pages read from the buffer pool. The category system variables of the MariaDB Knowledge Base has specific pages for system and status variables associated with various plugins.

Many plugins are installed by default, or available but not installed by default. They can be installed or uninstalled at runtime with SQL statements, like `INSTALL PLUGIN`, `UNINSTALL PLUGIN` and others; see Plugin SQL Statements. 3rd party plugins can be made available for installation by simply copying them to the plugin_dir.

It is important to note that different plugins may have different maturity levels. It is possible to prevent the installation of plugins we don’t consider production-ready by setting the plugin_maturity system variable. For plugins that are distributed with MariaDB, the maturity level is determined by the MariaDB team based on the bugs reported and fixed.

Some plugins are developed by 3rd parties. Even some 3rd party plugins are included in MariaDB official distributions - the ones available on mariadb.org.

In MariaDB every authorization method (including the default one) is provided by an authentication plugin. A user can be required to use a certain authentication plugin. This gives us much flexibility and control. Windows users may be interested in gsapi (which supports Windows authentication, Kerberos and NTLM) and named_pipe (which uses named pipe impersonation).

Other plugins that can be very useful include userstat, which includes statistics about resources and table usage, and METADATA_LOCK_INFO, which provides information about metadata locks.

Thread Pool

MariaDB supports thread pool. It works differently on UNIX and on Windows. On Windows, it is enabled by default and its implementation is quite similar to SQL Server. It uses the Windows native CreateThreadpool API.

If we don't use the thread pool, MariaDB will use its traditional method to handle connections. It consists of using a dedicated thread for each client connection. Creating a new thread has a cost in terms of CPU time. To mitigate this cost, after a client disconnects, the thread may be preserved for a certain time in the thread cache.

Whichever connection method we use, MariaDB has a maximum number of simultaneous connections, which can be changed at runtime. When the limit is reached, if more clients try to connect they will receive an error. This prevents MariaDB from consuming all the server resources and freezing or crashing. See Handling Too Many Connections.

Configuration

MariaDB has many settings that control the server behavior. These can be set up when starting mysqld (mysqld options), and the vast majority are also accessible as server system variables. These can be classified in these ways:

- **Dynamic** or **static**;
- **Global**, **session**, or both.

Note that server system variables are not to be confused with user-defined variables. The latter are not used for MariaDB

configuration.

Configuration Files

MariaDB can use several configuration files. Configuration files are searched in several locations, including in the user directory, and if present they all are read and used. They are read in a consistent order. These locations depend on the operating system; see Default Option File Locations. It is possible to tell MariaDB which files it should read; see Global Options Related to Option Files.

On Linux, by default the configuration files are called `my.cnf`. On Windows, by default the configuration files can be called `my.ini` or `my.cnf`. The former is more common.

If a variable is mentioned multiple times in different files, the occurrence that is read last will overwrite the others. Similarly, if a variable is mentioned several times in a single file, the occurrence that is read last overwrites the others.

The contents of each configuration file are organized by *option groups*. MariaDB Server and client programs read different groups. The read groups also depend on the MariaDB version. See Option Groups for the details. Most commonly, the `[server]` or `[mysqld]` groups are used to contain all server configuration. The `[client-server]` group can be used for options that are shared by the server and the clients (like the port to use), to avoid repeating those variables multiple times.

Dynamic and Static Variables

Dynamic variables have a value that can be changed at runtime, using the SET SQL statement. Static variables have a value that is decided at startup (see below) and cannot be changed without a restart.

The Server System Variables page states if variables are dynamic or static.

Scope

A global system variable is one that affects the general behavior of MariaDB. For example innodb_buffer_pool_size determines the size of the InnoDB buffer pool, which is used by read and write operations, no matter which user issued them. A session system variable is one that affects MariaDB behavior for the current connection; changing it will not affect other connected users, or future connections from the current user.

A variable could exist in both the global and session scopes. In this case, the session value is what affects the current connection. When a user connects, the current global value is copied to the session scope. Changing the global value afterward will not change existing connections.

The Server System Variables page states the scope of each variable.

Global variables and some session variables can only be modified by a user with the SUPER privilege (typically root).

Syntax

To see the value of a system variable:

```
-- global variables:
SELECT @@global.variable_name;
-- session variables:
SELECT @@session.variable_name;
-- or just use the shortcut:
SELECT @@variable_name;
```

A longer syntax, which is mostly useful to get multiple variables, makes use of the same pattern syntax that is used by the LIKE operator:

```
-- global variables whose name starts with 'innodb':
SHOW GLOBAL VARIABLES LIKE 'innodb%';
-- session variables whose name starts with 'innodb':
SHOW SESSION VARIABLES LIKE 'innodb%';
SHOW VARIABLES LIKE 'innodb%';
```

To modify the global or session value of a dynamic variable:

```
SET @@global.variable_name = 'new 'value';
SET @@session.variable_name = 'new 'value';
```

Notice that if we modify a global variable in this way, the new value will be lost at server restart. For this reason we probably want to change the value in the configuration file too.

For further information see:

- The SET statement.
- The SHOW VARIABLES statement.

Setting System Variables with Startup Parameters

System variables can be set at server startup without writing their values into a configuration file. This is useful if we want a value to be set once, until we change it or restart MariaDB. Values passed in this way override values written in the configuration files.

The general rule is that every global variable can be passed as an argument of `mysqld` by prefixing its name with `--` and by replacing every occurrence of `_` with `-` in its name.

For example, to pass `bind_address` as a startup argument:

```
mysqld --bind-address=127.0.0.1
```

Debugging Configuration

Mistyping a variable can prevent MariaDB from starting. We cannot set a variable that doesn't exist in the MariaDB version in use. In these cases, an error is written in the error log.

Having several configuration files and configuration groups, as well as being able to pass variables as command-line arguments, brings a lot of flexibility but can sometimes be confusing. When we are unsure about which values will be used, we can run:

```
mysqld --print-defaults
```

Status Variables

MariaDB status variables and some system tables allow external tools to monitor a server, building graphs on how they change over time, and allow the user to inspect what is happening inside the server.

Status variables cannot be directly modified by the user. Their values indicate how MariaDB is operating. Their scope can be:

- **Global**, meaning that the value is about some MariaDB activity.
- **Session**, meaning that the value measures activities taking place in the current session.

Many status variables exist in both scopes. For example,Cpu_time at global level indicates how much time the CPU was used by the MariaDB process (including all user sessions and all the background threads). At session level, it indicates how much time the CPU was used by the current session.

The status variables created by a plugin, usually, use the plugin name as a prefix.

The SHOW STATUS statement prints the values of the status variables that match a certain pattern.

```
-- Show all InnoDB global status variables
SHOW GLOBAL STATUS LIKE 'innodb%';
-- Show all InnoDB session status variables
SHOW SESSION STATUS LIKE 'innodb%';
SHOW STATUS LIKE 'innodb%';
-- Show global variables that contain the "size" substring:
SHOW GLOBAL STATUS LIKE '%size%';
```

Some status variables values are reset when FLUSH STATUS is executed. A possible use:

```
DELIMITER ||
BEGIN NOT ATOMIC
SET @i = 0;
WHILE @i < 60 DO
    SHOW GLOBAL STATUS LIKE 'Com_select';
    FLUSH STATUS;
    DO SLEEP(1);
    SET @i = @i + 1;
END WHILE;
END ||
```

2.1.14.2.2 SQL Server Features Not Available in MariaDB

Contents

When planning a migration between different DBMSs, one of the most important aspects to consider is that the new database system will probably miss some features supported by the old one. This is not relevant for all users. The most widely used features are supported by most DBMSs. However, it is important to make a list of unsupported features and check which of them are currently used by applications. In most cases it is possible to implement such features on the application side, or simply stop using them.

This page has a list of SQL Server features that are not supported in MariaDB. The list is not exhaustive.

Introduced in SQL Server versions older than 2016

- Full outer joins.
- `GROUP BY CUBE` syntax.
- `MERGE` statement.
- In MariaDB, indexes are always ascending. Defining them as `ASC` or `DESC` has no effect.
  - For single-column indexes, the performance difference between an `ORDER BY ... ASC` and `DESC` is negligible.
  - For multiple-column indexes, an index may be unusable for certain queries because `DESC` is not supported. In some cases, a generated column can be used to invert the order of an index (for example, the expression `0 - price` can be indexed to index the prices in a descending order).
- The WITH syntax is currently only supported for the `SELECT` statement.
- Filtered indexes (`CREATE INDEX ... WHERE`).
- Autonomous transactions.
- User-defined types.
- Rules.
- Triggers don't support the following features:
  - Triggers on DDL and login.
  - `INSTEAD OF` triggers.
  - The `DISABLE TRIGGER` syntax.
- Cursors advanced features.
  - Global cursors.
  - `DELETE ... CURRENT OF` , `UPDATE ... CURRENT OF` statements: MariaDB cursors are read-only.
  - Specifying a direction (MariaDB cursors can only advance by one row).
- Synonyms.
- Table variables.
- Queues.
- XML indexes, XML schema collection, XQuery.
- User access to system functionalities, for example:
  - Running system commands (`xp_cmdshell()`).
  - Sending emails (`sp_send_dbmail()`).
  - Sending HTTP requests.
- External languages, external libraries (MariaDB only supports procedural SQL and PL/SQL).
- Negative permissions (the `DENY` command).
- Snapshot replication. See Provisioning a Slave.

Introduced in SQL Server 2016

- Native data masking
- PolyBase (however, MariaDB 10.5 supports accessing Amazon S3 via the S3 storage engine and several DBMSs via CONNECT)
- R and Python services
- ColumnStore indexes. MariaDB has a storage engine called ColumnStore, but this is a completely different feature.

Introduced in SQL Server 2017

- Adaptive joins
- Graph SQL

2.1.14.2.3 SQL Server Features Implemented Differently in MariaDB

Contents

Modern DBMSs implement several advanced features. While an SQL standard exists, the complete feature list is different for every database system. Sometimes different features allow achieving the same purpose, but with a different logic and different limitations. This is something to take into account when planning a migration.

Some features are implemented by different DBMSs, with a similar logic and similar syntax. But there could be important differences that users should be aware of.

This page has a list of SQL Server features that MariaDB implements in a different way, and SQL Server features for which MariaDB has an alternative feature. Minor differences are not taken into account here. The list is not exhaustive.

SQL

- The list of supported data types is different.
- There are relevant differences in transaction isolation levels.
- `SNAPSHOT` isolation level is not supported. Instead, you can use `START TRANSACTION WITH CONSISTENT SNAPSHOT` to acquire a snapshot at the beginning of the transaction. This is compatible with all isolation levels. See How Isolation Levels are Implemented in MariaDB.
- JSON support is different.

Indexes and Performance

- Clustered indexes. In MariaDB, the physical order of rows is delegated to the storage engine. InnoDB uses the primary key as a clustered index.
- Hash indexes. Only some storage engines support `HASH` indexes.
  - The InnoDB storage engine has a feature called adaptive hash index, enabled by default. It means that in InnoDB all indexes are created as `BTREE`, and depending on how they are used, InnoDB could convert them from BTree to hash indexes, or the other way around. This happens in the background.
  - The MEMORY storage engine uses hash indexes by default, if we don't specify the `BTREE` keyword.
  - See Storage Engine Index Types for more information.
- Query store. MariaDB allows query performance analysis using the slow log and performance_schema. Some open source or commercial 3rd party tools read that information to produce statistics and make it easy to identify slow queries.

Tables

- Computed columns are called generated columns in MariaDB and are created with a different syntax. See also Implementation Differences Compared to Microsoft SQL Server.
- Temporal tables use a different (more standard) syntax on MariaDB. In MariaDB, the history is stored in the same

table as current data (but optionally in different partitions). MariaDB supports both SYSTEM_TIME and APPLICATION_TIME.
- Hidden columns are Invisible columns in MariaDB.
- Temporary tables are implemented and used differently.

High Availability

- `NOT FOR REPLICATION`
  - MariaDB supports replication filters to exclude some tables or databases from replication
  - It is possible to keep a table empty in a slave (or in the master) by using the BLACKHOLE storage engine.
  - The master can have columns that are not present in a slave (the other way around is also supported). Before using this feature, carefully read the Replication When the Master and Slave Have Different Table Definitions page.
  - With MariaDB it's possible to prevent a trigger from running on slaves .
  - It's possible to run events without replicating them. The same applies to some administrative statements.
  - MariaDB superusers can run statements without replicating them, by using the sql_log_bin system variable.
  - Constraints and triggers cannot be disabled for replication, but it is possible to drop them on the slaves.
  - The `IF EXISTS` syntax allows one to easily create a table on the master that already exists (possibly in a different version) on a slave.
- pollinginterval option. See Delayed Replication.

Security

- The list of permissions is different.
- Security policies. MariaDB allows one to achieve the same results by assigning permissions on views and stored procedures. However, this is not a common practice and it's more complicated than defining security policies. See Other Uses of Views.
- MariaDB does not support an `OUTPUT` clause. Instead, we can use DELETE RETURNING and, since MariaDB 10.5, INSERT RETURNING and REPLACE RETURNING.

Other Features

- Linked servers. MariaDB supports storage engines to read from, and write to, remote tables. When using the CONNECT engine, those tables could be in different DBMSs, including SQL Server.
- Job scheduler: MariaDB uses an event scheduler to schedule events instead.

2.1.14.2.4 MariaDB Features Not Available in SQL Server

Contents

Some MariaDB features are not available in SQL Server.

At first glance, it is not important to know about those features to migrate from SQL Server to MariaDB. However, this is not the case. Using MariaDB features that are not in SQL Server allows one to obtain more advantages from the migration, getting the most from MariaDB.

This page has a list of MariaDB features that are not supported in SQL Server. The list is not exhaustive.

Plugin Architecture

- Storage engines.
- Authentication plugins.
- Encryption plugins.
- ColumnStore is a columnar storage engine designed to scale horizontally. It runs on a specific edition of MariaDB, so currently it cannot be used in combination with other engines.

SQL

- The sql_mode variable determines in which cases an SQL statement should fail with an error, and in which cases it should succeed with a warning even if it is not entirely correct. For example, when a statement tries to insert a string in a column which is not big enough to contain it, it could fail, or it could insert a truncated string and emit a warning. It is a tradeoff between reliability and flexibility.
  - SQL_MODE=MSSQL allows one to use a small subset of SQL Server proprietary syntax.
- The `CREATE ... IF EXISTS`, `CREATE OR REPLACE`, `DROP ... IF NOT EXISTS` options are supported for most DDL statements.
- SHOW statements.
- SHOW CREATE statements.
- SHOW PROCESSLIST and PERFORMANCE_SCHEMA THREAD table provide much richer information, compared to SQL Server `sp_who()` and `sp_who2()` procedures.
- CHECKSUM TABLE statement.
- PL/SQL support (only for stored procedures and stored functions).
- Row constructors.
- `BEFORE` triggers.
- HANDLER statements, to scroll table rows ordered by an index or in their physical order.
- DO statement, to call functions without returning a result set.
- BENCHMARK() function, to measure the speed of an SQL expression.

See also Syntax Differences between MariaDB and SQL Server.

Types

- Character sets and collations don't depend on column type. They can be set globally, or at database, table or column level.
- Columns may use non-constant expressions as the DEFAULT value. TIMESTAMP columns may have a `DEFAULT` value.
- UNSIGNED numeric types.
- Dynamic columns (note that JSON is usually preferred to this feature).

See also SQL Server and MariaDB Types Comparison.

JSON

For compatibility with some other database systems, MariaDB supports the JSON pseudo-type. However, it is just an alias for:

```
LONGTEXT CHECK (JSON_VALID(column_name))
```

JSON_VALID() is the MariaDB equivalent of SQL Server's `ISJSON()`.

Features

- Flashback functionality allows one to "undo" the changes that happened after a certain point in time.
- Partitioned tables support the following features:
  - Tables can be partitioned based on multiple columns.
  - Several partitioning types are available.
  - Subpartitions.
- Progress reporting for some typically expensive statements.

2.1.14.2.5 Setting Up MariaDB for Testing for SQL Server Users

Contents

This page contains links and hints to setup MariaDB for testing. The page is designed for SQL Server users, assuming that

they are mostly familiar with Windows and they are not familiar with MariaDB.

Choosing a MariaDB Version

As a general rule, for new installations it's better to choose the latest Generally Available (GA) version.

If you need a feature that is only present in a version that is not yet production-ready, and the project will surely not go to production before that version is GA, it could make sense to use a non-GA version. In this case however, keep in mind that you are using a version that is only suitable for testing.

If you need to work with an existing production instance, you should of course use the same version in testing. However, deprecated versions should not be used in production, because they could be exposed to vulnerabilities that will never be fixed. See deprecation policies if you are not sure about the version you are using.

Setting up MariaDB on Windows

There are two different ways to use MariaDB on Windows natively: using Zip packages or MSI packages.

In both cases, 32-bit platforms are still supported.

Check the page Installation issues on Windows to verify if current versions of MariaDB have troubles on Windows. More generally, it is a good idea to check the Troubleshooting Installation Issues category.

ZIP Packages

Windows users don't necessarily need to install MariaDB to use it. They can download ready-to-use ZIP packages to avoid any change in the system (except for downloading MariaDB and writing databases on the disk). This is very useful for testing without risking some undesired side effect on the machine in use. And it avoids the hassle of installing Docker or virtual machines.

> **MariaDB starting with 10.4.3**
> Starting with MariaDB 10.4.3, it is necessary to run mysql_install_db.exe to install the data directory.

The drawback is that MariaDB will need to be started and stopped from the command line.

See Installing MariaDB Windows ZIP Packages.

MSI Packages

MSI packages provide a friendly graphical interface to install MariaDB. The installation process is easy but flexible. For example, the user can decide which components to install, whether to install it as a service or not, and if networking should be enabled. An interface to uninstall MariaDB is also provided.

See Installing MariaDB MSI Packages on Windows.

Installing MariaDB on Docker

Docker is a container platform that runs natively on Linux. A Docker image is a representation of a basic Linux system, which usually runs a single process - in our case, that process is MariaDB. A container is an instance of an image, which can be created or destroyed instantaneously. Once a container is started, it can be used just like a normal system.

Docker runs on all major operating systems. On Windows and MacOS it runs on a Linux virtual machine, but this additional complexity is transparent for the end user.

Docker's characteristics makes it optimal to test MariaDB functionalities without wasting time on installation and without making changes to the host system. However, it is not ideal to test MariaDB performance.

See Installing and Using MariaDB via Docker.

Reinitializing MariaDB Data Directory

While experimenting with MariaDB, you could end up with an unusable installation. This occurs for example if you deliberately delete files that you shouldn't delete. If it happens, there is no need to uninstall and reinstall MariaDB. Instead, you can simply delete the contents of the data directory and run mariadb-install-db. The program will recreate your system tables and the essential files.

To know where your data directory is, check the datadir system variable.

2.1.14.2.6 Syntax Differences between MariaDB and SQL Server

Contents

This article contains a non-exhaustive list of syntax differences between MariaDB and SQL Server, and is written for SQL Server users that are unfamiliar with MariaDB.

Compatibility Features

Some features are meant to improve syntax and semantics compatibility between MariaDB versions, between MariaDB and MySQL, and between MariaDB and other DBMSs. This section focuses on compatibility between MariaDB and SQL Server.

sql_mode and old_mode

SQL semantics and syntax, in MariaDB, are affected by the sql_mode variable. Its value is a comma-separated list of flags, and each of them, if specified, affects a different aspect of SQL syntax and semantics.

A particularly important flag for users familiar with SQL Server is MSSQL.

sql_mode can be changed locally, in which case it only affects the current session; or globally, in which case it will affect all new connections (but not the connections already established). sql_mode must be assigned a comma-separated list of flags.

A usage example:

```
# check the current global and local sql_mode values
SELECT @@global.sql_mode;
SELECT @@session.sql_mode;
# empty sql_mode for all usaers
SET GLOBAL sql_mode = '';
# add MSSQL flag to the sql_mode for the current session
SET SESSION sql_mode = CONCAT(sql_mode, ',MSSQL');
```

old_mode is very similar to sql_mode, but its purpose is to provide compatibility with older MariaDB versions. Its flags shouldn't affect compatibility with SQL Server (though it is theoretically possible that some of them do, as a side effect).

Executable Comments

MariaDB supports executable comments. These are designed to write generic queries that are only executed by MariaDB, and optionally only certain versions.

The following examples show how to insert SQL code that will be ignored by SQL Server but executed by MariaDB, or some of its versions.

- Executed by MariaDB and MySQL (see below):

```
SELECT * FROM tab /*! FORCE INDEX (idx_a) */ WHERE a = 1 OR b = 2;
```

- Executed by MariaDB only:

```
SELECT * /*M! , @in_transaction */ FROM tab;
```

- Executed by MariaDB starting from version 10.0.5:

```
DELETE FROM user WHERE id = 100 /*!M100005 RETURNING email */;
```

As explained in the Understanding MariaDB Architecture page, MariaDB was initially forked from MySQL. At that time, executable comments were already supported by MySQL. This is why the `/*! ... */` syntax is supported by both MariaDB and MySQL. But because MariaDB also supports specific syntax not supported by MySQL, it added the `/*M! ... */` syntax.

Generic Syntax

Here we discuss some differences between MariaDB and SQL Server syntax that may affect any user, as well as some hints to make queries compatible with a reasonable amount of work.

Delimiters

SQL Server uses two different terminators:

- The *batch terminator* is the `go` command. It tells Microsoft clients to send the text we typed to SQL Server.
- The *query terminator* is a semicolon (`;`) and it tells SQL Server where a query ends.

It is rarely necessary to use `;` in SQL Server. It is required for certain common table expressions, for example.

But the same doesn't apply to MariaDB. **Normally, with MariaDB you only use `;`.**

However, MariaDB also has some situations where you want to use a `;` but you don't want the mariadb command-line client to send the query yet. This can be done in any situation, but it is particularly useful when creating stored routines or using BEGIN NOT ATOMIC.

The reason is better explained with an example:

```
CREATE PROCEDURE p()
BEGIN
    SELECT * FROM t1;
    SELECT * FROM t2;
END;
```

If we enter this procedure in this way in the `mariadb` client, as soon as we type the first `;` (after the first `SELECT`) and press enter, the statement will be sent. MariaDB will try to parse it, and will return an error.

To avoid this, `mariadb` implements the DELIMITER statement. This client statement is never sent to MariaDB. Instead, the client uses it to find out when the typed query should be sent. Let's correct the above example:

```
DELIMITER ||

CREATE PROCEDURE p()
BEGIN
    SELECT * FROM t1;
    SELECT * FROM t2;
END;

DELIMITER ;
```

Names

In MariaDB, most names have a maximum length of 64 characters. When migrating an SQL Server database to MariaDB, check if some names exceed this limit (SQL Server maximum length is 128).

By default, MariaDB names are case-sensitive if the operating system has case-sensitive file names (Linux), and case-insensitive if the operating system is case-insensitive (Windows). SQL Server is case-insensitive by default on all operating systems.

When migrating a SQL Server database to MariaDB on Linux, to avoid problems you may want to set the lower_case_table_names system variable to 1, making table names, database names and aliases case-insensitive.

Names can be quoted inside backtick characters (`` ` ``). This character can be used in names, in which case it should be doubled. By default this is the only way to quote names.

To also enable the use of double quotes (`"`), modify sql_mode adding the `ANSI_QUOTES` flag. This is the equivalent of setting QUOTED_IDENTIFIER ON in SQL Server.

To also enable the use of SQL Server style quotes (`[` and `]`), modify sql_mode adding the `MSSQL` flag.

The case-sensitivity of stored procedures and functions is never a problem, as they are case-insensitive in SQL Server.

Quoting Strings

In SQL Server, by default strings can only be quoted with single-quotes (`'`), and to use a double quote in a string it should be doubled (`''`). This also works by default in MariaDB.

SQL Server also allows to use double quotes (`"`) to quote strings. This works by default in MariaDB, but as mentioned before it won't work if sql_mode contains the `ANSI_QUOTES` flag.

NULL

The default semantics of NULL in SQL Server and MariaDB is the same, by default.

However, SQL Server allows one to change it globally with SET ANSI_NULLS OFF, or at database level with ALTER DATABASE.

There is no way to achieve exactly the same result in MariaDB. To perform NULL-safe comparisons in MariaDB, one should replace the = operator with the <=> operator.

Also, note that MariaDB doesn't support the `UNKNOWN` pseudo-value. An expression like `NULL OR 0` returns `NULL` in MariaDB.

LIKE

In MariaDB, LIKE expressions only have two characters with special meanings: `%` and `_` . These two characters have the same meanings they have in SQL Server.

The additional characters recognized by SQL Server (`[` , `]` and `^`) are part of regular expressions. MariaDB supports the REGEXP operator, that supports the full regular expressions syntax.

Data Definition Language

Here we discuss some DDL differences that database administrators will want to be aware of.

While this section is meant to highlight the most noticeable DDL differences between MariaDB and SQL Server, there are many others, both in the syntax and in the semantics. See the ALTER statement documentation.

Altering Tables Online

Altering tables online can be a problem, especially when the tables are big and we don't want to cause a disruption.

MariaDB offers the following solutions to help:

- The ALTER TABLE ... ALGORITHM clause allows one to specify which algorithm should be used to run a certain operation. For example `INPLACE` tells MariaDB not to create a table copy (perhaps because we don't have enough disk space), and `INSTANT` tells MariaDB to execute the operation instantaneously. Not all algorithms are supported for certain operations. If the algorithm we've chosen cannot be used, the ALTER TABLE statement will fail with an error.
- The ALTER TABLE ... LOCK clause allows one to specify which lock type should be used. For example `NONE` tells MariaDB to avoid any lock on the table, and `SHARED` only allows one to acquire a share lock. If the operation requires a lock that is more strict than the one we are requesting, the ALTER TABLE statement will fail with an error. Sometimes this happens because the `LOCK` level we want is not available for the specified `ALGORITHM` .

To find out which operations require a table copy and which lock levels are necessary, see InnoDB Online DDL Overview.

An `ALTER TABLE` can be queued because a long-running statement (even a `SELECT`) required a metadata lock. Since this may cause troubles, sometimes we want the operation to simply fail if the wait is too long. This can be achieved with the WAIT and NOWAIT clauses, whose syntax is a bit different from SQL Server.

SQL Server `WITH ONLINE = ON` is equivalent to MariaDB `LOCK = NONE`. However, note that most ALTER TABLE statements support `ALGORITHM = INSTANT`, which is non-blocking and much faster (almost instantaneous, as the syntax suggests).

IF EXISTS, IF NOT EXISTS, OR REPLACE

Most DDL statements, including ALTER TABLE, support the following syntax:

- `DROP IF EXISTS`: A warning (not an error) is produced if the object does not exist.
- `OR REPLACE`: If the object exists, it is dropped and recreated; otherwise it is created. This operation is atomic, so at no point in time does the object not exist.
- `CREATE IF NOT EXISTS`: If the object already exists, a warning (not an error) is produced. The object will not be replaced.

These statements are functionally similar (but less verbose) than SQL Server snippets similar to the following:

```
IF NOT EXISTS (
        SELECT name
            FROM sysobjects
            WHERE name = 'my_table' AND xtype = 'U'
    )
    CREATE TABLE my_table (
        ...
    )
go
```

Altering Columns

With SQL Server, the only syntax to alter a table column is `ALTER TABLE ... ALTER COLUMN`. MariaDB provides more ALTER TABLE commands to obtain the same result:

- CHANGE COLUMN allows one to perform any change by specifying a new column definition, including the name.
- MODIFY COLUMN allows any change, except renaming the column. This is a slightly simpler syntax that we can use when we don't want to change a column name.
- ALTER COLUMN allows one to change or drop the `DEFAULT` value.
- RENAME COLUMN allows one to only change the column name.

Using a more specific syntax is less error-prone. For example, by using `ALTER TABLE ... ALTER COLUMN` we will not accidentally change the data type.

The word `COLUMN` is usually optional, except in the case of `RENAME COLUMN`.

SHOW Statements

MariaDB supports SHOW statements to quickly list all objects of a certain type (tables, views, triggers...). Most `SHOW` statements support a `LIKE` clause to filter data. For example, to list the tables in the current database whose name begins with 'wp_':

```
SHOW TABLES LIKE 'wp\_%';
```

This is the equivalent of this query, which would work on both MariaDB and SQL Server:

```
SELECT TABLE_SCHEMA, TABLE_NAME
    FROM INFORMATION_SCHEMA.TABLES
    WHERE TABLE_NAME LIKE 'wp\_';
```

SHOW CREATE Statements

In general, for each `CREATE` statement MariaDB also supports a `SHOW CREATE` statement. For example there is a SHOW CREATE TABLE that returns the CREATE TABLE statement that can be used to recreate a table.

Though SQL Server has no way to show the DDL statement to recreate an object, `SHOW CREATE` statements are functionally similar to `sp_helptext()`.

Database Comments

MariaDB does not support extended properties. Instead, it supports a `COMMENT` clause for most CREATE and ALTER statements.

For example, to create and then change a table comment:

```
CREATE TABLE counter (
    c INT UNSIGNED AUTO_INCREMENT PRIMARY KEY
)
    COMMENT 'Monotonic counter'
;
ALTER TABLE counter COMMENT
    'Counter. It can contain many values, we only care about the max';
```

Comments can be seen with `SHOW CREATE` statements, or by querying `information_schema` tables. For example:

```
SELECT TABLE_COMMENT
    FROM information_schema.TABLES
    WHERE TABLE_SCHEMA = 'test' AND TABLE_NAME = 'counter';
+-------------------------------------------------------------------+
| TABLE_COMMENT                                                     |
+-------------------------------------------------------------------+
| Counter. It can contain many values, we only care about the max |
+-------------------------------------------------------------------+
```

Error Handling

MariaDB SHOW ERRORS and SHOW WARNINGS statements can be used to show errors, or warning and errors. This is convenient for clients, but stored procedures cannot work with the output of these commands.

Instead, inside stored procedures you can:

- Use the GET DIAGNOSTICS command to assign error properties to variables. This is the equivalent of using SQL Server functions like `ERROR_NUMBER()` or `ERROR_STATE()`.
- Add a DECLARE HANDLER block to handle all errors, a class of errors, or a specific error. This is the equivalent of SQL Server `TRY ... CATCH`.
- An error or warning can be generated on purpose using SIGNAL. Inside a `DECLARE HANDLER` block, RESIGNAL can be used to issue the error again, and interrupt the execution of the block. These are the equivalents of SQL Server `RAISERROR()`.

Administration

Administration and maintenance commands in MariaDB use different syntax to SQL Server.

- OPTIMIZE TABLE rebuilds table data and indexes. It can be considered as the MariaDB equivalent of SQL Server's `ALTER INDEX REBUILD`. See Defragmenting InnoDB Tablespaces for more information. This statement is always locking. It supports WAIT and NOWAIT syntax,
- MariaDB has an ANALYZE TABLE command, which is an equivalent of `UPDATE STATISTICS`.

BULK INSERT

MariaDB has no `BULK INSERT` statement. Instead, it supports:

- LOAD DATA INFILE to load data from files in CSV or similar formats;
- LOAD XML INFILE to load data from XML files.

See also How to Quickly Insert Data Into MariaDB.

2.1.14.2.7 SQL Server and MariaDB Types Comparison

Contents

This page helps to map each SQL Server type to the matching MariaDB type.

Numbers

In MariaDB, numeric types can be declared as `SIGNED` or `UNSIGNED`. By default, numeric columns are `SIGNED`, so not specifying either will not break compatibility with SQL Server.

When using `UNSIGNED` values, there is a potential problem with subtractions. When subtracting an `UNSIGNED` valued from another, the result is usually of an `UNSIGNED` type. But if the result is negative, this will cause an error. To solve this problem, we can enable the NO_UNSIGNED_SUBTRACTION flag in sql_mode.

For more information see Numeric Data Type Overview.

Integer Numbers

SQL Server Types	Size (bytes)	MariaDB Types	Size (bytes)	Notes
`tinyint`	1	TINYINT	1	
`smallint`	2	SMALLINT	2	
		MEDIUMINT	3	Takes 3 bytes on disk, but 4 bytes in memory
`int`	1	INT / INTEGER	4	
`bigint`	8	BIGINT	8	

Real Numbers (approximated)

SQL Server Types	Precision	Size	MariaDB Types	Size
`float(1-24)`	7 digits	4	FLOAT(0-23)	4
`float(25-53)`	15 digist	8	FLOAT(24-53)	8

MariaDB supports an alternative syntax: `FLOAT(M, D)`. M is the total number of digits, and D is the number of digits after the decimal point.

See also: Floating-point Accuracy.

Aliases

In SQL Server `real` is an alias for `float(24)`.

In MariaDB DOUBLE, and DOUBLE PRECISION are aliases for `FLOAT(24-53)`.

Normally, `REAL` is also a synonym for `FLOAT(24-53)`. However, the sql_mode variable can be set with the `REAL_AS_FLOAT` flag to make `REAL` a synonym for `FLOAT(0-23)`.

Real Numbers (Exact)

SQL Server Types	Precision	Size (bytes)	MariaDB Types	Precision	Size (bytes)
`decimal`	0 - 38	Up to 17	DECIMAL	0 - 38	See table

MariaDB supports this syntax: `DECIMAL(M, D)`. M and D are both optional. M is the total number of digits (10 by default), and D is the number of digits after the decimal point (0 by default). In SQL Server, defaults are 18 and 0, respectively. The reason for this difference is that SQL standard imposes a default of 0 for D, but it leaves the implementation free to choose any default for M.

SQL Server `DECIMAL` is equivalent to MariaDB `DECIMAL(18)`.

Aliases

The following aliases for `DECIMAL` are recognized in both SQL Server and MariaDB: `DEC`, `NUMERIC`. MariaDB also allows one to use `FIXED`.

Money

SQL Server `money` and `smallmoney` types represent real numbers guaranteeing a very low level of approximation (five decimal digits are accurate), optionally associated with one of the supported currencies.

MariaDB doesn't have monetary types. To represent amounts of money:

- Store the currency in a separate column, if necessary. It's possible to use a foreign key to a currencies table, or the ENUM type.
- Use a non-approximated type:
  - DECIMAL is very convenient, as it allows one to store the number as-is. But calculations are potentially slower.
  - An integer type is faster for calculations. It is possible to store, for example, the amount of money multiplied by 100.

There is a small incompatibility that users should be aware about. `money` and `smallmoney` are accurate to about 4 decimal digits. This means that, if you use enough decimal digits, operations on these types may produce different results than the results they would produce on MariaDB types.

Bits

The BIT type is supported in MariaDB. Its maximum size is `BIT(64)`. The `BIT` type has a fixed length. If we insert a value which requires less bits than the ones that are allocated, zero-bits are padded on the left.

In MariaDB, binary values can be written in one of the following ways:

- `b'value'`
- `0value` where `value` is a sequence of 0 and 1 digits. Hexadecimal syntax can also be used. For more details, see Binary Literals and Hexadecimal Literals.

MariaDB and SQL Server have different sets of bitwise operators. See Bit Functions and Operators.

BOOLEAN Pseudo-Type

In SQL Server, it is common to use `bit` to represent boolean values. In MariaDB it is possible to do the same, but this is not a common practice.

A column can also be defined as BOOLEAN or `BOOL`, which is just a synonym for TINYINT. `TRUE` and `FALSE` keywords also exist, but they are synonyms for 1 and 0. To understand what this implies, see Boolean Literals.

In MariaDB `'True'` and `'False'` are always strings.

Date and Time

SQL Server Types	Range	Precision	Size (bytes)	MariaDB Types	Range	Size (bytes)	Precision	Notes
`date`	0001-01-01 - 9999-12-31	3	/	DATE	0001-01-01 - 9999-12-31	3	/	They cover the same range
`datetime`	1753-01-01 - 9999-12-31	8	0 to 3, rounded	DATETIME	001-01-01 - 9999-12-31	8	0 to 6	MariaDB values are not approximated, see below.
`datetime2`	001-01-01 - 9999-12-31	8	6 to 8	DATETIME	001-01-01 - 9999-12-31	8	0 to 6	MariaDB values are not approximated, see below.
`smalldatetime`				DATETIME				
`datetimeoffset`				DATETIME				
`time`				TIME				

You may also consider the following MariaDB types:

- TIMESTAMP has little to do with SQL Server's `timestamp`. In MariaDB it is the number of seconds elapsed since the beginning of 1970-01-01, with a decimal precision up to 6 digits (0 by default). The maximum allowed value is '2038-01-19 03:14:07'. Values are always stored in UTC. A TIMESTAMP column can optionally be automatically set to the current timestamp on insert, on update, or both. It is not meant to be a unique row identifier. Also, in MariaDB the range of TIMESTAMP values is
- YEAR is a 1-byte type representing years between 1901 and 2155, as well as 0000.

Zero Values

MariaDB allows a special value where all the parts of a date are zeroes: `'0000-00-00'`. This can be disallowed by setting sql_mode=NO_ZERO_DATE.

It is also possible to use values where only some date parts are zeroes, for example `'1994-01-00'` or `'1994-00-00'`. These values can be disallowed by setting sql_mode=NO_ZERO_IN_DATE. They are not affected by `NO_ZERO_DATE`.

Syntax

Several different date formats are understood. Typically used formats are `'YYYY-MM-DD'` and `YYYYMMDD`. Several separators are accepted.

The syntax defined in standard SQL and ODBC are understood - for example, `DATE '1994-01-01'` and `{d '1994-01-01'}`. Using these eliminates possible ambiguities in contexts where a temporal value could be interpreted as a string or as an integer.

See Date and Time Literals for the details.

Precision

For temporal types that include a day time, MariaDB allows a precision from 0 to 6 (microseconds), 0 being the default. The subsecond part is never approximated. It adds up to 3 bytes. See Data Type Storage Requirements for the details.

String and Binary

Binary Strings

SQL Server Types	Size (bytes)	MariaDB Types	Notes
`binary`	1 to 8000	VARBINARY or BLOB	See below for `BLOB` types
`varbinary`	1 to 8000	VARBINARY or BLOB	See below for `BLOB` types
`image`	2^31-1	VARBINARY or BLOB	See below for `BLOB` types

The `VARBINARY` type is similar to `VARCHAR`, but stores binary byte strings, just like SQL Server `binary` does.

For large binary strings, MariaDB has four `BLOB` types, with different sizes. See BLOB and TEXT Data Types for more information.

Character Strings

One important difference between SQL Server and MariaDB is that **in MariaDB character sets do not depend on types and collations**. Character sets can be set at database, table or column level. If this is not done, the default character sets applies, which is specified by the character_set_server system variable.

To create a MariaDB table that is identical to a SQL Server table, **it may be necessary to specify a character set for each string column**. However, in many cases using UTF-8 will work.

SQL Server Types	Size (bytes)	MariaDB Types	Size (bytes)	Character set
`char`	1 to 8000	CHAR	0 to 255	`utf8mb4` (1, 4)
`varchar`	1 to 8000	VARCHAR	0 to 65,532 (2)	`utf8mb4` (1)
`text`	2^31-1	TEXT	2^31-1	`ucs2`
`nchar`	2 to 8000	CHAR	0 to 255	`utf16` or `ucs2` (3, 4)
`nvarchar`	2 to 8000	VARCHAR	0 to 65,532 (2) (5)	`utf16` or `ucs2` (1) (3)
`ntext`	2^30 - 1	TEXT	2^31-1	`ucs2`

Notes:

1) If SQL Server uses a non-unicode collation, a subset of UTF-8 is used. So it is possible to use a smaller character set on MariaDB too.

2) InnoDB has a maximum row length of 65,535 bytes. TEXT columns do not contribute to the row size, because they are stored separately (except for the first 12 bytes).

3) In SQL Server, UTF-16 is used if data contains Supplementary Characters, otherwise UCS-2 is used. If not sure, use `utf16` in MariaDB.

4) In SQL Server, the value of `ANSI_PADDING` determines if `char` values should be padded with spaces to their maximum length. In MariaDB, this depends on the PAD_CHAR_TO_FULL_LENGTH sql_mode flag.

5) See JSON, below.

SQL Server Special Types

rowversion

MariaDB does not have the `rowversion` type.

If the only purpose is to check if a row has been modified since its last read, a TIMESTAMP column can be used instead. Its default value should be `ON UPDATE CURRENT_TIMESTAMP`. In this way, the timestamp will be updated whenever the column is modified.

A way to preserve much more information is to use a temporal table. Past versions of the row will be preserved.

sql_variant

MariaDB does not support the `sql_variant` type.

MariaDB is quite flexible about implicit and explicit type conversions. Therefore, for most cases storing the values as a string should be equivalent to using `sql_variant`.

Be aware that the maximum length of an `sql_variant` value is 8,000 bytes. In MariaDB, you may need to use `TINYBLOB`.

uniqueidentifier

While MariaDB does not support the `uniqueidentifier` type, the UUID type can typically be used for the same purpose.

`uniqueidentifier` columns contain 16-bit GUIDs. MariaDB UUID columns store UUIDv1 values (128 bits).

The UUID type was implemented in MariaDB 10.7. On older versions, you can generate unique values with the UUID() or UUID_SHORT() functions, and store them in `BIT(128)` or `BIT(64)` columns, respectively.

xml

MariaDB does not support the `xml` type.

XML data can be stored in string columns. MariaDB supports several XML functions.

JSON

With SQL Server, typically JSON documents are stored in `nvarchar` columns in a text form.

MariaDB has a JSON pseudo-type that maps to LONGTEXT. However, from MariaDB 10.5 the `JSON` pseudo-type also checks that the value is valid a JSON document.

MariaDB supports different JSON functions than SQL Server. MariaDB currently has more functions, and SQL Server syntax will not work. See JSON functions for more information.

MariaDB Specific Types

The following types are supported by MariaDB and don't have a direct equivalent in SQL Server. If you are migrating your database to MariaDB, you can consider using these types.

- INET6 - IPv6 addresses.
- INET4 - IPv4 addresses.

2.1.14.2.8 MariaDB Transactions and Isolation Levels for SQL Server Users

Contents

This page explains how transactions work in MariaDB, and highlights the main differences between MariaDB and SQL Server transactions.

Note that XA transactions are handled in a completely different way and are not covered in this page. See XA Transactions.

Missing Features

These SQL Server features are not available in MariaDB:

- Autonomous transactions;
- Distributed transactions.

Transactions, Storage Engines and the Binary Log

In MariaDB, transactions are optionally implemented by storage engines. The default storage engine, InnoDB, fully supports transactions. Other transactional storage engines include MyRocks and TokuDB. Most storage engines are not transactional, therefore they should not considered general purpose engines.

Most of the information in this page refers to generic MariaDB server behaviors or InnoDB. For MyRocks and TokuDB please check the proper KnowledgeBase sections.

Writing into a non-transactional table in a transaction can still be useful. The reason is that a metadata lock is acquired on the table for the duration of the transaction, so that ALTER TABLEs are queued.

It is possible to write into transactional and non-transactional tables within a single transaction. It is important to remember that non-transactional engines will have the following limitations:

- In case of rollback, changes to non-transactional engines won't be undone. We will receive a warning `1196` which reminds us of this.
- Data in transactional tables cannot be changed by other connections in the middle of a transaction, but data in non-transactional tables can.
- In case of a crash, committed data written into a transactional table can always be recovered, but this is not necessarily true for non-transactional tables.

If the binary log is enabled, writing into different transactional storage engines in a single transaction, or writing into transactional and non-transactional engines inside the same transaction, implies some extra work for MariaDB. It needs to perform a two-phase commit to be sure that changes to different tables are logged in the correct order. This affects the performance.

Transaction Syntax

The first read or write to an InnoDB table starts a transaction. No data access is possible outside a transaction.

By default autocommit is on, which means that the transaction is committed automatically after each SQL statement. We can disable it, and manually commit transactions:

```
SET SESSION autocommit = 0;
SELECT ... ;
DELETE ... ;
COMMIT;
```

Whether autocommit is enabled or not, we can start transactions explicitly, and they will not be automatically committed:

```
START TRANSACTION;
SELECT ... ;
DELETE ... ;
COMMIT;
```

`BEGIN` can also be used to start a transaction, but does not work in stored procedures.

Read-only transactions are also available using `START TRANSACTION READ ONLY`. This is a small performance optimization. MariaDB will issue an error when trying to write data in the middle of a read-only transaction.

Only DML statements are transactional and can be rolled back. This may change in a future version, see MDEV-17567 - Atomic DDL and MDEV-4259 - transactional DDL.

Changing autocommit and explicitly starting a transaction will implicitly commit the active transaction, if any. DDL statements, and several other statements, implicitly commit the active transaction. See SQL statements That Cause an Implicit Commit for the complete list of these statements.

A rollback can also be triggered implicitly, when certain errors occur.

You can experiment with transactions to check in which cases they implicitly commit or rollback. The in_transaction system variable can help: it is set to 1 when a transaction is in progress, or 0 when no transaction is in progress.

This section only covers the basic syntax for transactions. Much more options are available. For more information, see Transactions.

Constraint Checking

MariaDB supports the following constraints:

- Primary keys
- UNIQUE
- CHECK
- Foreign keys

In some databases, constraints can temporarily be violated during a transaction, and their enforcement can be deferred to the commit time. SQL Server does not support this, and always validates data against constraints at the end of each statement.

MariaDB does something different: it always checks constraints after each row change. There are cases this policy makes some statements fail with an error, even if those statements would work on SQL Server.

For example, suppose you have an `id` column that is the primary key, and you need to increase its value for some reason:

```
SELECT id FROM customer;
+----+
| id |
+----+
|  1 |
|  2 |
|  3 |
|  4 |
|  5 |
+----+

UPDATE customer SET id = id + 1;
ERROR 1062 (23000): Duplicate entry '2' for key 'PRIMARY'
```

The reason why this happens is that, as the first thing, MariaDB tries to change 1 to 2, but a value of 2 is already present in the primary key.

A solution is to use this non-standard syntax:

```
UPDATE customer SET id = id + 1 ORDER BY id DESC;
Query OK, 5 rows affected (0.00 sec)
Rows matched: 5  Changed: 5  Warnings: 0
```

Changing the ids in reversed order won't duplicate any value.

Similar problems can happen with `CHECK` constraints and foreign keys. To solve them, we can use a different approach:

```
SET SESSION check_constraint_checks = 0;
-- run some queries
-- that temporarily violate a CHECK clause
SET SESSION check_constraint_checks = 1;

SET SESSION foreign_key_checks = 0;
-- run some queries
-- that temporarily violate a foreign key
SET SESSION foreign_key_checks = 1;
```

The last solutions temporarily disable `CHECK` constraints and foreign keys. Note that, while this may solve practical problems, it is dangerous because:

- This doesn't disable a single `CHECK` or foreign key, but also others, that you don't expect to violate.
- This doesn't defer the constraint checks, but it simply disables them for a while. This means that, if you insert some invalid values, they will not be detected.

See check_constraint_checks and foreign_key_checks system variables.

Isolation Levels and Locks

For more information about MariaDB isolation levels see SET TRANSACTION.

Locking Reads

In MariaDB, the locks acquired by a read do not depend on the isolation level (with one exception noted below).

As a general rule:

- Plain SELECTs are not locking, they acquire snapshots instead.
- To force a read to acquire a shared lock, use SELECT ... LOCK IN SHARED MODE.
- To force a read to acquire an exclusive lock, use SELECT ... FOR UPDATE.

Changing the Isolation Level

The default, the isolation level in MariaDB is `REPEATABLE READ`. This can be changed with the tx_isolation system variable.

Applications developed for SQL Server and later ported to MariaDB may run with `READ COMMITTED` without problems. Using a stricter level would reduce scalability. To use `READ COMMITTED` by default, add the following line to the MariaDB configuration file:

```
tx_isolation = 'READ COMMITTED'
```

It is also possible to change the default isolation level for the current session:

```
SET SESSION tx_isolation = 'read-committed';
```

Or just for one transaction, by issuing the following statement before starting a transaction:

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

How Isolation Levels are Implemented in MariaDB

MariaDB supports the following isolation levels:

- `READ UNCOMMITTED`
- `READ COMMITTED`
- `REPEATABLE READ`
- `SERIALIZABLE`

MariaDB isolation levels differ from SQL Server in the following ways:

- `REPEATABLE READ` does not acquire share locks on all read rows, nor a range lock on the missing values that match a `WHERE` clause.
- It is not possible to change the isolation level in the middle of a transaction.
- `SNAPSHOT` isolation level is not supported. Instead, you can use `START TRANSACTION WITH CONSISTENT SNAPSHOT` to acquire a snapshot at the beginning of the transaction. This is compatible with all isolation levels.

Here is an example of `WITH CONSISTENT SNAPSHOT` usage:

```
-- session 1
SELECT * FROM t1;
+----+
| id |
+----+
|  1 |
+----+

SELECT * FROM t2;
+----+
| id |
+----+
|  1 |
+----+

START TRANSACTION WITH CONSISTENT SNAPSHOT;

-- session 2
INSERT INTO t1 VALUES (2);

-- session 1
SELECT * FROM t1;
+----+
| id |
+----+
|  1 |
+----+

-- session 2
INSERT INTO t2 VALUES (2);

-- session 1
SELECT * FROM t2;
+----+
| id |
+----+
|  1 |
+----+
```

As you can see, session 1 uses `WITH CONSISTENT SNAPSHOT`, thus it sees all tables as they were when the transaction begun.

Avoiding Lock Waits

When we try to read or modify a row that is exclusive-locked by another transaction, our transaction is queued until that lock is released. There could be more queued transactions waiting to acquire the same lock, in which case we will wait even more.

There is a timeout for such waits, defined by the innodb_lock_wait_timeout variable. If it is set to 0, statements that encounter a row lock will fail immediately. When the timeout is exceeded, MariaDB produces the following error:

```
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
```

It is important to note that this variable has two limitations (by design):

- It only affects transactional statements, not statements like `ALTER TABLE` or `TRUNCATE TABLE`.
- It only concerns row locks. It does not put a timeout on metadata locks, or table locks acquired - for example - with the LOCK TABLES statement.

Note however that lock_wait_timeout can be used for metadata locks.

There is a special syntax that can be used with `SELECT` and some non-transactional statements including `ALTER TABLE`: the WAIT and NOWAIT clauses. This syntax puts a timeout in seconds for all lock types, including row locks, table locks, and metadata locks. For example:

```
Session 1:
START TRANSACTION;
-- let's acquire a metadata lock
SELECT id FROM t WHERE 0;

Session 2:
DROP TABLE t WAIT 0;
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
```

InnoDB Transactions

InnoDB Lock Types

InnoDB locks are classified based on what exactly they lock, and which operations they lock.

The first classification is the following:

- Record Locks lock a row or, more precisely, an index entry.
- Gap Locks lock an interval between two index entries. Note that indexes have virtual values of -Infinum and Infinum, so a gap lock can cover the gap before the first or after the last index entry.
- Next-Key Locks lock an index entry and the gap between it and the next entry. They're a combination of record locks and gap locks.
- Insert Intention Locks are gap locks acquired before inserting a new row.

Lock modes are the following:

- Exclusive Locks (X) are generally acquired on writes, e.g. immediately before deleting a row. Only one exclusive lock can be acquired on a resource simultaneously.
- Shared Locks (S) can be acquired acquired on reads. Multiple shared locks can be acquired at the same time (because the rows are not supposed to change when shared-locked) but are incompatible with exclusive locks.
- Intention locks (IS, XS) are acquired when it is not possible to acquire an exclusive lock or a shared lock. When a lock on a row or gap is released, the oldest intention lock on that resource (if any) is converted to an X or S lock.

For more information see InnoDB Lock Modes.

Information Schema

Querying the information_schema is the best way to see which transactions have acquired some locks and which transactions are waiting for some locks to be released.

In particular, check the following tables:

- INNODB_LOCKS: requests for locks not yet fulfilled, or that are blocking another transaction.
- INNODB_LOCK_WAITS: queued requests to acquire a lock.
- INNODB_TRX: information about all currently executing InnoDB transactions, including SQL queries that are running.

Here is an example of their usage.

```
-- session 1
START TRANSACTION;
UPDATE t SET id = 15 WHERE id = 10;

-- session 2
DELETE FROM t WHERE id = 10;

-- session 1
USE information_schema;
SELECT l.*, t.*
    FROM information_schema.INNODB_LOCKS l
    JOIN information_schema.INNODB_TRX t
        ON l.lock_trx_id = t.trx_id
    WHERE trx_state = 'LOCK WAIT' \G
*************************** 1. row ***************************
                    lock_id: 840:40:3:2
                lock_trx_id: 840
                  lock_mode: X
                  lock_type: RECORD
                 lock_table: `test`.`t`
                 lock_index: PRIMARY
                 lock_space: 40
                  lock_page: 3
                   lock_rec: 2
                  lock_data: 10
                     trx_id: 840
                  trx_state: LOCK WAIT
                trx_started: 2019-12-23 18:43:46
     trx_requested_lock_id: 840:40:3:2
          trx_wait_started: 2019-12-23 18:43:46
                trx_weight: 2
       trx_mysql_thread_id: 46
                 trx_query: DELETE FROM t WHERE id = 10
       trx_operation_state: starting index read
         trx_tables_in_use: 1
         trx_tables_locked: 1
          trx_lock_structs: 2
     trx_lock_memory_bytes: 1136
           trx_rows_locked: 1
         trx_rows_modified: 0
   trx_concurrency_tickets: 0
       trx_isolation_level: REPEATABLE READ
         trx_unique_checks: 1
    trx_foreign_key_checks: 1
trx_last_foreign_key_error: NULL
          trx_is_read_only: 0
trx_autocommit_non_locking: 0
```

Deadlocks

InnoDB detects deadlocks automatically. Since this consumes CPU time, some users prefer to disable this feature by setting the innodb_deadlock_detect variable to 0. If this is done, locked transactions will wait until the they exceed the innodb_lock_wait_timeout. Therefore it is important to set innodb_lock_wait_timeout to a very low value, like 1.

When InnoDB detects a deadlock, it kills the transaction that modified the least amount of data. The client will receive the following error:

```
ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting transaction
```

The latest detected deadlock, and the killed transaction, can be viewed in the output of SHOW ENGINE InnoDB STATUS. Here's an example:

```
------------------------
LATEST DETECTED DEADLOCK
------------------------
2019-12-23 18:55:18 0x7f51045e3700
*** (1) TRANSACTION:
TRANSACTION 847, ACTIVE 10 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 4 lock struct(s), heap size 1136, 3 row lock(s), undo log entries 1
MySQL thread id 46, OS thread handle 139985942054656, query id 839 localhost root Updating
delete from t where id = 10
*** (1) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 40 page no 3 n bits 80 index PRIMARY of table `test`.`t` trx id 847
lock_mode X locks rec but not gap waiting
Record lock, heap no 2 PHYSICAL RECORD: n_fields 3; compact format; info bits 32
 0: len 4; hex 8000000a; asc     ;;
 1: len 6; hex 00000000034e; asc      N;;
 2: len 7; hex 760000019c0495; asc v      ;;

*** (2) TRANSACTION:
TRANSACTION 846, ACTIVE 25 sec starting index read
mysql tables in use 1, locked 1
3 lock struct(s), heap size 1136, 2 row lock(s), undo log entries 1
MySQL thread id 39, OS thread handle 139985942361856, query id 840 localhost root Updating
delete from t where id = 11
*** (2) HOLDS THE LOCK(S):
RECORD LOCKS space id 40 page no 3 n bits 80 index PRIMARY of table `test`.`t` trx id 846
lock_mode X locks rec but not gap
Record lock, heap no 2 PHYSICAL RECORD: n_fields 3; compact format; info bits 32
 0: len 4; hex 8000000a; asc     ;;
 1: len 6; hex 00000000034e; asc      N;;
 2: len 7; hex 760000019c0495; asc v      ;;

*** (2) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 40 page no 3 n bits 80 index PRIMARY of table `test`.`t` trx id 846
lock_mode X locks rec but not gap waiting
Record lock, heap no 3 PHYSICAL RECORD: n_fields 3; compact format; info bits 32
 0: len 4; hex 8000000b; asc     ;;
 1: len 6; hex 00000000034f; asc      O;;
 2: len 7; hex 770000019d031d; asc w      ;;

*** WE ROLL BACK TRANSACTION (2)
```

The latest detected deadlock never disappears from the output of `SHOW ENGINE InnoDB STATUS`. If you cannot see any, MariaDB hasn't detected any InnoDB deadlocks since the last restart.

Another way to monitor deadlocks is to set innodb_print_all_deadlocks to 1 (0 is the default). InnoDB will log all detected deadlocks into the error log.

2.1.14.2.9 MariaDB Authorization and Permissions for SQL Server Users

Contents

Understanding Accounts and Users

MariaDB authorizes access and check permissions on accounts, rather than users. Even if MariaDB supports standard SQL commands like CREATE USER and DROP USER, it is important to remember that it actually works with accounts.

An account is specified in the format `'user'@'host'`. The quotes are optional and allow one to include special characters, like dots. The host part can actually be a pattern, which follows the same syntax used in `LIKE` comparisons. Patterns are often convenient because they can match several hostnames.

Here are some examples.

Omitting the host part indicates an account that can access from any host. So the following statements are equivalent:

```
CREATE USER viviana;
CREATE USER viviana@'%';
```

However, such accounts may be unable to connect from localhost if an anonymous user `''@'%'` is present. See localhost and % for the details.

Accounts are not bound to a specific database. They are global. Once an account is created, it is possible to assign it permissions on any existing or non existing database.

The sql_mode system variable has a NO_AUTO_CREATE_USER flag. In recent MariaDB versions it is enabled by default. If it is not enabled, a GRANT statement specifying privileges for a non-existent account will automatically create that account.

For more information: Account Management SQL Commands.

Setting or Changing Passwords

Accounts with the same username can have different passwords.

By default, an account has no password. A password can be set, or changed, in the following way:

- By specifying it in CREATE USER.
- By the user, with SET PASSWORD.
- By root, with `SET PASSWORD` or ALTER USER.

With all these statements (`CREATE USER` , `ALTER USER` , `SET PASSWORD`) it is possible to specify the password in plain or as a hash:

```
-- specifying plain passwords:
CREATE USER tom@'%.example.com' IDENTIFIED BY 'plain secret';
ALTER USER tom@'%.example.com' IDENTIFIED BY 'plain secret';
SET PASSWORD = 'plain secret';
-- specifying hashes:
CREATE USER tom@'%.example.com' IDENTIFIED BY PASSWORD 'secret hash';
ALTER USER tom@'%.example.com' IDENTIFIED BY PASSWORD 'secret hash';
SET PASSWORD = PASSWORD('secret hash');
```

The PASSWORD() function uses the same algorithm used internally by MariaDB to generate hashes. Therefore it can be used to get a hash from a plain password. Note that this function should not be used by applications, as its output may depend on MariaDB version and configuration.

`SET PASSWORD` applies to the current account, by default. Superusers can change other accounts passwords in this way:

```
SET PASSWORD FOR tom@'%.example.com' = PASSWORD 'secret hash';
```

MariaDB starting with 10.4.3

Passwords can have an expiry date, set by default_password_lifetime. To set a different date for a particular user:

```
CREATE USER 'tom'@'%.example.com' PASSWORD EXPIRE INTERVAL 365 DAY;
```

To set no expiry date for a particular user:

```
CREATE USER 'tom'@'%.example.com' PASSWORD EXPIRE NEVER;
```

For more details, see User Password Expiry.

MariaDB starting with 10.4.2

It is also possible to lock an account with immediate effect:

```
CREATE USER 'tom'@'%.example.com' ACCOUNT LOCK;
```

See Account Locking for more details.

Authentication Plugins

MariaDB supports authentication plugins. These plugins implement user's login and authorization before they can use MariaDB.

Each user has one or more authentication plugins assigned. The default one is mysql_native_password. It is the traditional login using the username and password set in MariaDB, as described above.

MariaDB starting with 10.4

On UNIX systems, root is also assigned the unix_socket plugin, which allows a user logged in the operating system to be recognized by MariaDB.

Windows users may be interested in the named pipe and GSSAPI plugins. GSSAPI also requires the use of a plugin on the client side.

A plugin can be assigned to a user with `CREATE USER`, `ALTER USER` or `GRANT`, using the `IDENTIFIED VIA` syntax. For example:

```
CREATE USER username@hostname IDENTIFIED VIA gssapi;
GRANT SELECT ON db.* TO username@hostname IDENTIFIED VIA named_pipe;
```

TLS connections

A particular user can be required to use TLS connections. Additional requirements can be set:

- Having a valid X509 certificate.
- The certificate may be required to be issued by a particular authority.
- A particular certificate subject can be required.
- A particular certificate cipher suite can be required.

These requirements can be set with `CREATE USER`, `ALTER USER` or `GRANT`. For the syntax, see CREATE USER.

MariaDB can be bundled with several cryptography libraries, depending on its version. For more information about the libraries, see TLS and Cryptography Libraries Used by MariaDB.

For more information about secure connections, see Secure Connections Overview.

Permissions

Permissions can be granted to accounts. As mentioned before, the specified accounts can actually be patterns, and multiple accounts may match a pattern. For example, in this example we are creating three accounts, and we are assigning permissions to all of them:

```
CREATE USER 'tom'@'example.com';
CREATE USER 'tom'@'123.123.123.123;
CREATE USER 'tom'@'tomlaptop';
GRANT USAGE ON *.* TO tom@'%';
```

The following permission levels exist in MariaDB:

- Global privileges;
- Database privileges;
- Table privileges;
- Column privileges;
- Function and procedure privileges.

Note that database and schema are synonymous in MariaDB.

Permissions can be granted for non-existent objects that could exist in the future.

The list of supported privileges can be found in the GRANT page. Some highlights can be useful for SQL Server users:

- `USAGE` privilege has no effect. The `GRANT` command fails if we don't grant at least one privilege; but sometimes we want to run it for other purposes, for example to require a user to use TLS connections. In such cases, it is useful to grant `USAGE`.
- Normally we can obtain a list of all databases for which we have at least one permission. The `SHOW DATABASES` permission allows getting a list of all databases.
- There is no `SHOWPLAN` privilege in MariaDB. Instead, EXPLAIN requires the `SELECT` privilege for each accessed table and the `SHOW VIEW` privilege for each accessed view.
- The same permissions are needed to see a table structure (`SELECT`) or a view definition (`SHOW VIEW`).
- `REFERENCES` has no effect.

MariaDB does not support negative permissions (the `DENY` command).

Some differences concerning the SQL commands:

- In MariaDB `GRANT` and `REVOKE` statements can only assign/revoke permissions to one user at a time.
- While we can assign/revoke privileges at column level, we have to run a `GRANT` or `REVOKE` statement for each column. The `table (column_list)` syntax is not recognized by MariaDB.
- In MariaDB it is not needed (or possible) to specify a class type.

Roles

MariaDB supports roles. Permissions can be assigned to roles, and roles can be assigned to accounts.

An account may have zero or one default roles. A default role is a role that is automatically active for a user when they connect. To assign an account or remove a default role, these SQL statements can be used:

```
SET DEFAULT ROLE some_role FOR username@hostname;
SET DEFAULT ROLE NONE FOR username@hostname;
```

Normally a role is not a default role. If we assign a role in this way:

```
GRANT some_role TO username@hostname;
```

...the user will not have that role automatically enabled. They will have to enable it explicitly:

```
SET ROLE some_role;
```

MariaDB does not have predefined roles, like public.

For an introduction to roles, see Roles Overview.

2.1.14.2.10 Repairing MariaDB Tables for SQL Server Users

Contents

Repairing tables in MariaDB is not similar to repairing tables in SQL Server.

The first thing to understand is that every MariaDB table is handled by a storage engine. Storage engines are plugins that know how to physically read and write a table, so each storage engine allows one to repair tables in different ways. The default storage engine is InnoDB.

MariaDB provides specific SQL statements to deal with corrupted tables:

- CHECK TABLE checks if a table is corrupted;
- REPAIR TABLE repairs a table if it is corrupted.

As a general rule, there is no reason why a table that is corrupted on a master should also be corrupted on the slaves. Therefore, `REPAIR` is generally used with the `NO_WRITE_TO_BINLOG` option, to avoid replicating it to the slaves.

Partitioned Tables

Partitioned tables are normally split into multiple physical files (one per partition). Even if one of the partitions is corrupted, in most cases other partitions are healthy.

For this reason, `CHECK TABLE` and `REPAIR TABLE` don't work on partitioned tables. Instead, use ALTER TABLE to check or repair a single partition.

For example:

```
ALTER TABLE orders CHECK PARTITION p_2019, p_2020;
ALTER TABLE orders REPAIR PARTITION p_2019, p_2020;
```

Indexes

Indexes can get corrupted. However, as long as data is not corrupted, indexes can always be dropped and rebuilt with ALTER TABLE:

```
ALTER TABLE customer DROP INDEX idx_email;
ALTER TABLE customer ADD INDEX idx_email (email);
```

Checking and Repairing Tables

Here we discuss how to repair tables, depending on the storage engine.

InnoDB

InnoDB follows the "fail fast" philosophy. If table corruption is detected, by default InnoDB deliberately causes MariaDB to crash to avoid corruption propagation, logging an error into the error log. This happens even if the corruption is found with a `CHECK TABLE` statement. This behavior can be changed with the innodb_corrupt_table_action server variable.

To repair an InnoDB table after a crash:

1. Restart MariaDB with the `--innodb-force-recovery` option set to a low but non-zero value.
2. If MariaDB fails to start, retry with a higher value. Repeat until you succeed.

At this point, you can follow two different procedures, depending if you can use a backup or not. Provided that you have a usable backup, it is often the best option to bring the database up quickly. But if you want to reduce the data loss as much as possible, you prefer to follow the second method.

Restoring a backup:

1. Drop the whole database with DROP DATABASE.
2. Restore a backup of the database. The exact procedure depends on the type of backup.

Recovering existing data:

1. Dump data from the corrupter table, ordered by primary key. MariaDB could crash when it finds damaged data. Repeat the process skipping damaged data.
2. Save somewhere the table structure with SHOW CREATE TABLE.
3. Restart MariaDB.
4. Drop the table with DROP TABLE.
5. Recreate the table and restore the dump.

For more details, see InnoDB Recovery Modes.

Aria and MyISAM

MyISAM is not crash-safe. In case of a MariaDB crash, the changes applied to MyISAM tables but not yet flushed to the disk are lost.

Aria is crash-safe by default, which means that in case of a crash, after repairing any table that is damaged, no changes are lost. However, Aria tables are not crash-safe if created with `TRANSACTIONAL=0` or `ROW_FORMAT` set to `FIXED` or `DYNAMIC`.

System tables use the Aria storage engine and they are crash-safe.

To check if a MyISAM/Aria table is corrupted, we can use CHECK TABLE. To repair a MyISAM/Aria table, one can use REPAIR TABLE. Before running `REPAIR TABLE` against big tables, consider increasing myisam_repair_threads or aria_repair_threads.

MyISAM and Aria tables can also be automatically repaired when corruption is detected. This is particularly useful for Aria, in case corrupted system tables prevent MariaDB from starting. See myisam_recover_options and aria_recover_options. By default Aria runs the quickest repair type. Occasionally, to repair a system table, we may have to start MariaDB in this way:

```
mysqld --aria-recover-options=BACKUP,FORCE
```

It is also possible to stop MariaDB and repair MyISAM tables with myisamchk, and Aria tables with aria_chk. With default values, a repair can be unnecessarily very slow. Before running these tools, be sure to check the Memory and Disk Use

With myisamchk page.

Other Storage Engines

Notes on the different storage engines:

- For MyRocks, see MyRocks and CHECK TABLE.
- With ARCHIVE, `REPAIR TABLE` also improves the compression rate.
- For CSV, see Checking and Rpairing CSV Tables.
- Some special storage engines, like MEMORY or BLACKHOLE, do not support any form of check and repair.

2.1.14.2.11 MariaDB Backups Overview for SQL Server Users

Contents

MariaDB has the following types of backups:

- Logical backups (dumps).
- Hot backups with Mariabackup.
- Snapshots.
- Incremental backups.

Logical Backups (Dumps)

A *dump*, also called a *logical backup*, consists of the SQL statements needed to recreate MariaDB databases and their data into another server. A dump is the slowest form of backup to restore, because it implies executing all the SQL statements needed to recreate data. However it is also the most flexible, because restoring will work on any MariaDB version, because the SQL syntax is usually compatible. It is even possible to restore a dump into an older version, though the incompatible syntax (new features) will be ignored. Under certain conditions, MariaDB dumps may also be restored on other DBMSs, including SQL Server.

The compatibility between different versions and technologies is achieved by using executable comments, but we should be aware of how they work. If we use a feature introduced in version 11.1, for example, it will be included in the dump inside an executable comment. If we restore that backup on a server with MariaDB 10.11, the 11.1 feature will be ignored. This is the only way to restore backups in older MariaDB versions.

mariadb-dump

Logical backups are usually taken with mariadb-dump (previously called mysqldump).

mariadb-dump allows one to dump all databases, a single database, or a set of tables from a database. It is even possible to specify a `WHERE` clause, which under certain circumstances allows to obtain incremental dumps.

For consistency reasons, when using the default storage engine InnoDB, it is important to use the `--single-transaction` option. This will read all data in a single transaction. It's important however to understand that long transactions may have a big impact on performance.

The `--master-data` option adds the statements to setup a slave to the dump.

MariaDB also supports statements which make easy to write applications to obtain custom types of dumps. For most `CREATE <object_type>` statement, a corresponding `SHOW CREATE <object_type>` exists. For example, SHOW CREATE TABLE returns the `CREATE TABLE` statement that can be used to recreate a certain table, without data.

mydumper

mydumper is a 3rd party tools to take dumps from MariaDB and MySQL databases. It is much faster than mariadb-dump because it takes backups with several parallel threads, usually one thread for each available CPU core. It produces several files, that can be used to restore a database using the related tool myloader.

Since is it a 3rd party tool, it could be incompatible with some present or future MariaDB features.

Hot Backups (mariabackup)

Mariabackup is a tool for taking a backup of MariaDB files while MariaDB is working. A lock is only held for a small amount of time, so it is suitable to backup a server without causing disruptions. It works by taking corrupted backups and then bringing them to a consistent state by using the InnoDB undo log. Mariabackup also properly backups MyRocks tables and non-transactional storage engines.

Cold Backups and Snapshots

A copy of all MariaDB files is a working backup. Therefore, the easiest way to backup a dataset is to shutdown the server and copy all its files. It will be entirely possible to start another server with a copy of those files. This is often referred to as a *cold backup*. However, in most cases we don't want to do this, because it implies downtime for the server: it will not be working at least for the time necessary to copy the files.

Snapshots are usually a better idea, as they are a consistent copy of the files at a given moment in time, taken without stopping the normal operations.

A snapshot of the files can be taken at several levels: filesystem level, if the filesystem supports snapshots, for example zfs; Linux Logical Volume Manager (LVM) also supports snapshots; and virtual machines also allow one to take snapshots. Windows shadow copies are also snapshots, with a benefit: it is possible to restore a single file from a shadow copy. A snapshot is not an expensive operation, because it does not imply a copy of the files. The current files will not be modified anymore, and changes to them will be written in separate places.

The problem with snapshots is that they behave like a logical copy of the files as they are in a given point in time. But database files are not guaranteed to be consistent in every moment, because contents can be buffered before being flushed to the disk. You can think a database snapshot like a database after an operating system crash.

With non-transactional tables, some data is typically lost. Data changes that are present in a buffer before the snapshot, but not written on a disk, cannot be recovered in any way. Data changes in transactional tables, like InnoDB tables, can always be recovered after restoring a snapshot (or after a crash), as long as a commit was done. Tables will still need to be repaired, just like it happens after an SQL Server crash.

Snapshots can be taken while MariaDB is running. To restore them, stop MariaDB first - or kill the process, because you don't really care of the consequences in this case. Then restore a snapshot and start MariaDB again.

For more information about snapshots, check your filesystem, LVM or virtual machine documentation.

Incremental Backups

The term incremental backup in MariaDB indicates what SQL Server calls a *differential backup*. An important difference is that in SQL Server such backups are based on the transaction log, which wouldn't be possible in MariaDB because transaction logs are handled at storage engine level.

As mentioned here, MariaDB can use the binary log instead for backup purposes. Such incremental backups can be done manually. This means that:

- The binary log files are copied just like any other regular file.
- To copy those files it is necessary to have the proper permissions at filesystem level, not in MariaDB.
- Backups do not expire until we delete the last needed complete backup.

Replaying the Binary Log

The page Using mariadb-binlog shows how to use the mariadb-binlog utility to replay a binary log file.

The page also shows how to edit the binary log before replaying it. This allows one to undo an SQL statement that was executed by mistake, for example a `DROP TABLE` against a wrong table. The high level procedure is the following:

- Restore a backup that is older than the SQL statement to undo.
- Use `mariadb-binlog` to generate a file with the SQL statements that were executed after the backup.
- Edit the SQL file, erasing the unwanted statement.
- Run the SQL file.

Incremental Backups with mariabackup

The simplest way to take an incremental backup is to use Mariabackup. This tool is able to take and restore incremental backups. For the complete procedure to use, see Incremental Backup and Restore with Mariabackup.

Mariabackup can run on both Linux and Windows systems.

Flashback

Flashback is a feature that allows one to bring all databases, some databases or some tables back to a certain point in time. This can only be done if the binary log is enabled. Flashback is not a proper backup, but it can be used to restore a certain set of data.

Copying Individual Tables

It is entirely possible to restore a single table from a physical backup, or to copy the table to another server.

With the MyISAM storage engine it was very easy to move tables between different servers, as long as the MySQL or MariaDB version was the same.

InnoDB is nowadays the default storage engine, and it is more complex, as it supports transactions for example. It still supports restoring a table from a physical file, this feature is called *transportable tablespaces*. There is a particular procedure to follow, and some limitations. This is basically the MariaDB equivalent of detaching and re-attaching tables in SQL Server.

For more information, see InnoDB File-Per-Table Tablespaces.

By default. all table files are located in the *data directory*, which is defined by the system variable datadir. There may be exceptions, because a table's files can be located elsewhere using the `DATA DIRECTORY` and `INDEX DIRECTORY` options in `CREATE TABLE`.

Regardless of the storage engine used, each table's structure is generally stored in a file with the `.frm` extension.

The files used for partitioned tables are different from the files used for non-partitioned tables. See Partitions Files for details.

2.1.14.2.12 MariaDB Replication Overview for SQL Server Users

Contents

MariaDB supports the following types of replication:

- Asynchronous replication.
- Semi-synchronous replication.
- Galera Cluster.

MariaDB starting with 10.5.1

Note: in the snippets in this page, several SQL statements use the keyword `SLAVE`. This word is considered inappropriate by some persons or cultures, so from MariaDB 10.5 it is possible to use the `REPLICA` keyword, as a synonym.

Similar synonyms will be created in the future for status variables and system variables. See MDEV-18777 to track

the status of these changes.

Asynchronous Replication

The original MariaDB replication system is asynchronous primary-replica replication.

A primary needs to have the binary log enabled. The primary logs all data changes in the binary log. Every *event* (a binary log entry) is sent to all the replicas.

For a high-level description of the binary log for SQL Server users, see Understanding MariaDB Architecture.

The events can be written in two formats: as an SQL statement (*statement-based replication*, or SBR), or as a binary representation of the change (*row-based replication*, or RBR). The former is generally slower, because the statement needs to be re-executed by the replicas. It is also less reliable, because some SQL statements are not deterministic, so they could produce different results on the replicas. On the other hand row-based replication could make the binary log much bigger, and require more network traffic. For this reason, DML statements are always logged in statement format.

For more details on replication formats, see binary log formats.

The replicas have an I/O thread that receives the binary log events and writes them to the relay log. These events are then read by the SQL thread. This thread could directly apply the changes to the local databases, and this was the only option before MariaDB 10.0.5. If parallel replication is enabled, the SQL thread hands the events to the worker thread, that apply them to the databases. The latter method is recommended for performance reasons.

When a replica cannot apply an event to the local data, the SQL thread stops. This happens, for example, if the event is a row deletion but that row doesn't exist on the replica. There can be several reasons for this, for example non-deterministic statements, or a user deleted the row in the replica. To reduce the risk, it is recommended to set read_only to 1 in the replicas.

SHOW SLAVE STATUS has columns named `Slave_SQL_State` and `Slave_IO_State` that show, respectively, if the SQL thread and the IO thread are running. If they are not, the column `Last_IO_Errno` and `Last_IO_Error` (for the IO thread) or `Last_SQL_Errno` and `Last_SQL_Error` (for the SQL thread) show what the problem is.

In a replication chain, every server must have a unique server_id.

For more information on replication, see standard replication.

Binary Log Coordinates, Relay Log Coordinates and GTID

The binary log coordinates provide a way to identify a certain data change made by a server. Coordinates consist of a file name and the position of the latest event, expressed as an integer. The last event coordinates can be seen with the SHOW MASTER STATUS columns `File` and `Position`. mariadb-dump includes them in a dump if the `--master-data` option is used.

A replica uses primary binary log coordinates to identify the last event it read. This can be seen with the SHOW SLAVE STATUS columns `Master_Log_File` and `Read_Master_Log_Pos`.

The columns `Relay_Master_Log_File` and `Exec_Master_Log_Pos` identify the primary event that corresponds to the last event applied by the SQL thread.

The replica relay log also has coordinates. The coordinates of the last applied event can be seen with the `SHOW SLAVE STATUS` columns `Relay_Log_File` and `Relay_Log_Pos`.

To easily find out how far the replica is lagging behind the primary, we can look at `Seconds_Behind_Master`.

Coordinates represented in this way have a problem: they are different on each server. Each server can use files with different (or the same) names, depending on its configuration. And files can be rotated at different times, including when a user runs FLUSH LOGS. By enabling the GTID (global transaction id) an event will have the same id on the primary and on all the replicas.

When GTID is enabled, `SHOW SLAVE STATUS` shows two GTIDs: `Gtid_IO_Pos` is the last event written into the relay log, and `Gtid_Slave_Pos` is the last event applied by the SQL thread. There is no need for a column identifying the same event in the primary, because the id is the same.

Provisioning a Replica

MariaDB does not have an equivalent to SQL Server's snapshot replication.

To setup a replica, it is necessary to manually provision it. It can be provisioned from the primary in this way:

- A backup from the primary must be restored on the new replica;
- The binary log coordinates at the moment of the backup should be set as replication coordinates in the replica, via

CHANGE MASTER TO.

However, if there is at least one existing replica, it is better to use it to provision the new replica:

- A backup from the existing replica must be restored in the new replica;
- The backup should include the system tables. In this way it will not be necessary to set the correct coordinates manually.

For more information see Setting Up Replication and Setting up a Replica with Mariabackup.

Replication and Permissions

A replica connects to a primary using its credentials. See CHANGE MASTER TO.

The appropriate account must be created in the primary, and it needs to have the `REPLICATION SLAVE` permission.

See Setting Up Replication for more information.

Parallel Replication and Group Commit

MariaDB uses group commit, which means that a group of events are physically written in the binary log altogether. This reduces the number of IOPS (input/output operations per second). Group commit cannot be disabled, but it can be tuned with variables like binlog_commit_wait_count and binlog_commit_wait_usec.

Replicas can apply the changes using multiple threads. This is known as parallel replication. Before MariaDB 10.0.5 only one thread was used to apply changes. Since a primary can use many threads to write data, mono-thread replication is a well-known bottleneck. Parallel replication is not enabled by default. To use it, set the slave_parallel_threads variable to a number greater than 1. If replication is running, the replica threads must be stopped in order to change this value:

```
STOP SLAVE SQL_THREAD;
SET GLOBAL slave_parallel_threads = 4;
START SLAVE SQL_THREAD;
```

There are different parallel replication styles available: in-order and out-of-order. The exact mode in use is determined by the slave_parallel_mode system variable. In parallel replication, the events are not replicated exactly in the same order as they occurred in the primary. But with an in-order replication mode the commit phase is always applied simultaneously. In this way data in the replica always reflect data as they have been in the primary at a certain point in time. Out-of-order replication is faster because there is less queuing, but it's not completely consistent with the primary. If two transactions modified different sets of rows in the primary, they could become visible in the replica in a different order.

`conservative` relies on primary group commit: events in different groups are executed in a parallel way.

`optimistic` does not try to find out which transaction can be executed in a parallel way - except for transactions that conflicted on the primary. Instead, it always tries to apply many events together, and rolls transactions back when there is a conflict.

`aggressive` is similar to optimistic, but it does not take into account which transactions conflicted in the primary.

`minimal` applies commits together, but all other events are applied in order.

Out-of-order replication cannot be enabled automatically by changing a variable in the replica. Instead, it must be enabled by the applications that run transactions in the primary. They can do this if the GTID is enabled. They can set different values for the gtid_domain_id variable in different transactions. This shifts a lot of responsibility to the application layer; however, if the application is aware of which transactions are not going to conflict and this information allows one to sensibly increase the parallelism, and using out-of-order replication can be a good idea.

Even if out-of-order replication is not normally used, it can be a good idea to use it for long running transactions or ALTER TABLEs, so they can be applied at the same time as normal operations that are not conflicting.

The impact of the number of threads and mode on performance can be partly seen with SHOW PROCESSLIST, which shows the state of all threads. This includes the replication worker threads, and shows if they are blocking each other.

Differences Between the Primary and the Replicas

As a general rule, we want the primary and the replicas to contain exactly the same data. In this way, no conflicts are possible. Conflicts are the most likely cause of replication outages.

To reduce the possible causes of conflicts, the following best practices are recommended:

- Users must not change data in the replica directly. Set read_only to 1. Note that this won't prevent root from making changes.
- Use the same table definitions in the primary and in the replica.
- Use `ROW` binary log format on the primary.

Another cause of inconsistencies include MariaDB bugs and failover in case the primary crashes.

An open source third party tool is available to check if the primary and a replica are consistent. It is called pt-table-checksum. Another tool, pt-table-sync, can be used to eliminate the differences. Both are part of Percona Toolkit. The advice is to run pt-table-checksum periodically, and use pt-table-sync if inconsistencies are found.

If a replication outage occurs because an inconsistency is found, sometimes we want to quickly bring the replica up again as quickly as possible, and solve the core problem later. If GTID is not used, a way to do this is to run SET GLOBAL SQL_SLAVE_SKIP_COUNTER = 1, which skips the problematic replication event.

If GTID is used, the gtid_slave_pos variable can be used instead. See the link for an explanation of how it works.

There are ways to have different data on the replicas. For example:

- Multi-source replication is possible. In this way, a replica will replicate data from multiple primaries. This feature is described below.
- Replication filters are supported. This allows one to exclude or include in replication specific tables, entire databases, or tables whose name matches a certain pattern. This allows one to avoid replicating data that is present in the primary but can always be rebuilt.
- Differences in table definitions are also possible. For example, a replica could have more columns or less columns compared to the primary. In this way we can avoid replicating columns whose values can be rebuilt. Or we can add columns for analytics purposes, without having them in the primary. Be sure to understand the limitations and risks of this technique.

Delayed Replication

MariaDB supports delayed replication. This is the equivalent of setting a `pollinginterval` in SQL Server.

To delay replication in a MariaDB replica, use CHANGE MASTER TO to specify a delay in seconds.

For more information, see Delayed Replication.

Multi-Source Replication

Multi-source replication is an equivalent to peer-to-peer replication, available in SQL Server Enterprise Edition.

A MariaDB replica can replicate from any number of primaries. It is very important that different primaries don't have the same tables. Otherwise there could be conflicts between data changes made on different primaries, and this will result in a replication outage.

In multi-source replication different channels exist, one for each primary.

This changed the way SQL replication statements work. SHOW PROCESSLIST returns a different row for each channel. Several statements, like CHANGE MASTER TO, START SLAVE or STOP SLAVE. accept a parameter which specifies which replication channel they should affect. For example, to stop a channel called `wp1`:

```
STOP SLAVE "wp1";
```

Furthermore, variables that affect parallel replication can be prefixed with a channel name. This allow one to only use parallel replication for certain channels, or to tune it differently for each channel. For example, to enable parallel replication on a channel called `wp1`:

```
SET GLOBAL wp1.slave_parallel_threads = 4;
```

Dual Primary

It is possible to configure two servers in a way that each of them acts as a primary for the other server.

In this way, data could theoretically be inserted into any of these servers, and will then be replicated to the other server. However, in such a configuration conflicts are very likely. So it is impractical to use this technique to scale writes.

A dual primary (or primary-primary) configuration however can be useful for failover. In this case we talk about an *active primary* that receives reads and writes from the clients, and a *passive primary* that is not used until the active primary crashes.

Several problems should be considered in this scenario:

- If the active primary crashes, it is very possible that the passive primary did not receive all events yet, because replication is asynchronous. If the primary data are lost (for example because the disk is damaged), some data are also lost.
- If data is not lost, when we bring the primary up again, the latest events will be replicated by the other server. There could be conflicts that will break replication.

- When is the active primary considered down? Even if a server cannot reach it, the active primary could be running and it could be able to communicate with the passive primary. Switching the clients to the passive primary could lead to unnecessary problems. It is a good idea to always check `SHOW SLAVE STATUS` to be sure that the two primary are not communicating.
- If we want to have more replicas, we should attach some of them to the active primary, and some of them to the passive primary. The reason is that when a server crashes, its replicas stop receiving any data. Failover is still possible, but it's better to have some servers that will not need any failover.

A safe primary-primary configuration where both servers accept writes, however, is possible. This is the case is data never conflicts. For example, the two servers could accept writes on different databases. We will have to decide what should happens in case of a server crash:

- Writes can be stopped until the server is up again. Reads can be sent to the other server, but keep in mind that the most recently written data could be missing.
- Both writes and reads can failover to the other server. All the problems mentioned above may apply to this situation.

See Sveta Smirnova's slides at MariaDB Day 2020: "How Safe is Asynchronous Master-Master Setup?".

Semi-Synchronous Replication

Semi-synchronous replication was initially implemented as a plugin, in MySQL. Two different plugins needed to be used, one on the primary and the other on the replicas. Starting from MariaDB 10.3.3 it is built-in, which improved its performance.

The problem with standard replication is that there is no guarantee that it will not lag, even by long amounts of time. Semi-synchronous replication reduces this problem, at the cost of reducing the speed of the primary.

In semi-synchronous replication, when a transaction is committed on the primary, the primary does not immediately return control to the client. Instead, it sends the event to the replicas. After one replica reported that the commit was executed with success, the primary reports success to the client.

Semi-synchronous replication is useful for failover, therefore a dual primary setup is not needed in this case. If the primary crashes, the most up-to-date replica can be promoted to primary without losing any data.

Enabling Semi-Synchronous Replication

Semi-synchronous replication can be enabled at runtime in this way on the primary:

```
SET GLOBAL rpl_semi_sync_master_enabled = ON;
```

Semi-synchronous replication is not used until it has been enabled on the replicas also. If the replicas are already replicating, the io_thread needs to be stopped and restarted. This can be done as follows:

```
SET GLOBAL rpl_semi_sync_slave_enabled = ON;
STOP SLAVE IO_THREAD;
START SLAVE IO_THREAD;
```

Tuning the Wait Point and the Primary Timeout

The most important aspects to tune are the wait point and the primary timeout.

When the binary log is enabled, transactions must be committed both in the storage engine (usually InnoDB) and in the binary log. Semi-synchronous replication requires that the transaction is also acknowledged by at least one replica before the primary can report success to the client.

The wait point determines at which point the primary must stop and wait for a confirmation from a replica. This is an important decision from disaster recovery standpoint, in case the primary crashes when a transaction is not fully committed. The rpl_semi_sync_master_wait_point is used to set the wait point, Its allowed values are:

- `AFTER_SYNC` : After committing the transaction in the binary log, but before committing it to the storage engine. After a crash, a transaction may be present in the binary log even if it was not committed.
- `AFTER_COMMIT` . After committing a transaction both in the binary log and in the storage engine. In case of a crash, a transaction could possibly be committed in the primary but not replicated in the slaves. This is the default.

Primary timeout is meant to avoid that a primary remains stuck for a long time, or virtually forever, because no replica acknowledges a transaction. If primary timeout is reached, the primary switches to asynchronous replication. Before doing that, the primary writes an error in the error log and increments the Rpl_semi_sync_master_no_times status variable.

The timeout is set via the rpl_semi_sync_master_timeout variable.

Galera Cluster

Galera is a technology that implements virtually synchronous, primary-primary replication for a cluster of MariaDB servers.

Raft and the Primary Cluster

Nodes of the cluster communicate using the Raft protocol. In case the cluster is partitioned or some nodes crash, a cluster knows that it's still the *primary cluster* if it has the *quorum*: half of the nodes + 1. Only the primary cluster accepts reads and writes.

For this reason a cluster should consist of an odd number of nodes. Imagine for example that a cluster consists of two nodes: if one of them crashes of the connection between them is interrupted, there will be no primary cluster.

Transaction Certification

A transaction can be executed against any node. The node will use a 2-phase commit. After running the transaction locally, the node will ask other nodes to *certify* it. This means that other nodes will receive it, and will try to apply it, and will report success or a failure. The node that received the transaction will not wait for an answer from all the nodes. Once it succeeded on more than half of the nodes (the quorum) the node will run the final commit and data becomes visible.

It is desirable to write data on only one node (unless it fails), or write different databases on different nodes. This will minimize the risk of conflicts.

Galera Cache and SST

Data changes applied are recorded for some time in the *Galera cache*. This is an on-disk cache, written in a circularly written file.

The size of Galera cache can be tuned using the wsrep_provider_options system variable, which contains many flags. We need to tune gcache.size. To tune it, add a line similar to the following to a configuration file:

```
wsrep_provider_options = 'gcache.size=2G';
```

If a single transaction is bigger than half of the Galera cache, it needs to be written in a separate file, as *on-demand pages*. On-demand pages are regularly replaced. Whether a new page replaces an old one depends on another wsrep_provider_options flag: `wsrep_provider_options#gcachekeep_pages_size|gcache.keep_pages_size`, which limits the total size of on-demand pages.

When a node is restarted (after a crash or for maintenance reasons), it will need to receive all the changes that were written by other nodes since the moment it was unreachable. A node is therefore chosen as a donor, possibly using the gcssync_donor wsrep_provider_options flag.

If possible, the donor will send all the recent changes, reading them from the Galera cache and on-demand pages. However, sometimes the Galera cache is not big enough to contain all the needed changes, or the on-demand pages have been overwritten because `gcache.keep_pages_size` is not big enough. In these cases, a State Snapshot Transfer (SST) needs to be sent. This means that the donor will send the whole dataset to the restarted node. Most commonly, this happens using the mariabackup method.

Flow Control

While transaction certification is synchronous, certified transactions are applied locally in asynchronous fashion. However, a node should never lag too much behind others. To avoid that, a node may occasionally trigger a mechanism called *flow control* to ask other nodes to stop replication until its situation improves. Several wsrep_provider_options flags affect flow control.

gcs.fc_master_slave should normally be set to 1 if all writes are sent to a single node.

gcs.fc_limit is tuned automatically, unless gcs.fc_master_slave is set to 0. The *receive queue* (the transactions received and not yet applied) should not exceed this limit. When this happens, flow control is triggered by the node to pause other node's replication.

Once flow control is activated, gcs.fc_factor determines when it is released. It is a number from 0 to 1, and it represents a fraction. When the receive queue is below this fraction, the flow control is released.

Flow control and the receive queue can and should be monitored. The most useful metrics are:

- wsrep_flow_control_paused indicates how many times the replication has been paused as requested by other nodes, since the last `FLUSH STATUS`.
- wsrep_flow_control_sent indicates how many times this node requested other nodes to pause replication.

- wsrep_local_recv_queue is the size of the receive queue.

Configuration

Galera is implemented as a plugin. Starting from version 10.1, MariaDB comes with Galera pre-installed, but not in use by default. To enable it one has to set the wsrep_on system variable.

Like asynchronous replication, Galera uses the binary log. It also requires that data changes are logged in the `ROW` format.

For other required settings, see Mandatory Options.

Galera Limitations

Galera is not suitable for all databases and workloads.

- Galera only replicates InnoDB tables. Other storage engines should not be used.
- For performance reasons, it is highly desirable that all tables have a primary key.
- Long transactions will damage performance.
- Some applications use an integer AUTO_INCREMENT primary key. In case of failover from a crashed node to another, Galera does not guarantee that `AUTO_INCREMENT` follows a chronological order. Therere, applications should use TIMESTAMP columns for chronological order instead.

2.1.14.2.13 Moving Data Between SQL Server and MariaDB

Contents

There are several ways to move data between SQL Server and MariaDB. Here we will discuss them and we will highlight some caveats.

Moving Data Definition from SQL Server to MariaDB

To copy SQL Server data structures to MariaDB, one has to:

1. Generate a CSV file from SQL Server data.
2. Modify the syntax so that it works in MariaDB.
3. Run the file in MariaDB.

Variables That Affect DDL Statements

DDL statements are affected by some server system variables.

sql_mode determines the behavior of some SQL statements and expressions, including how strict error checking is, and some details regarding the syntax. Objects like stored procedures, stored functions triggers and views, are always executed with the sql_mode that was in effect during their creation. sql_mode='MSSQL' can be used to have MariaDB behaving as close to SQL Server as possible.

innodb_strict_mode enables the so-called InnoDB strict mode. Normally some errors in the CREATE TABLE options are ignored. When InnoDB strict mode is enabled, the creation of InnoDB tables will fail with an error when certain mistakes are made.

updatable_views_with_limit determines whether view updates can be made with an UPDATE or DELETE statement with a `LIMIT` clause if the view does not contain all primary or not null unique key columns from the underlying table.

Dumps and sys.sql_modules

SQL Server Management Studio allows one to create a working SQL script to recreate a database - something that MariaDB users refer to as a *dump*. Several options allow fine-tuning the generated syntax. It could be necessary to adjust some of these options to make the output compatible with MariaDB. It is possible to export schemas, data or both. One can create a single global file, or one file for each exported object. Normally, producing a single file is more practical.

Alternatively, the sp_helptext() procedure returns information about how to recreate a certain object. Similar information is also present in the sql_modules table (`definition` column), in the `sys` schema. Such information, however, is not a ready-to-use set of SQL statements.

Remember however that MariaDB does not support schemas. An SQL Server schema is approximately a MariaDB database.

To execute a dump, we can pass the file to `mariadb`, the MariaDB command-line client.

Provided that a dump file contains syntax that is valid in MariaDB, it can be executed in this way:

```
mariadb --show-warnings < dump.sql
```

`--show-warnings` tells MariaDB to output any warnings produced by the statements contained in the dump. Without this option, warnings will not appear on screen. Warnings don't stop the dump execution.

Errors will appear on screen. Errors will stop the dump execution, unless the `--force` option (or just `-f`) is specified.

For other `mariadb` options, see mariadb Command-line Client Options.

Another way to achieve the same purpose is to start the `mariadb` client in interactive mode first, and then run the `source` command. For example:

```
root@d5a54a082d1b:/# mariadb -uroot -psecret
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 22
Server version: 10.4.7-MariaDB-1:10.4.7+maria~bionic mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> \W
Show warnings enabled.
MariaDB [(none)]> source dump.sql
```

In this case, to show warnings we used the `\W` command, where "w" is uppercase. To hide warnings (which is the default), we can use `\w` (lowercase).

For other `mariadb` commands, see mariadb Commands.

CSV Data

If the table structures are already in MariaDB, we need only to import table data. While this can still be done as explained above, it may be more practical to export CSV files from SQL Server and import them into MariaDB.

SQL Server Management Studio and several other Microsoft tools allow one to export CSV files.

MariaDB allows importing CSV files with the LOAD DATA INFILE statement, which is essentially the MariaDB equivalent of `BULK INSERT`.

It can happen that we don't want to import the whole data, but some filtered or transformed version of it. In that case, we may prefer to use the CONNECT storage engine to access CSV files and query them. The results of a query can be inserted into a table using INSERT SELECT.

Moving Data from MariaDB to SQL Server

There are several ways to move data from MariaDB to SQL Server:

- If the tables don't exist at all in SQL Server, we need to generate a dump first. The dump can include data or not.
- If the tables are already in SQL Server, we can use CSV files instead of dumps to move the rows. CSV files are the most concise format to move data between different technologies.
- With the tables already in SQL Server, another way to move data is to insert the rows into CONNECT tables that "point" to remote SQL Server tables.

Using a Dump (Structure)

mariadb-dump can be used to generate dumps of all databases, a specified database, or a set of tables. It is even possible to only dump a set of rows by specifying the `WHERE` clause.

By specifying the `--no-data` option we can dump the table structures without data.

`--compatible=mssql` will produce an output that should be usable in SQL Server.

Using a Dump (Data)

mariadb-dump by default produces an output with both data and structure.

`--no-create-info` can be used to skip the CREATE TABLE statements.

`--compatible=mssql` will produce an output that should be usable in SQL Server.

`--single-transaction` should be specified to select the source data in a single transaction, so that a consistent dump is produced.

`--quick` speeds up the dump process when dumping big tables.

Using a CSV File

CSV files can also be used to export data to SQL Server. There are several ways to produce CSV files from MariaDB:

- The SELECT INTO OUTFILE statement.
- The CONNECT storage engine, with the CSV table type.
- The CSV storage engine (note that it doesn't support `NULL` and indexes).

Using CONNECT Tables

The CONNECT storage engine allows one to access external data, in many forms:

- Data files (CSV, JSON, XML, HTML and more).
- Remote databases, using the ODBC or JDBC standards, or MariaDB/MySQL native protocol.
- Some special data sources.

`CONNECT` was mentioned previously because it could allow one to read a CSV file and query it in SQL, filtering and transforming the data that we want to move into regular MariaDB tables.

However, `CONNECT` can also access remote SQL Server tables. We can read data from it, or even write data.

To enable `CONNECT` to work with SQL Server, we need to fulfill these requirements:

- Install the ODBC driver, downloadable form Microsoft website. The driver is also available for Linux and MacOS.
- Install unixODBC.
- Install `CONNECT` (unless it is already installed).

Here is an example of a `CONNECT` table that points to a SQL Server table:

```
CREATE TABLE city (
    id INT PRIMARY KEY,
    city_name VARCHAR(100),
    province_id INT NOT NULL
)
    ENGINE=CONNECT,
    TABLE_TYPE=ODBC,
    TABNAME='city'
    CONNECTION='Driver=SQL Server Native Client 13.0;Server=sql-server-
hostname;Database=world;UID=mariadb_connect;PWD=secret';
```

The key points here are:

- `ENGINE=CONNECT` tells MariaDB that we want to create a `CONNECT` table.
- `TABLE_TYPE` must be 'ODBC', so `CONNECT` knows what type of data source it has to use.
- `CONNECTION` is the connection string to use, including server address, username and password.
- `TABNAME` tells `CONNECT` what the remote table is called. The local name could be different.

`CONNECT` is able to query SQL Server to find out the remote table structure. We can use this feature to avoid specifying the column names and types:

```
CREATE TABLE city
    ENGINE=CONNECT,
    TABLE_TYPE=ODBC,
    TABNAME='city'
    CONNECTION='Driver=SQL Server Native Client 13.0;Server=sql-server-
hostname;Database=world;UID=mariadb_connect;PWD=secret';
```

However, we may prefer to manually specify the MariaDB types, sizes and character sets to use.

Linked Server

Instead of using MariaDB `CONNECT`, it is possible to use SQL Server Linked Server functionality. This will allow one to read data from a remote MariaDB database and copy it into local SQL Server tables. However, note that `CONNECT` allows more control on types and character sets mapping.

Refer to Linked Servers section in Microsoft documentation.

2.1.14.2.14 SQL_MODE=MSSQL

Contents

`SET SQL_MODE=MSSQL` implies all the following sql_mode flags:

- PIPES_AS_CONCAT
- ANSI_QUOTES
- IGNORE_SPACE
- NO_KEY_OPTIONS
- NO_TABLE_OPTIONS
- NO_FIELD_OPTIONS

Setting the sql_mode system variable to `MSSQL` allows the server to understand a small subset of Microsoft SQL Server's language. For the moment `MSSQL` mode only has limited functionality, but we plan to add more later according to demand.

Supported Syntax in MSSQL Mode

Using [] for Quoting

One can use [] instead of "" or `` for quoting identifiers:

```
SET SQL_MODE="MSSQL";
CREATE TABLE [t 1] ([a b] INT);
SHOW CREATE TABLE [t 1];
Table  Create Table
t 1    CREATE TABLE "t 1" (
  "a b" int(11) DEFAULT NULL
)
```

You can use '[' in identifiers. If you want to use ']' in identifiers you have to specify it twice.

2.1.14.3 Migrating to MariaDB from PostgreSQL

There are many different ways to migrate from PostgreSQL to MariaDB. This article will discuss some of those options.

Contents

MariaDB's CONNECT Storage Engine

MariaDB's CONNECT storage engine can be used to migrate from PostgreSQL to MariaDB. There are two primary ways that this can be done.

See Loading the CONNECT Storage Engine for information on how to install the CONNECT storage engine.

Tables with ODBC table_type

The CONNECT storage engine allows you to create tables that refer to tables on an external server, and it can fetch the data using a compatible ODBC driver. PostgreSQL does have a freely available ODBC driver called `psqlODBC`. Therefore, if you install `psqlODBC` on the MariaDB Server, and then configure the system's ODBC framework (such as unixODBC), then the MariaDB server will be able to connect to the remote PostgreSQL server. At that point, you can create tables with the `ENGINE=CONNECT` and `table_type=ODBC` table options set, so that you can access the PostgreSQL tables from MariaDB.

See CONNECT ODBC Table Type: Accessing Tables From Another DBMS for more information on how to do that.

Once the remote table is setup, you can migrate the data to local tables very simply. For example:

```
CREATE TABLE psql_tab (
   id int,
   str varchar(50)
) ENGINE = CONNECT
table_type=ODBC
tabname='tab'
connection='DSN=psql_server';

CREATE TABLE tab (
   id int,
   str varchar(50)
) ENGINE = InnoDB;

INSERT INTO tab SELECT * FROM psql_tab;
```

Tables with JDBC table_type

The CONNECT storage engine allows you to create tables that refer to tables on an external server, and it can fetch the data using a compatible JDBC driver. PostgreSQL does have a freely available JDBC driver. If you install this JDBC driver on the MariaDB server, then the MariaDB server will be able to connect to the remote PostgreSQL server via JDBC. At that point, you can create tables with the `ENGINE=CONNECT` and `table_type=JDBC` table options set, so that you can access the PostgreSQL tables from MariaDB.

See CONNECT JDBC Table Type: Accessing Tables from Another DBMS for more information on how to do that.

Once the remote table is setup, you can migrate the data to local tables very simply. For example:

```
CREATE TABLE psql_tab (
   id int,
   str varchar(50)
) ENGINE = CONNECT
table_type=JDBC
tabname='tab'
connection='jdbc:postgresql://psql_server/db1';

CREATE TABLE tab (
   id int,
   str varchar(50)
) ENGINE = InnoDB;

INSERT INTO tab SELECT * FROM psql_tab;
```

PostgreSQL's Foreign Data Wrappers

PostgreSQL's foreign data wrappers can also be used to migrate from PostgreSQL to MariaDB.

mysql_fdw

mysql_fdw allows you to create a table in PostgreSQL that actual refers to a remote MySQL or MariaDB server. Since MySQL and MariaDB are compatible at the protocol level, this should also support MariaDB.

The foreign data wrapper also supports writes, so you should be able to write to the remote MariaDB table to migrate your PostgreSQL data. For example:

```
CREATE TABLE tab (
   id int,
   str text
);

INSERT INTO tab VALUES (1, 'str1');

CREATE SERVER mariadb_server
   FOREIGN DATA WRAPPER mysql_fdw
   OPTIONS (host '10.1.1.101', port '3306');

CREATE USER MAPPING FOR postgres
   SERVER mariadb_server
   OPTIONS (username 'foo', password 'bar');

CREATE FOREIGN TABLE mariadb_tab (
   id int,
   str text
)
SERVER mariadb_server
OPTIONS (dbname 'db1', table_name 'tab');

INSERT INTO mariadb_tab SELECT * FROM tab;
```

PostgreSQL's COPY TO

PostgreSQL's `COPY TO` allows you to copy the data from a PostgreSQL table to a text file. This data can then be loaded into MariaDB with `LOAD DATA INFILE`.

MySQL Workbench

MySQL Workbench has a migration feature that requires an ODBC driver. PostgreSQL does have a freely available ODBC driver called `psqlODBC`.

See Set up and configure PostgreSQL ODBC drivers for the MySQL Workbench Migration Wizard for more information.

Known Issues

Migrating Functions and Procedures

PostgreSQL's functions and procedures use a language called `PL/pgSQL`. This language is quite different than the default `SQL/PSM` language used for MariaDB's stored procedures. `PL/pgSQL` is more similar to `PL/PSQL` from Oracle, so you may find it beneficial to try migrate with `SQL_MODE=ORACLE` set.

2.1.14.3.1 SQL_MODE=ORACLE

From MariaDB 10.3, setting the sql_mode system variable to `Oracle` allows the server to understand a subset of Oracle's PL/SQL language. For example:

```
SET SQL_MODE='ORACLE';
```

All traditional MariaDB SQL/PSM syntax should work as before, as long as it does not conflict with Oracle's PL/SQL syntax. All MariaDB functions should be supported in both normal and Oracle modes.

Prior to MariaDB 10.3, MariaDB does not support Oracle's PL/SQL language, and `SET SQL_MODE=ORACLE` is only an alias for the following sql_mode in those versions:

```
SET SQL_MODE='PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS,
NO_TABLE_OPTIONS, NO_FIELD_OPTIONS, NO_AUTO_CREATE_USER';
```

From MariaDB 10.3, `SET SQL_MODE=ORACLE` is same as:

```
SET SQL_MODE='PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE,ORACLE,NO_KEY_OPTIONS,
NO_TABLE_OPTIONS,NO_FIELD_OPTIONS,NO_AUTO_CREATE_USER,SIMULTANEOUS_ASSIGNMENT';
```

Contents

Supported Syntax in Oracle Mode

Stored Procedures and Stored Functions

Oracle mode makes the following changes to Stored Procedures and Stored Functions:

Oracle syntax	Description
CREATE PROCEDURE p1 `(param OUT INT)`	ANSI uses `(OUT param INT)`
CREATE PROCEDURE p1 `(a IN OUT INT)`	ANSI uses `(INOUT param INT)`
`AS` before function body	CREATE FUNCTION f1 RETURN NUMBER `AS` BEGIN...
`IS` before function body	CREATE FUNCTION f1 RETURN NUMBER `IS` BEGIN...
If function has no parameters then parentheses must be omitted	Example: CREATE PROCEDURE p1 AS BEGIN NULL; END;

CREATE PROCEDURE p1 AS BEGIN END `p1` ;	Optional routine name after END keyword. MDEV-12089
CREATE FUNCTION f1(a `VARCHAR`)	VARCHAR can be used without length for routine parameters and RETURN clause. The length is inherited from the argument at call time. MDEV-10596
CREATE AGGREGATE FUNCTION f1( )	Creates an aggregate function, which performs the function against a set of rows and returns one aggregate result.
No `CALL` needed in Stored Procedures	In Oracle mode one can call other stored procedures with name only. MDEV-12107
`RETURN` . Can also be used in stored procedures	ANSI uses `RETURNS` . MariaDB mode only supports RETURNS in stored functions

Cursors

Oracle mode makes the following changes to Cursors:

Oracle syntax	Description
CREATE PROCEDURE p1 AS CURSOR cur IS (SELECT a, b FROM t1); BEGIN FOR rec IN cur ...	Explicit cursor with FOR loop. MDEV-10581
CREATE PROCEDURE p1 AS `rec IN (SELECT a, b FROM t1)`	Implicit cursor with FOR loop. MDEV-12098
`CURSOR` c(prm_a VARCHAR2, prm_b VARCHAR2) ... `OPEN c(1,2)`	Cursor with parameters. MDEV-10597
`CURSOR` c(prm_a VARCHAR2, prm_b VARCHAR2) ... FOR `rec in c(1,2)`	Cursor with parameters and FOR loop. MDEV-12314
s %ISOPEN, %ROWCOUNT, %FOUND, %NOTFOUND	Explicit cursor attributes. MDEV-10582

LOOP

Oracle mode makes the following changes to LOOP:

Oracle syntax	Description
`FOR i IN 1..10 LOOP ... END LOOP`	Numeric FOR loop. MDEV-10580
`GOTO`	GOTO statement. MDEV-10697
`<<label>>` used with GOTO	ANSI uses `label:` . MDEV-10697
To leave loop block: `EXIT [ label ] [ WHEN bool_expr ]`	ANSI syntax is `IF bool_expr THEN LEAVE label`
[`<<label>>`] WHILE boolean_expression LOOP statement... END LOOP [label] ;	Oracle style `WHILE` loop
`CONTINUE` [label] [WHEN boolean_expression]	`CONTINUE` is only valid inside a loop

Variables

Oracle syntax	Version	Description
`var:= 10` ; Can also be used with MariaDB systemvariables	10.3	MariaDB uses `SET var= 10` ;
`var INT := 10`	10.3	Default variable value
`var1 table_name.column_name%TYPE`	10.3	Take data type from a table column. MDEV-10577
`var2 var1%TYPE`	10.3	Take data type from another variable
`rec1 table_name%ROWTYPE`	10.3	Take ROW structure from a table. MDEV-12133
`rec2 rec1%ROWTYPE`	10.3	Take ROW structure from ROW variable
CURSOR c1 IS SELECT a,b FROM t1; rec1 c1%ROWTYPE;	10.3	Take ROW structure from a cursor. MDEV-12011
Variables can be declared after cursor declarations	10.3	In MariaDB mode, variables must be declared before cursors. MDEV-10598
Triggers uses `:NEW` and `:OLD`	10.3	ANSI uses `NEW` and `OLD` . MDEV-10579

`SQLCODE`	10.3	Returns the number code of the most recent exception. Can only be used in Stored Procedures. MDEV-10578
`SQLERRM`	10.3	Returns the error message associdated to it's error number argument or `SQLCODE` if no argument is given. Can only be used in Stored Procedures. MDEV-10578
`SQL%ROWCOUNT`	10.3	Almost same as ROW_COUNT(). MDEV-10583
`ROWNUM`	10.6.1	Returns number of accepted rows

Exceptions

Oracle syntax	Description
BEGIN ... EXCEPTION WHEN OTHERS THEN BEGIN .. END; END;	Exception handlers are declared at the end of a block
TOO_MANY_ROWS, NO_DATA_FOUND, DUP_VAL_ON_INDEX	Predefined exceptions. MDEV-10839
RAISE TOO_MANY_ROWS ; .... EXCEPTION WHEN TOO_MANY_ROWS THEN ...	Exception can be used with RAISE and EXCEPTION...WHEN. MDEV-10840
CREATE OR REPLACE FUNCTION f1 (a INT) RETURN INT AS `e1 EXCEPTION` ...	User defined exceptions. MDEV-10587

BEGIN Blocks

Oracle syntax	Description
`BEGIN` to start a block	MariaDB uses `BEGIN NOT ATOMIC` for anyonymous blocks. MDEV-10655
`DECLARE` is used before `BEGIN`	DECLARE a INT; b VARCHAR(10); BEGIN v:= 10; END;
WHEN DUP_VAL_ON_INDEX THEN NULL ; NULL; WHEN OTHERS THEN NULL	Do not require BEGIN..END in multi-statement exception handlers in THEN clause. MDEV-12088

Simple Syntax Compatibility

Oracle syntax	Version	Description
`ELSIF`	10.3	ANSI uses `ELSEIF`
SELECT `UNIQUE`	10.3	Same as `SELECT DISTINCT` . MDEV-12086
TRUNCATE TABLE t1 [`DROP STORAGE`] or [`REUSE STORAGE`]	10.3	DROP STORAGE and REUSE STORAGE are allowed as optional keywords for TRUNCATE TABLE. MDEV-10588
Subqueries in a FROM clause without an alias	10.6	`SELECT * FROM (SELECT 1 FROM DUAL), (SELECT 2 FROM DUAL)`
UNION, EXCEPT and INTERSECT all have the same precedence.	10.3	INTERSECT has higher precedence than UNION and EXCEPT in non-Oracle modes.
`MINUS`	10.6	`MINUS` is a synonym for EXCEPT.

Functions

Oracle syntax	Version	Description
ADD_MONTHS()	10.6.1	Added as a wrapper for DATE_ADD() to enhance Oracle compatibility. All modes.
CAST(expr `as VARCHAR(N)`)	10.3	Cast expression to a `VARCHAR(N)` . MDEV-11275
DECODE	10.3	In Oracle mode, compares and matches search expressions
`LENGTH()` is same as CHAR_LENGTH()	10.3	MariaDB translates LENGTH() to OCTET_LENGTH(). In all modes one can use `LENGTHB()` as a synonym to `OCTET_LENGTH()`
CHR(num)	10.3	Returns a VARCHAR(1) with character set and collation according to @@character_set_database and @@collation_database
`substr('abc',0 ,3)` same as `substr('abc', 1 ,3)`	10.3	Position 0 for substr() is same as position 1

SYS_GUID	10.6.1	Generates a globally unique identifier. Similar to UUID but without the - . All modes.
TO_CHAR	10.6.1	Added to enhance Oracle compatibility. All modes.
TRIM, LTRIM, RTRIM, LPAD and RPAD	10.3	Returns NULL instead of an empty string if returning an empty result. These functions can also be accessed outside of ORACLE mode by suffixing _ORACLE onto the end of the function name, such as TRIM_ORACLE.

Prepared Statements

Oracle mode makes the following changes to Prepared Statements:

Oracle syntax	Description
PREPARE stmt FROM 'SELECT :1 , :2 '	ANSI uses ? . MDEV-10801
EXECUTE IMMEDIATE 'INSERT INTO t1 SELECT (:x,:y) FROM DUAL' USING 10,20	Dynamic placeholders. MDEV-10801

Synonyms for Basic SQL Types

Oracle type	MariaDB synonym
VARCHAR2	VARCHAR
NUMBER	DECIMAL
DATE (with time portion)	MariaDB DATETIME
RAW	VARBINARY
CLOB	LONGTEXT
BLOB	LONGBLOB

This was implemented as part of MDEV-10343.

If one does a SHOW CREATE TABLE in `ORACLE` mode on a table that has a native MariaDB `DATE` column, it will be displayed as mariadb_schema.date to not conflict with the Oracle `DATE` type.

Packages

The following syntax has been supported since MariaDB 10.3.5:

- CREATE PACKAGE
- CREATE PACKAGE BODY
- DROP PACKAGE
- DROP PACKAGE BODY
- SHOW CREATE PACKAGE
- SHOW CREATE PACKAGE BODY

NULL Handling

Oracle mode makes the following changes to NULL handling:

NULL As a Statement

`NULL` can be used as a statement:

```
IF a=10 THEN NULL; ELSE NULL; END IF
```

Translating Empty String Literals to NULL

In Oracle, empty string ('') and NULL are the same thing,

By using `sql_mode=EMPTY_STRING_IS_NULL` you can get a similar experience in MariaDB:

```
SET sql_mode=EMPTY_STRING_IS_NULL;
SELECT '' IS NULL; -- returns TRUE
INSERT INTO t1 VALUES (''); -- inserts NULL
```

Concat Operator Ignores NULL

CONCAT() and || ignore NULL in Oracle mode. Can also be accessed outside of ORACLE mode by using CONCAT_OPERATOR_ORACLE. MDEV-11880 and MDEV-12143.

Reserved Words

There are a number of extra reserved words in Oracle mode.

SHOW CREATE TABLE

The SHOW CREATE TABLE statement will not display MariaDB-specific table options, such as AUTO_INCREMENT or CHARSET, when Oracle mode is set.

2.1.14.4 Installing MariaDB on IBM Cloud

Contents

Get MariaDB on IBM Cloud

You should have an IBM Cloud account, otherwise you can register here. At the end of the tutorial you will have a cluster with MariaDB up and running. IBM Cloud uses Bitnami charts to deploy MariaDB on with helm

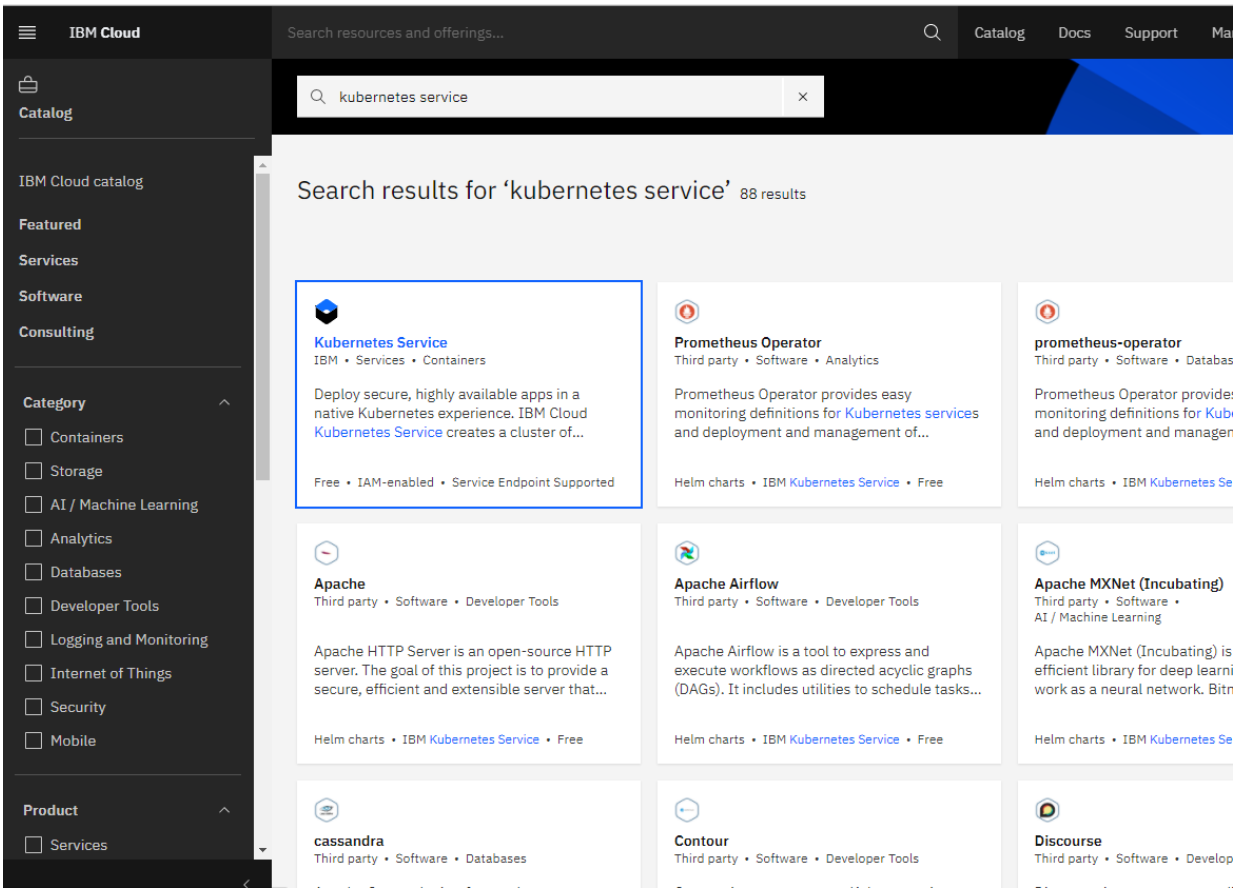
1. We will provision a new Kubernetes Cluster for you if, you already have one skip to step **2**

2. We will deploy the IBM Cloud Block Storage plug-in, if already have it skip to step **3**

3. MariaDB deployment

Step 1 provision Kubernetes Cluster

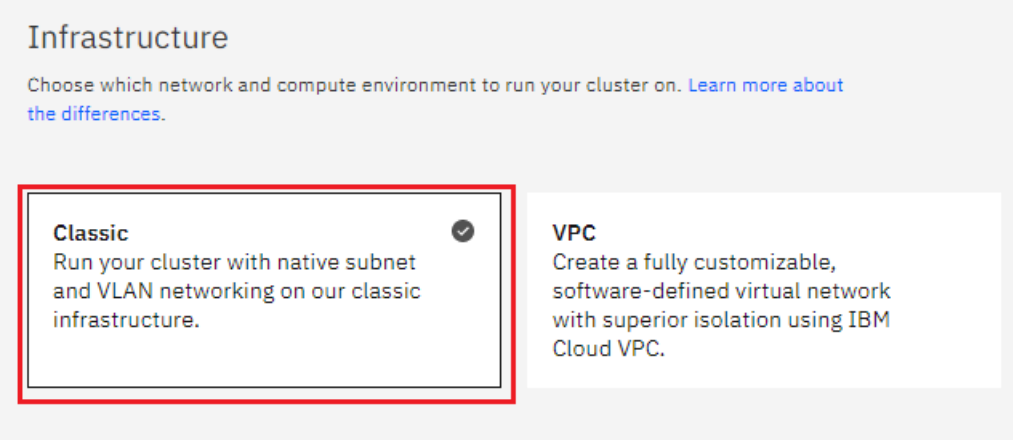
- Click the **Catalog** button on the top
- Select **Service** from the catalog
- Search for **Kubernetes Service** and click on it



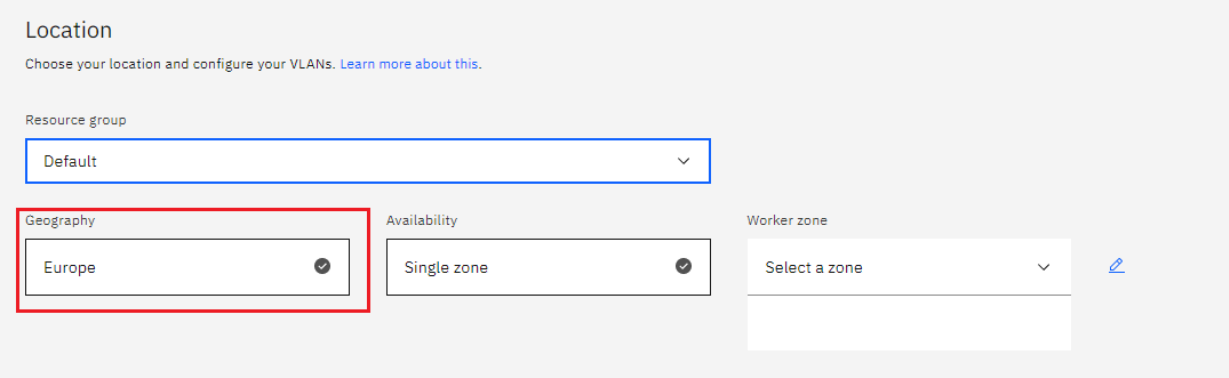
- You are now at the Kubernetes deployment page, you need to specify some details about the cluster
- Choose a plan **standard** or **free**, the free plan only has one worker node and no subnet, to provision a standard

cluster, you will need to upgrade you account to Pay-As-You-Go
- To upgrade to a Pay-As-You-Go account, complete the following steps:

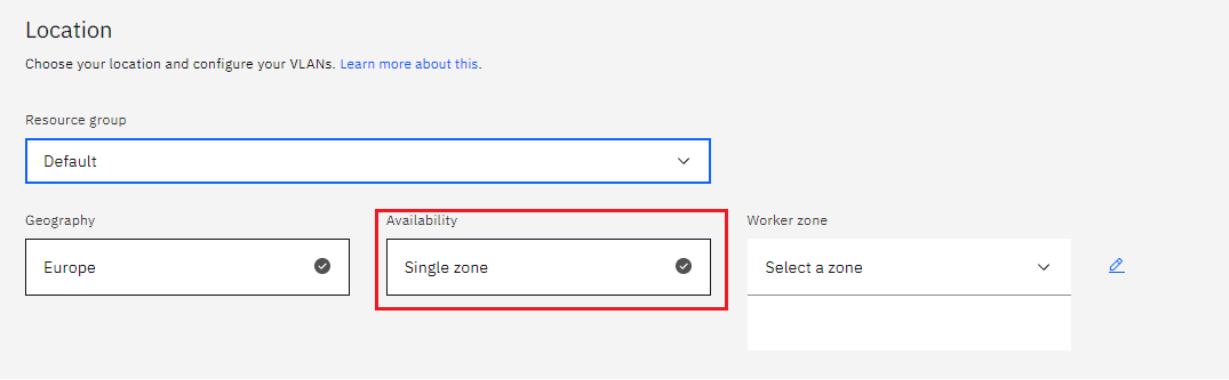
- In the console, go to Manage > Account.
- Select Account settings, and click Add credit card.
- Enter your payment information, click Next, and submit your information
- Choose **classic** or **VPC**, read the docs and choose the most suitable type for yourself



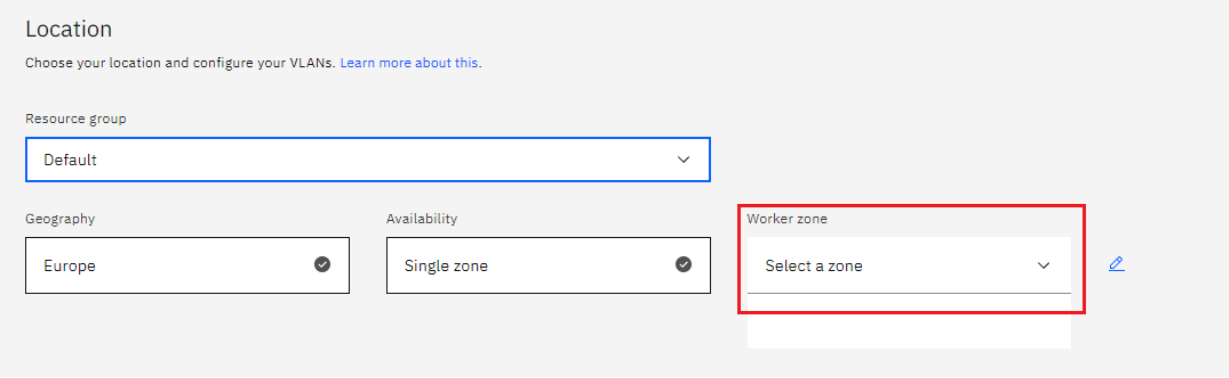
- Now choose your location settings, for more information please visit Locations
- Choose **Geography** (continent)



- Choose **Single** or **Multizone**, in single zone your data is only kept in on datacenter, on the other hand with Multizone it is distributed to multiple zones, thus safer in an unforseen zone failure

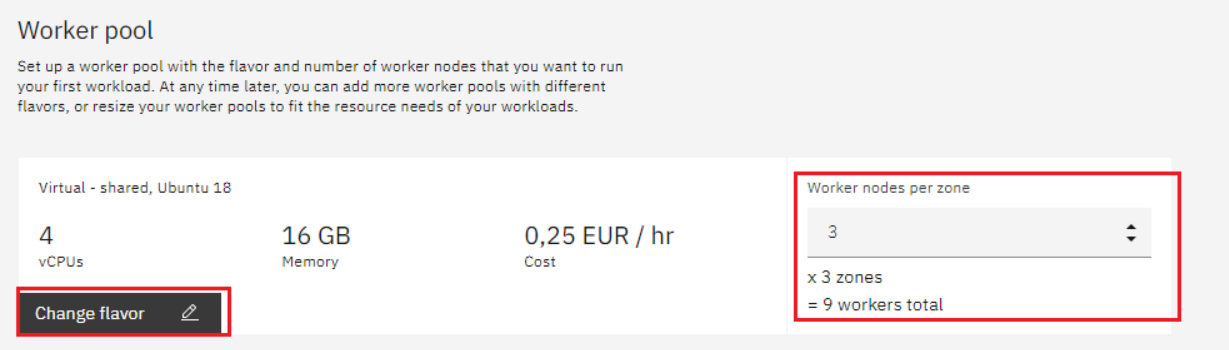


- Choose a **Worker Zone** if using Single zones or **Metro** if Multizone

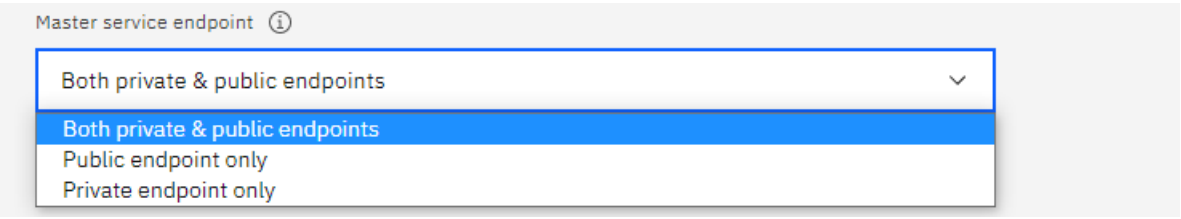


- If you wish to use Multizone please set up your account with VRF or enable Vlan spanning
- If at your current location selection, there is no available Virtual LAN, a new Vlan will be created for you

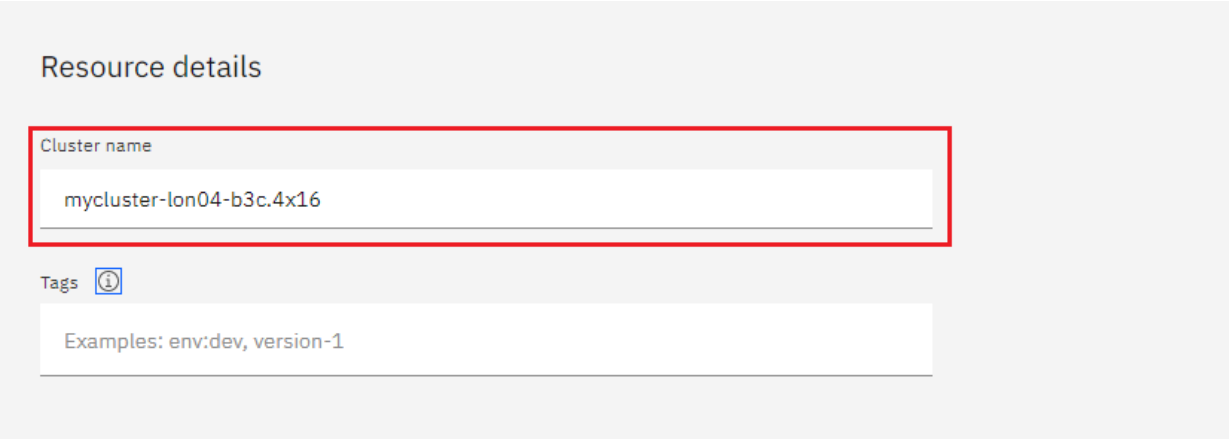
- Choose a **Worker node setup** or use the preselected one, set **Worker node amount per zone**



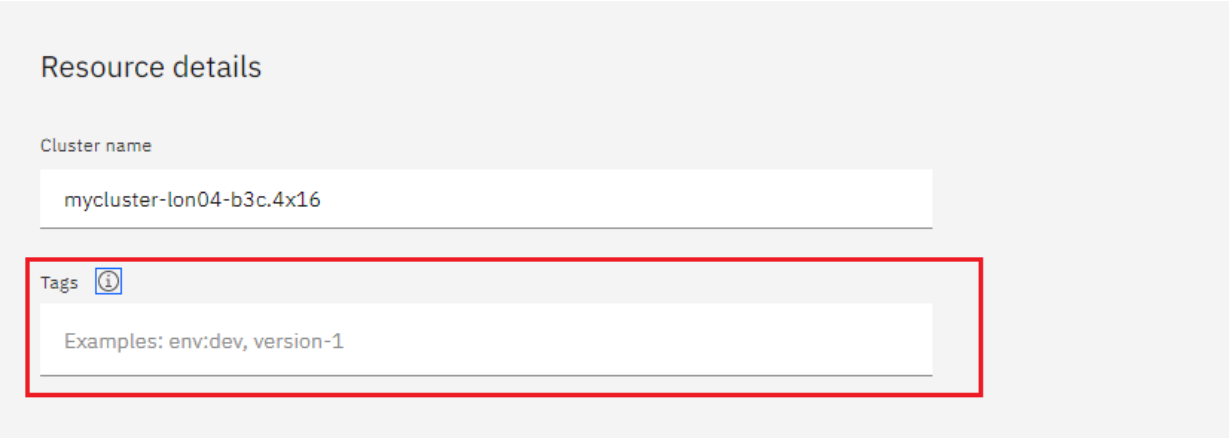
- Choose **Master Service Endpoint**, In VRF-enabled accounts, you can choose private-only to make your master accessible on the private network or via VPN tunnel. Choose public-only to make your master publicly accessible. When you have a VRF-enabled account, your cluster is set up by default to use both private and public endpoints. For more information visit endpoints.



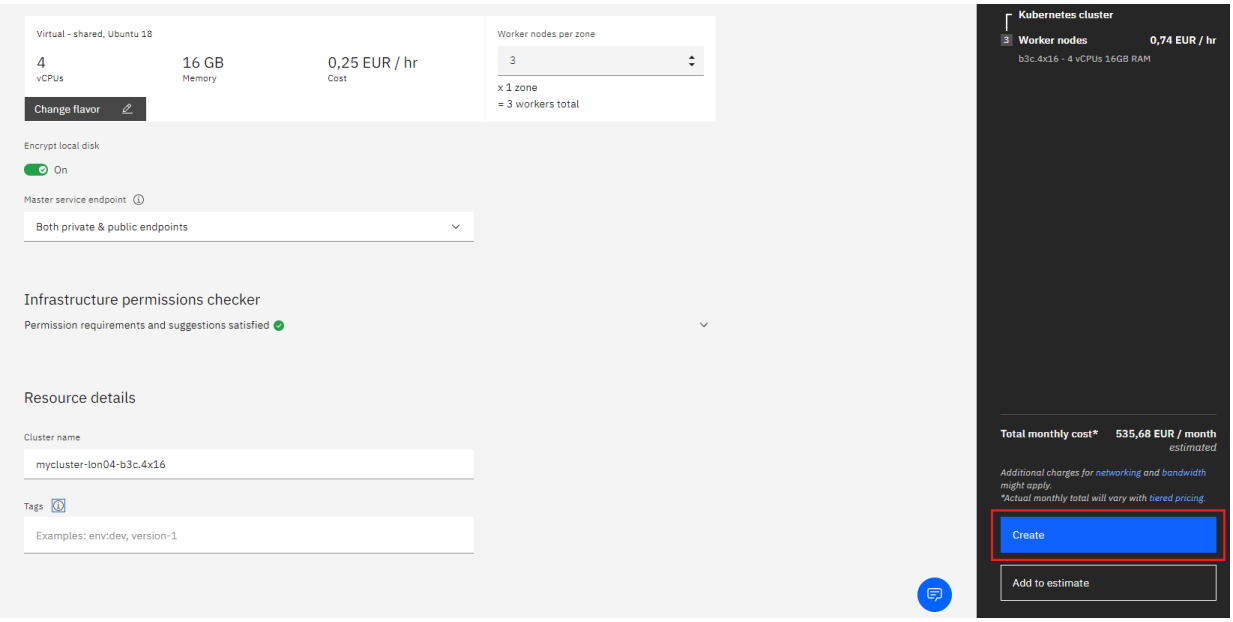
- Give cluster a **name**



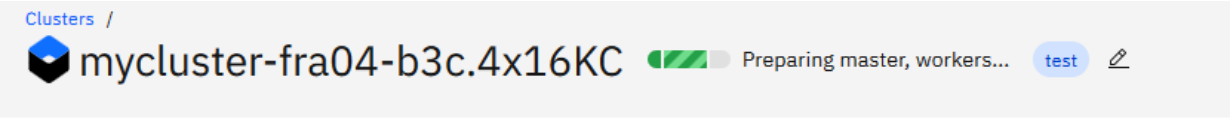
- Give desired **tags** to your cluster, for more information visit tags



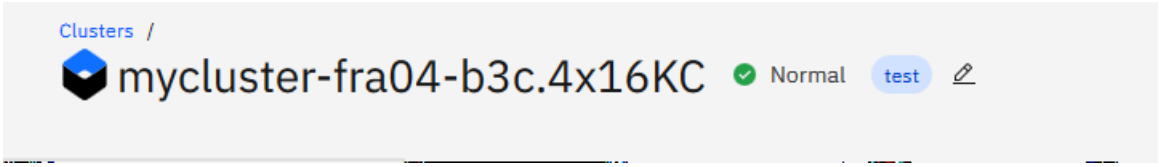
- Click **create**



- Wait for you cluster to be provisioned



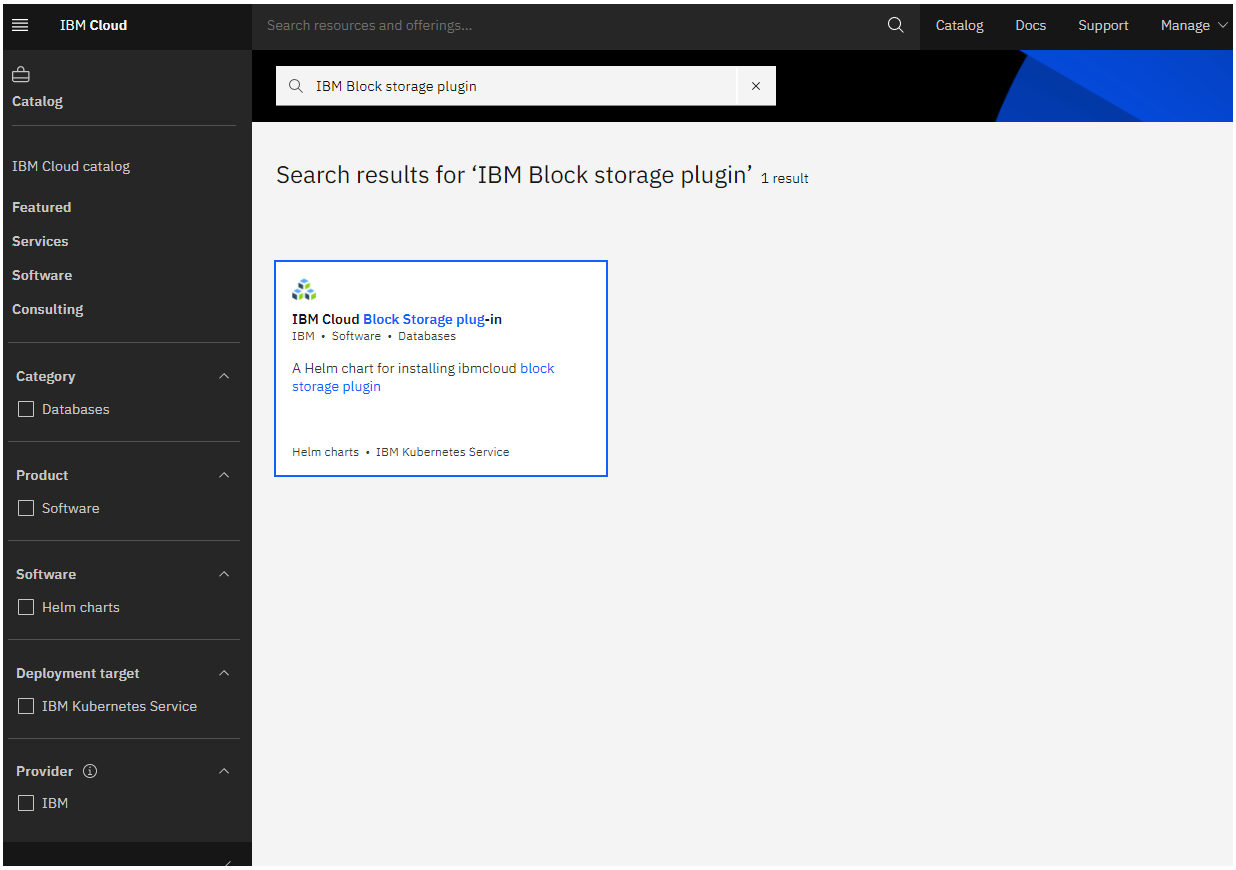
- Your cluster is ready for usage



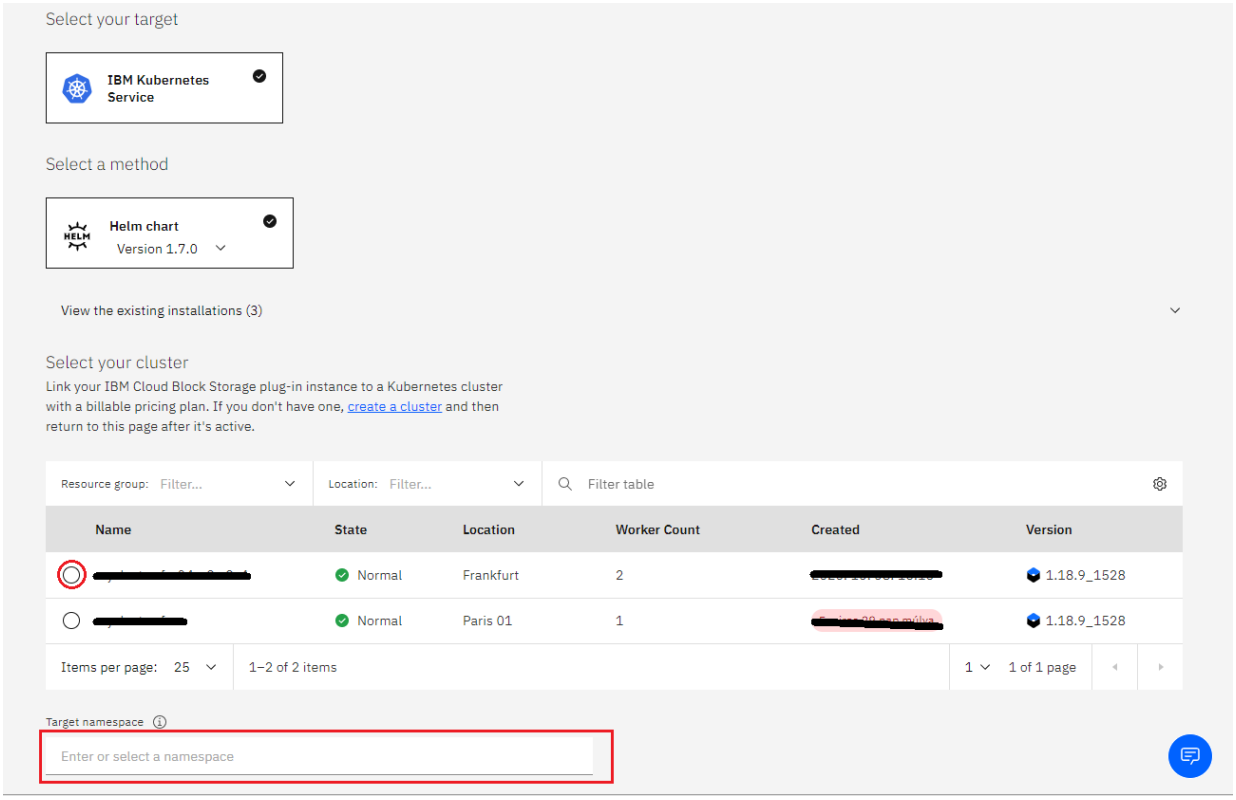
Step 2 deploy IBM Cloud Block Storage plug-in

The Block Storage plug-in is a persistent, high-performance iSCSI storage that you can add to your apps by using Kubernetes Persistent Volumes (PVs).

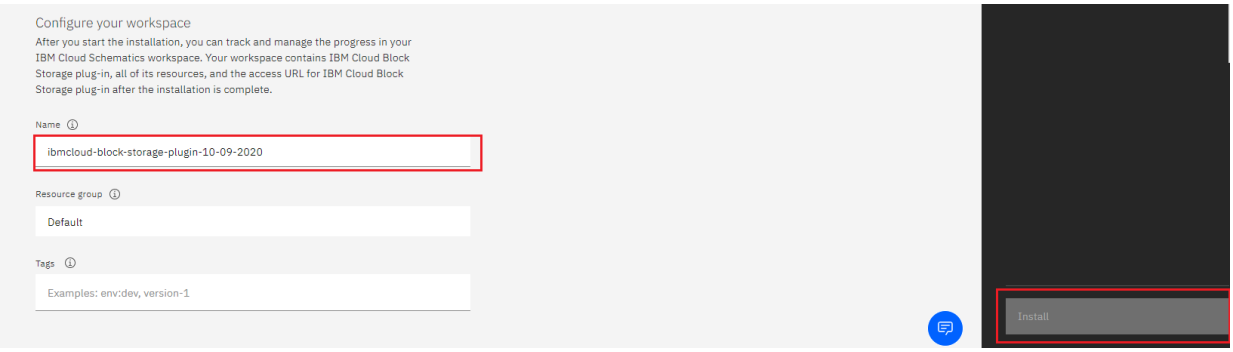
- Click the **Catalog** button on the top
- Select **Software** from the catalog
- Search for **IBM Cloud Block Storage plug-in** and click on it



- On the application page Click in the dot next to the cluster, you wish to use
- Click on **Enter or Select Namespace** and choose the default Namespace or use a custom one (if you get error please wait 30 minutes for the cluster to finalize)



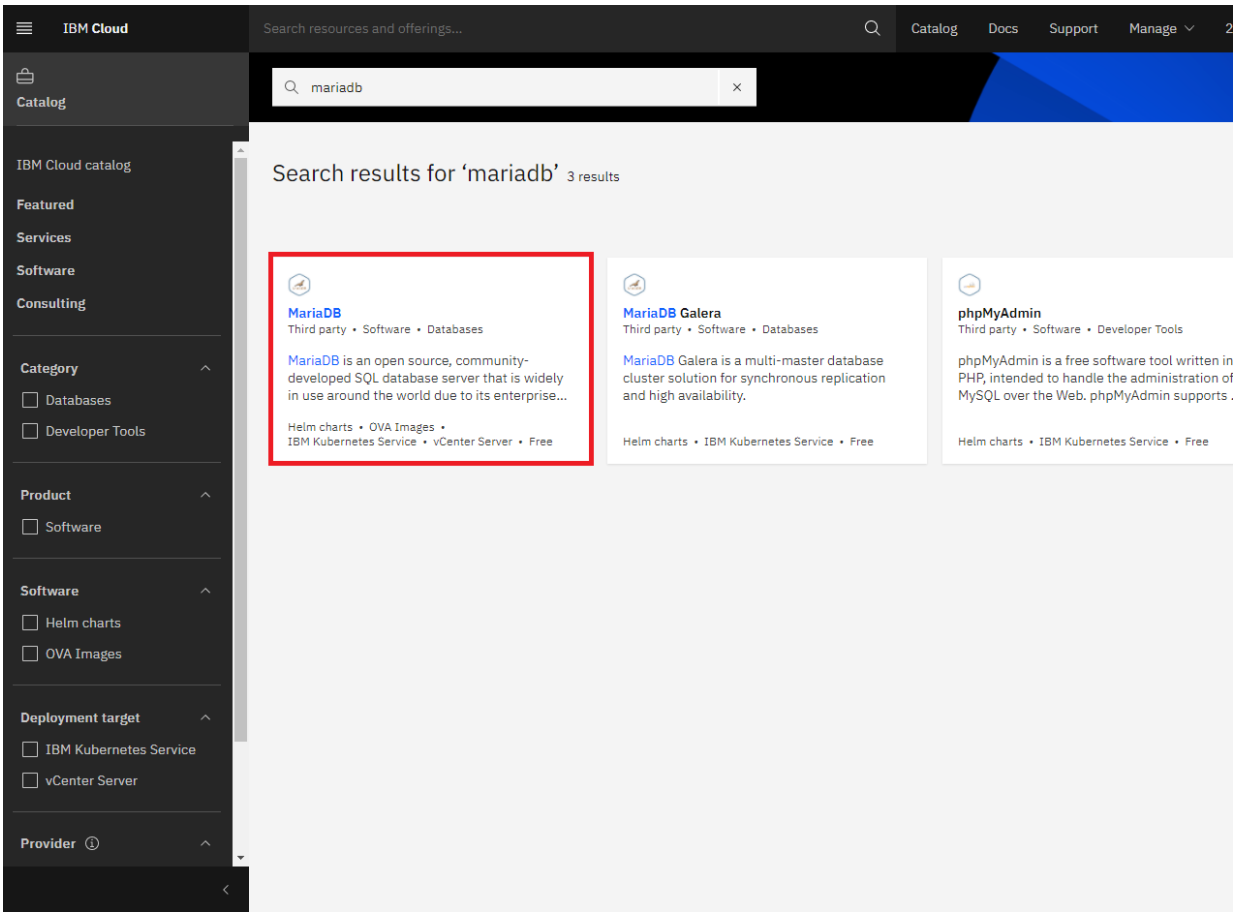
- Give a **name** to this workspace
- Click **install** and wait for the deployment



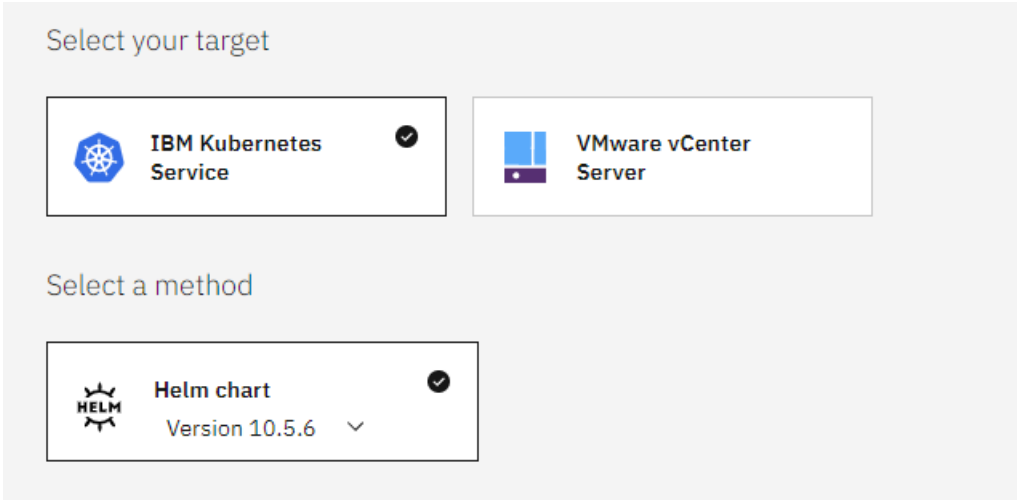
Step 3 deploy MariaDB

We will deploy MariaDB on our cluster

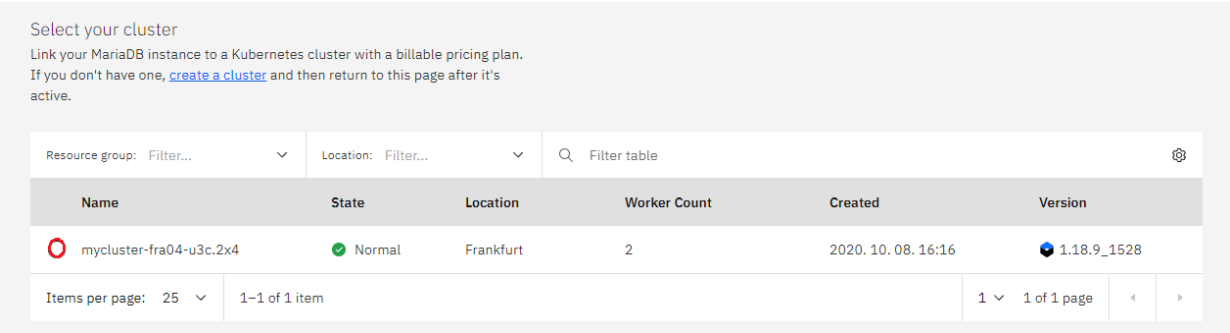
- Click the **Catalog** button on the top
- Select **Software** from the catalog
- Search for **MariaDB** and click on it



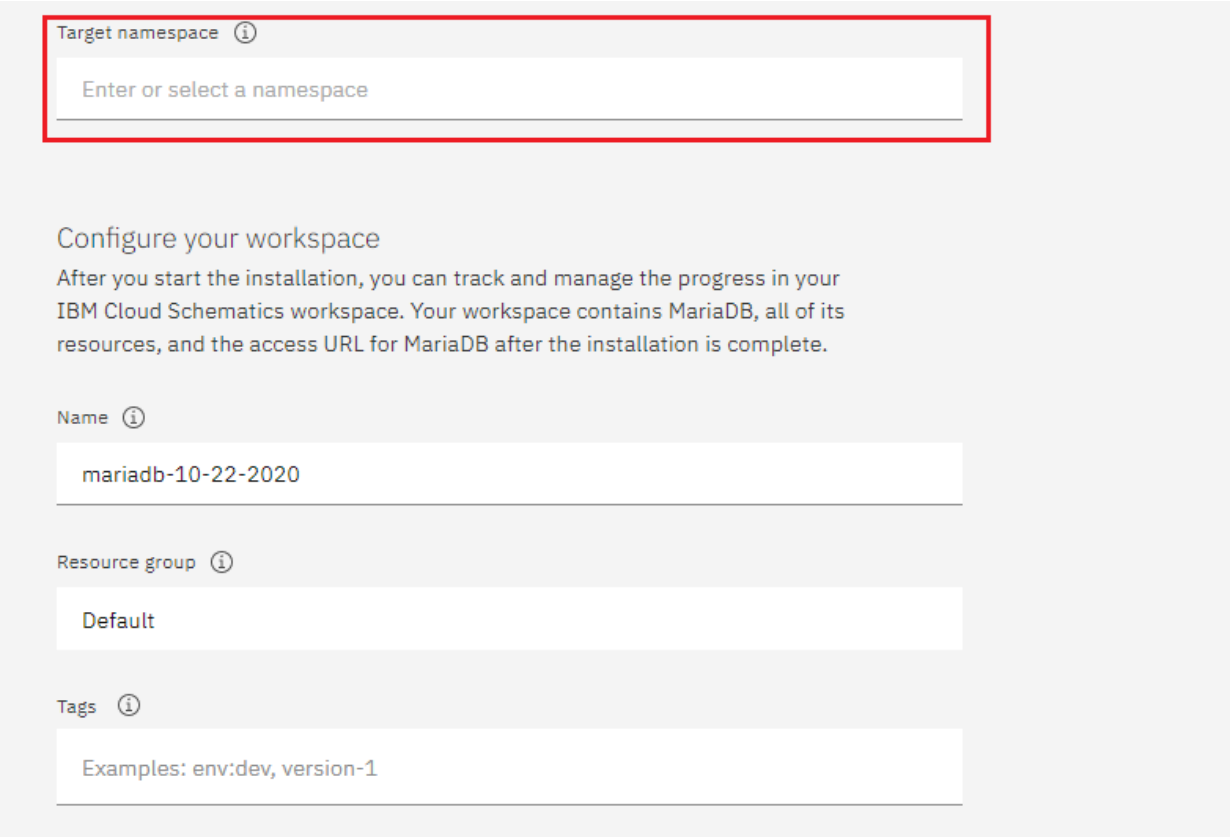
- Please select IBM Kubernetes Service



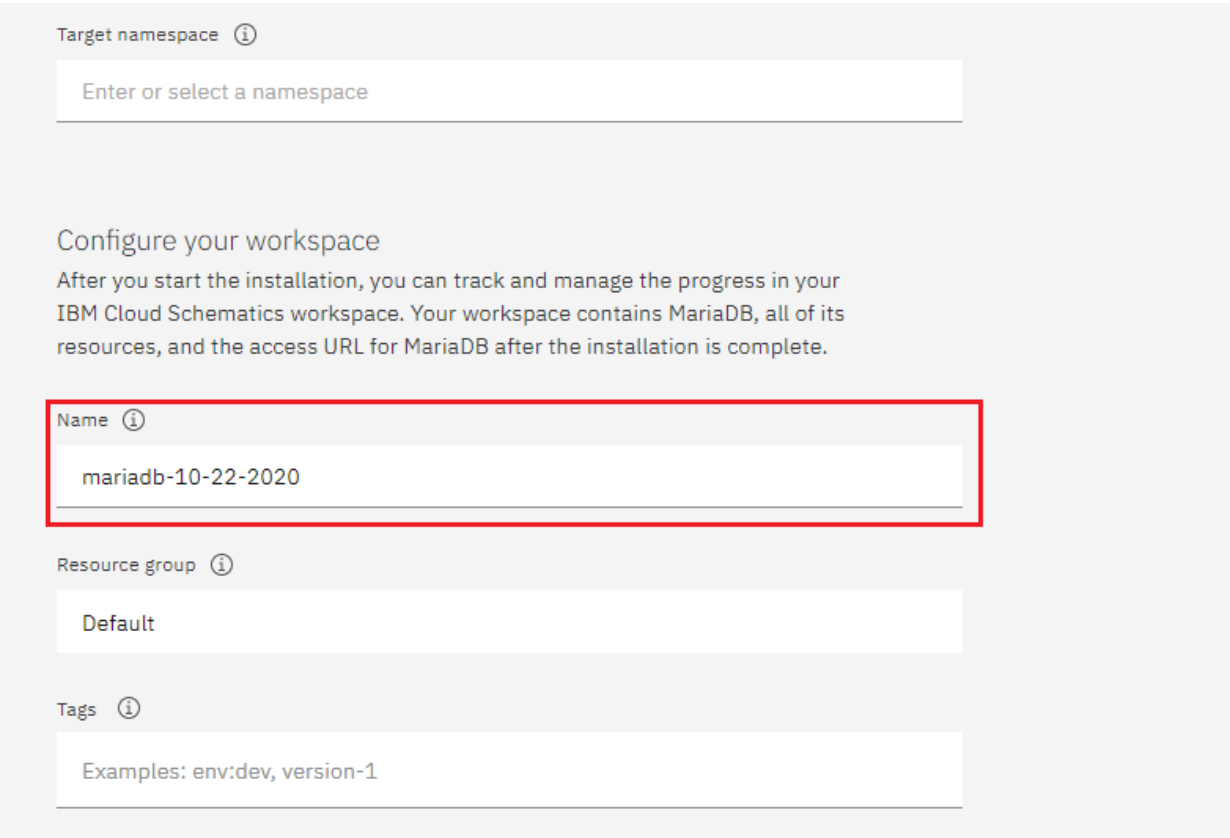
- On the application page Click in the dot next to the cluster, you wish to use



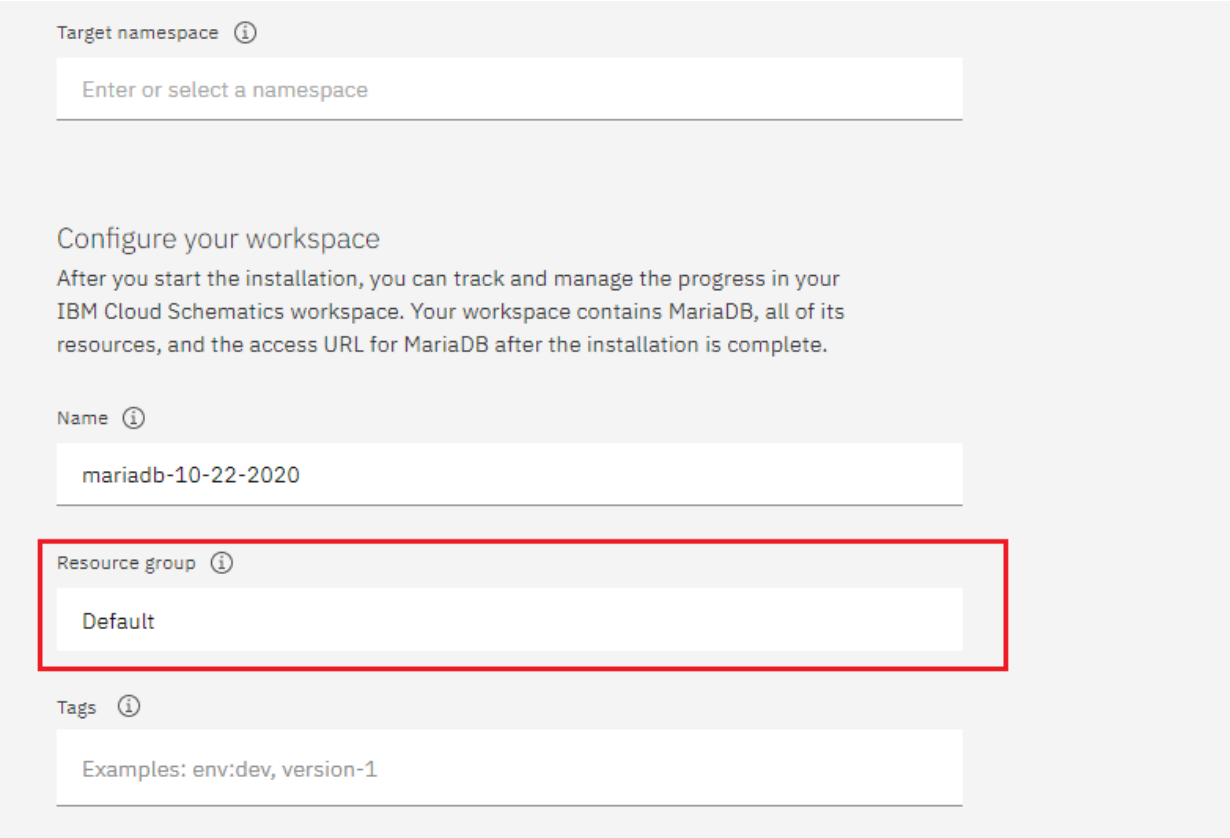
- Click on **Enter or Select Namespace** and choose the default Namespace or use a custom one



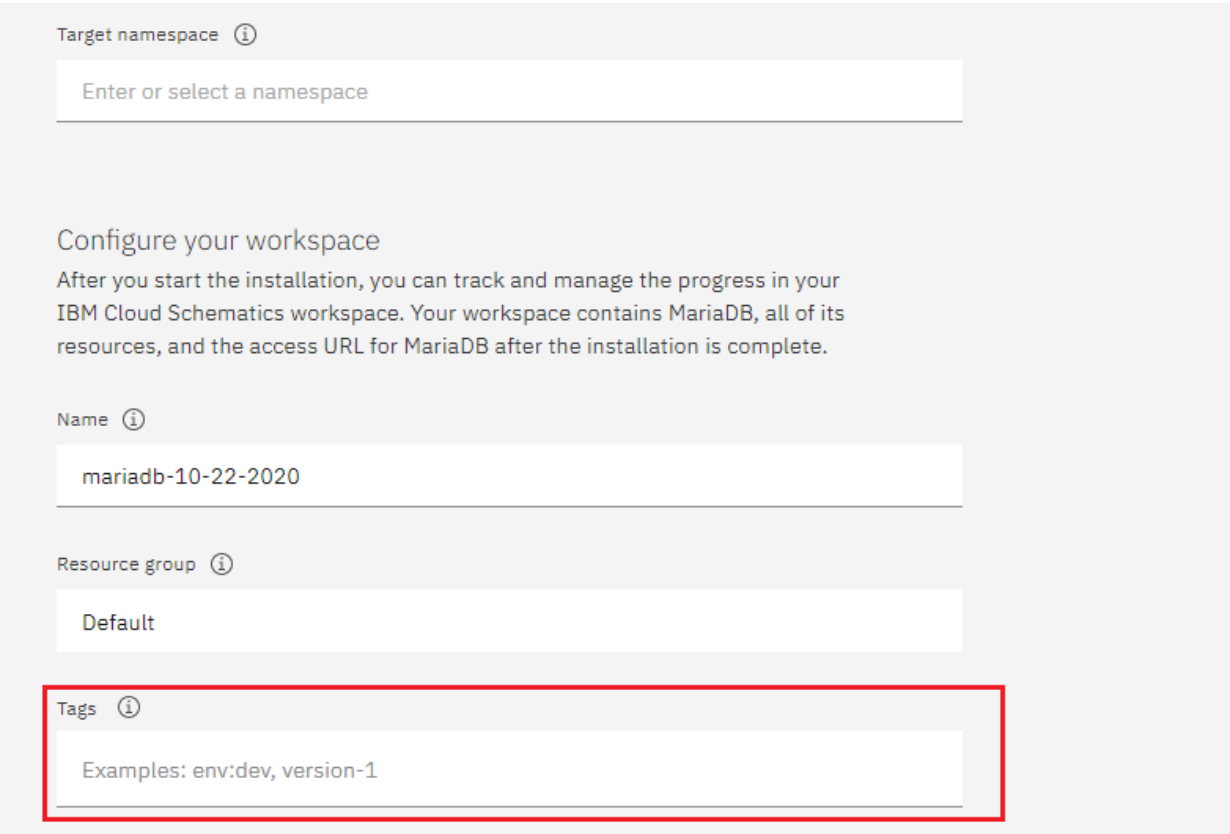
- Give a unique **name** to workspace, which you can easily recognize



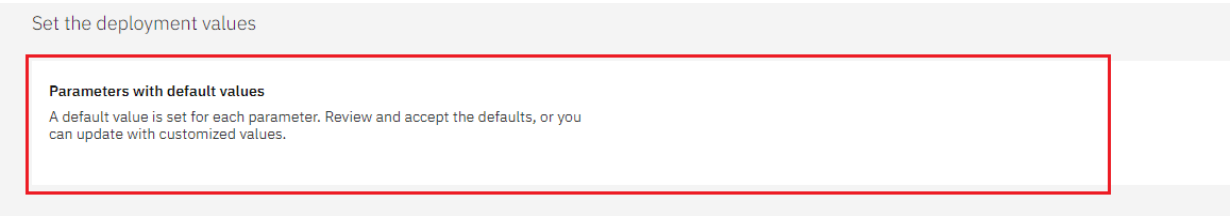
- Select which resource group you want to use, it's for access controll and billing purposes. For more information please visit resource groups



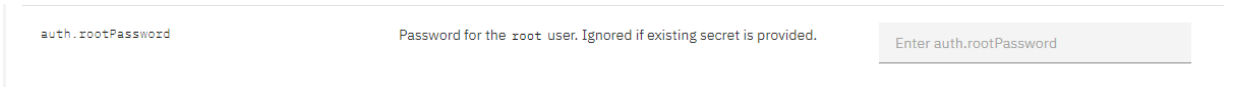
- Give **tags** to your MariaDB, for more information visit tags



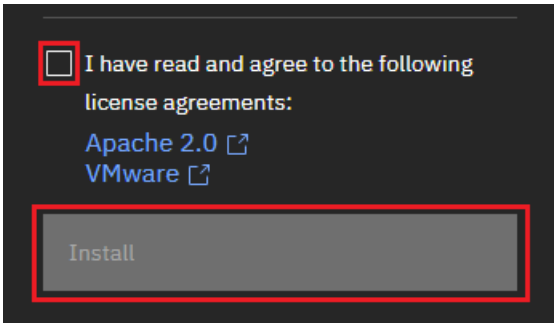
- Click on **Parameters with default values**, You can set deployment values or use the default ones



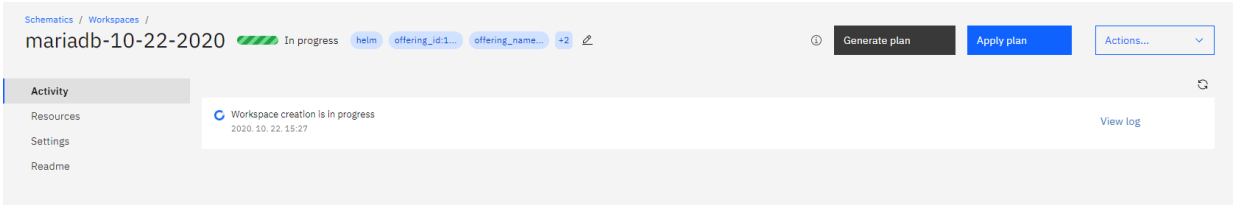
- Please set the MariaDB root password in the parameters



- After finishing everything, **tick** the box next to the agreements and click **install**



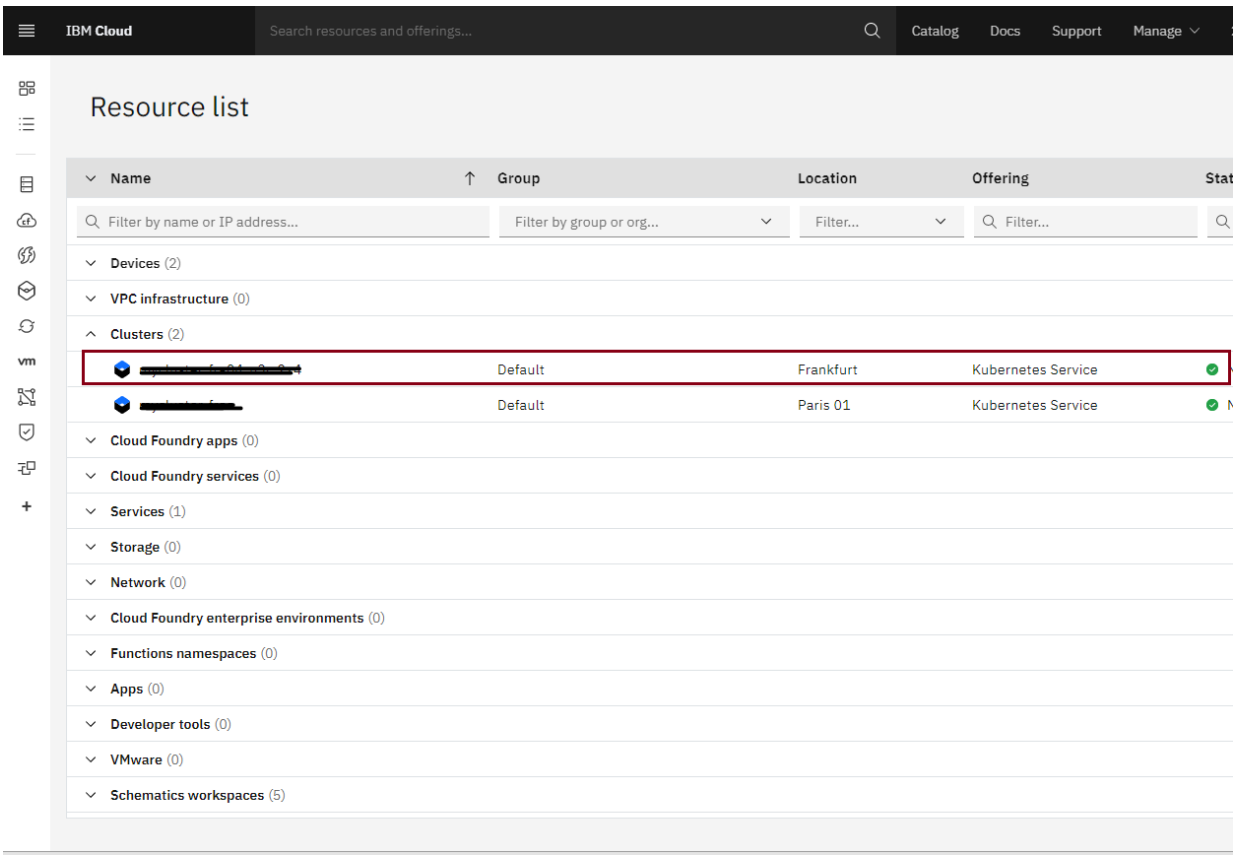
- The MariaDB workspace will start installing, wait a couple of minutes



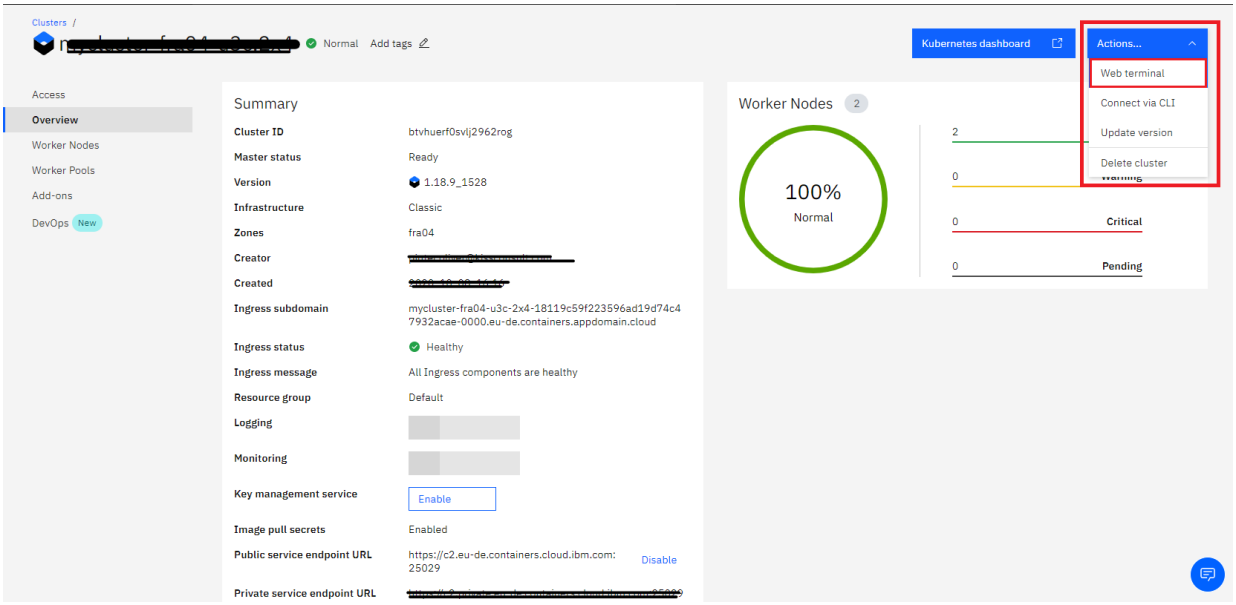
- Your MariaDB workspace has been successfully deployed

Verify MariaDB installation

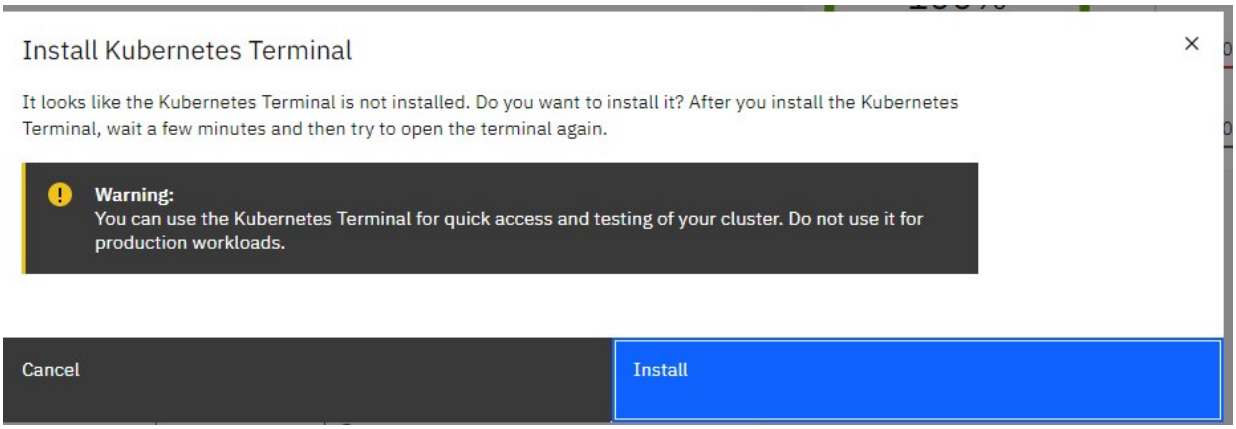
- Go to Resources in your browser
- Click on **Clusters**
- Click on your Cluster



- Now you are at you clusters overview, here Click on **Actions** and **Web terminal** from the dropdown menu



- Click **install** - wait couple of minutes



- Click on **Actions**
- Click **Web terminal** --> a terminal will open up

- **Type** in the terminal, please change NAMESPACE to the namespace you choose at the deployment setup:

```
$ kubectl get ns
```

```
pinter.oliver@k8s-terminal ~ (⎈ mycluster-fra04-u3c.2x4/btvhuerf0svlj2962rog:default)$ kubectl get ns
NAME                STATUS   AGE
default             Active   13d
discourse           Active   3d
grafana             Active   24m
ibm-cert-store      Active   13d
ibm-operators       Active   13d
ibm-system          Active   13d
kube-node-lease     Active   13d
kube-public         Active   13d
kube-system         Active   13d
mariadb             Active   20m
opencart            Active   3d3h
pinter.oliver@k8s-terminal ~ (⎈ mycluster-fra04-u3c.2x4/btvhuerf0svlj2962rog:default)$
```

```
$ kubectl get pod -n NAMESPACE -o wide
```

```
pinter.oliver@k8s-terminal ~ (⎈ mycluster-fra04-u3c.2x4/btvhuerf0svlj2962rog:default)$ kubectl get pods -n mariadb -o wide
NAME                READY   STATUS    RESTARTS   AGE   IP              NODE            NOMINATED NODE   READINESS GATES
mariadb2f-c27f-41-0   1/1     Running   0          21m   172.30.40.152   10.240.129.19   <none>           <none>
pinter.oliver@k8s-terminal ~ (⎈ mycluster-fra04-u3c.2x4/btvhuerf0svlj2962rog:default)$
```

```
$ kubectl get service -n NAMESPACE
```

```
pinter.oliver@k8s-terminal ~ (⎈ mycluster-fra04-u3c.2x4/btvhuerf0svlj2962rog:default)$ kubectl get service -n mariadb
NAME                TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
mariadb2f-c27f-41   ClusterIP   172.21.247.125   <none>        3306/TCP   22m
pinter.oliver@k8s-terminal ~ (⎈ mycluster-fra04-u3c.2x4/btvhuerf0svlj2962rog:default)$
```

- Enter your pod with bash , please replace PODNAME with your mariadb pod's name

```
$ kubectl exec --stdin --tty PODNAME -n NAMESPACE -- /bin/bash
```

```
pinter.oliver@k8s-terminal ~ (⎈ mycluster-fra04-u3c.2x4/btvhuerf0svlj2962rog:default)$ kubectl exec --stdin --tty mariadb2f-c27f-41-0 -n mariadb -- /bin/bash
```

- After you are in your pod please enter enter Mariadb and enter your root password after the prompt

```
$ mysql -u root -p
```

```
I have no name!@mariadb2f-c27f-41-0:/$ mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 370
Server version: 10.5.6-MariaDB Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

You have succesfully deployed MariaDB IBM Cloud!

2.1.6.3 mysqld Configuration Files and Groups

2.2 User & Server Security

Securing MariaDB
Securing your MariaDB installation

User Account Management
Administering user accounts in MariaDB

There are 4 related questions.

2.2.1 Securing MariaDB

> This section is about securing your MariaDB installation. If you are looking for the list of security vulnerabilities fixed in MariaDB, see Security Vulnerabilities Fixed in MariaDB.

There are a number of issues to consider when looking at improving the security of your MariaDB installation. These include:

Encryption
MariaDB supports encryption for data while at rest and while in transit.

Running mysqld as root
MariaDB should never normally be run as root

mysql_secure_installation
Symlink or old name for mariadb-secure-installation.

SecuRich
Library of security-related stored procedures.

SELinux
Security-Enhanced Linux (SELinux) is a Linux kernel module that provides a ...

There are 4 related questions.

2.2.1.1 Encryption

Data-in-Transit Encryption

Data can be encrypted in transit using the Transport Layer Security (TLS) protocol.

Data-at-Rest Encryption

MariaDB supports the use of data-at-rest encryption for tables and tablespa...

TLS and Cryptography Libraries Used by MariaDB

MariaDB supports several different TLS and cryptography libraries.

There are 3 related questions.

2.2.1.1.1 Data-in-Transit Encryption

Data can be encrypted in transit using the Transport Layer Security (TLS) protocol.

Secure Connections Overview

Data can be encrypted in transit using the TLS protocol.

Certificate Creation with OpenSSL

How to generate a self-signed certificate in OpenSSL.

Securing Connections for Client and Server

Enabling TLS encryption in transit on both the client and server.

Replication with Secure Connections

Enabling TLS encryption in transit for MariaDB replication.

Securing Communications in Galera Cluster

Enabling TLS encryption in transit for Galera Cluster.

SSL/TLS System Variables

List and description of Transport Layer Security (TLS)-related system variables.

SSL/TLS Status Variables

List and description of Transport Layer Security (TLS)-related status variables.

Using TLSv1.3

TLSv1.3 is a major rewrite of the protocol.

There are 6 related questions.

2.2.1.1.1.1 Secure Connections Overview

Contents

By default, MariaDB transmits data between the server and clients without encrypting it. This is generally acceptable when the server and client run on the same host or in networks where security is guaranteed through other means. However, in cases where the server and client exist on separate networks or they are in a high-risk network, the lack of encryption does introduce security concerns as a malicious actor could potentially eavesdrop on the traffic as it is sent over the network between them.

To mitigate this concern, MariaDB allows you to encrypt data in transit between the server and clients using the Transport Layer Security (TLS) protocol. TLS was formerly known as Secure Socket Layer (SSL), but strictly speaking the SSL protocol is a predecessor to TLS and, that version of the protocol is now considered insecure. The documentation still uses the term SSL often and for compatibility reasons TLS-related server system and status variables still use the prefix `ssl_`, but internally, MariaDB only supports its secure successors.

Checking MariaDB Server for TLS Support

In order for MariaDB Server to use TLS, it needs to be compiled with TLS support. All MariaDB packages distributed by MariaDB Foundation and MariaDB Corporation are compiled with TLS support.

If you aren't sure whether your MariaDB Server binary was compiled with TLS support, then you can check the value of the `have_ssl` system variable. For example:

```
SHOW GLOBAL VARIABLES LIKE 'have_ssl';
+---------------+----------+
| Variable_name | Value    |
+---------------+----------+
| have_ssl      | DISABLED |
+---------------+----------+
```

The possible values are:

- If it is `DISABLED`, then the server was compiled with TLS support, but TLS is not enabled.
- If it is `YES`, then the server was compiled with TLS support, and TLS is enabled.
- If it is `NO`, then the server was not compiled with TLS support.

TLS Libraries

When MariaDB is compiled with TLS and cryptography support, it is usually either statically linked with MariaDB's bundled TLS and cryptography library, which might be wolfSSL or yaSSL, or dynamically linked with the system's TLS and cryptography library, which might be OpenSSL, GnuTLS, or Schannel.

See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.

TLS Protocol Versions

There are 4 versions of the TLS protocol:

- `TLSv1.0`
- `TLSv1.1`
- `TLSv1.2`
- `TLSv1.3`

Enabling Specific TLS Protocol Versions

MariaDB starting with 10.4

The `tls_version` system variable was first introduced in MariaDB 10.4.6.

In some cases, it might make sense to only enable specific TLS protocol versions. For example, it would make sense if your organization has to comply with a specific security standard. It would also make sense if a vulnerability is found in a specific TLS protocol version, and you would like to ensure that your server does not use the vulnerable protocol version.

The PCI DSS v3.2 recommends using a minimum protocol version of TLSv1.2.

On the **server** side, users can enable specific TLS protocol versions by setting the `tls_version` system variable. This system variable accepts a comma-separated list of TLS protocol versions. A TLS protocol version will only be enabled if it is present in this list. All other TLS protocol versions will not be permitted. This system variable can be specified as a command-line argument to mysqld or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
tls_version = TLSv1.2,TLSv1.3
```

You can check which TLS protocol versions are enabled on a server by executing `SHOW GLOBAL VARIABLES`. For example:

```
SHOW GLOBAL VARIABLES LIKE 'tls_version';
```

On the **client** side, users can enable specific TLS protocol versions by setting the `--tls-version` option. This option accepts a comma-separated list of TLS protocol versions. A TLS protocol version will only be enabled if it is present in this list. All other TLS protocol versions will not be permitted. For example, to specify this option in a relevant client option group in an option file, you could set the following:

```
[client-mariadb]
...
tls_version = TLSv1.2,TLSv1.3
```

Or if you wanted to specify it on the command-line with the mariadb client, then you could execute something like this:

```
$ mariadb -u myuser -p -h myserver.mydomain.com \
   --ssl \
   --tls-version="TLSv1.2,TLSv1.3"
```

TLS Protocol Version Support

The TLS protocol versions that are supported depend on the underlying TLS library used by the specific MariaDB binary.

TLS Library	Supported TLS Protocol Versions
openSSL	TLSv1, TLSv1.1, TLSv1.2, TLSv1.3
wolfSSL	TLSv1, TLSv1.1, TLSv1.2, TLSv1.3
yaSSL	TLSv1, TLSv1.1
Schannel	TLSv1, TLSv1.1, TLSv1.2
GnuTLS	TLSv1, TLSv1.1, TLSv1.2, TLSv1.3

See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used by the server and by clients on each platform.

TLS Protocol Version Support in OpenSSL

MariaDB binaries built with the OpenSSL library (OpenSSL 1.0.1 or later) support TLSv1.1 and TLSv1.2 since MariaDB 5.5.41, MariaDB 10.0.15, and MariaDB 10.1.4.

MariaDB binaries built with the OpenSSL library (OpenSSL 1.1.1 or later) support TLSv1.3 since MariaDB 10.2.16 and MariaDB 10.3.8.

If your MariaDB Server binary is built with OpenSSL, then you can set the `ssl_cipher` system variable to values like `SSLv3` or `TLSv1.2` to allow all SSLv3.0 or all TLSv1.2 ciphers. However, this does not necessarily limit the protocol version to TLSv1.2. See MDEV-14101 for more information about that.

Note that the `TLSv1.3` ciphers cannot be excluded when using OpenSSL, even by using the `ssl_cipher` system variable. See Using TLSv1.3 for details.

SSLv3.0 is known to be vulnerable to the POODLE attack, so it should not be used. SSLv2.0 and SSLv3.0 are disabled for MariaDB Server binaries linked with OpenSSL since MariaDB 5.5.41, MariaDB 10.0.15, and MariaDB 10.1.4. If you are using a MariaDB version older than that and you cannot upgrade, then please see the section titled "SSL 3.0 Fallback protection" in OpenSSL Security Advisory - 15 Oct 2014.

TLS Protocol Version Support in wolfSSL

MariaDB binaries built with the bundled wolfSSL library support TLSv1.0, TLSv1.1, TLSv1.2, and TLSv1.3.

TLS Protocol Version Support in yaSSL

MariaDB binaries built with the bundled yaSSL library support SSLv3.0, TLSv1.0, and TLSv1.1.

SSLv3.0 is known to be vulnerable to the POODLE attack, so it should not be used. SSLv2.0 and SSLv3.0 are disabled for MariaDB Server binaries linked with yaSSL since MariaDB 5.5.41, MariaDB 10.0.15, and MariaDB 10.1.4.

TLS Protocol Version Support in Schannel

MariaDB binaries built with the Schannel library support different versions of TLS on different versions of Windows. See the Protocols in TLS/SSL (Schannel SSP) documentation from Microsoft to determine which versions of TLS are supported on each version of Windows.

TLS Protocol Version Support in GnuTLS

MariaDB binaries built with the GnuTLS library support TLSv1.0, TLSv1.1, TLSv1.2, and TLSv1.3.

Enabling TLS

See Securing Connections for Client and Server for information on how to enable TLS on the client and server.

Certificate Verification

Certificate verification is how TLS authenticates its connections by verifying that it is talking to who it says it is. There are multiple components to this verification process:

- Was the certificate signed by a trusted Certificate Authority (CA)?
- Is the certificate expired?
- Is the certificate on my Certificate Revocation List (CRL)?
- Does the certificate belong to who I believe that I'm communicating with?

Certificate Authorities (CAs)

Certificate Authorities (CAs) are entities that you trust to sign TLS certificates. Your organization might have its own internal CA, or it might use trusted third-party CAs.

CAs are specified on the server and client by using the `ssl_ca` and `ssl_capath` options.

The `ssl_ca` option defines a path to a PEM file that should contain one or more X509 certificates for trusted Certificate Authorities (CAs). This option requires that you use the absolute path, not a relative path.

The `ssl_capath` option defines a path to a directory that contains one or more PEM files that should each contain one X509 certificate for a trusted Certificate Authority (CA). This option requires that you use the absolute path, not a relative path. The `ssl_capath` option is only supported if the server or client was built with OpenSSL, wolfSSL, or yaSSL. If the client was built with GnuTLS or Schannel, then the `ssl_capath` option is not supported.

See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.

The directory specified by `ssl_capath` needs to be run through the `openssl rehash` command. For example, if the following is configured:

```
ssl_capath=/etc/my.cnf.d/certificates/ca/
```

Then you would have to execute the following:

```
openssl rehash /etc/my.cnf.d/certificates/ca/
```

Requiring a Specific Certificate Authority (CA)

The server can require a specific Certificate Authority (CA) for a client if the client's user account has been defined with `REQUIRE ISSUER`. See Securing Connections for Client and Server: Requiring TLS for more information.

Certificate Revocation Lists (CRLs)

Certificate Revocation Lists (CRLs) are lists of certificates that have been revoked by the Certificate Authority (CA) before they were due to expire.

CRLs are specified on the server and client by using the `ssl_crl` and `ssl_crlpath` options.

The `ssl_crl` option defines a path to a PEM file that should contain one or more X509 revoked certificates. This option requires that you use the absolute path, not a relative path. For servers, the `ssl_crl` option is only valid if the server was built with OpenSSL. If the server was built with wolfSSL or yaSSL, then the `ssl_crl` option is not supported. For clients, the `ssl_crl` option is only valid if the client was built with OpenSSL or Schannel. Likewise, if the client was built with GnuTLS, wolfSSL or yaSSL, then the `ssl_crl` option is not supported.

The `ssl_crlpath` option defines a path to a directory that contains one or more PEM files that should each contain one revoked X509 certificate. This option requires that you use the absolute path, not a relative path. The `ssl_crlpath` option is only supported if the server or client was built with OpenSSL. If the server was built with wolfSSL or yaSSL, then the `ssl_crlpath` option is not supported. Likewise, if the client was built with GnuTLS, Schannel, wolfSSL, or yaSSL, then the `ssl_crlpath` option is not supported.

See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.

The directory specified by `ssl_crlpath` needs to be run through the `openssl rehash` command. For example, if the following is configured:

```
ssl_crlpath=/etc/my.cnf.d/certificates/crl/
```

Then you would have to execute the following:

```
openssl rehash /etc/my.cnf.d/certificates/crl/
```

Server Certificate Verification

Clients and utilities verify a server certificate by checking the server's host name and IP address against certain attributes in the certificate. For most clients and utilities, server certificate verification is disabled by default, and it is only enabled if an option, such as `ssl-verify-server-cert` is specified.

To verify the server's certificate, clients and utilities will check the **Common Name (CN)** attribute located in the **Subject** field of the certificate against the server's host name and IP address. If the **Common Name (CN)** matches either of those, then the certificate is verified.

Server Certificate Verification with Subject Alternative Names (SANs)

The **Subject Alternative Name (SAN)** field, which is an X.509v3 extension, can also be used for server certificate verification, if it is present in the server certificate. This field is also sometimes called **subjectAltName**. When using a client

or utility that supports server certificate verification with **subjectAltName** fields, if the server certificate contains any **subjectAltName** fields, then those fields will also be checked against the server's host name and IP address.

Whether server certificate verification with **subjectAltName** fields is supported depends on the underlying TLS library used by the client or utility.

See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.

SAN Support with OpenSSL, wolfSSL, and yaSSL

For clients and utilities built with OpenSSL (OpenSSL 1.0.2 or later), support for server certificate verification with **subjectAltName** fields that contain the server's **host name** was added in MariaDB 10.1.23 and MariaDB 10.2.6. See MDEV-10594 for more information.

For clients and utilities built with OpenSSL (OpenSSL 1.0.2 or later), support for server certificate verification with **subjectAltName** fields that contain the server's **IP address** was added in MariaDB 10.1.39, MariaDB 10.2.24, MariaDB 10.3.15, and MariaDB 10.4.5. See MDEV-18131 for more information.

This support also applies to other TLS libraries that use OpenSSL's API. In OpenSSL's API, server certificate verification with **subjectAltName** fields depends on the `X509_check_host` and `X509_check_ip` functions. These functions are supported in the following TLS libraries:

- OpenSSL 1.0.2 or later
- wolfSSL

And they are **not** supported in the following TLS libraries:

- yaSSL

MariaDB's RPM packages were built with OpenSSL 1.0.1 on RHEL 7 and CentOS 7, even after OpenSSL 1.0.2 became available on those distributions. As a side effect, the clients and utilities bundled in these packages did not support server certificate verification with the **subjectAltName** field, even if the packages were installed on a system that had OpenSSL 1.0.2 installed. Starting with MariaDB MariaDB 10.1.39, MariaDB 10.2.23, MariaDB 10.3.14, and MariaDB 10.4.4, MariaDB's RPM packages on RHEL 7 and CentOS 7 are built with OpenSSL 1.0.2. See MDEV-18277 for more information.

SAN Support with Schannel

For clients and utilities linked with Schannel, support for server certificate verification with **subjectAltName** fields was added in MariaDB Connector/C 3.0.2. See CONC-250 for more information.

SAN Support with GnuTLS

For clients and utilities linked with GnuTLS, support for server certificate verification with **subjectAltName** fields was added in MariaDB Connector/C 3.0.0. See CONC-250 for more information.

Client Certificate Verification

The server verifies a client certificate by checking the client's known `SUBJECT` against the **Subject** attribute in the client's certificate. This is only done for user accounts that have been defined with `REQUIRE SUBJECT`. See Securing Connections for Client and Server: Requiring TLS for more information.

2.2.1.1.1.2 Certificate Creation with OpenSSL

Contents

Warning: the instructions below generate version 1 certificates only. These work fine with servers and clients using OpenSSL, but fail if WolfSSL is used instead, as is the case for our Windows MSI packages and our binary tarballs for Linux.

WolfSSL requires version 3 certificates instead when using TLS v1.2 or higher, and so won't work with certificates generated as shown here when using two-way TLS with explicit client certificates.

Generating version 3 certificates requires a few more minor steps, we will upgrade the instructions below soon to

include these.

See also: MDEV-25701

In order to secure communications with the MariaDB Server using TLS, you need to create a private key and an X509 certificate for the server. You may also want to create additional private keys and X509 certificates for any clients that need to connect to the server with TLS. This guide covers how to create a private key and a self-signed X509 certificate with OpenSSL.

Certificate Creation

The OpenSSL library provides a command-line tool called `openssl`, which can be used for performing various tasks with the library, such as generating private keys, creating X509 certificate requests, signing X509 certificates as a Certificate Authority (CA), and verifying X509 certificates.

Creating a Certificate Authority Private Key and Certificate

The Certificate Authority (CA) is typically an organization (such as Let's Encrypt) that signs the X509 certificate and validates ownership of the domain. However, when you would like to use self-signed certificates, you need to create the private key and certificate for the CA yourself, and then you can use them to sign your own X509 certificates.

To start, generate a private key for the CA using the `openssl genrsa` command. For example:

```
# openssl genrsa 2048 > ca-key.pem
```

After that, you can use the private key to generate the X509 certificate for the CA using the `openssl req` command. For example:

```
# openssl req -new -x509 -nodes -days 365000 \
      -key ca-key.pem -out ca.pem
```

The above commands create two files in the working directory: The `ca-key.pem` private key and the `ca.pem` X509 certificate are both are used by the CA to create self-signed X509 certificates below.

Creating a Private Key and a Self-signed Certificate

Once you have the CA's private key and X509 certificate, you can create the self-signed X509 certificates to use for the MariaDB Server, client, replication and other purposes.

To start, generate a private key and create a certificate request using the `openssl req` command. For example:

```
# openssl req -newkey rsa:2048 -days 365000 \
      -nodes -keyout server-key.pem -out server-req.pem
```

After that, process the key to remove the passphrase using the `openssl rsa` command. For example:

```
# openssl rsa -in server-key.pem -out server-key.pem
```

Lastly, using the certificate request and the CA's private key and X509 certificate, you can generate a self-signed X509 certificate from the certificate request using the `openssl x509` command. For example:

```
# openssl x509 -req -in server-req.pem -days 365000 \
      -CA ca.pem -CAkey ca-key.pem -set_serial 01 \
      -out server-cert.pem
```

This creates a `server-cert.pem` file, which is the self-signed X509 certificate.

Certificate Verification

Once you have created the CA's X509 certificate and a self-signed X509 certificate, you can verify that the X509 certificate was correctly generated using the `openssl verify` command. For example:

```
# openssl verify -CAfile ca.pem server-cert.pem
server-cert.pem: OK
```

You can add as many X509 certificates to check against the CA's X509 certificate as you want to verify. A value of `OK` indicates that you can use it was correctly generated and is ready for use with MariaDB.

2.2.1.1.1.3 Securing Connections for Client and Server

Contents

By default, MariaDB transmits data between the server and clients without encrypting it. This is generally acceptable when the server and client run on the same host or in networks where security is guaranteed through other means. However, in cases where the server and client exist on separate networks or they are in a high-risk network, the lack of encryption does introduce security concerns as a malicious actor could potentially eavesdrop on the traffic as it is sent over the network between them.

To mitigate this concern, MariaDB allows you to encrypt data in transit between the server and clients using the Transport Layer Security (TLS) protocol. TLS was formerly known as Secure Socket Layer (SSL), but strictly speaking the SSL protocol is a predecessor to TLS and, that version of the protocol is now considered insecure. The documentation still uses the term SSL often and for compatibility reasons TLS-related server system and status variables still use the prefix `ssl_`, but internally, MariaDB only supports its secure successors.

In order to secure connections between the server and client, you need to ensure that your server was compiled with TLS support. See Secure Connections Overview to determine how to check whether a server was compiled with TLS support.

You also need an X509 certificate, a private key, and the Certificate Authority (CA) chain to verify the X509 certificate for the server. If you want to use two-way TLS, then you will also need an X509 certificate, a private key, and the Certificate Authority (CA) chain to verify the X509 certificate for the client. If you want to use self-signed certificates that are created with OpenSSL, then see Certificate Creation with OpenSSL for information on how to create those.

Enabling TLS

Enabling TLS for MariaDB Server

In order to enable TLS on a MariaDB server that was compiled with TLS support, there are a number of system variables that you need to set, such as:

- You need to set the path to the server's X509 certificate by setting the ssl_cert system variable.
- You need to set the path to the server's private key by setting the ssl_key system variable.
- You need to set the path to the certificate authority (CA) chain that can verify the server's certificate by setting either the ssl_ca or the ssl_capath system variables.
- If you want to restrict the server to certain ciphers, then you also need to set the ssl_cipher system variable.

For example, to set these variables for the server, add the system variables to a relevant server option group in an option file:

```
[mariadb]
...
ssl_cert = /etc/my.cnf.d/certificates/server-cert.pem
ssl_key = /etc/my.cnf.d/certificates/server-key.pem
ssl_ca = /etc/my.cnf.d/certificates/ca.pem
```

And then restart the server to make the changes persistent.

Once the server is back up, you can check that TLS is enabled by checking the value of the have_ssl system variable. For example:

```
SHOW VARIABLES LIKE 'have_ssl';

+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| have_ssl      | YES   |
+---------------+-------+
```

Reloading the Server's Certificates and Keys Dynamically

MariaDB starting with 10.4

The `FLUSH SSL` command was first added in MariaDB 10.4.

In MariaDB 10.4 and later, the `FLUSH SSL` command can be used to dynamically reinitialize the server's TLS context.

See FLUSH SSL for more information.

Enabling TLS for MariaDB Clients

Different clients and utilities may use different methods to enable TLS.

For many of the standard clients and utilities that come bundled with MariaDB, you can enable two-way TLS by adding the same options that were set for the server to a relevant client option group in an option file. For example:

```
[client-mariadb]
...
ssl_cert = /etc/my.cnf.d/certificates/client-cert.pem
ssl_key = /etc/my.cnf.d/certificates/client-key.pem
ssl_ca = /etc/my.cnf.d/certificates/ca.pem
```

The specific options that you would need to set would depend on whether you want one-way TLS or two-way TLS, and whether you want to verify the server certificate.

The same options may also enable TLS on non-standard clients and utilities that are linked with either libmysqlclient or MariaDB Connector/C.

Enabling Two-Way TLS for MariaDB Clients

Two-way TLS means that both the client and server provide a private key and an X509 certificate. It is called "two-way" TLS because both the client and server can be authenticated. For example, to specify these options in a relevant client option group in an option file, you could set the following:

```
[client-mariadb]
...
ssl_cert = /etc/my.cnf.d/certificates/client-cert.pem
ssl_key = /etc/my.cnf.d/certificates/client-key.pem
ssl_ca = /etc/my.cnf.d/certificates/ca.pem
ssl-verify-server-cert
```

Or if you wanted to specify them on the command-line with the mariadb client, then you could execute something like this:

```
$ mariadb -u myuser -p -h myserver.mydomain.com \
   --ssl-cert=/etc/my.cnf.d/certificates/client-cert.pem \
   --ssl-key=/etc/my.cnf.d/certificates/client-key.pem \
   --ssl-ca=/etc/my.cnf.d/certificates/ca.pem \
   --ssl-verify-server-cert
```

Two-way SSL is required for an account if the `REQUIRE X509`, `REQUIRE SUBJECT`, and/or `REQUIRE ISSUER` clauses are specified for the account.

Enabling One-Way TLS for MariaDB Clients

Enabling One-Way TLS for MariaDB Clients with Server Certificate Verification

One-way TLS means that only the server provides a private key and an X509 certificate. When TLS is used without a client certificate, it is called "one-way" TLS, because only the server can be authenticated, so authentication is only possible in one direction. However, encryption is still possible in both directions. Server certificate verification means that the client verifies that the certificate belongs to the server. For example, to specify these options in a a relevant client option group in an option file, you could set the following:

```
[client-mariadb]
...
ssl_ca = /etc/my.cnf.d/certificates/ca.pem
ssl-verify-server-cert
```

Or if you wanted to specify them on the command-line with the mariadb client, then you could execute something like this:

```
$ mariadb -u myuser -p -h myserver.mydomain.com \
   --ssl-ca=/etc/my.cnf.d/certificates/ca.pem \
   --ssl-verify-server-cert
```

Enabling One-Way TLS for MariaDB Clients without Server Certificate Verification

One-way TLS means that only the server provides a private key and an X509 certificate. When TLS is used without a client certificate, it is called "one-way" TLS, because only the server can be authenticated, so authentication is only possible in one direction. However, encryption is still possible in both directions. For example, to specify these options in a a relevant client option group in an option file, you could set the following:

```
[client-mariadb]
...
ssl
```

Or if you wanted to specify them on the command-line with the mariadb client, then you could execute something like this:

```
$ mariadb -u myuser -p -h myserver.mydomain.com \
   --ssl
```

Enabling TLS for MariaDB Connector/C Clients

See the documentation on MariaDB Connector/C's TLS Options for information on how to enable TLS for clients that use MariaDB Connector/C.

Enabling TLS for MariaDB Connector/ODBC Clients

See the documentation on MariaDB Connector/ODBC's TLS-Related Connection Parameters for information on how to enable TLS for clients that use MariaDB Connector/ODBC.

Enabling TLS for MariaDB Connector/J Clients

See the documentation on Using TLS/SSL with MariaDB Connector/J for information on how to enable TLS for clients that use MariaDB Connector/J.

Verifying that a Connection is Using TLS

You can verify that a connection is using TLS by checking the connection's Ssl_cipher status variable. If it is non-empty, then the connection is using TLS. For example:

```
SHOW SESSION STATUS LIKE 'Ssl_cipher';
+---------------+--------------------------+
| Variable_name | Value                    |
+---------------+--------------------------+
| Ssl_cipher    | DHE-RSA-AES256-GCM-SHA384 |
+---------------+--------------------------+
1 row in set (0.00 sec)
```

Requiring TLS

From MariaDB 10.5.2, the require_secure_transport system variable is available. When set (by default it is off), connections attempted using insecure transport will be rejected. Secure transports are SSL/TLS, Unix sockets or named pipes. Note that requirements set for specific user accounts will take precedence over this setting.

Requiring TLS for Specific User Accounts

You can set certain TLS-related restrictions for specific user accounts. For instance, you might use this with user accounts that require access to sensitive data while sending it across networks that you do not control. These restrictions can be enabled for a user account with the CREATE USER, ALTER USER, or GRANT statements. For example:

- A user account must connect via TLS if the user account is defined with the `REQUIRE SSL` clause.

```
ALTER USER 'alice'@'%'
   REQUIRE SSL;
```

- A user account must connect via TLS with a specific cipher if the user account is defined with the `REQUIRE CIPHER` clause.

```
ALTER USER 'alice'@'%'
   REQUIRE CIPHER 'ECDH-RSA-AES256-SHA384';
```

- A user account must connect via TLS with a valid client certificate if the user account is defined with the `REQUIRE X509` clause.

```
ALTER USER 'alice'@'%'
   REQUIRE X509;
```

- A user account must connect via TLS with a specific client certificate if the user account is defined with the `REQUIRE SUBJECT` clause.

```
ALTER USER 'alice'@'%'
   REQUIRE SUBJECT '/CN=alice/O=My Dom, Inc./C=US/ST=Oregon/L=Portland';
```

- A user account must connect via TLS with a client certificate that must be signed by a specific certificate authority if the user account is defined with the `REQUIRE ISSUER` clause.

```
ALTER USER 'alice'@'%'
   REQUIRE SUBJECT '/CN=alice/O=My Dom, Inc./C=US/ST=Oregon/L=Portland'
   AND ISSUER '/C=FI/ST=Somewhere/L=City/ O=Some Company/CN=Peter
Parker/emailAddress=p.parker@marvel.com';
```

Requiring TLS for Specific User Accounts from Specific Hosts

A user account can have different definitions depending on what host the user account is logging in from. Therefore, it is possible to have different TLS requirements for the same username for different hosts. For example:

```
CREATE USER 'alice'@'localhost'
   REQUIRE NONE;

CREATE USER 'alice'@'%'
   REQUIRE SUBJECT '/CN=alice/O=My Dom, Inc./C=US/ST=Oregon/L=Portland'
   AND ISSUER '/C=FI/ST=Somewhere/L=City/ O=Some Company/CN=Peter
Parker/emailAddress=p.parker@marvel.com'
   AND CIPHER 'ECDHE-ECDSA-AES256-SHA384';
```

In the above example, the `alice` user account does not require TLS when logging in from localhost. However, when the `alice` user account logs in from any other host, they must use TLS with the given cipher, and they must provide a valid client certificate with the given subject that must have been signed by the given issuer.

2.2.1.1.1.4 Replication with Secure Connections

The terms *master* and *slave* have historically been used in replication, but the terms terms *primary* and *replica* are now preferred. The old terms are used still used in parts of the documentation, and in MariaDB commands, although MariaDB 10.5 has begun the process of renaming. The documentation process is ongoing. See MDEV-18777 to follow progress on this effort.

Contents

By default, MariaDB replicates data between primaries and replicas without encrypting it. This is generally acceptable when the primary and replica run are in networks where security is guaranteed through other means. However, in cases where the primary and replica exist on separate networks or they are in a high-risk network, the lack of encryption does introduce security concerns as a malicious actor could potentially eavesdrop on the traffic as it is sent over the network between them.

To mitigate this concern, MariaDB allows you to encrypt replicated data in transit between primaries and replicas using the Transport Layer Security (TLS) protocol. TLS was formerly known as Secure Socket Layer (SSL), but strictly speaking the SSL protocol is a predecessor to TLS and, that version of the protocol is now considered insecure. The documentation still uses the term SSL often and for compatibility reasons TLS-related server system and status variables still use the prefix `ssl_`, but internally, MariaDB only supports its secure successors.

In order to secure connections between the primary and replica, you need to ensure that both servers were compiled with TLS support. See Secure Connections Overview to determine how to check whether a server was compiled with TLS support.

You also need an X509 certificate, a private key, and the Certificate Authority (CA) chain to verify the X509 certificate for the primary. If you want to use two-way TLS, then you will also an X509 certificate, a private key, and the Certificate Authority (CA) chain to verify the X509 certificate for the replica. If you want to use self-signed certificates that are created with OpenSSL, then see Certificate Creation with OpenSSL for information on how to create those.

Securing Replication Traffic

In order to secure replication traffic, you will need to ensure that TLS is enabled on the primary. If you want to use two-way TLS, then you will also need to ensure that TLS is enabled on the replica. See Securing Connections for Client and Server for information on how to do that.

For example, to set the TLS system variables for each server, add them to a relevant server option group in an option file on each server:

```
[mariadb]
...
ssl_cert = /etc/my.cnf.d/certificates/server-cert.pem
ssl_key = /etc/my.cnf.d/certificates/server-key.pem
ssl_ca = /etc/my.cnf.d/certificates/ca.pem
```

And then restart the server to make the changes persistent.

At this point, you can reconfigure the replicas to use TLS to encrypt replicated data in transit. There are two methods available to do this:

- Executing the CHANGE MASTER statement to set the relevant TLS options.
- Setting TLS client options in an option file.

Executing CHANGE MASTER

TLS can be enabled on a replication replica by executing the CHANGE MASTER statement. In order to do so, there are a

number of options that you would need to set. The specific options that you would need to set would depend on whether you want one-way TLS or two-way TLS, and whether you want to verify the server certificate.

Enabling Two-Way TLS with CHANGE MASTER

Two-way TLS means that both the client and server provide a private key and an X509 certificate. It is called "two-way" TLS because both the client and server can be authenticated. In this case, the "client" is the replica. To configure two-way TLS, you would need to set the following options:

- You need to set the path to the server's certificate by setting the MASTER_SSL_CERT option.
- You need to set the path to the server's private key by setting the MASTER_SSL_KEY option.
- You need to set the path to the certificate authority (CA) chain that can verify the server's certificate by setting either the MASTER_SSL_CA or the MASTER_SSL_CAPATH options.
- If you want server certificate verification, then you also need to set the MASTER_SSL_VERIFY_SERVER_CERT option (enabled by default from MariaDB 11.3).
- If you want to restrict the server to certain ciphers, then you also need to set the MASTER_SSL_CIPHER option.

If the replica threads are currently running, you first need to stop them by executing the STOP SLAVE statement. For example:

```
STOP SLAVE;
```

Then, execute the CHANGE MASTER statement to configure the replica to use TLS. For example:

```
CHANGE MASTER TO
   MASTER_SSL_CERT = '/path/to/client-cert.pem',
   MASTER_SSL_KEY = '/path/to/client-key.pem',
   MASTER_SSL_CA = '/path/to/ca/ca.pem',
   MASTER_SSL_VERIFY_SERVER_CERT=1;
```

At this point, you can start replication by executing the START SLAVE statement. For example:

```
START SLAVE;
```

The replica now uses TLS to encrypt data in transit as it replicates it from the primary.

Enabling One-Way TLS with CHANGE MASTER

Enabling One-Way TLS with CHANGE MASTER with Server Certificate Verification

One-way TLS means that only the server provides a private key and an X509 certificate. When TLS is used without a client certificate, it is called "one-way" TLS, because only the server can be authenticated, so authentication is only possible in one direction. However, encryption is still possible in both directions. Server certificate verification means that the client verifies that the certificate belongs to the server. In this case, the "client" is the replica. This mode is enabled by default starting from MariaDB 11.3. To configure one-way TLS in earlier versions, you would need to set the following options:

- You need to set the path to the certificate authority (CA) chain that can verify the server's certificate by setting either the MASTER_SSL_CA or the MASTER_SSL_CAPATH options.
- You need to set the MASTER_SSL_VERIFY_SERVER_CERT option.
- If you want to restrict the server to certain ciphers, then you also need to set the MASTER_SSL_CIPHER option.

If the replica threads are currently running, you first need to stop them by executing the STOP SLAVE statement. For example:

```
STOP SLAVE;
```

Then, execute the CHANGE MASTER statement to configure the replica to use TLS. For example:

```
CHANGE MASTER TO
   MASTER_SSL_CA = '/path/to/ca/ca.pem',
   MASTER_SSL_VERIFY_SERVER_CERT=1;
```

At this point, you can start replication by executing the START SLAVE statement. For example:

```
START SLAVE;
```

The replica now uses TLS to encrypt data in transit as it replicates it from the primary.

Enabling One-Way TLS with CHANGE MASTER without Server Certificate Verification

One-way TLS means that only the server provides a private key and an X509 certificate. When TLS is used without a client certificate, it is called "one-way" TLS, because only the server can be authenticated, so authentication is only possible in one direction. However, encryption is still possible in both directions. In this case, the "client" is the replica. To configure two-way TLS without server certificate verification, you would need to set the following options:

- You need to configure the replica to use TLS by setting the MASTER_SSL option.
- If you want to restrict the server to certain ciphers, then you also need to set the MASTER_SSL_CIPHER option.
- Starting from MariaDB 11.3 you need to disable the MASTER_SSL_VERIFY_SERVER_CERT option.

If the replica threads are currently running, you first need to stop them by executing the STOP SLAVE statement. For example:

```
STOP SLAVE;
```

Then, execute the CHANGE MASTER statement to configure the replica to use TLS. For example:

```
CHANGE MASTER TO
   MASTER_SSL=1, MASTER_SSL_VERIFY_SERVER_CERT=0;
```

At this point, you can start replication by executing the START SLAVE statement. For example:

```
START SLAVE;
```

The replica now uses TLS to encrypt data in transit as it replicates it from the primary.

2.2.1.1.1.5 Securing Communications in Galera Cluster

Contents

By default, Galera Cluster replicates data between each node without encrypting it. This is generally acceptable when the cluster nodes runs on the same host or in networks where security is guaranteed through other means. However, in cases where the cluster nodes exist on separate networks or they are in a high-risk network, the lack of encryption does introduce security concerns as a malicious actor could potentially eavesdrop on the traffic or get a complete copy of the data by triggering an SST.

To mitigate this concern, Galera Cluster allows you to encrypt data in transit as it is replicated between each cluster node using the Transport Layer Security (TLS) protocol. TLS was formerly known as Secure Socket Layer (SSL), but strictly speaking the SSL protocol is a predecessor to TLS and, that version of the protocol is now considered insecure. The documentation still uses the term SSL often and for compatibility reasons TLS-related server system and status variables still use the prefix `ssl_`, but internally, MariaDB only supports its secure successors.

In order to secure connections between the cluster nodes, you need to ensure that all servers were compiled with TLS support. See Secure Connections Overview to determine how to check whether a server was compiled with TLS support.

For each cluster node, you also need a certificate, private key, and the Certificate Authority (CA) chain to verify the certificate. If you want to use self-signed certificates that are created with OpenSSL, then see Certificate Creation with OpenSSL for information on how to create those.

Securing Galera Cluster Replication Traffic

In order to enable TLS for Galera Cluster's replication traffic, there are a number of wsrep_provider_options that you need to set, such as:

- You need to set the path to the server's certificate by setting the `socket.ssl_cert` wsrep_provider_option.
- You need to set the path to the server's private key by setting the `socket.ssl_key` wsrep_provider_option.
- You need to set the path to the certificate authority (CA) chain that can verify the server's certificate by setting the `socket.ssl_ca` wsrep_provider_option.

- If you want to restrict the server to certain ciphers, then you also need to set the `socket.ssl_cipher` wsrep_provider_option.

It is also a good idea to set MariaDB Server's regular TLS-related system variables, so that TLS will be enabled for regular client connections as well. See Securing Connections for Client and Server for information on how to do that.

For example, to set these variables for the server, add the system variables to a relevant server option group in an option file:

```
[mariadb]
...
ssl_cert = /etc/my.cnf.d/certificates/server-cert.pem
ssl_key = /etc/my.cnf.d/certificates/server-key.pem
ssl_ca = /etc/my.cnf.d/certificates/ca.pem
wsrep_provider_options="socket.ssl_cert=/etc/my.cnf.d/certificates/server-
cert.pem;socket.ssl_key=/etc/my.cnf.d/certificates/server-
key.pem;socket.ssl_ca=/etc/my.cnf.d/certificates/ca.pem"
```

And then restart the server to make the changes persistent.

By setting both MariaDB Server's TLS-related system variables and Galera Cluster's TLS-related wsrep_provider_options, the server can secure both external client connections and Galera Cluster's replication traffic.

Securing State Snapshot Transfers

The method that you would use to enable TLS for State Snapshot Transfers (SSTs) would depend on the value of `wsrep_sst_method`.

mariabackup

See mariabackup SST Method: TLS for more information.

xtrabackup-v2

See xtrabackup-v2 SST Method: TLS for more information.

mysqldump

This SST method simply uses the mariadb-dump (previously mysqldump) utility, so TLS would be enabled by following the guide at Securing Connections for Client and Server: Enabling TLS for MariaDB Clients

rsync

This SST method supports encryption in transit via `stunnel`. See Introduction to State Snapshot Transfers (SSTs): rsync for more information.

2.2.1.1.1.6 SSL/TLS System Variables

Contents

The system variables listed on this page relate to encrypting data during transfer between servers and clients using the Transport Layer Security (TLS) protocol. Often, the term Secure Sockets Layer (SSL) is used interchangeably with TLS, although strictly speaking the SSL protocol is the predecessor of TLS and is no longer considered secure.

For compatibility reasons, the TLS system variables in MariaDB still use the `ssl_` prefix, but MariaDB only supports its more secure successors. For more information on SSL/TLS in MariaDB, see Secure Connections Overview.

Variables

have_openssl

- **Description:** This variable shows whether the server is linked with OpenSSL rather than MariaDB's bundled TLS library, which might be wolfSSL or yaSSL.
  - In MariaDB 10.0.1 and later, if this system variable shows `YES`, then the server is linked with OpenSSL.
  - In MariaDB 10.0.0 and before, this system variable was an alias for the `have_ssl` system variable.
  - See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.
- **Scope:** Global
- **Dynamic:** No

---

have_ssl

- **Description:** This variable shows whether the server supports using TLS to secure connections.
  - If the value is `YES`, then the server supports TLS, and TLS is enabled.
  - If the value is `DISABLED`, then the server supports TLS, but TLS is **not** enabled.
  - If the value is `NO`, then the server was not compiled with TLS support, so TLS cannot be enabled.
  - When TLS is supported, check the `have_openssl` system variable to determine whether the server is using OpenSSL or MariaDB's bundled TLS library. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.
- **Scope:** Global
- **Dynamic:** No

---

ssl_ca

- **Description:** Defines a path to a PEM file that should contain one or more X509 certificates for trusted Certificate Authorities (CAs) to use for TLS. This system variable requires that you use the absolute path, not a relative path. This system variable implies the `ssl` option.
  - See Secure Connections Overview: Certificate Authorities (CAs) for more information.
- **Commandline:** `--ssl-ca=file_name`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `file name`

---

ssl_capath

- **Description:** Defines a path to a directory that contains one or more PEM files that should each contain one X509 certificate for a trusted Certificate Authority (CA) to use for TLS. This system variable requires that you use the absolute path, not a relative path. The directory specified by this variable needs to be run through the `openssl rehash` command. This system variable implies the `ssl` option.
  - See Secure Connections Overview: Certificate Authorities (CAs) for more information.
- **Commandline:** `--ssl-capath=directory_name`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `directory name`

---

ssl_cert

- **Description:** Defines a path to the X509 certificate file to use for TLS. This system variable requires that you use the absolute path, not a relative path. This system variable implies the `ssl` option.
- **Commandline:** `--ssl-cert=name`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `file name`
- **Default Value:** None

ssl_cipher

- **Description:** List of permitted ciphers or cipher suites to use for TLS. Besides cipher names, if MariaDB was compiled with OpenSSL, this variable could be set to "SSLv3" or "TLSv1.2" to allow all SSLv3 or all TLSv1.2 ciphers. Note that the TLSv1.3 ciphers cannot be excluded when using OpenSSL, even by using this system variable. See Using TLSv1.3 for details. This system variable implies the `ssl` option.
- **Commandline:** `--ssl-cipher=name`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** None

ssl_crl

- **Description:** Defines a path to a PEM file that should contain one or more revoked X509 certificates to use for TLS. This system variable requires that you use the absolute path, not a relative path.
  - See Secure Connections Overview: Certificate Revocation Lists (CRLs) for more information.
  - This variable is only valid if the server was built with OpenSSL. If the server was built with wolfSSL or yaSSL, then this variable is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.
- **Commandline:** `--ssl-crl=name`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `file name`
- **Default Value:** None

ssl_crlpath

- **Description:** Defines a path to a directory that contains one or more PEM files that should each contain one revoked X509 certificate to use for TLS. This system variable requires that you use the absolute path, not a relative path. The directory specified by this variable needs to be run through the `openssl rehash` command.
  - See Secure Connections Overview: Certificate Revocation Lists (CRLs) for more information.
  - This variable is only supported if the server was built with OpenSSL. If the server was built with wolfSSL or yaSSL, then this variable is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.
- **Commandline:** `--ssl-crlpath=name`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `directory name`
- **Default Value:** None

ssl_key

- **Description:** Defines a path to a private key file to use for TLS. This system variable requires that you use the absolute path, not a relative path. This system variable implies the ssl option.
- **Commandline:** `--ssl-key=name`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** None

tls_version

- **Description:** This system variable accepts a comma-separated list (with no whitespaces) of TLS protocol versions. A TLS protocol version will only be enabled if it is present in this list. All other TLS protocol versions will not be permitted.
  - See Secure Connections Overview: TLS Protocol Versions for more information.
- **Commandline:** `--tls-version=value`
- **Scope:** Global
- **Dynamic:** No

- **Data Type:** `enumerated`
- **Default Value:** TLSv1.1,TLSv1.2,TLSv1.3
- **Valid Values:** TLSv1.0,TLSv1.1,TLSv1.2,TLSv1.3
- **Introduced:** MariaDB 10.4.6

`version_ssl_library`

- **Description:** The version of the TLS library that is being used. Note that the version returned by this system variable does not always necessarily correspond to the exact version of the OpenSSL package installed on the system. OpenSSL shared libraries tend to contain interfaces for multiple versions at once to allow for backward compatibility. Therefore, if the OpenSSL package installed on the system is newer than the OpenSSL version that the MariaDB server binary was built with, then the MariaDB server binary might use one of the interfaces for an older version.
  - See TLS and Cryptography Libraries Used by MariaDB: Checking the Server's OpenSSL Version for more information.
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** None

2.2.1.1.1.7 SSL/TLS Status Variables

Contents

The status variables listed on this page relate to encrypting data during transfer with the Transport Layer Security (TLS) protocol. Often, the term Secure Socket Layer (SSL) is used interchangeably with TLS, although strictly speaking, the SSL protocol is a predecessor to TLS and is no longer considered secure.

For compatibility reasons, the TLS status variables in MariaDB still use the `Ssl_` prefix, but MariaDB only supports its more secure successors. For more information on SSL/TLS in MariaDB, see Secure Connections Overview.

Variables

`Ssl_accept_renegotiates`

- **Description:** Number of negotiations needed to establish the TLS connection. The global value can be flushed by `FLUSH STATUS`.

- **Scope:** Global
- **Data Type:** `numeric`

`Ssl_accepts`

- **Description:** Number of accepted TLS handshakes. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

`Ssl_callback_cache_hits`

- **Description:** Number of sessions retrieved from the session cache. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

`Ssl_cipher`

- **Description:** The TLS cipher currently in use.
- **Scope:** Global, Session
- **Data Type:** `string`

`Ssl_cipher_list`

- **Description:** List of the available TLS ciphers.
- **Scope:** Global, Session
- **Data Type:** `string`

`Ssl_client_connects`

- **Description:** Number of TLS handshakes started in client mode. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

`Ssl_connect_renegotiates`

- **Description:** Number of negotiations needed to establish the connection to a TLS-enabled master. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

`Ssl_ctx_verify_depth`

- **Description:** Number of tested TLS certificates in the chain. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

`Ssl_ctx_verify_mode`

- **Description:** Mode used for TLS context verification.The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

`Ssl_default_timeout`

- **Description:** Default timeout for TLS, in seconds.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Ssl_finished_accepts

- **Description:** Number of successful TLS sessions in server mode. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

Ssl_finished_connects

- **Description:** Number of successful TLS sessions in client mode. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

Ssl_server_not_after

- **Description:** Last valid date for the TLS certificate.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** `MariaDB 10.0`

Ssl_server_not_before

- **Description:** First valid date for the TLS certificate.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** `MariaDB 10.0`

Ssl_session_cache_hits

- **Description:** Number of TLS sessions found in the session cache. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

Ssl_session_cache_misses

- **Description:** Number of TLS sessions not found in the session cache. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

Ssl_session_cache_mode

- **Description:** Mode used for TLS caching by the server.
- **Scope:** Global
- **Data Type:** `string`

Ssl_session_cache_overflows

- **Description:** Number of sessions removed from the session cache because it was full. The global value can be flushed by `FLUSH STATUS`.

- **Scope:** Global
- **Data Type:** `numeric`

`Ssl_session_cache_size`

- **Description:** Size of the session cache. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

`Ssl_session_cache_timeouts`

- **Description:** Number of sessions which have timed out. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

`Ssl_sessions_reused`

- **Description:** Number of sessions reused. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Ssl_used_session_cache_entries`

- **Description:** Current number of sessions in the session cache. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

`Ssl_verify_depth`

- **Description:** TLS verification depth.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Ssl_verify_mode`

- **Description:** TLS verification mode.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Ssl_version`

- **Description:** TLS version in use.
- **Scope:** Global, Session
- **Data Type:** `string`

2.2.1.1.1.8 Using TLSv1.3

Contents

OpenSSL 1.1.1 introduced support for TLSv1.3. TLSv1.3 is a major rewrite of the TLS protocol. Some even argued it should've been called TLSv2.0. One of the changes is that it introduces a new set of cipher suites that only work with TLSv1.3. Additionally, TLSv1.3 does not support cipher suites from previous TLS protocol versions.

This incompatible change had a non-obvious consequence. If a user had been explicitly specifying cipher suites to disable old and obsolete TLS protocol version, then that user may have also inadvertently prevented TLSv1.3 from working, unless the user remembered to add the TLSv1.3 cipher suites to their cipher list. After upgrading to OpenSSL 1.1.1, this user might

believe they are using TLSv1.3, when their existing cipher suite configuration might be preventing it.

To avoid this problem, OpenSSL developers decided that TLSv1.3 cipher suites should not be affected by the normal cipher-selecting API. This means that `ssl_cipher` system variable has no effect on the TLSv1.3 cipher suites.

See this OpenSSL blog post and GitHub issue for more information.

2.2.1.1.2 Data-at-Rest Encryption

MariaDB supports the use of data-at-rest encryption for tables and tablespaces. For a minor performance overhead of 3-5%, this makes it almost impossible for someone with access to the host system or who steals a hard drive to read the original data.

Data-at-Rest Encryption Overview

Having data encrypted will make it hard for someone to steal your data.

Why Encrypt MariaDB Data?

When to use encryption for MariaDB data.

Key Management and Encryption Plugins

MariaDB uses plugins to handle key management and encryption of data.

Encrypting Binary Logs

Data-at-rest encryption for binary logs and relay logs.

Aria Encryption

Configuration and use of data-at-rest encryption with the Aria storage engine.

InnoDB Encryption

Articles on using data-at-rest encryption with the InnoDB storage engine.

There are 4 related questions.

2.2.1.1.2.1 Data-at-Rest Encryption Overview

Contents

Overview

Having tables encrypted makes it almost impossible for someone to access or steal a hard disk and get access to the original data. MariaDB got Data-at-Rest Encryption with MariaDB 10.1. This functionality is also known as "Transparent Data Encryption (TDE)".

This assumes that encryption keys are stored on another system.

Using encryption has an overhead of roughly *3-5%*.

Which Storage Engines Does MariaDB Encryption Support?

MariaDB encryption is fully supported for the InnoDB storage engines. Encryption is also supported for the Aria storage engine, but only for tables created with `ROW_FORMAT=PAGE` (the default), and for the binary log (replication log).

MariaDB allows the user to configure flexibly what to encrypt. In or InnoDB, one can choose to encrypt:

- everything — all tablespaces (with all tables)
- individual tables
- everything, excluding individual tables

Additionally, one can choose to encrypt InnoDB log files (recommended).

Limitations

These limitations exist in the data-at-rest encryption implementation:

- Only **data** and only **at rest** is encrypted. Metadata (for example `.frm` files) and data sent to the client are not encrypted (but see Secure Connections).
- Only the MariaDB server knows how to decrypt the data, in particular
  - mariadb-binlog can read encrypted binary logs only when --read-from-remote-server is used (MDEV-8813).
  - Percona XtraBackup cannot back up instances that use encrypted InnoDB. However, MariaDB's fork, MariaDB Backup, can back up encrypted instances.
- The disk-based Galera gcache is not encrypted in the community version of MariaDB Server (MDEV-9639). However, this file is encrypted in MariaDB Enterprise Server 10.4.
- The Audit plugin cannot create encrypted output. Send it to syslog and configure the protection there instead.
- File-based general query log and slow query log cannot be encrypted (MDEV-9639).
- The Aria log is not encrypted (MDEV-8587). This affects only non-temporary Aria tables though.
- The MariaDB error log is not encrypted. The error log can contain query text and data in some cases, including crashes, assertion failures, and cases where InnoDB write monitor output to the log to aid in debugging. It can be sent to syslog too, if needed.

Encryption Key Management

MariaDB's data-at-rest encryption requires the use of a key management and encryption plugin. These plugins are responsible both for the management of encryption keys and for the actual encryption and decryption of data.

MariaDB supports the use of multiple encryption keys. Each encryption key uses a 32-bit integer as a key identifier. If the specific plugin supports key rotation, then encryption keys can also be rotated, which creates a new version of the encryption key.

How MariaDB manages encryption keys depends on which encryption key management solution you choose. Currently, MariaDB has four options:

- File Key Management Plugin
- AWS Key Management Plugin
- Eperi Key Management Plugin
- Hashicorp Key Management Plugin

Once you have an key management and encryption plugin set up and configured for your server, you can begin using encryption options to better secure your data.

Encrypting Data

Encryption occurs whenever MariaDB writes pages to disk. Encrypting table data requires that you install a key management and encryption plugin, such as the File Key Management plugin. Once you have a plugin set up and configured, you can enable encryption for your InnoDB and Aria tables.

Encrypting Table Data

MariaDB supports data-at-rest encryption for InnoDB and Aria storage engines. Additionally, it supports encrypting the InnoDB redo log and internal on-disk temporary tables that use the Aria storage engine..

- Encrypting Data for InnoDB
- Encrypting Data for Aria

Encrypting Temporary Files

MariaDB also creates temporary files on disk. For example, a binary log cache will be written to a temporary file if the binary log cache exceeds `binlog_cache_size` or `binlog_stmt_cache_size`, and temporary files are also often used for filesorts during query execution. Since MariaDB 10.1.5, these temporary files can also be encrypted if encrypt_tmp_files=ON is set.

Since MariaDB 10.1.27, MariaDB 10.2.9 and MariaDB 10.3.2, temporary files created internally by InnoDB, such as those used for merge sorts and row logs can also be encrypted if innodb_encrypt_log=ON is set. These files are encrypted regardless of whether the tables involved are encrypted or not, and regardless of whether encrypt_tmp_files is set or not.

Encrypting Binary Logs

MariaDB can also encrypt binary logs (including relay logs).

- Encrypting Binary Logs

Encryption and Page Compression

Data-at-rest encryption and InnoDB page compression can be used together. When they are used together, data is first compressed, and then it is encrypted. In this case you save space and still have your data protected.

Thanks

- Tablespace encryption was donated to the MariaDB project by Google.
- Per-table encryption and key identifier support was donated to the MariaDB project by eperi.

We are grateful to these companies for their support of MariaDB!

2.2.1.1.2.2 Why Encrypt MariaDB Data?

Nearly everyone owns data of immense value: customer data, construction plans, recipes, product designs and other information. These data are stored in clear text on your storage media. Everyone with file system access is able to read and modify the data. If this data falls into the wrong hands (criminals or competitors) this may result in serious consequences.

With encryption you protect Data At Rest (see the Wikipedia article). That way, the database files are protected against unauthorized access.

When Does Encryption Help to Protect Your Data?

Encryption helps in case of threats against the database files:

- An attacker gains access to the system and copies the database files to avoid the MariaDB authorization check.
- MariaDB is operated by a service provider who should not gain access to the sensitive data.

When is Encryption No Help?

Encryption provides no additional protection against threats caused by authorized database users. Specifically, SQL injections aren't prevented.

What to Encrypt?

All data that is not supposed to fall into possible attackers hands should be encrypted. Especially information, subject to strict data protection regulations, is to be protected by encryption (e.g. in the healthcare sector: patient records). Additionally data being of interest for criminals should be protected. Data which should be encrypted are:

- Personal related information
- Customer details
- Financial and credit card data
- Public authorities data
- Construction plans and research and development results

How to Handle Key Management?

There are currently three options for key management:

- File Key Management Plugin
- AWS Key Management Plugin
- eperi Gateway for Databases

See Encryption Key Management for details.

5.4.8 Key Management and Encryption Plugins

2.2.1.1.2.4 Encrypting Binary Logs

Contents

MariaDB Server can encrypt the server's binary logs and relay logs. This ensures that your binary logs are only accessible through MariaDB.

Basic Configuration

Since MariaDB 10.1.7, MariaDB can also encrypt binary logs (including relay logs). Encryption of binary logs is configured by the encrypt_binlog system variable.

Users of data-at-rest encryption will also need to have a key management and encryption plugin configured. Some examples are File Key Management Plugin and AWS Key Management Plugin.

```
[mariadb]
...

# File Key Management
plugin_load_add = file_key_management
file_key_management_filename = /etc/mysql/encryption/keyfile.enc
file_key_management_filekey = FILE:/etc/mysql/encryption/keyfile.key
file_key_management_encryption_algorithm = AES_CTR

# Binary Log Encryption
encrypt_binlog=ON
```

Encryption Keys

Key management and encryption plugins support using multiple encryption keys. Each encryption key can be defined with a different 32-bit integer as a key identifier.

MariaDB uses the encryption key with ID 1 to encrypt binary logs.

Key Rotation

Some key management and encryption plugins allow you to automatically rotate and version your encryption keys. If a plugin support key rotation, and if it rotates the encryption keys, then InnoDB's background encryption threads can re-encrypt InnoDB pages that use the old key version with the new key version. However, the binary log does **not** have a similar mechanism, which means that existing binary logs remain encrypted with the older key version, but new binary logs will be encrypted with the new key version. For more information, see MDEV-20098.

In order for key rotation to work, both the backend key management service (KMS) and the corresponding key management and encryption plugin have to support key rotation. See Encryption Key Management: Support for Key Rotation in Encryption Plugins to determine which plugins currently support key rotation.

Enabling Encryption

Encryption of binary logs can be enabled by doing the following process.

- First, stop the server.
- Then, set `encrypt_binlog=ON` in the MariaDB configuration file.

- Then, start the server.

From that point forward, any new binary logs will be encrypted. To delete old unencrypted binary logs, you can use RESET MASTER or PURGE BINARY LOGS.

Disabling Encryption

Encryption of binary logs can be disabled by doing the following process.

- First, stop the server.
- Then, set `encrypt_binlog=OFF` in the MariaDB configuration file.
- Then, start the server.

From that point forward, any new binary logs will be unencrypted. If you would like the server to continue to have access to old encrypted binary logs, then make sure to keep your key management and encryption plugin loaded.

Understanding Binlog Encryption

When starting with binary log encryption, MariaDB Server logs a `Format_descriptor_log_event` and a `START_ENCRYPTION_EVENT`, then encrypts all subsequent events for the binary log.

Each event's header and footer are created and processed to produce encrypted blocks. These encrypted blocks are produced before transactions are committed and before the events are flushed to the binary log. As such, they exist in an encrypted state in memory buffers and in the `IO_CACHE` files for user connections.

Effects of Data-at-Rest Encryption on Replication

When using encrypted binary logs with replication, it is completely supported to have different encryption keys on the master and slave. The master decrypts encrypted binary log events as it reads them from disk, and before its binary log dump thread sends them to the slave, so the slave actually receives the unencrypted binary log events.

If you want to ensure that binary log events are encrypted as they are transmitted between the master and slave, then you will have to use TLS with the replication connection.

Effects of Data-at-Rest Encryption on mariadb-binlog

mariadb-binlog does not currently have the ability to decrypt encrypted binary logs on its own (see MDEV-8813 about that). In order to use mariadb-binlog with encrypted binary logs, you have to use the --read-from-remote-server command-line option, so that the server can decrypt the binary logs for mariadb-binlog.

Note, using the `--read-from-remote-server` option on versions of the `mariadb-binlog` utility that do not have the MDEV-20574 fix (<=MariaDB 10.4.9, MariaDB 10.3.19, MariaDB 10.2.28) can corrupt binlog positions when the binary log is encrypted.

2.2.1.1.2.5 Aria Encryption

Configuration and use of data-at-rest encryption with the Aria storage engine.

Aria Encryption Overview

Data-at-rest encryption for user-created tables and internal on-disk tempor...

Aria Enabling Encryption

In order to enable data-at-rest encryption for tables using the Aria stora...

Aria Disabling Encryption

The process involved in safely disabling data-at-rest encryption for your ...

Aria Encryption Keys

As with other storage engines that support data-at-rest encryption, Aria r...

5.3.4.10 Aria Encryption Overview

2.2.1.1.2.5.2 Aria Enabling Encryption

Contents

In order to enable data-at-rest encryption for tables using the Aria storage engine, you first need to configure the server to use an Encryption Key Management plugin. Once this is done, you can enable encryption by setting the relevant system variables.

Encrypting User-created Tables

With tables that the user creates, you can enable encryption by setting the `aria_encrypt_tables` system variable to `ON`, then restart the Server. Once this is set, Aria automatically enables encryption on all tables you create after with the `ROW_FORMAT` table option set to `PAGE`.

Currently, Aria does not support encryption on tables where the `ROW_FORMAT` table option is set to the `FIXED` or `DYNAMIC` values.

Unlike InnoDB, Aria does not support the `ENCRYPTED` table option (see MDEV-18049 about that). Encryption for Aria can only be enabled globally using the `aria_encrypt_tables` system variable.

Encrypting Existing Tables

In cases where you have existing Aria tables that you would like to encrypt, the process is a little more complicated. Unlike InnoDB, Aria does not utilize background encryption threads to automatically perform encryption changes (see MDEV-18971 about that). Therefore, to encrypt existing tables, you need to identify each table that needs to be encrypted, and then you need to manually rebuild each table.

First, set the `aria_encrypt_tables` system variable to encrypt new tables.

```
SET GLOBAL aria_encrypt_tables=ON;
```

Identify Aria tables that have the `ROW_FORMAT` table option set to `PAGE`.

```
SELECT TABLE_SCHEMA, TABLE_NAME
FROM information_schema.TABLES
WHERE ENGINE='Aria'
  AND ROW_FORMAT='PAGE'
  AND TABLE_SCHEMA != 'information_schema';
```

For each table in the result-set, issue an `ALTER TABLE` statement to rebuild the table.

```
ALTER TABLE test.aria_table ENGINE=Aria ROW_FORMAT=PAGE;
```

This statement causes Aria to rebuild the table using the `ROW_FORMAT` table option. In the process, with the new default setting, it encrypts the table when it writes to disk.

Encrypting Internal On-disk Temporary Tables

During the execution of queries, MariaDB routinely creates internal temporary tables. These internal temporary tables initially use the MEMORY storage engine, which is entirely stored in memory. When the table size exceeds the allocation defined by the `max_heap_table_size` system variable, MariaDB writes the data to disk using another storage engine. If you have the `aria_used_for_temp_tables` set to `ON`, MariaDB uses Aria in writing the internal temporary tables to disk.

Encryption for internal temporary tables is handled separately from encryption for user-created tables. To enable encryption for these tables, set the `encrypt_tmp_disk_tables` system variable to `ON`. Once set, all internal temporary tables that are written to disk using Aria are automatically encrypted.

Manually Encrypting Tables

Currently, Aria does not support manually encrypting tables through the `ENCRYPTED` and `ENCRYPTION_KEY_ID` table

options. For more information, see MDEV-18049.

In cases where you want to encrypt tables manually or set the specific encryption key, use InnoDB.

2.2.1.1.2.5.3 Aria Disabling Encryption

Contents

The process involved in safely disabling data-at-rest encryption for your Aria tables is very similar to that of enabling encryption. To disable, you need to set the relevant system variables and then rebuild each table into an unencrypted state.

Don't remove the Encryption Key Management plugin from your configuration file until you have unencrypted all tables in your database. MariaDB cannot read encrypted tables without the relevant encryption key.

Disabling Encryption on User-created Tables

With tables that the user creates, you can disable encryption by setting the `aria_encrypt_tables` system variable to `OFF`. Once this is set, MariaDB no longer encrypts new tables created with the Aria storage engine.

```
SET GLOBAL aria_encrypt_tables = OFF;
```

Unlike InnoDB, Aria does not currently use background encryption threads. Before removing the Encryption Key Management plugin from the configuration file, you first need to manually rebuild each table to an unencrypted state.

To find the encrypted tables, query the Information Schema, filtering the `TABLES` table for those that use the Aria storage engine and the `PAGE ROW_FORMAT`.

```
SELECT TABLE_SCHEMA, TABLE_NAME
FROM information_schema.TABLES
WHERE ENGINE = 'Aria'
  AND ROW_FORMAT = 'PAGE'
  AND TABLE_SCHEMA != 'information_schema';
```

Each table in the result-set was potentially written to disk in an encrypted state. Before removing the configuration for the encryption keys, you need to rebuild each of these to an unencrypted state. This can be done with an `ALTER TABLE` statement.

```
ALTER TABLE test.aria_table ENGINE = Aria ROW_FORMAT = PAGE;
```

Once all of the Aria tables are rebuilt, they're safely unencrypted.

Disabling Encryption for Internal On-disk Temporary Tables

MariaDB routinely creates internal temporary tables. When these temporary tables are written to disk and the `aria_used_for_temp_tables` system variable is set to `ON`, MariaDB uses the Aria storage engine.

To decrypt these tables, set the `encrypt_tmp_disk_tables` to `OFF`. Once set, all internal temporary tables that are created from that point on are written unencrypted to disk.

2.2.1.1.2.5.4 Aria Encryption Keys

Contents

As with other storage engines that support data-at-rest encryption, Aria relies on an Encryption Key Management plugin to handle its encryption keys. Where the support is available, Aria can use multiple keys.

Encryption Keys

MariaDB keeps track of each encryption key internally using a 32-bit integer, which serves as the key identifier. Unlike InnoDB, Aria does not support the `ENCRYPTION_KEY_ID` table option (for more information, see MDEV-18049), which allows the user to specify the encryption key to use. Instead, Aria defaults to specific encryption keys provided by the Encryption Key Management plugin.

- When working with user-created tables, Aria encrypts them to disk using the ID 1 key.
- When working with internal temporary tables written to disk, Aria encrypts them to disk using the ID 2 key, unless there is no ID 2 key, then it falls back on the ID 1 key.

Key Rotation

Some key management and encryption plugins allow you to automatically rotate and version your encryption keys. If a plugin support key rotation, and if it rotates the encryption keys, then InnoDB's background encryption threads can re-encrypt InnoDB pages that use the old key version with the new key version. However, Aria does **not** have a similar mechanism, which means that the tables remain encrypted with the older key version. For more information, see MDEV-18971.

> In order for key rotation to work, both the backend key management service (KMS) and the corresponding key management and encryption plugin have to support key rotation. See Encryption Key Management: Support for Key Rotation in Encryption Plugins to determine which plugins currently support key rotation.

2.2.1.1.2.6 InnoDB Encryption

Data-at-rest encryption configuration and use with the InnoDB storage engine.

InnoDB Encryption Overview

Data-at-rest encryption for tables that use the InnoDB storage engine.

Enabling InnoDB Encryption

Configuration and procedure for enabling data-at-rest encryption for InnoDB tables.

Disabling InnoDB Encryption

Configuration and procedure to disable data-at-rest encryption for InnoDB tables.

InnoDB Background Encryption Threads

InnoDB performs some encryption and decryption operations with background encryption threads.

InnoDB Encryption Keys

InnoDB uses encryption key management plugins to support the use of multiple encryption keys.

InnoDB Encryption Troubleshooting

Troubleshooting InnoDB encryption

5.3.2.25 InnoDB Encryption Overview

2.2.1.1.2.6.2 Enabling InnoDB Encryption

Contents

In order to enable data-at-rest encryption for tables using the InnoDB storage engines, you first need to configure the Server to use an Encryption Key Management plugin. Once this is done, you can enable encryption by setting the innodb_encrypt_tables system variable to encrypt the InnoDB system and file tablespaces and setting the innodb_encrypt_log system variable to encrypt the InnoDB Redo Log.

Setting these system variables enables the encryption feature for InnoDB tables on your server. To use the feature, you

need to use the ENCRYPTION_KEY_ID table option to set what encryption key you want to use and set the ENCRYPTED table option to enable encryption.

> When encrypting any InnoDB tables, the best practice is also enable encryption for the Redo Log. If you have encrypted InnoDB tables and have not encrypted the Redo Log, data written to an encrypted table may be found unencrypted in the Redo Log.

Enabling Encryption for Automatically Encrypted Tablespaces

The innodb_encrypt_tables system variable controls the configuration of automatic encryption of InnoDB tables. It has the following possible values:

Option	Description
OFF	Disables table encryption.
ON	Enables table encryption, but allows unencrypted tables to be created.
FORCE	Enables table encryption, and doesn't allow unencrypted tables to be created. Added in MariaDB 10.1.4.

When innodb_encrypt_tables is set to ON, InnoDB tables are automatically encrypted by default. For example, the following statements create an encrypted table and confirm that it is encrypted:

```
SET GLOBAL innodb_encryption_threads=4;

SET GLOBAL innodb_encrypt_tables=ON;

SET SESSION innodb_default_encryption_key_id=100;

CREATE TABLE tab1 (
   id int PRIMARY KEY,
   str varchar(50)
);

SELECT NAME, ENCRYPTION_SCHEME, CURRENT_KEY_ID
FROM information_schema.INNODB_TABLESPACES_ENCRYPTION
WHERE NAME='db1/tab1';
+----------+-------------------+----------------+
| NAME     | ENCRYPTION_SCHEME | CURRENT_KEY_ID |
+----------+-------------------+----------------+
| db1/tab1 |                 1 |            100 |
+----------+-------------------+----------------+
```

When innodb_encrypt_tables is set to ON, an unencrypted InnoDB table can be created by setting the ENCRYPTED table option to NO for the table. For example, the following statements create an unencrypted table and confirm that it is not encrypted:

```
SET GLOBAL innodb_encryption_threads=4;

SET GLOBAL innodb_encrypt_tables=ON;

SET SESSION innodb_default_encryption_key_id=100;

CREATE TABLE tab1 (
   id int PRIMARY KEY,
   str varchar(50)
) ENCRYPTED=NO;

SELECT NAME, ENCRYPTION_SCHEME, CURRENT_KEY_ID
FROM information_schema.INNODB_TABLESPACES_ENCRYPTION
WHERE NAME='db1/tab1';
+----------+-------------------+----------------+
| NAME     | ENCRYPTION_SCHEME | CURRENT_KEY_ID |
+----------+-------------------+----------------+
| db1/tab1 |                 0 |            100 |
+----------+-------------------+----------------+
```

When innodb_encrypt_tables is set to FORCE, InnoDB tables are automatically encrypted by default, and unencrypted InnoDB tables can **not** be created. In this scenario, if you set the ENCRYPTED table option to NO for a table, then you will

encounter an error. For example:

```
SET GLOBAL innodb_encryption_threads=4;

SET GLOBAL innodb_encrypt_tables='FORCE';

SET SESSION innodb_default_encryption_key_id=100;

CREATE TABLE tab1 (
   id int PRIMARY KEY,
   str varchar(50)
) ENCRYPTED=NO;
ERROR 1005 (HY000): Can't create table `db1`.`tab1` (errno: 140 "Wrong create options")

SHOW WARNINGS;
+---------+------+-----------------------------------------------------------------------+
| Level   | Code | Message                                                               |
+---------+------+-----------------------------------------------------------------------+
| Warning |  140 | InnoDB: ENCRYPTED=NO implies ENCRYPTION_KEY_ID=1                      |
| Warning |  140 | InnoDB: ENCRYPTED=NO cannot be used with innodb_encrypt_tables=FORCE |
| Error   | 1005 | Can't create table `db1`.`tab1` (errno: 140 "Wrong create options")   |
| Warning | 1030 | Got error 140 "Wrong create options" from storage engine InnoDB       |
+---------+------+-----------------------------------------------------------------------+
4 rows in set (0.00 sec)
```

When innodb_encrypt_tables is set to `ON` or `FORCE`, then you must ensure that innodb_encryption_threads is set to a non-zero value, so that InnoDB can perform any necessary encryption operations in the background. See background operations for more information about that. innodb_encryption_rotate_key_age must also be set to a non-zero value for the initial encryption operations to happen in the background. See disabling key rotations for more information about that.

Enabling Encryption for Manually Encrypted Tablespaces

If you do not want to automatically encrypt every InnoDB table, then it is possible to manually enable encryption for just the subset of InnoDB tables that you would like to encrypt. MariaDB provides the ENCRYPTED and ENCRYPTION_KEY_ID table options that can be used to manually enable encryption for specific InnoDB tables. These table options can be used with CREATE TABLE and ALTER TABLE statements. These table options can only be used with InnoDB tables that have their own InnoDB's file-per-table tablespaces, meaning that tables that were created with innodb_file_per_table=ON set.

Table Option	Value	Description
`ENCRYPTED`	Boolean	Defines whether to encrypt the table
`ENCRYPTION_KEY_ID`	32-bit integer	Defines the identifier for the encryption key to use

You can manually enable or disable encryption for a table by using the ENCRYPTED table option. If you only need to protect a subset of InnoDB tables with encryption, then it can be a good idea to manually encrypt each table that needs the extra protection, rather than encrypting all InnoDB tables globally with innodb_encrypt_tables. This allows you to balance security with speed, as it means the encryption and decryption performance overhead only applies to those tables that require the additional security.

If a manually encrypted InnoDB table contains a FULLTEXT INDEX, then the internal table for the full-text index will not also be manually encrypted. To encrypt internal tables for InnoDB full-text indexes, you must enable automatic InnoDB encryption by setting innodb_encrypt_tables to `ON` or `FORCE`.

You can also manually specify a encryption key for a table by using the ENCRYPTION_KEY_ID table option. This allows you to use different encryption keys for different tables. For example, you might create a table using a statement like this:

```
CREATE TABLE tab1 (
   id int PRIMARY KEY,
   str varchar(50)
) ENCRYPTED=YES ENCRYPTION_KEY_ID=100;

SELECT NAME, ENCRYPTION_SCHEME, CURRENT_KEY_ID
FROM information_schema.INNODB_TABLESPACES_ENCRYPTION
WHERE NAME='db1/tab1';
+----------+-------------------+----------------+
| NAME     | ENCRYPTION_SCHEME | CURRENT_KEY_ID |
+----------+-------------------+----------------+
| db1/tab1 |                 1 |            100 |
+----------+-------------------+----------------+
```

If the ENCRYPTION_KEY_ID table option is not specified, then the table will be encrypted with the key identified by the innodb_default_encryption_key_id system variable. For example, you might create a table using a statement like this:

```
SET SESSION innodb_default_encryption_key_id=100;

CREATE TABLE tab1 (
   id int PRIMARY KEY,
   str varchar(50)
) ENCRYPTED=YES;

SELECT NAME, ENCRYPTION_SCHEME, CURRENT_KEY_ID
FROM information_schema.INNODB_TABLESPACES_ENCRYPTION
WHERE NAME='db1/tab1';
+----------+-------------------+----------------+
| NAME     | ENCRYPTION_SCHEME | CURRENT_KEY_ID |
+----------+-------------------+----------------+
| db1/tab1 |                 1 |            100 |
+----------+-------------------+----------------+
```

In the event that you have an existing table and you want to manually enable encryption for that table, then you can do the same with an ALTER TABLE statement. For example:

```
CREATE TABLE tab1 (
   id int PRIMARY KEY,
   str varchar(50)
) ENCRYPTED=NO;

SELECT NAME, ENCRYPTION_SCHEME, CURRENT_KEY_ID
FROM information_schema.INNODB_TABLESPACES_ENCRYPTION
WHERE NAME='db1/tab1';
+----------+-------------------+----------------+
| NAME     | ENCRYPTION_SCHEME | CURRENT_KEY_ID |
+----------+-------------------+----------------+
| db1/tab1 |                 0 |            100 |
+----------+-------------------+----------------+

ALTER TABLE tab1
   ENCRYPTED=YES ENCRYPTION_KEY_ID=100;

SELECT NAME, ENCRYPTION_SCHEME, CURRENT_KEY_ID
FROM information_schema.INNODB_TABLESPACES_ENCRYPTION
WHERE NAME='db1/tab1';
+----------+-------------------+----------------+
| NAME     | ENCRYPTION_SCHEME | CURRENT_KEY_ID |
+----------+-------------------+----------------+
| db1/tab1 |                 1 |            100 |
+----------+-------------------+----------------+
```

InnoDB does not permit manual encryption changes to tables in the system tablespace using ALTER TABLE. Encryption of the system tablespace can only be configured by setting the value of the innodb_encrypt_tables system variable. This means that when you want to encrypt or decrypt the system tablespace, you must also set a non-zero value for the innodb_encryption_threads system variable, and you must also set the innodb_system_rotate_key_age system variable to `1` to ensure that the system tablespace is properly encrypted or decrypted by the background threads. See MDEV-14398 for more information.

Enabling Encryption for Temporary Tablespaces

The innodb_encrypt_temporary_tables system variable controls the configuration of encryption for the temporary tablespace. It has the following possible values:

Option	Description
OFF	Disables temporary table encryption.
ON	Enables temporary table encryption.

This system variable can be specified as a command-line argument to mysqld or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
innodb_encrypt_temporary_tables=ON
```

Enabling Encryption for the Redo Log

InnoDB uses the Redo Log in crash recovery. By default, these events are written to file in an unencrypted state. In configuring MariaDB for data-at-rest encryption, ensure that you also enable encryption for the Redo Log.

To encrypt the Redo Log, first stop the server process. Then, set the innodb_encrypt_log to ON in a relevant server option group in an option file. For example:

```
[mariadb]
...
innodb_encrypt_log = ON
```

Then, start MariaDB. When the server starts back up, it checks to recover InnoDB in the event of a crash. Once it is back online, it begins writing encrypted data to the Redo Log.

> In MariaDB 10.3 and before, InnoDB does not support key rotation for the Redo Log. Key rotation for the Redo Log is supported in MariaDB 10.4 and later. See InnoDB Encryption Keys: Key Rotation for more information.

2.2.1.1.2.6.3 Disabling InnoDB Encryption

Contents

The process involved in safely disabling encryption for your InnoDB tables is a little more complicated than that of enabling encryption. Turning off the relevant system variables doesn't decrypt the tables. If you turn it off and remove the encryption key management plugin, it'll render the encrypted data inaccessible.

In order to safely disable encryption, you first need to decrypt the tablespaces and the Redo Log, then turn off the system variables. The specifics of this process depends on whether you are using automatic or manual encryption of the InnoDB tablespaces.

Disabling Encryption for Automatically Encrypted Tablespaces

When an InnoDB tablespace has the ENCRYPTED table option set to DEFAULT and the innodb_encrypt_tables system variable is set to ON or FORCE, the tablespace's encryption is automatically managed by the background encryption threads. When you want to disable encryption for these tablespaces, you must ensure that the background encryption threads decrypt the tablespaces before removing the encryption keys. Otherwise, the tablespace remains encrypted and becomes inaccessible once you've removed the keys.

To safely decrypt the tablespaces, first, set the innodb_encrypt_tables system variable to OFF:

```
SET GLOBAL innodb_encrypt_tables = OFF;
```

Next, set the innodb_encryption_threads system variable to a non-zero value:

```
SET GLOBAL innodb_encryption_threads = 4;
```

Then, set the innodb_encryption_rotate_key_age system variable to `1`:

```
SET GLOBAL innodb_encryption_rotate_key_age = 1;
```

Once set, any InnoDB tablespaces that have the ENCRYPTED table option set to `DEFAULT` will be decrypted in the background by the InnoDB background encryption threads.

Decryption Status

You can check the status of the decryption process using the INNODB_TABLESPACES_ENCRYPTION table in the information_schema database.

```
SELECT COUNT(*) AS "Number of Encrypted Tablespaces"
FROM information_schema.INNODB_TABLESPACES_ENCRYPTION
WHERE ENCRYPTION_SCHEME != 0
   OR ROTATING_OR_FLUSHING != 0;
```

This query shows the number of InnoDB tablespaces that currently using background encryption threads. Once the count reaches 0, then all of your InnoDB tablespaces are unencrypted. Be sure to also remove encryption on the Redo Log and the Aria storage engine before removing the encryption key management settings from your configuration file.

Disabling Encryption for Manually Encrypted Tablespaces

In the case of manually encrypted InnoDB tablespaces, (that is, those where the ENCRYPTED table option is set to `YES`), you must issue an ALTER TABLE statement to decrypt each tablespace before removing the encryption keys. Otherwise, the tablespace remains encrypted and becomes inaccessible without the keys.

First, query the Information Schema TABLES table to find the encrypted tables. This can be done with a `WHERE` clause filtering the `CREATE_OPTIONS` column.

```
SELECT TABLE_SCHEMA AS "Database", TABLE_NAME AS "Table"
FROM information_schema.TABLES
WHERE ENGINE='InnoDB'
      AND CREATE_OPTIONS LIKE '%`ENCRYPTED`=YES%';
```

For each table in the result-set, issue an ALTER TABLE statement, setting the ENCRYPTED table option to `NO`.

```
SELECT NAME, ENCRYPTION_SCHEME, CURRENT_KEY_ID
FROM information_schema.INNODB_TABLESPACES_ENCRYPTION
WHERE NAME='db1/tab1';
+----------+-------------------+----------------+
| NAME     | ENCRYPTION_SCHEME | CURRENT_KEY_ID |
+----------+-------------------+----------------+
| db1/tab1 |                 1 |            100 |
+----------+-------------------+----------------+

ALTER TABLE tab1
   ENCRYPTED=NO;

SELECT NAME, ENCRYPTION_SCHEME, CURRENT_KEY_ID
FROM information_schema.INNODB_TABLESPACES_ENCRYPTION
WHERE NAME='db1/tab1';
+----------+-------------------+----------------+
| NAME     | ENCRYPTION_SCHEME | CURRENT_KEY_ID |
+----------+-------------------+----------------+
| db1/tab1 |                 0 |            100 |
+----------+-------------------+----------------+
```

Once you have removed encryption from all the tables, your InnoDB deployment is unencrypted. Be sure to also remove encryption from the Redo Log as well as Aria and any other storage engines that support encryption before removing the encryption key management settings from your configuration file.

> InnoDB does not permit manual encryption changes to tables in the system tablespace using ALTER TABLE.

Encryption of the system tablespace can only be configured by setting the value of the innodb_encrypt_tables system variable. This means that when you want to encrypt or decrypt the system tablespace, you must also set a non-zero value for the innodb_encryption_threads system variable, and you must also set the innodb_system_rotate_key_age system variable to `1` to ensure that the system tablespace is properly encrypted or decrypted by the background threads. See MDEV-14398 for more information.

Disabling Encryption for Temporary Tablespaces

The innodb_encrypt_temporary_tables system variable controls the configuration of encryption for the temporary tablespace. To disable it, remove the system variable from your server's option file, and then restart the server.

Disabling Encryption for the Redo Log

InnoDB uses the Redo Log in crash recovery. By default, these events are written to file in an unencrypted state. In removing data-at-rest encryption for InnoDB, be sure to also disable encryption for the Redo Log before removing encryption key settings. Otherwise the Redo Log can become inaccessible without the encryption keys.

First, check the value of the innodb_fast_shutdown system variable with the SHOW VARIABLES statement. For example:

```
SHOW VARIABLES LIKE 'innodb_fast_shutdown';
+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| innodb_fast_shutdown |     2 |
+----------------------+-------+
```

When the value is set to `2`, InnoDB performs an unclean shutdown, so it will need the Redo Log at the next server startup. Ensure that the variable is set to `0`, `1`, or `3`. For performance reasons, `1` is usually the best option. It can be changed dynamically with SET GLOBAL. For example:

```
SET GLOBAL innodb_fast_shutdown = 1;
```

Then, set the innodb_encrypt_log system variable to `OFF` in a server option group in an option file. Once this is done, restart the MariaDB Server. When the Server comes back online, it begins writing unencrypted data to the Redo Log.

2.2.1.1.2.6.4 InnoDB Background Encryption Threads

InnoDB performs some encryption and decryption operations with background encryption threads. The innodb_encryption_threads system variable controls the number of threads that the storage engine uses for encryption-related background operations, including encrypting and decrypting pages after key rotations or configuration changes, and scrubbing data to permanently delete it.

Contents

Background Operations

InnoDB performs the following encryption and decryption operations using background encryption threads:

- When rotating encryption keys, InnoDB's background encryption threads re-encrypt pages that use key versions older than innodb_encryption_rotate_key_age to the new key version.
- When changing the innodb_encrypt_tables system variable to `FORCE`, InnoDB's background encryption threads encrypt the system tablespace and any file-per-table tablespaces that have the `ENCRYPTED` table option set to `DEFAULT`.
- When changing the innodb_encrypt_tables system variable to `OFF`, InnoDB's background encryption threads decrypt the system tablespace and any file-per-table tablespacs that have the `ENCRYPTED` table option set to `DEFAULT`.

The innodb_encryption_rotation_iops system variable can be used to configure how many I/O operations you want to allow for the operations performed by InnoDB's background encryption threads.

Whenever you change the value on the innodb_encrypt_tables system variable, InnoDB's background encryption

threads perform the necessary encryption or decryption operations. Because of this, you must have a non-zero value set for the innodb_encryption_threads system variable. InnoDB also considers these operations to be key rotations internally. Because of this, you must have a non-zero value set for the innodb_encryption_rotate_key_age system variable. For more information, see disabling key rotations.

Non-background Operations

InnoDB performs the following encryption and decryption operations **without** using background encryption threads:

- When a file-per-table tablespaces and using ALTER TABLE to manually set the ENCRYPTED table option to `YES`, InnoDB does **not** use background threads to encrypt the tablespaces.
- Similarly, when using file-per-table tablespaces and using ALTER TABLE to manually set the ENCRYPTED table option to `NO`, InnoDB does **not** use background threads to decrypt the tablespaces.

In these cases, InnoDB performs the encryption or decryption operation using the server thread for the client connection that executes the statement. This means that you can update encryption on file-per-table tablespaces with an ALTER TABLE statement, even when the innodb_encryption_threads and/or the innodb_rotate_key_age system variables are set to `0`.

InnoDB does not permit manual encryption changes to tables in the system tablespace using ALTER TABLE. Encryption of the system tablespace can only be configured by setting the value of the innodb_encrypt_tables system variable. This means that when you want to encrypt or decrypt the system tablespace, you must also set a non-zero value for the innodb_encryption_threads system variable, and you must also set the innodb_system_rotate_key_age system variable to `1` to ensure that the system tablespace is properly encrypted or decrypted by the background threads. See MDEV-14398 for more information.

Checking the Status of Background Operations

InnoDB records the status of background encryption operations in the INNODB_TABLESPACES_ENCRYPTION table in the information_schema database.

For example, to see which InnoDB tablespaces are currently being decrypted or encrypted on by background encryption, you can check which InnoDB tablespaces have the `ROTATING_OR_FLUSHING` column set to `1`:

```
SELECT SPACE, NAME
FROM information_schema.INNODB_TABLESPACES_ENCRYPTION
WHERE ROTATING_OR_FLUSHING = 1;
```

And to see how many InnoDB tablespaces are currently being decrypted or encrypted by background encryption threads, you can call the COUNT() aggregate function.

```
SELECT COUNT(*) AS 'encrypting'
FROM information_schema.INNODB_TABLESPACES_ENCRYPTION
WHERE ROTATING_OR_FLUSHING = 1;
```

And to see how many InnoDB tablespaces are currently being decrypted or encrypted by background encryption threads, while comparing that to the total number of InnoDB tablespaces and the total number of encrypted InnoDB tablespaces, you can join the table with the INNODB_SYS_TABLESPACES table in the information_schema database:

```
/* information_schema.INNODB_TABLESPACES_ENCRYPTION does not always have rows for all
tablespaces,
  so let's join it with information_schema.INNODB_SYS_TABLESPACES */
WITH tablespace_ids AS (
   SELECT SPACE
   FROM information_schema.INNODB_SYS_TABLESPACES ist
   UNION
   /* information_schema.INNODB_SYS_TABLESPACES doesn't have a row for the system tablespace
(MDEV-20802) */
   SELECT 0 AS SPACE
)
SELECT NOW() as 'time',
   'tablespaces', COUNT(*) AS 'tablespaces',
   'encrypted', SUM(IF(ite.ENCRYPTION_SCHEME IS NOT NULL, ite.ENCRYPTION_SCHEME, 0)) AS
'encrypted',
   'encrypting', SUM(IF(ite.ROTATING_OR_FLUSHING IS NOT NULL, ite.ROTATING_OR_FLUSHING, 0)) AS
'encrypting'
FROM tablespace_ids
LEFT JOIN information_schema.INNODB_TABLESPACES_ENCRYPTION ite
   ON tablespace_ids.SPACE = ite.SPACE
```

2.2.1.1.2.6.5 InnoDB Encryption Keys

Contents

InnoDB uses encryption key management plugins to support the use of multiple encryption keys.

Encryption Keys

Each encryption key has a 32-bit integer that serves as a key identifier.

The default key is set using the innodb_default_encryption_key_id system variable.

Encryption keys can also be specified with the ENCRYPTION_KEY_ID table option for tables that use file-per-table tablespaces.

InnoDB encrypts the temporary tablespace using the encryption key with the ID `1`.

InnoDB encrypts the Redo Log using the encryption key with the ID `1`.

Keys with Manually Encrypted Tablespaces

With tables that use manually enabled encryption, one way to set the specific encryption key for the table is to use the ENCRYPTION_KEY_ID table option. For example:

```
CREATE TABLE tab1 (
   id int PRIMARY KEY,
   str varchar(50)
) ENCRYPTED=YES ENCRYPTION_KEY_ID=100;

SELECT NAME, ENCRYPTION_SCHEME, CURRENT_KEY_ID
FROM information_schema.INNODB_TABLESPACES_ENCRYPTION
WHERE NAME='db1/tab1';
+----------+-------------------+----------------+
| NAME     | ENCRYPTION_SCHEME | CURRENT_KEY_ID |
+----------+-------------------+----------------+
| db1/tab1 |                 1 |            100 |
+----------+-------------------+----------------+
```

If the ENCRYPTION_KEY_ID table option is not set for a table that uses manually enabled encryption, then it will inherit the value from the innodb_default_encryption_key_id system variable. For example:

```
SET SESSION innodb_default_encryption_key_id=100;

CREATE TABLE tab1 (
   id int PRIMARY KEY,
   str varchar(50)
) ENCRYPTED=YES;

SELECT NAME, ENCRYPTION_SCHEME, CURRENT_KEY_ID
FROM information_schema.INNODB_TABLESPACES_ENCRYPTION
WHERE NAME='db1/tab1';
+----------+-------------------+----------------+
| NAME     | ENCRYPTION_SCHEME | CURRENT_KEY_ID |
+----------+-------------------+----------------+
| db1/tab1 |                 1 |            100 |
+----------+-------------------+----------------+
```

Keys with Automatically Encrypted Tablespaces

With tables that use automatically enabled encryption, one way to set the specific encryption key for the table is to use the innodb_default_encryption_key_id system variable. For example:

```
SET GLOBAL innodb_encryption_threads=4;

SET GLOBAL innodb_encrypt_tables=ON;

SET SESSION innodb_default_encryption_key_id=100;

CREATE TABLE tab1 (
   id int PRIMARY KEY,
   str varchar(50)
);

SELECT NAME, ENCRYPTION_SCHEME, CURRENT_KEY_ID
FROM information_schema.INNODB_TABLESPACES_ENCRYPTION
WHERE NAME='db1/tab1';
+----------+-------------------+----------------+
| NAME     | ENCRYPTION_SCHEME | CURRENT_KEY_ID |
+----------+-------------------+----------------+
| db1/tab1 |                 1 |            100 |
+----------+-------------------+----------------+
```

InnoDB tables that are part of the system tablespace can only be encrypted using the encryption key set by the innodb_default_encryption_key_id system variable.

If the table is in a file-per-table tablespace, and if innodb_encrypt_tables is set to `ON` or `FORCE`, and if innodb_encryption_threads is set to a value greater than `0`, then you can also set the specific encryption key for the table by using the ENCRYPTION_KEY_ID table option. For example:

```
SET GLOBAL innodb_encryption_threads=4;

SET GLOBAL innodb_encrypt_tables=ON;

CREATE TABLE tab1 (
   id int PRIMARY KEY,
   str varchar(50)
) ENCRYPTION_KEY_ID=100;

SELECT NAME, ENCRYPTION_SCHEME, CURRENT_KEY_ID
    -> FROM information_schema.INNODB_TABLESPACES_ENCRYPTION
    -> WHERE NAME='db1/tab1';
+----------+-------------------+----------------+
| NAME     | ENCRYPTION_SCHEME | CURRENT_KEY_ID |
+----------+-------------------+----------------+
| db1/tab1 |                 1 |            100 |
+----------+-------------------+----------------+
```

However, if innodb_encrypt_tables is set to `OFF` or if innodb_encryption_threads is set to `0`, then this will not work. See InnoDB Encryption Troubleshooting: Setting Encryption Key ID For an Unencrypted Table for more information.

Key Rotation

Some key management and encryption plugins allow you to automatically rotate and version your encryption keys. If a plugin support key rotation, and if it rotates the encryption keys, then InnoDB's background encryption threads can re-encrypt InnoDB pages that use the old key version with the new key version.

You can set the maximum age for an encryption key using the innodb_encryption_rotate_key_age system variable. When this variable is set to a non-zero value, background encryption threads constantly check pages to determine if any page is encrypted with a key version that's too old. When the key version is too old, any page encrypted with the older version of the key is automatically re-encrypted in the background to use a more current version of the key. Bear in mind, this constant checking can sometimes result in high CPU usage.

Key rotation for the InnoDB Redo Log is only supported in MariaDB 10.4.0 and later. For more information, see MDEV-12041.

> In order for key rotation to work, both the backend key management service (KMS) and the corresponding key management and encryption plugin have to support key rotation. See Encryption Key Management: Support for Key Rotation in Encryption Plugins to determine which plugins currently support key rotation.

Disabling Background Key Rotation Operations

In the event that you encounter issues with background key encryption, you can disable it by setting the innodb_encryption_rotate_key_age system variable to `0`. You may find this useful when the constant key version checks lead to excessive CPU usage. It's also useful in cases where your encryption key management plugin does not support key rotation, (such as with the file_key_management plugin). For more information, see MDEV-14180.

There are, however, issues that can arise when the background key rotation is disabled.

Pending Encryption Operations

Prior to MariaDB 10.2.24, MariaDB 10.3.15, and MariaDB 10.4.5, when you update the value on the innodb_encrypt_tables system variable InnoDB internally treats the subsequent background operations to encrypt and decrypt tablespaces as background key rotations. See MDEV-14398 for more information.

In older versions of MariaDB, if you have recently changed the value of the innodb_encrypt_tables system variable, then you must ensure that any pending background encryption or decryption operations are complete before disabling key rotation. You can check the status of background encryption operations by querying the INNODB_TABLESPACES_ENCRYPTION table in the information_schema database.

See InnoDB Background Encryption Threads: Checking the Status of Background Operations for some example queries.

Otherwise, in older versions of MariaDB, if you disable key rotation while there are background encryption threads at work, it may result in unencrypted tables that you want encrypted or vice versa.

For more information, see MDEV-14398.

2.2.1.1.2.6.6 InnoDB Encryption Troubleshooting

Contents

Wrong Create Options

With InnoDB tables using encryption, there are several cases where a CREATE TABLE or ALTER TABLE statement can throw Error 1005, due to the InnoDB error 140, `Wrong create options`. For instance,

```
CREATE TABLE `test`.`table1` ( `id` int(4) primary key , `name` varchar(50));
ERROR 1005 (HY000): Can't create table `test`.`table1` (errno: 140 "Wrong create options")
```

When this occurs, you can usually get more information about the cause of the error by following it with a SHOW WARNINGS statement.

This error is known to occur in the following cases:

- Encrypting a table by setting the ENCRYPTED table option to `YES` when the innodb_file_per_table is set to `OFF` .In this case, SHOW WARNINGS would return the following:

```
SHOW WARNINGS;
+---------+------+-----------------------------------------------------------------------+
| Level   | Code | Message                                                               |
+---------+------+-----------------------------------------------------------------------+
| Warning |  140 | InnoDB: ENCRYPTED requires innodb_file_per_table                      |
| Error   | 1005 | Can't create table `db1`.`tab3` (errno: 140 "Wrong create options") |
| Warning | 1030 | Got error 140 "Wrong create options" from storage engine InnoDB     |
+---------+------+-----------------------------------------------------------------------+
3 rows in set (0.00 sec)
```

- Encrypting a table by setting the ENCRYPTED table option to `YES` , and the innodb_default_encryption_key_id system variable or the ENCRYPTION_KEY_ID table option refers to a non-existent key identifier. In this case, SHOW WARNINGS would return the following:

```
SHOW WARNINGS;
+---------+------+-----------------------------------------------------------------------+
| Level   | Code | Message                                                               |
+---------+------+-----------------------------------------------------------------------+
| Warning |  140 | InnoDB: ENCRYPTION_KEY_ID 500 not available                           |
| Error   | 1005 | Can't create table `db1`.`tab3` (errno: 140 "Wrong create options") |
| Warning | 1030 | Got error 140 "Wrong create options" from storage engine InnoDB     |
+---------+------+-----------------------------------------------------------------------+
3 rows in set (0.00 sec)
```

- In some versions, this could happen while creating a table with the ENCRYPTED table option set to `DEFAULT` while the innodb_encrypt_tables system variable is set to `OFF` , and the innodb_default_encryption_key_id system variable or the ENCRYPTION_KEY_ID table option are **not** set to `1` . In this case, SHOW WARNINGS would return the following:

```
SHOW WARNINGS;
+---------+------+-----------------------------------------------------------------------+
| Level   | Code | Message                                                               |
+---------+------+-----------------------------------------------------------------------+
| Warning |  140 | InnoDB: innodb_encrypt_tables=OFF only allows ENCRYPTION_KEY_ID=1   |
| Error   | 1005 | Can't create table `db1`.`tab3` (errno: 140 "Wrong create options") |
| Warning | 1030 | Got error 140 "Wrong create options" from storage engine InnoDB     |
+---------+------+-----------------------------------------------------------------------+
3 rows in set (0.00 sec)
```

Starting in MariaDB 10.1.39, MariaDB 10.2.23, and MariaDB 10.3.14, creating a table with the ENCRYPTED table option set to `DEFAULT` while the innodb_encrypt_tables system variable is set to `OFF` , and the innodb_default_encryption_key_id system variable or the ENCRYPTION_KEY_ID table option are **not** set to `1` will no longer fail, and it will no longer throw a warning.

For more information, see MDEV-18601.

Setting Encryption Key ID For an Unencrypted Table

If you set the ENCRYPTION_KEY_ID table option for a table that is unencrypted because the innodb_encrypt_tables system variable is set to `OFF` and the ENCRYPTED table option set to `DEFAULT` , then this encryption key ID will be saved in the table's `.frm` file, but the encryption key will not be saved to the table's `.ibd` file.

As a side effect, with the current encryption design, if the innodb_encrypt_tables system variable is later set to `ON` , and InnoDB goes to encrypt the table, then the InnoDB background encryption threads will not read this encryption key ID from the `.frm` file. Instead, the threads may encrypt the table with the encryption key with ID `1` , which is internally considered the default encryption key when no key is specified. For example:

```
SET GLOBAL innodb_encrypt_tables=OFF;

CREATE TABLE tab1 (
   id INT PRIMARY KEY,
   str VARCHAR(50)
) ENCRYPTION_KEY_ID=100;

SET GLOBAL innodb_encrypt_tables=ON;

SELECT NAME, ENCRYPTION_SCHEME, CURRENT_KEY_ID
FROM information_schema.INNODB_TABLESPACES_ENCRYPTION
WHERE NAME='db1/tab1';
+----------+-------------------+----------------+
| NAME     | ENCRYPTION_SCHEME | CURRENT_KEY_ID |
+----------+-------------------+----------------+
| db1/tab1 |                 1 |              1 |
+----------+-------------------+----------------+
```

A similar problem is that, if you set the ENCRYPTION_KEY_ID table option for a table that is unencrypted because the ENCRYPTED table option is set to `NO`, then this encryption key ID will be saved in the table's `.frm` file, but the encryption key will not be saved to the table's `.ibd` file.

Recent versions of MariaDB will throw warnings in the case where the ENCRYPTED table option is set to `NO`, but they will allow the operation to succeed. For example:

```
CREATE TABLE tab1 (
   id INT PRIMARY KEY,
   str VARCHAR(50)
) ENCRYPTED=NO ENCRYPTION_KEY_ID=100;
Query OK, 0 rows affected, 1 warning (0.01 sec)

SHOW WARNINGS;
+---------+------+-----------------------------------------------------+
| Level   | Code | Message                                             |
+---------+------+-----------------------------------------------------+
| Warning |  140 | InnoDB: ENCRYPTED=NO implies ENCRYPTION_KEY_ID=1 |
+---------+------+-----------------------------------------------------+
1 row in set (0.00 sec)
```

However, in this case, if you change the ENCRYPTED table option to `YES` or `DEFAULT` with ALTER TABLE, then it will actually use the proper key. For example:

```
SET GLOBAL innodb_encrypt_tables=ON;

ALTER TABLE tab1 ENCRYPTED=DEFAULT;

SELECT NAME, ENCRYPTION_SCHEME, CURRENT_KEY_ID
FROM information_schema.INNODB_TABLESPACES_ENCRYPTION
WHERE NAME = 'db1/tab1';
+----------+-------------------+----------------+
| NAME     | ENCRYPTION_SCHEME | CURRENT_KEY_ID |
+----------+-------------------+----------------+
| db1/tab1 |                 1 |            100 |
+----------+-------------------+----------------+
```

For more information, see MDEV-17230, MDEV-18601, and MDEV-19086.

Tablespaces Created on MySQL 5.1.47 or Earlier

MariaDB's data-at-rest encryption implementation re-used previously unused fields in InnoDB's buffer pool pages to identify the encryption key version and the post-encryption checksum. Prior to MySQL 5.1.48, these unused fields were not initialized in memory due to performance concerns. These fields still had zero values most of the time, but since they were not explicitly initialized, that means that these fields could have occasionally had non-zero values that could have been written into InnoDB's tablespace files. If MariaDB were to encounter an unencrypted page from a tablespace file that was created on an early version of MySQL that also had non-zero values in these fields, then it would mistakenly think that the page was encrypted.

The fix for MDEV-12112 that was included in MariaDB 10.1.38, MariaDB 10.2.20, and MariaDB 10.3.12 changed the way that MariaDB distinguishes between encrypted and unencrypted pages, so that it is less likely to mistake an

unencrypted page for an encrypted page.

In MariaDB 10.4.3 and later, if innodb_checksum_algorithm is set to `full_crc32` or `strict_full_crc32`, and if the table does not use ROW_FORMAT=COMPRESSED, then data files will be guaranteed to be zero-initialized.

For more information, see MDEV-18097.

Spatial Indexes

MariaDB 10.4.3 introduces support for encrypting spatial indexes. To enable, set the innodb_checksum_algorithm to `full_crc32` or to `strict_full_crc32`. Note that MariaDB only encrypts spatial indexes when the ROW_FORMAT table option is **not** set to COMPRESSED.

In older versions of MariaDB, spatial index encryption is unsupported. Tables that contain spatial indexes store them unencrypted.

For more information, see MDEV-12026.

2.2.1.1.3 TLS and Cryptography Libraries Used by MariaDB

Contents

When MariaDB Server is compiled with TLS and cryptography support, it is usually either statically linked with MariaDB's bundled TLS and cryptography library or dynamically linked with the system's OpenSSL library. MariaDB's bundled TLS library is either wolfSSL or yaSSL, depending on the server version.

When a MariaDB client or client library is compiled with TLS and cryptography support, it is usually either statically linked with MariaDB's bundled TLS and cryptography library or dynamically linked with the system's TLS and cryptography library, which might be OpenSSL, GnuTLS, or Schannel.

Checking Dynamically vs. Statically Linked

Dynamically linking MariaDB to the system's TLS and cryptography library can often be beneficial, since this allows you to fix bugs in the system's TLS and cryptography library independently of MariaDB. For example, when information on the Heartbleed Bug in OpenSSL was released in 2014, the bug could be mitigated by simply updating your system to use a fixed version of the OpenSSL library, and then restarting the MariaDB Server.

You can verify that `mysqld` is in fact dynamically linked to the OpenSSL shared library on your system by using the `ldd` command:

```
$ ldd $(which mysqld) | grep -E '(libssl|libcrypto)'
        libssl.so.10 => /lib64/libssl.so.10 (0x00007f8736386000)
        libcrypto.so.10 => /lib64/libcrypto.so.10 (0x00007f8735f25000)
```

If the command does not return any results, then either your `mysqld` is statically linked to the TLS and cryptography library on your system or your `mysqld` is not built with TLS and cryptography support at all.

Checking If the Server Uses OpenSSL

In MariaDB 10.0 and later, if you aren't sure whether your server is linked with OpenSSL or the bundled TLS library, then you can check the value of the `have_openssl` system variable. For example:

```
SHOW GLOBAL VARIABLES LIKE 'have_openssl';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| have_openssl  | YES   |
+---------------+-------+
```

Checking the Server's OpenSSL Version

In MariaDB 10.1 and later, if you want to see what version of OpenSSL your server is using, then you can check the value of the `version_ssl_library` system variable. For example:

```
SHOW GLOBAL VARIABLES LIKE 'version_ssl_library';
+--------------------+--------------------------------+
| Variable_name      | Value                          |
+--------------------+--------------------------------+
| version_ssl_library | OpenSSL 1.0.1e-fips 11 Feb 2013 |
+--------------------+--------------------------------+
```

Note that the version returned by this system variable does not always necessarily correspond to the exact version of the OpenSSL package installed on the system. OpenSSL shared libraries tend to contain interfaces for multiple versions at once to allow for backward compatibility. Therefore, if the OpenSSL package installed on the system is newer than the OpenSSL version that the MariaDB Server binary was built with, then the MariaDB Server binary might use one of the interfaces for an older version. See MDEV-15848 for more information. For example:

```
$ cat /etc/redhat-release
Red Hat Enterprise Linux Server release 7.5 (Maipo)
$ rpm -q openssl
openssl-1.0.2k-12.el7.x86_64
$ mysql -u root --batch --execute="SHOW GLOBAL VARIABLES LIKE 'version_ssl_library';"
Variable_name   Value
version_ssl_library     OpenSSL 1.0.1e-fips 11 Feb 2013
$ ldd $(which mysqld) | grep libcrypto
        libcrypto.so.10 => /lib64/libcrypto.so.10 (0x00007f3dd3482000)
$ readelf -a /lib64/libcrypto.so.10 | grep SSLeay_version
  1374: 000000000006f5d0    21 FUNC    GLOBAL DEFAULT   13 SSLeay_version@libcrypto.so.10
  1375: 000000000006f5f0    21 FUNC    GLOBAL DEFAULT   13 SSLeay_version@OPENSSL_1.0.1
  1377: 000000000006f580    70 FUNC    GLOBAL DEFAULT   13 SSLeay_version@@OPENSSL_1.0.2
```

FIPS Certification

Federal Information Processing Standards (FIPS) are standards published by the U.S. federal government that are used to establish requirements for various aspects of computer systems. FIPS 140-2 is a set of standards for security requirements for cryptographic modules.

This standard is relevant when discussing the TLS and cryptography libraries used by MariaDB. Some of these libraries have been certified to meet the standards set by FIPS 140-2.

FIPS Certification by OpenSSL

The OpenSSL library has a special FIPS mode that has been certified to meet the FIPS 140-2 standard. In FIPS mode, only algorithms and key sizes that meet the FIPS 140-2 standard are enabled by the library.

MariaDB does not yet support enabling FIPS mode within the database server. See MDEV-20260 for more information. Therefore, if you would like to use OpenSSL's FIPS mode with MariaDB, then you would either need to enable FIPS mode at the kernel level or enable it via the OpenSSL configuration file, system-wide or only for the MariaDB process.. See the following resources for more information on how to do that:

- Red Hat Enterprise Linux 7: Security Guide: Chapter 8. Federal Standards and Regulations
- Ubuntu Security Certifications Documentation: FIPS for Ubuntu 16.04 and 18.04
- OpenSSL 1.0.2, configuration file method
- OpenSSL 3.0 configuration file method

FIPS Certification by wolfSSL

The standard version of the wolfSSL library has not been certified to meet the FIPS 140-2 standard, but a special "FIPS-ready" version has been certified. Unfortunately, the "FIPS-ready" version of wolfSSL uses a license that is incompatible with MariaDB's license, so it cannot be used with MariaDB.

FIPS Certification by yaSSL

The yaSSL library has not been certified to meet the FIPS 140-2 standard.

Libraries Used by Each Platform and Package

MariaDB Server

MariaDB Server on Windows

MariaDB starting with 10.4.6

MariaDB Server is statically linked with the bundled wolfSSL library in MSI and ZIP packages on Windows.

MariaDB until 10.4.5

MariaDB Server is statically linked with the bundled yaSSL library in MSI and ZIP packages on Windows.

MariaDB Server on Linux

MariaDB Server in Binary Tarballs

MariaDB starting with 10.4.6

In MariaDB 10.4.6 and later, MariaDB Server is statically linked with the bundled wolfSSL library in binary tarballs on Linux.

MariaDB until 10.4.5

In MariaDB 10.4.5 and before, MariaDB Server is statically linked with the bundled yaSSL library in binary tarballs on Linux.

MariaDB Server in DEB Packages

MariaDB Server is dynamically linked with the system's OpenSSL library in `.deb` packages.

MariaDB until 10.3

In MariaDB 10.3 and before, MariaDB Server is statically linked with the bundled yaSSL library in `.deb` packages provided by Debian's and Ubuntu's default repositories.

See Differences in MariaDB in Debian (and Ubuntu) for more information.

MariaDB Server in RPM Packages

MariaDB Server is dynamically linked with the system's OpenSSL library in `.rpm` packages.

MariaDB Clients and Utilities

In MariaDB 10.2 and later, MariaDB Connector/C has been included with MariaDB Server, and the bundled and the

clients and utilities are linked with it. On some platforms, MariaDB Connector/C and these clients and utilities may use a different TLS library than the one used by MariaDB Server and libmysqlclient.

MariaDB Clients and Utilities on Windows

MariaDB starting with 10.4.6

In MariaDB 10.4.6 and later, MariaDB's clients and utilities and MariaDB Connector/C are are dynamically linked with the system's Schannel libraries in MSI and ZIP packages on Windows. libmysqlclient is still statically linked with the bundled wolfSSL library.

MariaDB Clients and Utilities on Linux

MariaDB Clients and Utilities in Binary Tarballs

MariaDB starting with 10.4.6

In MariaDB 10.4.6 and later, MariaDB's clients and utilities and MariaDB Connector/C are statically linked with the GnuTLS library in binary tarballs on Linux. libmysqlclient is still statically linked with the bundled wolfSSL library.

MariaDB Clients and Utilities in DEB Packages

MariaDB's clients and utilities, libmysqlclient, and MariaDB Connector/C are dynamically linked with the system's OpenSSL library in `.deb` packages.

See Differences in MariaDB in Debian (and Ubuntu) for more information.

MariaDB Clients and Utilities in RPM Packages

MariaDB's clients and utilities, libmysqlclient, and MariaDB Connector/C are dynamically linked with the system's OpenSSL library in `.rpm` packages.

Updating Dynamically Linked OpenSSL Libraries on Linux

When the MariaDB Server or clients and utilities are dynamically linked to the system's OpenSSL library, it makes it very easy to update the libraries. The information below will show how to update these libraries for each platform.

Updating Dynamically Linked OpenSSL Libraries with yum/dnf

On RHEL, CentOS, Fedora, and other similar Linux distributions, it is highly recommended to update the libraries using `yum` or `dnf`. Starting with RHEL 8 and Fedora 22, `yum` has been replaced by `dnf`, which is the next major version of `yum`. However, `yum` commands still work on many systems that use `dnf`. For example:

Update the package by executing the following command:

```
sudo yum update openssl
```

And then restart MariaDB server and any clients or applications that use the library.

Updating Dynamically Linked OpenSSL Libraries with apt-get

On Debian, Ubuntu, and other similar Linux distributions, it is highly recommended to recommended to update the libraries using `apt-get`. For example:

First update the package cache by executing the following command:

```
sudo apt update
```

And then update the package by executing the following command:

```
sudo apt-get update openssl
```

And then restart MariaDB server and any clients or applications that use the library.

Updating Dynamically Linked OpenSSL Libraries with zypper

On SLES, OpenSUSE, and other similar Linux distributions, it is highly recommended to recommended to update the libraries using `zypper`. For example:

Update the package by executing the following command:

```
sudo zypper update openssl
```

And then restart MariaDB server and any clients or applications that use the library.

2.2.1.2 Running mysqld as root

MariaDB should never normally be run as the system's root user (this is unrelated to the MariaDB root user). If it is, any user with the FILE privilege can create or modify any files on the server as root.

MariaDB will normally return the error **Fatal error: Please read "Security" section of the manual to find out how to run mysqld as root!** if you attempt to run mysqld as root. If you need to override this restriction for some reason, start mysqld with the `user=root` option.

Better practice, and the default in most situations, is to use a separate user, exclusively used for MariaDB. In most distributions, this user is called `mysql`.

1.3.43.14 mysql_secure_installation

2.2.1.4 Security-Enhanced Linux with MariaDB

Contents

Security-Enhanced Linux (SELinux) is a Linux kernel module that provides a framework for configuring mandatory access control (MAC) system for many resources on the system. It is enabled by default on some Linux distributions, including RHEL, CentOS, Fedora, and other similar Linux distribution. SELinux prevents programs from accessing files, directories or ports unless it is configured to access those resources.

Verifying Whether SELinux Is Enabled

To verify whether SELinux is enabled, execute the `getenforce` command. For example:

```
getenforce
```

Temporarily Putting mysqld Into Permissive Mode

When you are troubleshooting issues that you think SELinux might be causing, it can help to temporarily put `mysqld_t` into permissive mode. This can be done by executing the `semanage` command. For example:

```
sudo semanage permissive -a mysqld_t
```

If that solved the problem, then it means that the current SELinux policy is the culprit. You need to adjust the SELinux policy or labels for MariaDB.

Configuring a MariaDB Server SELinux Policy

MariaDB Server should work with your default distribution policy (which is usually part of the `selinux-policy` or `selinux-policy-targeted` system package). If you use `mysqld_safe`, you will need an additional policy file, `mariadb.pp`, which is installed together with the MariaDB Server. It will be loaded automatically if you have `/usr/sbin/semodule` installed, but you can load it manually anytime with

```
/usr/sbin/semodule -i /usr/share/mysql/policy/selinux/mariadb.pp
```

Note that this policy file extends, but not replaces the system policy.

Setting File Contexts

SELinux uses file contexts as a way to determine who should be able to access that file.

File contexts are managed with the `semanage fcontext` and `restorecon` commands.

On many systems, the `semanage` utility is installed by the `policycoreutils-python` package, and the `restorecon` utility is installed by the `policycoreutils` package. You can install these with the following command:

```
sudo yum install policycoreutils policycoreutils-python
```

A file or directory's current context can be checked by executing `ls` with the `--context` or `--scontext` options.

Setting the File Context for the Data Directory

If you use a custom directory for `datadir`, then you may need to set the file context for that directory. The SELinux file context for MariaDB data files is `mysqld_db_t`. You can determine if this file context is present on your system and which files or directories it is associated with by executing the following command:

```
sudo semanage fcontext --list | grep mysqld_db_t
```

If you would like to set the file context for your custom directory for your `datadir`, then that can be done by executing the `semanage fcontext` and `restorecon` commands. For example:

```
sudo semanage fcontext -a -t mysqld_db_t "/mariadb/data(/.*)?"
sudo restorecon -Rv /mariadb/data
```

If you would like to check the current file context, you can do so by by executing `ls` with the `--context` or `--scontext` options. For example:

```
ls --directory --scontext /mariadb/data
```

Setting the File Context for Log Files

If you use a custom directory for log files, then you may need to set the file context for that directory. The SELinux file context for MariaDB log files is `mysqld_log_t`. You can determine if this file context is present on your system and which files or directories it is associated with by executing the following command:

```
sudo semanage fcontext --list | grep mysqld_log_t
```

If you would like to set the file context for your custom directory for log files, then that can be done by executing the `semanage fcontext` and `restorecon` commands. For example:

```
sudo semanage fcontext -a -t mysqld_log_t "/var/log/mysql(/.*)?"
sudo restorecon -Rv /var/log/mysql
```

If you would like to check the current file context, you can do so by by executing `ls` with the `--context` or `--scontext` options. For example:

```
ls --directory --scontext /var/log/mysql
```

Setting the File Context for Option Files

If you use a custom directory for option files, then you may need to set the file context for that directory. The SELinux file context for MariaDB option files is `mysqld_etc_t`. You can determine if this file context is present on your system and which files or directories it is associated with by executing the following command:

```
sudo semanage fcontext --list | grep mysqld_etc_t
```

If you would like to set the file context for your custom directory for option files, then that can be done by executing the `semanage fcontext` and `restorecon` commands. For example:

```
sudo semanage fcontext -a -t mysqld_etc_t "/etc/mariadb(/.*)?"
sudo restorecon -Rv /etc/mariadb
```

If you would like to check the current file context, you can do so by by executing `ls` with the `--context` or `--scontext` options. For example:

```
ls --directory --scontext /etc/mariadb
```

Allowing Access to the Tmpfs File Context

If you wanted to mount your `tmpdir` on a `tmpfs` file system or wanted to use a `tmpfs` file system on `/run/shm`, then you might need to allow `mysqld_t` to have access to a couple tmpfs-related file contexts. For example:

```
cd /usr/share/mysql/policy/selinux/
tee ./mysqld_tmpfs.te <<EOF
module mysqld_tmpfs 1.0;

require {
type tmpfs_t;
type mysqld_t;
class dir { write search read remove_name open getattr add_name };
class file { write getattr read lock create unlink open };
}

allow mysqld_t tmpfs_t:dir { write search read remove_name open getattr add_name };

allow mysqld_t tmpfs_t:file { write getattr read lock create unlink open }
EOF
sudo checkmodule -M -m mysqld_tmpfs.te -o mysqld_tmpfs.mod
sudo semodule_package -m mysqld_tmpfs.mod -o mysqld_tmpfs.pp
sudo semodule -i mysqld_tmpfs.pp
```

Troubleshooting SELinux Issues

You might need to troubleshoot SELinux-related issues in cases, such as:

- MariaDB is using a non-default port.
- MariaDB is reading from or writing to some files (datadir, log files, option files, etc.) located at non-default paths.
- MariaDB is using a plugin that requires access to resources that default installations do not use.

File System Permission Errors

If the file system permissions for some MariaDB directory look fine, but the MariaDB error log still has errors that look similar to the following:

```
130321 11:50:51 mysqld_safe Starting mysqld daemon with databases from /datadir
...
2013-03-21 11:50:52 2119 [Warning] Can't create test file /datadir/
2013-03-21 11:50:52 2119 [Warning] Can't create test file /datadir/
...
2013-03-21 11:50:52 2119 [ERROR] /usr/sbin/mysqld: Can't create/write to file
    '/datadir/boxy.pid' (Errcode: 13 - Permission denied)
2013-03-21 11:50:52 2119 [ERROR] Can't start server: can't create PID file:
    Permission denied
130321 11:50:52 mysqld_safe mysqld from pid file /datadir/boxy.pid ended
```

Then check SELinux's `/var/log/audit/audit.log` for log entries that look similar to the following:

```
type=AVC msg=audit(1363866652.030:24): avc:  denied  { write } for  pid=2119
    comm="mysqld" name="datadir" dev=dm-0 ino=394
    scontext=unconfined_u:system_r:mysqld_t:s0
    tcontext=unconfined_u:object_r:default_t:s0 tclass=dir
```

If you see any entries that look similar to this, then you most likely need to adjust the file contexts for some files or directories. See Setting File Contexts for more information on how to do that.

SELinux and MariaDB On a Different Port

TCP and UDP ports are enabled for permission to bind too. If you are using a different port, or some Galera ports, configure SELinux to be able to use those ports:

```
sudo semanage port -a -t mysqld_port_t -p tcp 3307
```

Generating SELinux Policies with audit2allow

In some cases, a MariaDB system might need non-standard policies. It is possible to create these policies from the SELinux audit log using the `audit2allow` utility. The `semanage` and `semodule` utilities will also be needed.

On many systems, the `audit2allow` and `semanage` utilities are installed by the `policycoreutils-python` package, and the `semodule` utility is installed by the `policycoreutils` package. You can install these with the following command:

```
sudo yum install policycoreutils policycoreutils-python
```

The following process can be used to generate a policy from the audit log:

- Remove dontaudits from the policy:

```
sudo semodule -DB
```

- Temporarily put `mysqld_t` into permissive mode. For example:

```
sudo semanage permissive -a mysqld_t
```

- Start MariaDB.
- Do whatever was causing SELinux errors.
- Use the generated audit log to create a policy:

```
sudo grep mysqld /var/log/audit/audit.log | audit2allow -M mariadb_local
sudo semodule -i mariadb_local.pp
```

- Pull `mysqld_t` out of permissive mode. For example:

```
sudo semanage permissive -d mysqld_t
```

- Restore dontaudits for the policy:

```
sudo setmodule -B
```

The same procedure can be used if MariaDB starts but SELinux prevents it from functioning correctly. For example, SELinux may prevent PAM plugin from authenticating users. The solution is the same — enable auditing, switch to permissive, do, whatever SELinux didn't allow you to, create a policy from the audit log.

When you discover any needed SELinux permissions, please report the needed permissions to your operating system bug tracking so all users can benefit from your work (e.g. Red Hat Bugzilla https://bugzilla.redhat.com/).

2.2.2 User Account Management

Account Management SQL Commands
CREATE/DROP USER, GRANT, REVOKE, SET PASSWORD etc.

Data-in-Transit Encryption
Data can be encrypted in transit using the Transport Layer Security (TLS) protocol.

Roles
Roles bundle privileges together to ease account management.

Catalogs
Catalogs permit several unrelated users or customers to share a single MariaDB Server instance.

Account Locking
Account locking permits privileged administrators to lock/unlock user accounts.

Authentication from MariaDB 10.4
Authentication changes from MariaDB 10.4.

User Password Expiry
Password expiry permits administrators to expire user passwords.

There are 3 related questions.

1.1.1.1 Account Management SQL Commands

2.2.1.1.1 Data-in-Transit Encryption

2.2.2.3 Roles

Roles bundle privileges together to ease account management

Roles Overview
Bundling privileges together.

CREATE ROLE
Add new roles.

DROP ROLE
Drop a role.

CURRENT_ROLE
Current role name.

SET ROLE
Enable a role.

SET DEFAULT ROLE
Sets a default role for a specified (or current) user.

GRANT
Create accounts and set privileges or roles.

REVOKE
Remove privileges or roles.

mysql.roles_mapping Table
MariaDB roles information.

Information Schema APPLICABLE_ROLES Table

Roles available to be used.

Information Schema ENABLED_ROLES Table

Enabled roles for the current session.

SecuRich

Library of security-related stored procedures.

There are 2 related questions.

2.2.2.3.1 Roles Overview

Contents

Description

A role bundles a number of privileges together. It assists larger organizations where, typically, a number of users would have the same privileges, and, previously, the only way to change the privileges for a group of users was by changing each user's privileges individually.

Alternatively, multiple external users could have been assigned the same user, and there would have been no way to see which actual user was responsible for which action.

With roles, managing this is easy. For example, there could be a number of users assigned to a journalist role, with identical privileges. Changing the privileges for all the journalists is a matter of simply changing the role's privileges, while the individual user is still linked with any changes that take place.

Roles are created with the CREATE ROLE statement, and dropped with the DROP ROLE statement. Roles are then assigned to a user with an extension to the GRANT statement, while privileges are assigned to a role in the regular way with GRANT. Similarly, the REVOKE statement can be used to both revoke a role from a user, or revoke a privilege from a role.

Once a user has connected, he can obtain all privileges associated with a role by **setting** a role with the SET ROLE statement. The CURRENT_ROLE function returns the currently set role for the session, if any.

Only roles granted directly to a user can be set, roles granted to other roles cannot. Instead the privileges granted to a role, which is, in turn, granted to another role (grantee), will be immediately available to any user who sets this second grantee role.

The SET DEFAULT ROLE statement allows one to set a default role for a user. A default role is automatically enabled when a user connects (an implicit SET ROLE statement is executed immediately after a connection is established).

Roles were implemented as a GSoC 2013 project by Vicentiu Ciorbaru.

System Tables

Information about roles and who they've been granted to can be found in the Information Schema APPLICABLE_ROLES table as well as the mysql.ROLES_MAPPING table.

The Information Schema ENABLED_ROLES table shows the enabled roles for the current session.

Examples

Creating a role and granting a privilege:

```
CREATE ROLE journalist;

GRANT SHOW DATABASES ON *.* TO journalist;

GRANT journalist to hulda;
```

Note, that hulda has no `SHOW DATABASES` privilege, even though she was granted the journalist role. She needs to **set** the role first:

```
SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
+--------------------+

SELECT CURRENT_ROLE;
+--------------+
| CURRENT_ROLE |
+--------------+
| NULL         |
+--------------+

SET ROLE journalist;

SELECT CURRENT_ROLE;
+--------------+
| CURRENT_ROLE |
+--------------+
| journalist   |
+--------------+

SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| ...                |
| information_schema |
| mysql              |
| performance_schema |
| test               |
| ...                |
+--------------------+

SET ROLE NONE;
```

Roles can be granted to roles:

```
CREATE ROLE writer;

GRANT SELECT ON data.* TO writer;

GRANT writer TO journalist;
```

But one does not need to set a role granted to a role. For example, hulda will automatically get all writer privileges when she sets the journalist role:

```
SELECT CURRENT_ROLE;
+--------------+
| CURRENT_ROLE |
+--------------+
| NULL         |
+--------------+

SHOW TABLES FROM data;
Empty set (0.01 sec)

SET ROLE journalist;

SELECT CURRENT_ROLE;
+--------------+
| CURRENT_ROLE |
+--------------+
| journalist   |
+--------------+

SHOW TABLES FROM data;
+-----------------------------+
| Tables_in_data              |
+-----------------------------+
| set1                        |
| ...                         |
+-----------------------------+
```

Roles and Views (and Stored Routines)

When a user sets a role, he, in a sense, has two identities with two associated sets of privileges. But a view (or a stored routine) can have only one definer. So, when a view (or a stored routine) is created with the `SQL SECURITY DEFINER`, one can specify whether the definer should be `CURRENT_USER` (and the view will have none of the privileges of the user's role) or `CURRENT_ROLE` (in this case, the view will use role's privileges, but none of the user's privileges). As a result, sometimes one can create a view that is impossible to use.

```
CREATE ROLE r1;

GRANT ALL ON db1.* TO r1;

GRANT r1 TO foo@localhost;

GRANT ALL ON db.* TO foo@localhost;

SELECT CURRENT_USER
+---------------+
| current_user  |
+---------------+
| foo@localhost |
+---------------+

SET ROLE r1;

CREATE TABLE db1.t1 (i int);

CREATE VIEW db.v1 AS SELECT * FROM db1.t1;

SHOW CREATE VIEW db.v1;
+------+-----------------------------------------------------------------------------------------------------------------+----------------------+----------------------+
| View | Create View                                                                                                     | character_set_client | collation_connection |
+------+-----------------------------------------------------------------------------------------------------------------+----------------------+----------------------+
| v1   | CREATE ALGORITHM=UNDEFINED DEFINER=`foo`@`localhost` SQL SECURITY DEFINER VIEW `db`.`v1` AS SELECT `db1`.`t1`.`i` AS `i` from `db1`.`t1` | utf8                 | utf8_general_ci      |
+------+-----------------------------------------------------------------------------------------------------------------+----------------------+----------------------+

CREATE DEFINER=CURRENT_ROLE VIEW db.v2 AS SELECT * FROM db1.t1;

SHOW CREATE VIEW db.b2;
+------+-----------------------------------------------------------------------------------------------------------------+----------------------+----------------------+
| View | Create View                                                                                                     | character_set_client | collation_connection |
+------+-----------------------------------------------------------------------------------------------------------------+----------------------+----------------------+
| v2   | CREATE ALGORITHM=UNDEFINED DEFINER=`r1` SQL SECURITY DEFINER VIEW `db`.`v2` AS select `db1`.`t1`.`a` AS `a` from `db1`.`t1` | utf8                 | utf8_general_ci      |
+------+-----------------------------------------------------------------------------------------------------------------+----------------------+----------------------+
```

Other Resources

- Roles Review by Peter Gulutzan

1.1.1.1.8 CREATE ROLE

1.1.1.1.9 DROP ROLE

1.2.8.3.7 CURRENT_ROLE

1.1.1.1.10 SET ROLE

1.1.1.1.11 SET DEFAULT ROLE

1.1.1.1.4 GRANT

1.1.1.1.6 REVOKE

1.1.1.2.9.3.20 mysqlroles_mapping Table

1.1.1.2.9.1.1.5 Information Schema APPLICABLE_ROLES Table

1.1.1.2.9.1.1.14 Information Schema ENABLED_ROLES Table

2.2.2.4 Catalogs

Catalogs are an upcoming feature intended primarily for Cloud Service Providers with many customers, each having many MariaDB Server users and databases.

Catalogs will permit several unrelated users or customers to share a single MariaDB Server instance.

Catalogs Overview

Catalogs offer multi-tenancy.

Starting with Catalogs

Installing MariaDB with catalogs and adding new catalogs.

Catalog Status Variables

With catalogs enabled, status information is collected for the whole server...

DROP CATALOG

Deletes a catalog.

USE CATALOG

Changes to another catalog.

2.2.2.4.1 Catalogs Overview

Contents

Catalogs are an upcoming feature that will be included in a future release of MariaDB. The MariaDB catalogs will be a multi-tenancy feature where a single instance MariaDB server handles multiple independent tenants (customers), who have their own users, schemas etc. See MDEV-31542 "Add multi-tenancy catalogs to MariaDB" for details.

Background

For hosting providers, a common solution, to drive down cost, is to have one MariaDB server support several different customers by creating one named schema for each of them.

This has however a lot of limitations:

- The user cannot have exactly the same schema(s) on the cloud as they have on premise.
- The user cannot use multiple schemas.
- The user cannot take a backup of all their data (not even with mariadb-dump). This is because the 'mysql' schema, which includes users, stored procedures etc. cannot be copied as its data is shared among all server users.
- The user cannot access the general or error log.

The suggested solution to solve all of the above and thus create a better multi-tenant database is to add support for catalogs to MariaDB.

By each user having their own catalog, they will get very close to the same user experience as if they would have the MariaDB server for themselves.

Catalogs make it possible for hosting providers to have 10-100x more 'not that active' database users on a server compared to having a container or MariaDB server per customer (which limits a 192G server to about 100 customers with a 1G InnoDB buffer each).

User Experience With Catalogs

- Each user is assigned one catalog. The user can specify their catalog in their my.cnf file or as an argument to clients or when connecting to MariaDB server.
- Users can mariadb-dump of all their tables (including the 'mysql' database) and apply it on their own on premise MariaDB or to another 'MariaDB catalog' to duplicate their setup.
- Each catalog has its own privilege system. This allows a MariaDB admin to create users independently in their catalog to users in any other catalog. This also implies that the catalog has to be part of the connect information as otherwise the server does not know which user table to use.
- If the user is using applications that don't yet support catalogs, they can specify the catalog as part of the database when connecting to the server ('catalog.database') or by connecting to a specific port that is associated with a catalog.
- After logging in, a normal user can only see the objects (databases, tables, users etc) from their database. They cannot access other catalogs or change catalogs.
- A normal user cannot change the active catalog with a command. They need to logout from the current catalog and login to another.

For the end user, the MariaDB server will act as a normal a standalone server, with the following differences:

- When connecting to the server, a normal user must specify the catalog. If the connector software does not support catalogs, then the catalog should be specified in the database string. If the catalog is not specified, the 'def' catalog is assumed.
- LOAD DATA INFILE and SELECT … INTO OUTFILE can be configured to only be used with the catalog directory or a directory in it.
- SHUTDOWN command is only for the 'catalog root users'
- Replication (MASTER and SLAVE commands) are only for 'catalog root users'
- Errors from background task (like write error) will be logged into the system error log, not the catalog error log.
- SHOW STATUS will show status data for the whole server, not only for the active catalog.
- The server will handle legacy applications by extending the default database in the connection to contain the catalog in the form "catalog/database". See Appendix for details.
- Tables that are only read from the 'def.mysql' schema:
  - plugin
  - help_* tables
  - time_zone* tables
  - gtid_slave_pos (replication state)
  - innodb_index_stats (innodb internal)
  - servers (federated)
  - transaction_registry (innodb internal)
  - func (udf)
  - performance_schema

New 'catalog root user'

- The 'def' catalog is reserved to store permissions for 'catalog root users', which can access any catalog. * These are meant for admin users that need to do tasks like shutdown, upgrade, create/drop catalogs, managing primaries and replicas etc.
- Only the 'catalog root user' can change to another catalog with 'set catalog catalog_name'.
- A normal user can do 'set catalog current-catalog'. This will be needed to be able to execute a mariadb-dump that includes this command.

New Storage Layout

MariaDB server will be able to run either on 'original mode', where the data layout is exactly as it was before, or on 'catalog' mode, with a new data layout:

When running mariadb-install-db with --use-catalogs, it will create the following new data structure:

- data_directory/
  - engine system data files
  - system files
  - replication files
  - general.log
  - error.log
  - mariadb/
    - mysql/
    - privilege tables
  - catalog1
  - general.log
  - error.log
  - mysql/
    - privilege tables
  - database1/
    - tables for database1
  - database2/
    - tables for database2
  - catalog2/
    - general.log
    - error.log
  - mysql/
    - privilege tables
  - database1/
    - tables for database1
  - database2/
    - tables for database2

The disk structure when not using catalogs is:

- data_directory/
  - engine system data files
  - system files
  - replication files
  - general.log
  - error.log
  - mysql/
    - privilege tables
  - database1/
    - tables for database1
  - database2/
    - tables for database2

The above shows:

- There is a 'mariadb' catalog that stores admin users that can access all catalogs, shutdown servers, create new catalogs etc. The 'system root' user uses this when connecting.
- Each catalog has their own users, privilege tables, databases, error log and general logs

The MariaDB server will automatically start in catalog mode if it notices the new directory structure.

Catalog SQL Commands/Functions

- USE CATALOG catalog_name;
- CREATE CATALOG
- DROP CATALOG
- ALTER CATALOG
- SHOW CATALOGS (and also information_schema.catalogs)
- SHOW CREATE CATALOG catalog_name;
- SELECT CATALOG();

Changes Needed in MariaDB Codebase

Client changes:

- Add --catalog option to all standard MariaDB clients
- Add support for looping over all existing catalogs to:
  - mariadb-dump
  - mariadb-backup
  - mariadb-upgrade

Changes to mariadb-install-db:

- Allow one to create multiple catalogs at once: -–catalogs=”catalog1,catalog2”
- Init MariaDB with catalog support: —use-catalogs

Changes to mariadb (mysql client):

- Add support for 'USE CATALOG xxx'' (and later 'use database xxx').

Changes to mysql-test-run:

- Add support of running tests with catalogs (normal tests are run without catalogs)

Changes to MariaDB server (See MDEV-31542):

- Add support for 'catalog' in the connection string. For old clients, the user can specify the catalog as part of the database. If catalog is not specified, the 'def' catalog (like now) is assumed.
- Add CATALOG() function that returns the current catalog.
- Add ‘USE CATALOG xxx’
- Add 'USE DATABASE xxx'
- Create a global CATALOG object to hold all information related to the catalog.
- Add the current catalog to the 'thd' object.
- Add catalog argument to all functions that take 'database' as an argument.
- Add SHOW CATALOGS and information_schema.catalogs
- Move all relevant global variables (users, privileges, mdl-locks(?), open log files) to be stored in the CATALOG structure.
- Add 'catalog privilege', for ‘catalog super users’ to allow them to access data in any catalog.
- Add support for accessing tables with 'catalog.schema.table' (needed for catalog super users).
- For normal users, only show processes for the current catalog in 'show processlist'.
- Add loops over all catalogs for information schema for the 'catalog root user'.
- Update performance schema to take catalogs into account.
- Work with external connectors to get them to support connecting with a catalog.
- Check/update all storage engines to ensure they work also with catalogs.

Notes:

- The storage handler calls will probably not be changed. The storage engine will get the catalog name as part of the database name (catalog/database).
- We don't need a 'catalog' column for tables in the 'mysql' schema (like mysql.proc) as these are stored per catalog.

Some Implementation Ideas

- Instead of sending a catalog string to function, use a pointer to the global catalog object. Do the same later for databases. This allows use to precompute things like 'filename' for catalogs and databases and we don't have to do this for every table open. It also allows us to later support logging information at a catalog and database level.
- Don't take a MDL lock for the catalog for each table. The metadata lock for the catalog will be taken when a user logs in or changes catalog.
- Add system variables ‘current_catalog’ and ‘current_database’ and allow users to change these.
- Add support for ‘catalog ports’ that are connected to catalog. This allows users to connect to a specific catalog from any client software.

Limitations (in addition to limitations listed in “User experience with catalogs”)

- Database names cannot contain ‘.’ when connecting from clients without the new catalog connect option.
- One cannot refer to other catalogs in triggers, stored procedures, events etc. This is because a transaction cannot span catalogs.
- Only the catalog root user can use mariadb-backup. This is a normal restriction as one has to be system root to be able to use mariadb-backup.
- Events are global (to save resources). Catalog users can enable/disable events for their catalog.

Stage 2 (not in first release)

- Support usage statistics per catalog and whole server (the last for the ‘catalog root user’). This allows the DBA to see the number of queries, type of queries etc. Some ‘system’ and ‘global innodb’ statistics will only be shown globally (number of open files, number of sync calls etc).
- Support a my.cnf file in each catalog directory to handle catalog (customer) unique defaults.
- Add quotas per catalog for tables and temporary files.
- Add more support to limit users from overusing resources (cpu, tables, databases, number of connections etc)
- Support 'drop catalog'. (This is in Stage 2 as there may be some issues to drop already active CATALOG objects)
- Add optional catalog support to the S3 engine
- More things will be added later.

Stage 3

- Allow users to manage their own replication stream (maybe?).
- Allow users to have different options for the S3 engine
- More things will be added later.

Appendix

Legacy Connector Support

SQLALchemy test:

```
In [1]: from sqlalchemy.engine import make_url
In [2]: u = make_url('mariadb+mariadbconnector://app_user:Password123!@127.0.0.1:3306/catalog/com
In [3]: u.database
Out[3]: 'catalog/company'
```

The following tests ensured that inside the server (mysql_change_db), the “catalog/test” was picked up as the database.

PHP PDO test:

```
$ php -r '$db = new \PDO("mysql:host=localhost;user=dan;dbname=catalog/test;charset=utf8mb4;unix_
```

PHP mysqli test:

```
php -r '$dbcon = mysqli_connect("localhost","dan","nopass","catalog/test",3306,"/tmp/build-mariad
```

Nodejs test:

```
var mysql = require('mysql')
var con = mysql.createConnection({
  socketPath: "/tmp/build-mariadb-server-10.4.sock",
  user: "dan",
  password: "yourpassword",
  database: "catalog/test",
})
con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
})
```

(need to map out a few other connectors here to make sure it’s supported well in this form).

Ref: https://mariadb.com/kb/en/connection/#handshake-response-packet

Migration of existing MariaDB original mode to the new catalog layout

As shared hosting services have a naming scheme from user/schema to database name in MariaDB, to provide a migration to the new catalog layout, the following steps will be required:

- Use mariadb-dump to dump the original data
- On the new server execute:
- mariadb-install-db –catalogs=’catalog_name’
- mariadb –catalog catalog_name < dump_file

This is needed as InnoDB needs to know where the new files are located.

Migration of one catalog user to another MariaDB server

Create a migration tool set / procedure that does the following

- Execute FLUSH TABLES FOR EXPORT for all tables in a catalog.
- Take a copy of the catalog directory
- Copy the data to a new catalog directory to the new server
- Run ALTER TABLE ... IMPORT TABLESPACE on each InnoDB table

Note that for partitioned tables the process will be a bit more complex, see above link.

This procedure will be a bit easier after an in-the-works patch for InnoDB related to IMPORT will be pushed. (Should happen before we start on the catalog project)

Other Things

- Drizzle's default catalog was called "local". MariaDB's default will be called 'def', as this is what we already have as the default catalog in information_schema, in current connectors and other places.
- CONNECT engine will need testing against catalogs and maybe a small code change to support them. It could also be a way to join from one catalog to another.

2.2.2.4.2 Starting with Catalogs

MariaDB starting with 11.3
Catalog support is planned for 11.3.

Contents

Background

mariadb-install-db initializes the MariaDB data directory and creates the system tables in the mysql database.

When used with the `--catalog` options it will initialize MariaDB server to use catalogs. The mariadbd server will automatically discover if catalogs are used or not.

Note that **one cannot change** a 'normal server' to a server with catalogs or a server with catalogs to a 'normal server'. In the future we will add tools that will allow one to easily move an existing server inside a catalog or move an server inside a catalog to a standalone server.

Initializing a New Server with Catalog Support

To initialize a server with 4 catalogs (the `def` catalog, that holds the catalog root user (CRU) is automatically created):

```
mariadb_install_db --catalogs="cat1 cat2 cat3" --datadir=/my/data/
```

The above will create a directory `/my/data` and the 4 directories under it, one for each catalog.

Adding More Catalogs to a Running Server

When adding more catalogs to an existing server, `mariadb_install_db` will start the mariadb client to execute the needed commands on the running server. This is why one has to supply user and password to `mariadb_install_db`.

```
mariadb_install_db --catalogs="cat4 cat5 cat6" --datadir=/my/data --catalog-user=monty --catalog-
```

2.2.2.4.3 Catalog Status Variables

When using a MariaDB Server with catalogs support, all status information is collected for the whole server, per catalog and per user.

```
SHOW SERVER STATUS;
```

shows the status for the whole server. Note that only the super user in the 'def' catalog has privileges for the above statement.

```
SHOW GLOBAL STATUS;
SHOW CATALOG STATUS;
```

Both commands show the status for the current catalog. The reason that `GLOBAL` shows catalog status is that because catalogs are 'multi-tenant', a catalog user should not be able to see the status from other users (for most things).

```
SHOW [SESSION] STATUS;
```

Shows the status for the current connection.

The main "new thing" is that catalogs enable SAS providers to see the status for a single tenant (catalog user). This makes it much easier to find 'bad neighbors' (tenants that cause problems for other tenants) so that they can be moved to other servers.

When the MariaDB server is not configured for catalogs, the following commands are equivalent:

```
SHOW GLOBAL STATUS
SHOW SERVER STATUS
SHOW CATALOG STATUS
```

2.2.2.4.4 DROP CATALOG

Syntax

```
DROP CATALOG catalog_name
```

Description

Deletes a catalog. Can only be performed by a super user in the 'def' catalog. If the current catalog is dropped, the user is moved to the 'def' catalog. 'def' catalog cannot be dropped.

Currently, there cannot be any databases in a catalog to be dropped. This will be fixed soon.

2.2.2.4.5 USE CATALOG

Syntax

```
USE CATALOG catalog_name
```

Description

Changes to another catalog. Can only be done by a super user in the 'def' catalog. Changing catalog will update catalog status and reset all session status.

A tenant (a user in any other catalog than 'def') cannot change to another catalog. However tenants can execute `USE CATALOG current_catalog`. This is to allow the user to import SQL scripts that use `USE CATALOG...`.

2.2.2.5 Account Locking

MariaDB starting with 10.4.2

Account locking was introduced in MariaDB 10.4.2.

Contents

Description

Account locking permits privileged administrators to lock/unlock user accounts. No new client connections will be permitted if an account is locked (existing connections are not affected).

User accounts can be locked at creation, with the CREATE USER statement, or modified after creation with the ALTER USER statement. For example:

```
CREATE USER 'lorin'@'localhost' ACCOUNT LOCK;
```

or

```
ALTER USER 'marijn'@'localhost' ACCOUNT LOCK;
```

The server will return an ER_ACCOUNT_HAS_BEEN_LOCKED error when locked users attempt to connect:

```
mysql -ulorin
  ERROR 4151 (HY000): Access denied, this account is locked
```

The ALTER USER statement is also used to unlock a user:

```
ALTER USER 'lorin'@'localhost' ACCOUNT UNLOCK;
```

The SHOW CREATE USER statement will show whether the account is locked:

```
SHOW CREATE USER 'marijn'@'localhost';
+-----------------------------------------------+
| CREATE USER for marijn@localhost              |
+-----------------------------------------------+
| CREATE USER 'marijn'@'localhost' ACCOUNT LOCK |
+-----------------------------------------------+
```

as well as querying the mysql.global_priv table:

```
SELECT CONCAT(user, '@', host, ' => ', JSON_DETAILED(priv)) FROM mysql.global_priv
  WHERE user='marijn';
+-----------------------------------------------------------------------------------------+
| CONCAT(user, '@', host, ' => ', JSON_DETAILED(priv))                                    |
+-----------------------------------------------------------------------------------------+
| marijn@localhost => {
    "access": 0,
    "plugin": "mysql_native_password",
    "authentication_string": "",
    "account_locked": true,
    "password_last_changed": 1558017158
} |
+-----------------------------------------------------------------------------------------+
```

2.2.2.6 Authentication from MariaDB 10.4

MariaDB starting with 10.4

MariaDB 10.4 introduced a number of changes to the authentication process, intended to make things easier and more intuitive.

Contents

Overview

There are four **new main features in 10.4** relating to authentication:

- It is possible to use more than one authentication plugin for each user account. For example, this can be useful to slowly migrate users to the more secure ed25519 authentication plugin over time, while allowing the old mysql_native_password authentication plugin as an alternative for the transitional period.
- The `root@localhost` user account created by mariadb-install-db is created with the ability to use two authentication plugins.
    - First, it is configured to try to use the unix_socket authentication plugin. This allows the `root@localhost` user to login without a password via the local Unix socket file defined by the socket system variable, as long as the login is attempted from a process owned by the operating system `root` user account.
    - Second, if authentication fails with the unix_socket authentication plugin, then it is configured to try to use the mysql_native_password authentication plugin. However, an invalid password is initially set, so in order to authenticate this way, a password must be set with SET PASSWORD.
    - However, just using the unix_socket authentication plugin may be fine for many users, and it is very secure. You may want to try going without password authentication to see how well it works for you. Remember, the best way to keep your password safe is not to have one!
- All user accounts, passwords, and global privileges are now stored in the mysql.global_priv table. The mysql.user table still exists and has exactly the same set of columns as before, but it's now a view that references the mysql.global_priv table. Tools that analyze the mysql.user table should continue to work as before. From MariaDB 10.4.13, the dedicated `mariadb.sys` user is created as the definer of this view. Previously `root` was the definer, which resulted in privilege problems when this username was changed.
- MariaDB 10.4 adds supports for User Password Expiry, which is not active by default.

Description

As a result of the above changes, the open-for-everyone all-powerful root account is finally gone. And installation scripts will no longer demand that you "PLEASE REMEMBER TO SET A PASSWORD FOR THE MariaDB root USER !", because the root account is securely created automatically.

Two all-powerful accounts are created by default — root and the OS user that owns the data directory, typically mysql. They are created as:

```
CREATE USER root@localhost IDENTIFIED VIA unix_socket OR mysql_native_password USING 'invalid'
CREATE USER mysql@localhost IDENTIFIED VIA unix_socket OR mysql_native_password USING 'invalid'
```

Using unix_socket means that if you are the system root user, you can login as root@locahost without a password. This technique was pioneered by Otto Kekäläinen in Debian MariaDB packages and has been successfully used in Debian since as early as MariaDB 10.0.

It is based on a simple fact that asking the system root for a password adds no extra security — root has full access to all the data files and all process memory anyway. But not asking for a password means, there is no root password to forget (no need for the numerous tutorials on "how to reset MariaDB root password"). And if you want to script some tedious database work, there is no need to store the root password in plain text for the script to use (no need for debian-sys-maint user).

Still, some users may wish to log in as MariaDB root without using sudo. Hence the old authentication method — conventional MariaDB password — is still available. By default it is disabled ("invalid" is not a valid password hash), but one can set the password with a usual SET PASSWORD statement. And still retain the password-less access via sudo.

If you install MariaDB locally (say from a tarball), you would not want to use sudo to be able to login. This is why MariaDB creates a second all-powerful user with the same name as a system user that owns the data directory. In local (not system-wide) installations, this will be the user who installed MariaDB — they automatically get convenient password-less root-like access, because they can access all the data files anyway.

Even if MariaDB is installed system-wide, you may not want to run your database maintenance scripts as system root — now you can run them as system mysql user. And you will know that they will never destroy your entire system, even if you make a typo in a shell script.

However, seasoned MariaDB DBAs who are used to the old ways do need to make some changes. See the examples below for common tasks.

Cookbook

After installing MariaDB system-wide the first thing you've got used to doing is logging in into the unprotected root account and protecting it, that is, setting the root password:

```
$ sudo dnf install MariaDB-server
$ mysql -uroot
...
MariaDB> set password = password("XH4VmT3_jt");
```

This is not only unnecessary now, it will simply not work — there is no unprotected root account. To login as root use

```
$ sudo dnf install MariaDB-server
$ sudo mysql
```

Note that it implies you are connecting via the unix socket, not tcp. If you happen to have `protocol=tcp` in a system-wide `/etc/my.cnf` file, use `sudo mysql --protocol=socket`.

After installing MariaDB locally you've also used to connect to the unprotected root account using `mysql -uroot`. This will not work either, simply use `mysql` without specifying a username.

If you've forgotten your root password, no problem — you can still connect using sudo and change the password. And if you've also removed unix_socket authentication, to restore access do as follows:

- restart MariaDB with --skip-grant-tables
- login into the unprotected server
- run FLUSH PRIVILEGES (note, before 10.4 this would've been the last step, not anymore). This disables `--skip-grant-tables` and allows you to change the stored authentication method
- run SET PASSWORD FOR root@localhost to change the root password.

To view inside privilege tables, the old mysql.user table still exists. You can select from it as before, although you cannot update it anymore. It doesn't show alternative authentication plugins and this was one of the reasons for switching to the mysql.global_priv table — complex authentication rules did not fit into rigid structure of a relational table. You can select from the new table, for example:

```
select concat(user, '@', host, ' => ', json_detailed(priv)) from mysql.global_priv;
```

Reverting to the Previous Authentication Method for root@localhost

If you don't want the `root@localhost` user account created by mariadb-install-db to use unix_socket authentication by default, then there are a few ways to revert to the previous mysql_native_password authentication method for this user account.

Configuring mariadb-install-db to Revert to the Previous Authentication Method

One way to revert to the previous mysql_native_password authentication method for the `root@localhost` user account is to execute mariadb-install-db with a special option. If mariadb-install-db is executed while `--auth-root-authentication-method=normal` is specified, then it will create the default user accounts using the default behavior of MariaDB 10.3 and before.

This means that the `root@localhost` user account will use mysql_native_password authentication by default. There are some other differences as well. See mariadb-install-db: User Accounts Created by Default for more information.

For example, the option can be set on the command-line while running mariadb-install-db:

```
mariadb-install-db --user=mysql --datadir=/var/lib/mysql --auth-root-authentication-method=normal
```

The option can also be set in an option file in an option group supported by mariadb-install-db. For example:

```
[mysql_install_db]
auth_root_authentication_method=normal
```

If the option is set in an option file and if mariadb-install-db is executed, then mariadb-install-db will read this option from the option file, and it will automatically set this option.

Altering the User Account to Revert to the Previous Authentication Method

If you have already installed MariaDB, and if the `root@localhost` user account is already using unix_socket authentication, then you can revert to the old mysql_native_password authentication method for the user account by executing the following:

```
ALTER USER root@localhost IDENTIFIED VIA mysql_native_password USING PASSWORD("verysecret")
```

2.2.2.7 User Password Expiry

MariaDB starting with 10.4.3

User password expiry was introduced in MariaDB 10.4.3.

Contents

Password expiry permits administrators to expire user passwords, either manually or automatically.

System Variables

There are two system variables which affect password expiry: default_password_lifetime, which determines the amount of time between requiring the user to change their password. `0`, the default, means automatic password expiry is not active.

The second variable, disconnect_on_expired_password determines whether a client is permitted to connect if their password has expired, or whether they are permitted to connect in sandbox mode, able to perform a limited subset of queries related to resetting the password, in particular SET PASSWORD and SET.

Setting a Password Expiry Limit for a User

Besides automatic password expiry, as determined by default_password_lifetime, password expiry times can be set on an individual user basis, overriding the global using the CREATE USER or ALTER USER statements, for example:

```
CREATE USER 'monty'@'localhost' PASSWORD EXPIRE INTERVAL 120 DAY;
```

```
ALTER USER 'monty'@'localhost' PASSWORD EXPIRE INTERVAL 120 DAY;
```

Limits can be disabled by use of the `NEVER` keyword, for example:

```
CREATE USER 'monty'@'localhost' PASSWORD EXPIRE NEVER;
```

```
ALTER USER 'monty'@'localhost' PASSWORD EXPIRE NEVER;
```

A manually set limit can be restored the system default by use of `DEFAULT`, for example:

```
CREATE USER 'monty'@'localhost' PASSWORD EXPIRE DEFAULT;
```

```
ALTER USER 'monty'@'localhost' PASSWORD EXPIRE DEFAULT;
```

SHOW CREATE USER

The SHOW CREATE USER statement will display information about the password expiry status of the user. Unlike MySQL, it will not display if the user is unlocked, or if the password expiry is set to default.

```
CREATE USER 'monty'@'localhost' PASSWORD EXPIRE INTERVAL 120 DAY;
CREATE USER 'konstantin'@'localhost' PASSWORD EXPIRE NEVER;
CREATE USER 'amse'@'localhost' PASSWORD EXPIRE DEFAULT;

SHOW CREATE USER 'monty'@'localhost';
+-------------------------------------------------------------------+
| CREATE USER for monty@localhost                                   |
+-------------------------------------------------------------------+
| CREATE USER 'monty'@'localhost' PASSWORD EXPIRE INTERVAL 120 DAY |
+-------------------------------------------------------------------+

SHOW CREATE USER 'konstantin'@'localhost';
+-----------------------------------------------------------+
| CREATE USER for konstantin@localhost                      |
+-----------------------------------------------------------+
| CREATE USER 'konstantin'@'localhost' PASSWORD EXPIRE NEVER |
+-----------------------------------------------------------+

SHOW CREATE USER 'amse'@'localhost';
+--------------------------------+
| CREATE USER for amse@localhost |
+--------------------------------+
| CREATE USER 'amse'@'localhost' |
+--------------------------------+
```

Checking When Passwords Expire

The following query can be used to check when the current passwords expire for all users:

```
WITH password_expiration_info AS (
  SELECT User, Host,
  IF(
   IFNULL(JSON_EXTRACT(Priv, '$.password_lifetime'), -1) = -1,
   @@global.default_password_lifetime,
   JSON_EXTRACT(Priv, '$.password_lifetime')
  ) AS password_lifetime,
  JSON_EXTRACT(Priv, '$.password_last_changed') AS password_last_changed
  FROM mysql.global_priv
)
SELECT pei.User, pei.Host,
  pei.password_lifetime,
  FROM_UNIXTIME(pei.password_last_changed) AS password_last_changed_datetime,
  FROM_UNIXTIME(
   pei.password_last_changed +
   (pei.password_lifetime * 60 * 60 * 24)
  ) AS password_expiration_datetime
  FROM password_expiration_info pei
  WHERE pei.password_lifetime != 0
   AND pei.password_last_changed IS NOT NULL
UNION
SELECT pei.User, pei.Host,
  pei.password_lifetime,
  FROM_UNIXTIME(pei.password_last_changed) AS password_last_changed_datetime,
  0 AS password_expiration_datetime
  FROM password_expiration_info pei
  WHERE pei.password_lifetime = 0
   OR pei.password_last_changed IS NULL;
```

--connect-expired-password Client Option

The mariadb client `--connect-expired-password` option notifies the server that the client is prepared to handle expired password sandbox mode (even if the `--batch` option was specified).

2.3 Backing Up and Restoring Databases

There are a number of ways to backup a MariaDB server.

Backup and Restore Overview
Backing up and restoring MariaDB.

Replication as a Backup Solution
Replication can be used to support the backup strategy.

mariadb-dump
Dump a database or a collection of databases in a portable format.

Mariabackup
Physical backups, supports Data-at-Rest and InnoDB compression.

Backup and Restore via dbForge Studio
The fastest and easiest way to perform these operations with MariaDB databases.

mariadb-hotcopy
Fast backup program on local machine. Deprecated.

There are 13 related questions.

2.3.1 Backup and Restore Overview

Contents

This article briefly discusses the main ways to backup MariaDB. For detailed descriptions and syntax, see the individual pages. More detail is in the process of being added.

Logical vs Physical Backups

Logical backups consist of the SQL statements necessary to restore the data, such as CREATE DATABASE, CREATE TABLE and INSERT.

Physical backups are performed by copying the individual data files or directories.

The main differences are as follows:

- logical backups are more flexible, as the data can be restored on other hardware configurations, MariaDB versions or even on another DBMS, while physical backups cannot be imported on significantly different hardware, a different DBMS, or potentially even a different MariaDB version.
- logical backups can be performed at the level of database and table, while physical databases are the level of directories and files. In the MyISAM and InnoDB storage engines, each table has an equivalent set of files. (In versions prior to MariaDB 5.5, by default a number of InnoDB tables are stored in the same file, in which case it is not possible to backup by table. See innodb_file_per_table.)
- logical backups are larger in size than the equivalent physical backup.
- logical backups takes more time to both backup and restore than the equivalent physical backup.
- log files and configuration files are not part of a logical backup

Backup Tools

Mariadb-backup

Mariadb-backup is a fork of Percona XtraBackup with added support for MariaDB 10.1 compression and data-at-rest encryption. It is included with MariaDB 10.1.23 and later.

mariadb-dump

mariadb-dump (previously mysqldump) performs a logical backup. It is the most flexible way to perform a backup and restore, and a good choice when the data size is relatively small.

For large datasets, the backup file can be large, and the restore time lengthy.

mariadb-dump dumps the data into SQL format (it can also dump into other formats, such as CSV or XML) which can then easily be imported into another database. The data can be imported into other versions of MariaDB, MySQL, or even another DBMS entirely, assuming there are no version or DBMS-specific statements in the dump.

mariadb-dump dumps triggers along with tables, as these are part of the table definition. However, stored procedures, views, and events are not, and need extra parameters to be recreated explicitly (for example, `--routines` and `--events`). Procedures and functions are however also part of the system tables (for example mysql.proc).

InnoDB Logical Backups

InnoDB uses the buffer pool, which stores data and indexes from its tables in memory. This buffer is very important for performance. If InnoDB data doesn't fit the memory, it is important that the buffer contains the most frequently accessed data. However, last accessed data is candidate for insertion into the buffer pool. If not properly configured, when a table scan happens, InnoDB may copy the whole contents of a table into the buffer pool. The problem with logical backups is that they always imply full table scans.

An easy way to avoid this is by increasing the value of the innodb_old_blocks_time system variable. It represents the number of milliseconds that must pass before a recently accessed page can be put into the "new" sublist in the buffer pool. Data which is accessed only once should remain in the "old" sublist. This means that they will soon be evicted from the buffer pool. Since during the backup process the "old" sublist is likely to store data that is not useful, one could also consider resizing it by changing the value of the innodb_old_blocks_pct system variable.

It is also possible to explicitly dump the buffer pool on disk before starting a logical backup, and restore it after the process. This will undo any negative change to the buffer pool which happens during the backup. To dump the buffer pool, the innodb_buffer_pool_dump_now system variable can be set to ON. To restore it, the innodb_buffer_pool_load_now system variable can be set to ON.

Examples

Backing up a single database

```
shell> mariadb-dump db_name > backup-file.sql
```

Restoring or loading the database

```
shell> mariadb db_name < backup-file.sql
```

See the mariadb-dump page for detailed syntax and examples.

mariadb-hotcopy

> mariadb-hotcopy is deprecated.

mariadb-hotcopy performs a physical backup, and works only for backing up MyISAM and ARCHIVE tables. It can only be run on the same machine as the location of the database directories.

Examples

```
shell> mariadb-hotcopy db_name [/path/to/new_directory]
shell> mariadb-hotcopy db_name_1 ... db_name_n /path/to/new_directory
```

Percona XtraBackup

In MariaDB 10.1 and later, Mariabackup is the recommended backup method to use instead of Percona XtraBackup.

In MariaDB 10.3, Percona XtraBackup is **not supported**. See Percona XtraBackup Overview: Compatibility with MariaDB for more information.

In MariaDB 10.2 and MariaDB 10.1, Percona XtraBackup is only **partially supported**. See Percona XtraBackup Overview: Compatibility with MariaDB for more information.

Percona XtraBackup is a tool for performing fast, hot backups. It was designed specifically for XtraDB/InnoDB databases, but can be used with any storage engine (although not with MariaDB 10.1 encryption and compression). It is not included by default with MariaDB.

Filesystem Snapshots

Some filesystems, like Veritas, support snapshots. During the snapshot, the table must be locked. The proper steps to obtain a snapshot are:

- From the mariadb client, execute FLUSH TABLES WITH READ LOCK. The client must remain open.
- From a shell, execute `mount vxfs snapshot`
- The client can execute UNLOCK TABLES.
- Copy the snapshot files.
- From a shell, unmount the snapshot with `umount snapshot`.

LVM

Widely-used physical backup method, using a Perl script as a wrapper. See http://www.lenzg.net/mylvmbackup/.

Percona TokuBackup

For details, see:

- TokuDB Hot Backup – Part 1
- TokuDB Hot Backup – Part 2
- TokuDB Hot Backup Now a MySQL Plugin

dbForge Studio for MySQL

Besides the system utilities, it is possible to use third-party GUI tools to perform backup and restore operations. In this context, it is worth mentioning dbForge Studio for MySQL, a feature-rich database IDE that is fully compatible with MariaDB and delivers extensive backup functionality.

The backup and restore module of the Studio allows precise configuration and management of full and partial backups up to particular database objects. The feature of scheduling regular backups offers specific settings to handle errors and keep a log of them. Additionally, settings and configurations can be saved for later reuse.

These operations are wizard-aided allowing users to set up all tasks in a visual mode.

3.1.6 Replication as a Backup Solution

1.3.6.2 mysqldump

2.3.4 Mariabackup

Mariabackup is an open source tool provided by MariaDB for performing physical online backups of InnoDB, MyRocks, Aria and MyISAM tables.

Mariabackup Overview

The Mariabackup utility performs physical backups and supports Data-at-Rest...

Mariabackup Options
Options for Mariabackup.

Full Backup and Restore with Mariabackup
Taking complete backups of databases and restoring from a complete backup.

Incremental Backup and Restore with Mariabackup
Backing up incremental changes of a database

Partial Backup and Restore with Mariabackup
Taking partial backups of databases and restoring from a partial backup.

Restoring Individual Tables and Partitions with Mariabackup
Restoring individual tables and partitions from a backup.

Setting up a Replica with Mariabackup
Setting up a replica with Mariabackup.

Files Backed Up By Mariabackup
Mariabackup backs up many different files in order to perform its operations.

Files Created by Mariabackup
Mariabackup creates many different files in order to perform its operations.

Using Encryption and Compression Tools With Mariabackup
Mariabackup supports streaming to stdout, allowing easy integration with popular tools.

How Mariabackup Works
Description of the different Mariabackup stages, what they do and why they are needed.

Mariabackup and BACKUP STAGE Commands
How Mariabackup could use BACKUP STAGE commands.

mariabackup SST Method
The mariabackup SST method uses the Mariabackup utility for performing SSTs.

Manual SST of Galera Cluster Node With Mariabackup
It can be helpful to perform a "manual SST" with Mariabackup when Galera's normal SSTs fail.

Individual Database Restores with MariaBackup from Full Backup
Restoring individual databases with MariaBackup from full backup.

There are 9 related questions.

2.3.4.1 Mariabackup Overview

Mariabackup is an open source tool provided by MariaDB for performing physical online backups of InnoDB, Aria and MyISAM tables. For InnoDB, “hot online” backups are possible. It was originally forked from Percona XtraBackup 2.3.8. It is available on Linux and Windows.

Contents

Backup Support for MariaDB-Exclusive Features

MariaDB 10.1 introduced features that are exclusive to MariaDB, such as InnoDB Page Compression and Data-at-Rest Encryption. These exclusive features have been very popular with MariaDB users. However, existing backup solutions from the MySQL ecosystem, such as Percona XtraBackup, did not support full backup capability for these features.

To address the needs of our users, we decided to develop a backup solution that would fully support these popular MariaDB-exclusive features. We did this by creating Mariabackup, which is based on the well-known and commonly used backup tool called Percona XtraBackup. Mariabackup was originally extended from version 2.3.8.

Supported Features

Mariabackup supports all of the main features of Percona XtraBackup 2.3.8, plus:

- Backup/Restore of tables using Data-at-Rest Encryption.
- Backup/Restore of tables using InnoDB Page Compression.
- mariabackup SST method with Galera Cluster.
- Microsoft Windows support.
- Backup/Restore of tables using the MyRocks storage engine starting with MariaDB 10.2.16 and MariaDB 10.3.8. See Files Backed up by Mariabackup: MyRocks Data Files for more information.

Supported Features in MariaDB Enterprise Backup

MariaDB Enterprise Backup supports some additional features, such as:

- Minimizes locks during the backup to permit more concurrency and to enable faster backups.
  - This relies on the usage of `BACKUP STAGE` commands and DDL logging.
  - This includes no locking during the copy phase of `ALTER TABLE` statements, which tends to be the longest phase of these statements.
- Provides optimal backup support for all storage engines that store things on local disk.

Differences Compared to Percona XtraBackup

- Percona XtraBackup copies its InnoDB redo log files to the file `xtrabackup_logfile`, while Mariabackup uses the

file `ib_logfile0`.

- Percona XtraBackup's libgcrypt-based encryption of backups is not supported by Mariabackup.
- There is no symbolic link from `mariabackup` to `innobackupex`, as there is for `xtrabackup`. Instead, `mariabackup` has the `--innobackupex` command-line option to enable innobackupex-compatible options.
- The `--compact` and `--rebuild_indexes` options are not supported.
- Support for `--stream=tar` was removed from Mariabackup in MariaDB 10.1.24.
- The `xbstream` utility has been renamed to `mbstream`. However, to select this output format when creating a backup, Mariabackup's `--stream` option still expects the `xbstream` value.
- Mariabackup does not support lockless binlog.

Difference in Versioning Schemes

Each Percona XtraBackup release has two version numbers--the Percona XtraBackup version number and the version number of the MySQL Server release that it is based on. For example:

```
xtrabackup version 2.2.8 based on MySQL server 5.6.22
```

Each Mariabackup release only has one version number, and it is the same as the version number of the MariaDB Server release that it is based on. For example:

```
mariabackup based on MariaDB server 10.2.15-MariaDB Linux (x86_64)
```

See Compatibility of Mariabackup Releases with MariaDB Server Releases for more information on Mariabackup versions.

Compatibility of Mariabackup Releases with MariaDB Server Releases

> It is not generally possible, or supported, to prepare a backup in a different MariaDB version than the database version at the time when backup was taken. For example, if you backup MariaDB 10.4, you should use mariabackup version 10.4, rather than e.g 10.5.

A MariaDB Server version can often be backed up with most other Mariabackup releases in the same release series. For example, MariaDB 10.2.21 and MariaDB 10.2.22 are both in the MariaDB 10.2 release series, so MariaDB Server from MariaDB 10.2.21 could be backed up by Mariabackup from MariaDB 10.2.22, or vice versa.

However, occasionally, a MariaDB Server or Mariabackup release will include bug fixes that will break compatibility with previous releases. For example, the fix for MDEV-13564 changed the InnoDB redo log format in MariaDB 10.2.19 which broke compatibility with previous releases. To be safest, a MariaDB Server release should generally be backed up with the Mariabackup release that has the same version number.

> Mariabackup from MariaDB 10.1 releases may also be able to back up MariaDB Server from MariaDB 5.5 and MariaDB 10.0 releases in many cases. However, this is not fully supported. See MDEV-14936 for more information.

Installing Mariabackup

Installing on Linux

The `mariabackup` executable is included in binary tarballs on Linux.

Installing with a Package Manager

Mariabackup can also be installed via a package manager on Linux. In order to do so, your system needs to be configured to install from one of the MariaDB repositories.

You can configure your package manager to install it from MariaDB Corporation's MariaDB Package Repository by using the MariaDB Package Repository setup script.

You can also configure your package manager to install it from MariaDB Foundation's MariaDB Repository by using the MariaDB Repository Configuration Tool.

Installing with yum/dnf

On RHEL, CentOS, Fedora, and other similar Linux distributions, it is highly recommended to install the relevant RPM package from MariaDB's repository using `yum` or `dnf`. Starting with RHEL 8 and Fedora 22, `yum` has been replaced by `dnf`, which is the next major version of `yum`. However, `yum` commands still work on many systems that use `dnf`. For example:

```
sudo yum install MariaDB-backup
```

Installing with apt-get

On Debian, Ubuntu, and other similar Linux distributions, it is highly recommended to install the relevant DEB package from MariaDB's repository using `apt-get`. For example:

```
sudo apt-get install mariadb-backup
```

Installing with zypper

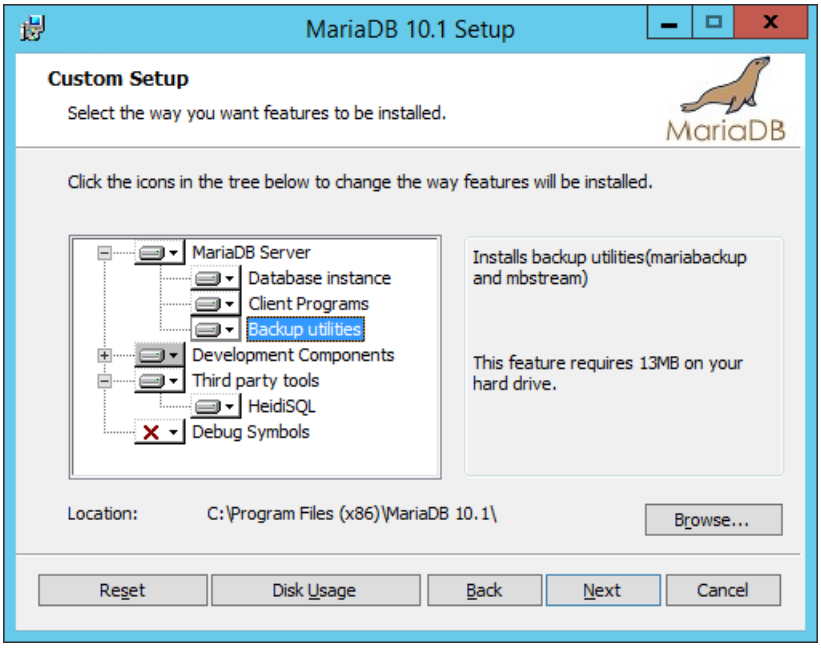
On SLES, OpenSUSE, and other similar Linux distributions, it is highly recommended to install the relevant RPM package from MariaDB's repository using `zypper`. For example:

```
sudo zypper install MariaDB-backup
```

Installing on Windows

The `mariabackup` executable is included in MSI and ZIP packages on Windows.

When using the Windows MSI installer, `mariabackup` can be installed by selecting *Backup utilities*:



Using Mariabackup

The command to use `mariabackup` and the general syntax is:

```
mariabackup <options>
```

For in-depth explanations on how to use Mariabackup, see:

- Full Backup and Restore with Mariabackup
- Incremental Backup and Restore with Mariabackup
- Partial Backup and Restore with Mariabackup
- Restoring Individual Tables and Partitions with Mariabackup
- Setting up a Replication Slave with Mariabackup
- Using Encryption and Compression Tools With Mariabackup

Options

Options supported by Mariabackup can be found here.

`mariabackup` will currently silently ignore unknown command-line options, so be extra careful about accidentally including typos in options or accidentally using options from later `mariabackup` versions. The reason for this is that `mariabackup` currently treats command-line options and options from option files equivalently. When it reads from these option files, it has to read a lot of options from the server option groups read by `mysqld`. However, `mariabackup` does not know about many of the options that it normally reads in these option groups. If `mariabackup` raised an error or warning when it encountered an unknown option, then this process would generate a large amount of log messages under normal use. Therefore, `mariabackup` is designed to silently ignore the unknown options instead. See MDEV-18215 about that.

Option Files

In addition to reading options from the command-line, Mariabackup can also read options from option files.

The following options relate to how MariaDB command-line tools handles option files. They must be given as the first argument on the command-line:

Option	Description
`--print-defaults`	Print the program argument list and exit.
`--no-defaults`	Don't read default options from any option file.
`--defaults-file=#`	Only read default options from the given option file.
`--defaults-extra-file=#`	Read this file after the global files are read.
`--defaults-group-suffix=#`	In addition to the default option groups, also read option groups with this suffix.

Server Option Groups

Mariabackup reads server options from the following option groups from option files:

Group	Description
`[mariabackup]`	Options read by Mariabackup. Available starting with MariaDB 10.1.31 and MariaDB 10.2.13.
`[mariadb-backup]`	Options read by Mariabackup. Available starting with MariaDB 10.4.14 and MariaDB 10.5.4.
`[xtrabackup]`	Options read by Mariabackup and Percona XtraBackup.
`[server]`	Options read by MariaDB Server. Available starting with MariaDB 10.1.38, MariaDB 10.2.22, and MariaDB 10.3.13.
`[mysqld]`	Options read by `mysqld`, which includes both MariaDB Server and MySQL Server.
`[mysqld-X.Y]`	Options read by a specific version of `mysqld`, which includes both MariaDB Server and MySQL Server. For example, `[mysqld-10.4]`. Available starting with MariaDB 10.1.38, MariaDB 10.2.22, and MariaDB 10.3.13.
`[mariadb]`	Options read by MariaDB Server. Available starting with MariaDB 10.1.38, MariaDB 10.2.22, and MariaDB 10.3.13.
`[mariadb-X.Y]`	Options read by a specific version of MariaDB Server. For example, `[mariadb-10.4]`. Available starting with MariaDB 10.1.38, MariaDB 10.2.22, and MariaDB 10.3.13.
`[mariadbd]`	Options read by MariaDB Server. Available starting with MariaDB 10.4.14 and MariaDB 10.5.4.
`[mariadbd-X.Y]`	Options read by a specific version of MariaDB Server. For example, `[mariadbd-10.4]`. Available starting with MariaDB 10.4.14 and MariaDB 10.5.4.
`[client-server]`	Options read by all MariaDB client programs and the MariaDB Server. This is useful for options like socket and port, which is common between the server and the clients. Available starting with MariaDB 10.1.38, MariaDB 10.2.22, and MariaDB 10.3.13.
`[galera]`	Options read by MariaDB Server, but only if it is compiled with Galera Cluster support. In MariaDB 10.1 and later, all builds on Linux are compiled with Galera Cluster support. When using one of these builds, options from this option group are read even if the Galera Cluster functionality is not enabled. Available starting with MariaDB 10.1.38, MariaDB 10.2.22, and MariaDB 10.3.13 on systems compiled with Galera Cluster support.

Client Option Groups

Mariabackup reads client options from the following option groups from option files:

Group	Description
`[mariabackup]`	Options read by Mariabackup. Available starting with MariaDB 10.1.31 and MariaDB 10.2.13.
`[mariadb-backup]`	Options read by Mariabackup. Available starting with MariaDB 10.4.14 and MariaDB 10.5.4.
`[xtrabackup]`	Options read by Mariabackup and Percona XtraBackup.
`[client]`	Options read by all MariaDB and MySQL client programs, which includes both MariaDB and MySQL clients. For example, `mysqldump`.
`[client-server]`	Options read by all MariaDB client programs and the MariaDB Server. This is useful for options like socket and port, which is common between the server and the clients. Available starting with MariaDB 10.1.38, MariaDB 10.2.22, and MariaDB 10.3.13.
`[client-mariadb]`	Options read by all MariaDB client programs. Available starting with MariaDB 10.1.38, MariaDB 10.2.22, and MariaDB 10.3.13.

Authentication and Privileges

Mariabackup needs to authenticate with the database server when it performs a backup operation (i.e. when the `--backup` option is specified). For most use cases, the user account that performs the backup needs to have the following global privileges on the database server.

In 10.5 and later the required privileges are:

```
CREATE USER 'mariabackup'@'localhost' IDENTIFIED BY 'mypassword';
GRANT RELOAD, PROCESS, LOCK TABLES, BINLOG MONITOR ON *.* TO 'mariabackup'@'localhost';
```

Prior to 10.5, the required privileges are:

```
CREATE USER 'mariabackup'@'localhost' IDENTIFIED BY 'mypassword';
GRANT RELOAD, PROCESS, LOCK TABLES, REPLICATION CLIENT ON *.* TO 'mariabackup'@'localhost';
```

If your database server is also using the MyRocks storage engine, then the user account that performs the backup will also need the `SUPER` global privilege. This is because Mariabackup creates a checkpoint of this data by setting the `rocksdb_create_checkpoint` system variable, which requires this privilege. See MDEV-20577 for more information.

To use the `--history` option, the backup user also needs to have the following privileges granted:

```
GRANT CREATE, INSERT ON mysql.* TO 'mariabackup'@'localhost';
```

Prior to MariaDB 10.11, the necessary permissions to use `--history` were:

```
GRANT CREATE, INSERT ON PERCONA_SCHEMA.* TO 'mariabackup'@'localhost';
```

If you're upgrading from an older version and you want to use the new default table without losing your backup history, you can move and rename the current table in this way:

```
RENAME TABLE PERCONA_SCHEMA.xtrabackup_history TO mysql.mariadb_backup_history;
```

The user account information can be specified with the `--user` and `--password` command-line options. For example:

```
$ mariabackup --backup \
   --target-dir=/var/mariadb/backup/ \
   --user=mariabackup --password=mypassword
```

The user account information can also be specified in a supported client option group in an option file. For example:

```
[mariabackup]
user=mariabackup
password=mypassword
```

Mariabackup does not need to authenticate with the database server when preparing or restoring a backup.

File System Permissions

Mariabackup has to read MariaDB's files from the file system. Therefore, when you run Mariabackup as a specific operating system user, you should ensure that user account has sufficient permissions to read those files.

If you are using Linux and if you installed MariaDB with a package manager, then MariaDB's files will probably be owned by the `mysql` user and the `mysql` group.

Using Mariabackup with Data-at-Rest Encryption

Mariabackup supports Data-at-Rest Encryption.

Mariabackup will query the server to determine which key management and encryption plugin is being used, and then it will load that plugin itself, which means that Mariabackup needs to be able to load the key management and encryption plugin's shared library.

Mariabackup will also query the server to determine which encryption keys it needs to use.

In other words, Mariabackup is able to figure out a lot of encryption-related information on its own, so normally one doesn't need to provide any extra options to backup or restore encrypted tables.

Mariabackup backs up encrypted and unencrypted tables as they are on the original server. If a table is encrypted, then the table will remain encrypted in the backup. Similarly, if a table is unencrypted, then the table will remain unencrypted in the backup.

The primary reason that Mariabackup needs to be able to encrypt and decrypt data is that it needs to apply InnoDB redo log records to make the data consistent when the backup is prepared. As a consequence, Mariabackup does not perform many encryption or decryption operations when the backup is initially taken. MariaDB performs more encryption and decryption operations when the backup is prepared. This means that some encryption-related problems (such as using the wrong encryption keys) may not become apparent until the backup is prepared.

Using Mariabackup for Galera SSTs

The `mariabackup` SST method uses the Mariabackup utility for performing SSTs. See mariabackup SST method for more information.

Files Backed up by Mariabackup

Mariabackup backs up many different files in order to perform its backup operation. See Files Backed up by Mariabackup for a list of these files.

Files Created by Mariabackup

Mariabackup creates several different types of files during the backup and prepare phases. See Files Created by Mariabackup for a list of these files.

Known Issues

Unsupported Server Option Groups

Prior to MariaDB 10.1.38, MariaDB 10.2.22, and MariaDB 10.3.13, Mariabackup doesn't read server options from all option groups supported by the server. In those versions, it only looks for server options in the following server option groups:

Group	Description
`[xtrabackup]`	Options read by Percona XtraBackup and Mariabackup.
`[mariabackup]`	Options read by Percona XtraBackup and Mariabackup. Available starting with MariaDB 10.1.31 and MariaDB 10.2.13.
`[mysqld]`	Options read by `mysqld`, which includes both MariaDB Server and MySQL Server.

Those versions do not read server options from the following option groups supported by the server:

Group	Description
`[server]`	Options read by MariaDB Server. Available starting with MariaDB 10.1.38, MariaDB 10.2.22, and MariaDB 10.3.13.
`[mysqld-X.Y]`	Options read by a specific version of `mysqld`, which includes both MariaDB Server and MySQL Server. For example, `[mysqld-5.5]` Available starting with MariaDB 10.1.38, MariaDB 10.2.22, and MariaDB 10.3.13.
`[mariadb]`	Options read by MariaDB Server. Available starting with MariaDB 10.1.38, MariaDB 10.2.22, and MariaDB 10.3.13.
`[mariadb-X.Y]`	Options read by a specific version of MariaDB Server. For example, `[mariadb-10.3]` Available starting with MariaDB 10.1.38, MariaDB 10.2.22, and MariaDB 10.3.13.
`[client-server]`	Options read by all MariaDB client programs and the MariaDB Server. This is useful for options like socket and port, which is common between the server and the clients. Available starting with MariaDB 10.1.38, MariaDB 10.2.22, and MariaDB 10.3.13.
`[galera]`	Options read by MariaDB Server, but only if it is compiled with Galera Cluster support. In MariaDB 10.1 and later, all builds on Linux are compiled with Galera Cluster support. When using one of these builds, options from this option group are read even if the Galera Cluster functionality is not enabled. Available starting with MariaDB 10.1.38, MariaDB 10.2.22, and MariaDB 10.3.13 on systems compiled with Galera Cluster support.

See MDEV-18347 for more information.

No Default Datadir

Prior to MariaDB 10.1.36, MariaDB 10.2.18, and MariaDB 10.3.10, if you were performing a `--copy-back` operation, and if you did not explicitly specify a value for the `datadir` option either on the command line or one of the supported server option groups in an option file, then Mariabackup would not default to the server's default `datadir`. Instead, Mariabackup would fail with an error. For example:

```
Error: datadir must be specified.
```

The solution is to explicitly specify a value for the `datadir` option either on the command line or in one of the supported server option groups in an option file. For example:

```
[mysqld]
datadir=/var/lib/mysql
```

In MariaDB 10.1.36, MariaDB 10.2.18, and MariaDB 10.3.10 and later, Mariabackup will default to the server's default `datadir` value.

See MDEV-12956 for more information.

Concurrent DDL and Backup Issues

Prior to MariaDB 10.2.19 and MariaDB 10.3.10, if concurrent DDL was executed while the backup was taken, then that could cause various kinds of problems to occur.

One example is that if DDL caused any tablespace IDs to change (such as `TRUNCATE TABLE` or `RENAME TABLE`), then that could cause the effected tables to be inconsistent in the backup. In this scenario, you might see errors about mismatched tablespace IDs when the backup is prepared.

For example, the errors might look like this:

```
2018-12-07 07:49:32 7f51b3184820  InnoDB: Error: table 'DB1/TAB_TEMP'
InnoDB: in InnoDB data dictionary has tablespace id 1355633,
InnoDB: but a tablespace with that id does not exist. There is
InnoDB: a tablespace of name DB1/TAB_TEMP and id 1354713, though. Have
InnoDB: you deleted or moved .ibd files?
InnoDB: Please refer to
InnoDB: http://dev.mysql.com/doc/refman/5.6/en/innodb-troubleshooting-datadict.html
InnoDB: for how to resolve the issue.
```

Or they might look like this:

```
2018-07-12 21:24:14 139666981324672 [Note] InnoDB: Ignoring data file 'db1/tab1.ibd' with space
ID 200485, since the redo log references db1/tab1.ibd with space ID 200484.
```

Some of the problems related to concurrent DDL are described below.

Problems solved by setting `--lock-ddl-per-table` (Mariabackup command-line option added in MariaDB 10.2.9):

- If a table is dropped during the backup, then it might still exists after the backup is prepared.
- If a table exists when the backup starts, but it is dropped before the backup copies it, then the tablespace file can't be copied, so the backup would fail.

Problems solved by setting `innodb_log_optimize_ddl=OFF` (MariaDB Server system variable added in MariaDB 10.2.17 and removed in 10.6.0):

- If the backup noticed concurrent DDL, then it might fail with "ALTER TABLE or OPTIMIZE TABLE was executed during backup".

Problems solved by `innodb_safe_truncate=ON` (MariaDB Server system variable in MariaDB 10.2.19 and removed in 10.3.0):

- If a table is created during the backup, then it might not exist in the backup after prepare.
- If a table is renamed during the backup after the tablespace file was copied, then the table may not exist after the backup is prepared.
- If a table is dropped and created under the same name during the backup after the tablespace file was copied, then the table will have the wrong tablespace ID when the backup is prepared.

Note that, with the removal of `innodb_log_optimize_ddl` and `innodb_safe_truncate`, the above problems were definitely solved.

Problems solved by other bug fixes:

- If `--lock-ddl-per-table` is used and if a table is concurrently being dropped or renamed, then Mariabackup can fail to acquire the MDL lock.

These problems are only fixed in MariaDB 10.2 and later, so it is not recommended to execute concurrent DDL when using Mariabackup with MariaDB 10.1.

See MDEV-13563, MDEV-13564, MDEV-16809, and MDEV-16791 for more information.

Manual Restore with Pre-existing InnoDB Redo Log files

Prior to MariaDB 10.2.10, Mariabackup users could run into issues if they restored a backup by manually copying the files from the backup into the `datadir` while the directory still contained pre-existing InnoDB redo log files. The backup itself did not contain InnoDB redo log files with the traditional `ib_logfileN` file names, so the pre-existing log files would remain in the `datadir`. If the server were started with these pre-existing log files, then it could perform crash recovery with them, which could cause the database to become inconsistent or corrupt.

In these MariaDB versions, this problem could be avoided by not restoring the backup by manually copying the files and instead restoring the backup by using Mariabackup and providing the `--copy-back` option, since Mariabackup deletes pre-existing InnoDB redo log files from the `datadir` during the restore process.

In MariaDB 10.2.10 and later, Mariabackup prevents this issue by creating an empty InnoDB redo log file called `ib_logfile0` as part of the `--prepare` stage. That way, if the backup is manually restored, any pre-existing InnoDB redo log files would get overwritten by the empty one.

See MDEV-13311 for more information.

Too Many Open Files

If Mariabackup uses more file descriptors than the system is configured to allow, then users can see errors like the following:

```
2019-02-12 09:48:38 7ffff7fdb820  InnoDB: Operating system error number 23 in a file operation.
InnoDB: Error number 23 means 'Too many open files in system'.
InnoDB: Some operating system error numbers are described at
InnoDB: http://dev.mysql.com/doc/refman/5.6/en/operating-system-error-codes.html
InnoDB: Error: could not open single-table tablespace file ./db1/tab1.ibd
InnoDB: We do not continue the crash recovery, because the table may become
InnoDB: corrupt if we cannot apply the log records in the InnoDB log to it.
InnoDB: To fix the problem and start mysqld:
InnoDB: 1) If there is a permission problem in the file and mysqld cannot
InnoDB: open the file, you should modify the permissions.
InnoDB: 2) If the table is not needed, or you can restore it from a backup,
InnoDB: then you can remove the .ibd file, and InnoDB will do a normal
InnoDB: crash recovery and ignore that table.
InnoDB: 3) If the file system or the disk is broken, and you cannot remove
InnoDB: the .ibd file, you can set innodb_force_recovery > 0 in my.cnf
InnoDB: and force InnoDB to continue crash recovery here.
```

Prior to MariaDB 10.1.39, MariaDB 10.2.24, and MariaDB 10.3.14, Mariabackup would actually ignore the error and continue the backup. In some of those cases, Mariabackup would even report a successful completion of the backup to the user. In later versions, Mariabackup will properly throw an error and abort when this error is encountered. See MDEV-19060 for more information.

When this error is encountered, one solution is to explicitly specify a value for the `open-files-limit` option either on the command line or in one of the supported server option groups in an option file. For example:

```
[mariabackup]
open_files_limit=65535
```

An alternative solution is to set the soft and hard limits for the user account that runs Mariabackup by adding new limits to `/etc/security/limits.conf`. For example, if Mariabackup is run by the `mysql` user, then you could add lines like the following:

```
mysql soft nofile 65535
mysql hard nofile 65535
```

After the system is rebooted, the above configuration should set new open file limits for the `mysql` user, and the user's `ulimit` output should look like the following:

```
$ ulimit -Sn
65535
$ ulimit -Hn
65535
```

Versions

Mariabackup/Server Version	Maturity
MariaDB 10.2.10+, MariaDB 10.1.26+	Stable
MariaDB 10.2.7+, MariaDB 10.1.25	Beta
MariaDB 10.1.23	Alpha

2.3.4.2 Mariabackup Options

Contents

10. --core-file
11. --databases
12. --databases-exclude
13. --databases-file
14. -h, --datadir
15. --debug-sleep-before-unlock
16. --decompress
17. --debug-sync
18. --defaults-extra-file
19. --defaults-file
20. --defaults-group
21. --encrypted-backup
22. --export
23. --extra-lsndir
24. --force-non-empty-directories
25. --ftwrl-wait-query-type
26. --ftwrl-wait-threshold
27. --ftwrl-wait-timeout
28. --galera-info
29. --history
30. -H, --host
31. --include
32. --incremental
33. --incremental-basedir
34. --incremental-dir
35. --incremental-force-scan
36. --incremental-history-name
37. --incremental-history-uuid
38. --incremental-lsn
39. --innobackupex
40. --innodb
41. --innodb-adaptive-hash-index
42. --innodb-autoextend-increment
43. --innodb-buffer-pool-filename
44. --innodb-buffer-pool-size
45. --innodb-checksum-algorithm
46. --innodb-data-file-path
47. --innodb-data-home-dir
48. --innodb-doublewrite
49. --innodb-encrypt-log
50. --innodb-file-io-threads
51. --innodb-file-per-table
52. --innodb-flush-method
53. --innodb-io-capacity
54. --innodb-log-checksums
55. --innodb-log-buffer-size
56. --innodb-log-files-in-group
57. --innodb-log-group-home-dir
58. --innodb-max-dirty-pages-pct
59. --innodb-open-files
60. --innodb-page-size
61. --innodb-read-io-threads
62. --innodb-undo-directory
63. --innodb-undo-tablespaces
64. --innodb-use-native-aio
65. --innodb-write-io-threads
66. --kill-long-queries-timeout
67. --kill-long-query-type
68. --lock-ddl-per-table
69. --log
70. --log-bin
71. --log-copy-interval
72. --log-innodb-page-corruption
73. --move-back
74. --mysqld
75. --no-backup-locks
76. --no-lock

77. --no-timestamp
78. --no-version-check
79. --open-files-limit
80. --parallel
81. -p, --password
82. --plugin-dir
83. --plugin-load
84. -P, --port
85. --prepare
86. --print-defaults
87. --print-param
88. --rollback-xa
89. --rsync
90. --safe-slave-backup
91. --safe-slave-backup-timeout
92. --secure-auth
93. --skip-innodb-adaptive-hash-index
94. --skip-innodb-doublewrite
95. --skip-innodb-log-checksums
96. --skip-secure-auth
97. --slave-info
98. -S, --socket
99. --ssl
100. --ssl-ca
101. --ssl-capath
102. --ssl-cert
103. --ssl-cipher
104. --ssl-crl
105. --ssl-crlpath
106. --ssl-key
107. --ssl-verify-server-cert
108. --stream
109. --tables
110. --tables-exclude
111. --tables-file
112. --target-dir
113. --throttle
114. --tls-version
115. -t, --tmpdir
116. --use-memory
117. --user
118. --version

There are a number of options available in `Mariabackup`.

List of Options

`--apply-log`

Prepares an existing backup to restore to the MariaDB Server. This is only valid in `innobackupex` mode, which can be enabled with the `--innobackupex` option.

Files that Mariabackup generates during `--backup` operations in the target directory are not ready for use on the Server. Before you can restore the data to MariaDB, you first need to prepare the backup.

In the case of full backups, the files are not point in time consistent, since they were taken at different times. If you try to restore the database without first preparing the data, InnoDB rejects the new data as corrupt. Running Mariabackup with the `--prepare` command readies the data so you can restore it to MariaDB Server. When working with incremental backups, you need to use the `--prepare` command and the `--incremental-dir` option to update the base backup with the deltas from an incremental backup.

```
$ mariabackup --innobackupex --apply-log
```

Once the backup is ready, you can use the `--copy-back` or the `--move-back` commands to restore the backup to the server.

--apply-log-only

If this option is used when preparing a backup, then only the redo log apply stage will be performed, and other stages of crash recovery will be ignored. This option is used with incremental backups.

This option is only supported in MariaDB 10.1. In MariaDB 10.2 and later, this option is not needed or supported.

--backup

Backs up your databases.

Using this command option, Mariabackup performs a backup operation on your database or databases. The backups are written to the target directory, as set by the `--target-dir` option.

```
$ mariabackup --backup
      --target-dir /path/to/backup \
      --user user_name --password user_passwd
```

Mariabackup can perform full and incremental backups. A full backup creates a snapshot of the database in the target directory. An incremental backup checks the database against a previously taken full backup, (defined by the `--incremental-basedir` option) and creates delta files for these changes.

In order to restore from a backup, you first need to run Mariabackup with the `--prepare` command option, to make a full backup point-in-time consistent or to apply incremental backup deltas to base. Then you can run Mariabackup again with either the `--copy-back` or `--move-back` commands to restore the database.

For more information, see Full Backup and Restore and Incremental Backup and Restore.

--binlog-info

Defines how Mariabackup retrieves the binary log coordinates from the server.

```
--binlog-info[=OFF | ON | LOCKLESS | AUTO]
```

The `--binlog-info` option supports the following retrieval methods. When no retrieval method is provided, it defaults to `AUTO`.

Option	Description
`OFF`	Disables the retrieval of binary log information
`ON`	Enables the retrieval of binary log information, performs locking where available to ensure consistency
`LOCKLESS`	Unsupported option
`AUTO`	Enables the retrieval of binary log information using `ON` or `LOCKLESS` where supported

Using this option, you can control how Mariabackup retrieves the server's binary log coordinates corresponding to the backup.

When enabled, whether using `ON` or `AUTO`, Mariabackup retrieves information from the binlog during the backup process. When disabled with `OFF`, Mariabackup runs without attempting to retrieve binary log information. You may find this useful when you need to copy data without metadata like the binlog or replication coordinates.

```
$ mariabackup --binlog-info --backup
```

Currently, the `LOCKLESS` option depends on features unsupported by MariaDB Server. See the description of the `xtrabackup_binlog_pos_innodb` file for more information. If you attempt to run Mariabackup with this option, then it causes the utility to exit with an error.

--close-files

Defines whether you want to close file handles.

Using this option, you can tell Mariabackup that you want to close file handles. Without this option, Mariabackup keeps files open in order to manage DDL operations. When working with particularly large tablespaces, closing the file can make the backup more manageable. However, it can also lead to inconsistent backups. Use at your own risk.

```
$ mariabackup --close-files --prepare
```

--compress

This option was deprecated starting with MariaDB 10.1.31 and 10.2.13 as it relies on the no longer maintained QuickLZ library. It is recommended to instead backup to a stream (stdout), and use a 3rd party compression library to compress the stream, as described in Using Encryption and Compression Tools With Mariabackup.

Defines the compression algorithm for backup files.

```
--compress[=compression_algorithm]
```

The `--compress` option only supports the now deprecated `quicklz` algorithm.

Option	Description
`quicklz`	Uses the QuickLZ compression algorithm

```
$ mariabackup --compress --backup
```

If a backup is compressed using this option, then Mariabackup will record that detail in the `xtrabackup_info` file.

--compress-chunk-size

Deprecated, for details see the `--compress` option.

Defines the working buffer size for compression threads.

```
--compress-chunk-size=#
```

Mariabackup can perform compression operations on the backup files before writing them to disk. It can also use multiple threads for parallel data compression during this process. Using this option, you can set the chunk size each thread uses during compression. It defaults to 64K.

```
$ mariabackup --backup --compress \
     --compress-threads=12 --compress-chunk-size=5M
```

To further configure backup compression, see the `--compress` and `--compress-threads` options.

--compress-threads

Deprecated, for details see the `--compress` option.

Defines the number of threads to use in compression.

```
--compress-threads=#
```

Mariabackup can perform compression operations on the backup files before writing them to disk. Using this option, you can define the number of threads you want to use for this operation. You may find this useful in speeding up the compression of particularly large databases. It defaults to single-threaded.

```
$ mariabackup --compress --compress-threads=12 --backup
```

To further configure backup compression, see the `--compress` and `--compress-chunk-size` options.

--copy-back

Restores the backup to the data directory.

Using this command, Mariabackup copies the backup from the target directory to the data directory, as defined by the `--datadir` option. You must stop the MariaDB Server before running this command. The data directory must be empty. If you want to overwrite the data directory with the backup, use the `--force-non-empty-directories` option.

Bear in mind, before you can restore a backup, you first need to run Mariabackup with the `--prepare` option. In the case of full backups, this makes the files point-in-time consistent. With incremental backups, this applies the deltas to the base backup. Once the backup is prepared, you can run `--copy-back` to apply it to MariaDB Server.

```
$ mariabackup --copy-back --force-non-empty-directories
```

Running the `--copy-back` command copies the backup files to the data directory. Use this command if you want to save the backup for later. If you don't want to save the backup for later, use the `--move-back` command.

--core-file

Defines whether to write a core file.

Using this option, you can configure Mariabackup to dump its core to file in the event that it encounters fatal signals. You may find this useful for review and debugging purposes.

```
$ mariabackup --core-file --backup
```

--databases

Defines the databases and tables you want to back up.

```
--databases="database[.table][ database[.table] ...]"
```

Using this option, you can define the specific database or databases you want to back up. In cases where you have a particularly large database or otherwise only want to back up a portion of it, you can optionally also define the tables on the database.

```
$ mariabackup --backup \
      --databases="example.table1 example.table2"
```

In cases where you want to back up most databases on a server or tables on a database, but not all, you can set the specific databases or tables you don't want to back up using the `--databases-exclude` option.

If a backup is a partial backup, then Mariabackup will record that detail in the `xtrabackup_info` file.

In `innobackupex` mode, which can be enabled with the `--innobackupex` option, the `--databases` option can be used as described above, or it can be used to refer to a file, just as the `--databases-file` option can in the normal mode.

--databases-exclude

Defines the databases you don't want to back up.

```
--databases-exclude="database[.table][ database[.table] ...]"
```

Using this option, you can define the specific database or databases you want to exclude from the backup process. You may find it useful when you want to back up most databases on the server or tables on a database, but would like to exclude a few from the process.

```
$ mariabackup --backup \
      --databases="example" \
      --databases-exclude="example.table1 example.table2"
```

To include databases in the backup, see the `--databases` option option

If a backup is a partial backup, then Mariabackup will record that detail in the `xtrabackup_info` file.

--databases-file

Defines the path to a file listing databases and/or tables you want to back up.

```
--databases-file="/path/to/database-file"
```

Format the databases file to list one element per line, with the following syntax:

```
database[.table]
```

In cases where you need to back up a number of databases or specific tables in a database, you may find the syntax for the `--databases` and `--databases-exclude` options a little cumbersome. Using this option you can set the path to a file listing the databases or databases and tables you want to back up.

For instance, imagine you list the databases and tables for a backup in a file called `main-backup`.

```
$ cat main-backup
example1
example2.table1
example2.table2

$ mariabackup --backup --databases-file=main-backup
```

If a backup is a partial backup, then Mariabackup will record that detail in the `xtrabackup_info` file.

-h, --datadir

Defines the path to the database root.

```
--datadir=PATH
```

Using this option, you can define the path to the source directory. This is the directory that Mariabackup reads for the data it backs up. It should be the same as the MariaDB Server `datadir` system variable.

```
$ mariabackup --backup -h /var/lib64/mysql
```

--debug-sleep-before-unlock

This is a debug-only option used by the Xtrabackup test suite.

--decompress

Deprecated, for details see the `--compress` option.

This option requires that you have the `qpress` utility installed on your system.

Defines whether you want to decompress previously compressed backup files.

When you run Mariabackup with the `--compress` option, it compresses the subsequent backup files, using the QuickLZ algorithm. Using this option, Mariabackup decompresses the compressed files from a previous backup.

For instance, run a backup with compression,

```
$ mariabackup --compress --backup
```

Then decompress the backup,

```
$ mariabackup --decompress
```

You can enable the decryption of multiple files at a time using the `--parallel` option. By default, Mariabackup does not remove the compressed files from the target directory. If you want to delete these files, use the `--remove-original` option.

--debug-sync

Defines the debug sync point. This option is only used by the Mariabackup test suite.

--defaults-extra-file

Defines the path to an extra default option file.

```
--defaults-extra-file=/path/to/config
```

Using this option, you can define an extra default option file for Mariabackup. Unlike `--defaults-file`, this file is read after the default option files are read, allowing you to only overwrite the existing defaults.

```
$ mariabackup --backup \
      --defaults-file-extra=addition-config.cnf \
      --defaults-file=config.cnf
```

--defaults-file

Defines the path to the default option file.

```
--defaults-file=/path/to/config
```

Using this option, you can define a default option file for Mariabackup. Unlike the `--defaults-extra-file` option, when this option is provided, it completely replaces all default option files.

```
$ mariabackup --backup \
     --defaults-file="config.cnf
```

--defaults-group

Defines the option group to read in the option file.

```
--defaults-group="name"
```

In situations where you find yourself using certain Mariabackup options consistently every time you call it, you can set the options in an option file. The `--defaults-group` option defines what option group Mariabackup reads for its options.

Options you define from the command-line can be set in the configuration file using minor formatting changes. For instance, if you find yourself perform compression operations frequently, you might set `--compress-threads` and `--compress-chunk-size` options in this way:

```
[mariabackup]
compress_threads = 12
compress_chunk_size = 64K
```

Now whenever you run a backup with the `--compress` option, it always performs the compression using 12 threads and 64K chunks.

```
$ mariabackup --compress --backup
```

See Mariabackup Overview: Server Option Groups and Mariabackup Overview: Client Option Groups for a list of the option groups read by Mariabackup by default.

--encrypted-backup

When this option is used with `--backup`, if Mariabackup encounters a page that has a non-zero `key_version` value, then Mariabackup assumes that the page is encrypted.

Use `--skip-encrypted-backup` instead to allow Mariabackup to copy unencrypted tables that were originally created before MySQL 5.1.48.

This option was added in MariaDB 10.2.22, MariaDB 10.3.13, and MariaDB 10.4.2.

--export

If this option is provided during the `--prepare` stage, then it tells Mariabackup to create `.cfg` files for each InnoDB file-per-table tablespace. These `.cfg` files are used to import transportable tablespaces in the process of restoring partial backups and restoring individual tables and partitions.

The `--export` option could require rolling back incomplete transactions that had modified the table. This will likely create a "new branch of history" that does not correspond to the server that had been backed up, which makes it impossible to apply another incremental backup on top of such additional changes. The option should only be applied when doing a `--prepare` of the last incremental.

```
$ mariabackup --prepare --export
```

MariaDB until 10.2.8

In MariaDB 10.2.8 and before, Mariabackup did not support the `--export` option. See MDEV-13466 about that. In earlier versions of MariaDB, this means that Mariabackup could not create `.cfg` files for InnoDB file-per-table tablespaces during the `--prepare` stage. You can still import file-per-table tablespaces without the `.cfg` files in many cases, so it may still be possible in those versions to restore partial backups or to restore individual tables and partitions with just the `.ibd` files. If you have a full backup and you need to create `.cfg` files for InnoDB file-per-table tablespaces, then you can do so by preparing the backup as usual without the `--export` option, and then restoring the backup, and then starting the server. At that point, you can use the server's built-in features to copy the transportable tablespaces.

--extra-lsndir

Saves an extra copy of the `xtrabackup_checkpoints` and `xtrabackup_info` files into the given directory.

```
--extra-lsndir=PATH
```

When using the `--backup` command option, Mariabackup produces a number of backup files in the target directory. Using this option, you can have Mariabackup produce additional copies of the `xtrabackup_checkpoints` and `xtrabackup_info` files in the given directory.

```
$ mariabackup --extra-lsndir=extras/ --backup
```

This is especially usefull when using `--stream` for streaming output, e.g. for compression and/or encryption using external tools in combination with incremental backups, as the `xtrabackup_checkpoints` file necessary to determine the LSN to continue the incremental backup from is still accessible without uncompressing / decrypting the backup file first. Simply pass in the `--extra-lsndir` of the previous backup as `--incremental-basedir`

--force-non-empty-directories

Allows `--copy-back` or `--move-back` command options to use non-empty target directories.

When using Mariabackup with the `--copy-back` or `--move-back` command options, they normally require a non-empty target directory to avoid conflicts. Using this option with either of command allows Mariabackup to use a non-empty directory.

```
$ mariabackup --force-on-empty-directories --copy-back
```

Bear in mind that this option does not enable overwrites. When copying or moving files into the target directory, if Mariabackup finds that the target file already exists, it fails with an error.

--ftwrl-wait-query-type

Defines the type of query allowed to complete before Mariabackup issues the global lock.

```
--ftwrl-wait-query-type=[ALL | UPDATE | SELECT]
```

The `--ftwrl-wait-query-type` option supports the following query types. The default value is `ALL`.

Option	Description
`ALL`	Waits until all queries complete before issuing the global lock
`SELECT`	Waits until `SELECT` statements complete before issuing the global lock
`UPDATE`	Waits until `UPDATE` statements complete before issuing the global lock

When Mariabackup runs, it issues a global lock to prevent data from changing during the backup process. When it encounters a statement in the process of executing, it waits until the statement is finished before issuing the global lock. Using this option, you can modify this default behavior to ensure that it waits only for certain query types, such as for `SELECT` and `UPDATE` statements.

```
$ mariabackup --backup  \
      --ftwrl-wait-query-type=UPDATE
```

`--ftwrl-wait-threshold`

Defines the minimum threshold for identifying long-running queries for FTWRL.

```
--ftwrl-wait-threshold=#
```

When Mariabackup runs, it issues a global lock to prevent data from changing during the backup process and ensure a consistent record. If it encounters statements still in the process of executing, it waits until they complete before setting the lock. Using this option, you can set the threshold at which Mariabackup engages FTWRL. When it `--ftwrl-wait-timeout` is not 0 and a statement has run for at least the amount of time given this argument, Mariabackup waits until the statement completes or until the `--ftwrl-wait-timeout` expires before setting the global lock and starting the backup.

```
$ mariabackup --backup \
     --ftwrl-wait-timeout=90 \
     --ftwrl-wait-threshold=30
```

`--ftwrl-wait-timeout`

Defines the timeout to wait for queries before trying to acquire the global lock. In MariaDB 10.4 and later, the global lock refers to `BACKUP STAGE BLOCK_COMMIT`. In MariaDB 10.3 and before, the global lock refers to `FLUSH TABLES WITH READ LOCK (FTWRL)`.

```
--ftwrl-wait-timeout=#
```

When Mariabackup runs, it acquires a global lock to prevent data from changing during the backup process and ensure a consistent record. If it encounters statements still in the process of executing, it can be configured to wait until the statements complete before trying to acquire the global lock.

If the `--ftwrl-wait-timeout` is set to 0, then Mariabackup tries to acquire the global lock immediately without waiting. This is the default value.

If the `--ftwrl-wait-timeout` is set to a non-zero value, then Mariabackup waits for the configured number of seconds until trying to acquire the global lock.

Starting in MariaDB 10.5.3, MariaDB 10.4.13, MariaDB 10.3.23, and MariaDB 10.2.32, Mariabackup will exit if it can't acquire the global lock after waiting for the configured number of seconds. In earlier versions, it could wait for the global lock indefinitely, even if `--ftwrl-wait-timeout` was set to a non-zero value.

```
$ mariabackup --backup \
      --ftwrl-wait-query-type=UPDATE \
      --ftwrl-wait-timeout=5
```

`--galera-info`

Defines whether you want to back up information about a Galera Cluster node's state.

When this option is used, Mariabackup creates an additional file called `xtrabackup_galera_info`, which records information about a Galera Cluster node's state. It records the values of the `wsrep_local_state_uuid` and `wsrep_last_committed` status variables.

You should only use this option when backing up a Galera Cluster node. If the server is not a Galera Cluster node, then this

option has no effect.

```
$ mariabackup --backup --galera-info
```

--history

Defines whether you want to track backup history in the `PERCONA_SCHEMA.xtrabackup_history` table.

```
--history[=name]
```

When using this option, Mariabackup records its operation in a table on the MariaDB Server. Passing a name to this option allows you group backups under arbitrary terms for later processing and analysis.

```
$ mariabackup --backup --history=backup_all
```

Currently, the table it uses by default is named `mysql.mariadb_backup_history`. Prior to MariaDB 10.11, the default table was `PERCONA_SCHEMA.xtrabackup_history`.

Mariabackup will also record this in the `xtrabackup_info` file.

-H, --host

Defines the host for the MariaDB Server you want to backup.

```
--host=name
```

Using this option, you can define the host to use when connecting to a MariaDB Server over TCP/IP. By default, Mariabackup attempts to connect to the local host.

```
$ mariabackup --backup \
      --host="example.com"
```

--include

This option is a regular expression to be matched against table names in databasename.tablename format. It is equivalent to the `--tables` option. This is only valid in `innobackupex` mode, which can be enabled with the `--innobackupex` option.

--incremental

Defines whether you want to take an increment backup, based on another backup. This is only valid in `innobackupex` mode, which can be enabled with the `--innobackupex` option.

```
mariabackup --innobackupex --incremental
```

Using this option with the `--backup` command option makes the operation incremental rather than a complete overwrite. When this option is specified, either the `--incremental-lsn` or `--incremental-basedir` options can also be given. If neither option is given, option `--incremental-basedir` is used by default, set to the first timestamped backup directory in the backup base directory.

```
$ mariabackup --innobackupex --backup --incremental \
     --incremental-basedir=/data/backups \
     --target-dir=/data/backups
```

If a backup is a incremental backup, then Mariabackup will record that detail in the `xtrabackup_info` file.

--incremental-basedir

Defines whether you want to take an incremental backup, based on another backup.

```
--incremental-basedir=PATH
```

Using this option with the `--backup` command option makes the operation incremental rather than a complete overwrite. Mariabackup will only copy pages from `.ibd` files if they are newer than the backup in the specified directory.

```
$ mariabackup --backup \
     --incremental-basedir=/data/backups \
     --target-dir=/data/backups
```

If a backup is a incremental backup, then Mariabackup will record that detail in the `xtrabackup_info` file.

`--incremental-dir`

Defines whether you want to take an incremental backup, based on another backup.

```
--increment-dir=PATH
```

Using this option with `--prepare` command option makes the operation incremental rather than a complete overwrite. Mariabackup will apply `.delta` files and log files into the target directory.

```
$ mariabackup --prepare \
      --increment-dir=backups/
```

If a backup is a incremental backup, then Mariabackup will record that detail in the `xtrabackup_info` file.

`--incremental-force-scan`

Defines whether you want to force a full scan for incremental backups.

When using Mariabackup to perform an incremental backup, this option forces it to also perform a full scan of the data pages being backed up, even when there's bitmap data on the changes. MariaDB 10.2 and later does not support changed page bitmaps, so this option is useless in those versions. See MDEV-18985 for more information.

```
$ mariabackup --backup \
     --incremental-basedir=/path/to/target \
     --incremental-force-scan
```

`--incremental-history-name`

Defines a logical name for the backup.

```
--incremental-history-name=name
```

Mariabackup can store data about its operations on the MariaDB Server. Using this option, you can define the logical name it uses in identifying the backup.

```
$ mariabackup --backup \
     --incremental-history-name=morning_backup
```

Currently, the table it uses by default is named `mysql.mariadb_backup_history`. Prior to MariaDB 10.11, the default table was `PERCONA_SCHEMA.xtrabackup_history`.

Mariabackup will also record this in the `xtrabackup_info` file.

`--incremental-history-uuid`

Defines a UUID for the backup.

```
--incremental-history-uuid=name
```

Mariabackup can store data about its operations on the MariaDB Server. Using this option, you can define the UUID it uses in identifying a previous backup to increment from. It checks `--incremental-history-name`, `--incremental-basedir`, and `--incremental-lsn`. If Mariabackup fails to find a valid lsn, it generates an error.

```
$ mariabackup --backup \
      --incremental-history-uuid=main-backup012345678
```

Currently, the table it uses is named `PERCONA_SCHEMA.xtrabackup_history`, but expect that name to change in future releases. See MDEV-19246 for more information.

Mariabackup will also record this in the `xtrabackup_info` file.

--incremental-lsn

Defines the sequence number for incremental backups.

```
--incremental-lsn=name
```

Using this option, you can define the sequence number (LSN) value for `--backup` operations. During backups, Mariabackup only copies `.ibd` pages newer than the specified values.

WARNING: Incorrect LSN values can make the backup unusable. It is impossible to diagnose this issue.

--innobackupex

> Deprecated in MariaDB 10.3.0.

Enables `innobackupex` mode, which is a compatibility mode.

```
$ mariabackup --innobackupex
```

In `innobackupex` mode, Mariabackup has the following differences:

- To prepare a backup, the `--apply-log` option is used instead of the `--prepare` option.
- To create an incremental backup, the `--incremental` option is supported.
- The `--no-timestamp` option is supported.
- To create a partial backup, the `--include` option is used instead of the `--tables` option.
- To create a partial backup, the `--databases` option can still be used, but it's behavior changes slightly.
- The `--target-dir` option is not used to specify the backup directory. The backup directory should instead be specified as a standalone argument.

The primary purpose of `innobackupex` mode is to allow scripts and tools to more easily migrate to Mariabackup if they were originally designed to use the `innobackupex` utility that is included with Percona XtraBackup. It is not recommended to use this mode in new scripts, since it is not guaranteed to be supported forever. See MDEV-20552 for more information.

--innodb

This option has no effect. Set only for MySQL option compatibility.

--innodb-adaptive-hash-index

Enables InnoDB Adaptive Hash Index.

Mariabackup initializes its own embedded instance of InnoDB using the same configuration as defined in the configuration file. Using this option you can explicitly enable the InnoDB Adaptive Hash Index. This feature is enabled by default for Mariabackup. If you want to disable it, use `--skip-innodb-adaptive-hash-index`.

```
$ mariabackup --backup \
      --innodb-adaptive-hash-index
```

--innodb-autoextend-increment

Defines the increment in megabytes for auto-extending the size of tablespace file.

```
--innodb-autoextend-increment=36
```

Mariabackup initializes its own embedded instance of InnoDB using the same configuration as defined in the configuration file. Using this option, you can set the increment in megabytes for automatically extending the size of tablespace data file in InnoDB.

```
$ mariabackup --backup \
     --innodb-autoextend-increment=35
```

--innodb-buffer-pool-filename

Using this option has no effect. It is available to provide compatibility with the MariaDB Server.

--innodb-buffer-pool-size

Defines the memory buffer size InnoDB uses the cache data and indexes of the table.

```
--innodb-buffer-pool-size=124M
```

Mariabackup initializes its own embedded instance of InnoDB using the same configuration as defined in the configuration file. Using this option, you can configure the buffer pool for InnoDB operations.

```
$ mariabackup --backup \
      --innodb-buffer-pool-size=124M
```

--innodb-checksum-algorithm

innodb_checksum_algorithm was deprecated in MariaDB 10.3.29, MariaDB 10.4.19, MariaDB 10.5.10 and removed in MariaDB 10.6.

In earlier versions, it is used to define the checksum algorithm.

```
--innodb-checksum-algorithm=crc32
                           | strict_crc32
                           | innodb
                           | strict_innodb
                           | none
                           | strict_none
```

Mariabackup initializes its own embedded instance of InnoDB using the same configuration as defined in the configuration file. Using this option, you can specify the algorithm Mariabackup uses when checksumming on InnoDB tables. Currently, MariaDB supports the following algorithms `CRC32`, `STRICT_CRC32`, `INNODB`, `STRICT_INNODB`, `NONE`, `STRICT_NONE`.

```
$ mariabackup --backup \
      ---innodb-checksum-algorithm=strict_innodb
```

--innodb-data-file-path

Defines the path to individual data files.

```
--innodb-data-file-path=/path/to/file
```

Mariabackup initializes its own embedded instance of InnoDB using the same configuration as defined in the configuration file. Using this option you can define the path to InnoDB data files. Each path is appended to the `--innodb-data-home-dir` option.

```
$ mariabackup --backup \
     --innodb-data-file-path=ibdata1:13M:autoextend \
     --innodb-data-home-dir=/var/dbs/mysql/data
```

--innodb-data-home-dir

Defines the home directory for InnoDB data files.

```
--innodb-data-home-dir=PATH
```

Mariabackup initializes its own embedded instance of InnoDB using the same configuration as defined in the configuration file. Using this option you can define the path to the directory containing InnoDB data files. You can specific the files using the `--innodb-data-file-path` option.

```
$ mariabackup --backup \
     --innodb-data-file-path=ibdata1:13M:autoextend \
     --innodb-data-home-dir=/var/dbs/mysql/data
```

--innodb-doublewrite

Enables doublewrites for InnoDB tables.

Mariabackup initializes its own embedded instance of InnoDB using the same configuration as defined in the configuration file. When using this option, Mariabackup improves fault tolerance on InnoDB tables with a doublewrite buffer. By default, this feature is enabled. Use this option to explicitly enable it. To disable doublewrites, use the `--skip-innodb-doublewrite` option.

```
$ mariabackup --backup \
     --innodb-doublewrite
```

--innodb-encrypt-log

Defines whether you want to encrypt InnoDB logs.

Mariabackup initializes its own embedded instance of InnoDB using the same configuration as defined in the configuration file. Using this option, you can tell Mariabackup that you want to encrypt logs from its InnoDB activity.

--innodb-file-io-threads

Defines the number of file I/O threads in InnoDB.

```
--innodb-file-io-threads=#
```

Mariabackup initializes its own embedded instance of InnoDB using the same configuration as defined in the configuration file. Using this option, you can define the number of file I/O threads Mariabackup uses on InnoDB tables.

```
$ mariabackup --backup \
     --innodb-file-io-threads=5
```

--innodb-file-per-table

Defines whether you want to store each InnoDB table as an `.ibd` file.

Mariabackup initializes its own embedded instance of InnoDB using the same configuration as defined in the configuration file. Using this option causes Mariabackup to store each InnoDB table as an `.ibd` file in the target directory.

--innodb-flush-method

Defines the data flush method. Ignored from MariaDB 11.0.

```
--innodb-flush-method=fdatasync
                     | O_DSYNC
                     | O_DIRECT
                     | O_DIRECT_NO_FSYNC
                     | ALL_O_DIRECT
```

Mariabackup initializes its own embedded instance of InnoDB using the same configuration as defined in the configuration file. Using this option, you can define the data flush method Mariabackup uses with InnoDB tables.

```
$ mariabackup --backup \
      --innodb-flush-method==_DIRECT_NO_FSYNC
```

Note, the `O_DIRECT_NO_FSYNC` method is only available with MariaDB 10.0 and later. The `ALL_O_DIRECT` method available with version 5.5 and later, but only with tables using the XtraDB storage engine.

--innodb-io-capacity

Defines the number of IOP's the utility can perform.

```
--innodb-io-capacity=#
```

Mariabackup initializes its own embedded instance of InnoDB using the same configuration as defined in the configuration file. Using this option, you can limit the I/O activity for InnoDB background tasks. It should be set around the number of I/O operations per second that the system can handle, based on drive or drives being used.

```
$ mariabackup --backup \
     --innodb-io-capacity=200
```

--innodb-log-checksums

Defines whether to include checksums in the InnoDB logs.

Mariabackup initializes its own embedded instance of InnoDB using the same configuration as defined in the configuration file. Using this option, you can explicitly set Mariabackup to include checksums in the InnoDB logs. The feature is enabled by default. To disable it, use the `--skip-innodb-log-checksums` option.

```
$ mariabackup --backup \
      --innodb-log-checksums
```

--innodb-log-buffer-size

This option has no functionality in Mariabackup. It exists for MariaDB Server compatibility.

--innodb-log-files-in-group

This option has no functionality in Mariabackup. It exists for MariaDB Server compatibility.

--innodb-log-group-home-dir

Defines the path to InnoDB log files.

```
--innodb-log-group-home-dir=PATH
```

Mariabackup initializes its own embedded instance of InnoDB using the same configuration as defined in the configuration file. Using this option, you can define the path to InnoDB log files.

```
$ mariabackup --backup \
     --innodb-log-group-home-dir=/path/to/logs
```

--innodb-max-dirty-pages-pct

Defines the percentage of dirty pages allowed in the InnoDB buffer pool.

```
--innodb-max-dirty-pages-pct=#
```

Mariabackup initializes its own embedded instance of InnoDB using the same configuration as defined in the configuration file. Using this option, you can define the maximum percentage of dirty, (that is, unwritten) pages that Mariabackup allows in the InnoDB buffer pool.

```
$ mariabackup --backup \
     --innodb-max-dirty-pages-pct=80
```

--innodb-open-files

Defines the number of files kept open at a time.

```
--innodb-open-files=#
```

Mariabackup initializes its own embedded instance of InnoDB using the same configuration as defined in the configuration file. Using this option, you can set the maximum number of files InnoDB keeps open at a given time during backups.

```
$ mariabackup --backup \
      --innodb-open-files=10
```

--innodb-page-size

Defines the universal page size.

```
--innodb-page-size=#
```

Mariabackup initializes its own embedded instance of InnoDB using the same configuration as defined in the configuration file. Using this option, you can define the universal page size in bytes for Mariabackup.

```
$ mariabackup --backup \
     --innodb-page-size=16k
```

--innodb-read-io-threads

Defines the number of background read I/O threads in InnoDB.

```
--innodb-read-io-threads=#
```

Mariabackup initializes its own embedded instance of InnoDB using the same configuration as defined in the configuration file. Using this option, you can set the number of I/O threads MariaDB uses when reading from InnoDB.

```
$ mariabackup --backup \
      --innodb-read-io-threads=4
```

--innodb-undo-directory

Defines the directory for the undo tablespace files.

```
--innodb-undo-directory=PATH
```

Mariabackup initializes its own embedded instance of InnoDB using the same configuration as defined in the configuration file. Using this option, you can define the path to the directory where you want MariaDB to store the undo tablespace on InnoDB tables. The path can be absolute.

```
$ mariabackup --backup \
     --innodb-undo-directory=/path/to/innodb_undo
```

--innodb-undo-tablespaces

Defines the number of undo tablespaces to use.

```
--innodb-undo-tablespaces=#
```

Mariabackup initializes its own embedded instance of InnoDB using the same configuration as defined in the configuration file. Using this option, you can define the number of undo tablespaces you want to use during the backup.

```
$ mariabackup --backup \
      --innodb-undo-tablespaces=10
```

--innodb-use-native-aio

Defines whether you want to use native AI/O.

Mariabackup initializes its own embedded instance of InnoDB using the same configuration as defined in the configuration file. Using this option, you can enable the use of the native asynchronous I/O subsystem. It is only available on Linux operating systems.

```
$ mariabackup --backup \
      --innodb-use-native-aio
```

--innodb-write-io-threads

Defines the number of background write I/O threads in InnoDB.

```
--innodb-write-io-threads=#
```

Mariabackup initializes its own embedded instance of InnoDB using the same configuration as defined in the configuration file. Using this option, you can set the number of background write I/O threads Mariabackup uses.

```
$ mariabackup --backup \
     --innodb-write-io-threads=4
```

--kill-long-queries-timeout

Defines the timeout for blocking queries.

```
--kill-long-queries-timeout=#
```

When Mariabackup runs, it issues a `FLUSH TABLES WITH READ LOCK` statement. It then identifies blocking queries. Using this option you can set a timeout in seconds for these blocking queries. When the time runs out, Mariabackup kills the queries.

The default value is 0, which causes Mariabackup to not attempt killing any queries.

```
$ mariabackup --backup \
      --kill-long-queries-timeout=10
```

--kill-long-query-type

Defines the query type the utility can kill to unblock the global lock.

```
--kill-long-query-type=ALL | UPDATE | SELECT
```

When Mariabackup encounters a query that sets a global lock, it can kill the query in order to free up MariaDB Server for the backup. Using this option, you can choose the types of query it kills: `SELECT`, `UPDATE`, or both set with `ALL`. The default is `ALL`.

```
$ mariabackup --backup \
      --kill-long-query-type=UPDATE
```

--lock-ddl-per-table

Prevents DDL for each table to be backed up by acquiring MDL lock on that. NOTE: Unless --no-lock option was also specified, conflicting DDL queries , will be killed at the end of backup This is done avoid deadlock between "FLUSH TABLE WITH READ LOCK", user's DDL query (ALTER, RENAME), and MDL lock on table. Only available in MariaDB 10.2.9 and later.

--log

This option has no functionality. It is set to ensure compatibility with MySQL.

--log-bin

Defines the base name for the log sequence.

```
--log-bin[=name]
```

Using this option you, you can set the base name for Mariabackup to use in log sequences.

--log-copy-interval

Defines the copy interval between checks done by the log copying thread.

```
--log-copy-interval=#
```

Using this option, you can define the copy interval Mariabackup uses between checks done by the log copying thread. The given value is in milliseconds.

```
$ mariabackup --backup \
      --log-copy-interval=50
```

--log-innodb-page-corruption

Continue backup if InnoDB corrupted pages are found. The pages are logged in `innodb_corrupted_pages` and backup is finished with error. --prepare will try to fix corrupted pages. If `innodb_corrupted_pages` exists after --prepare in base backup directory, backup still contains corrupted pages and can not be considered as consistent.

Added in MariaDB 10.2.37, MariaDB 10.3.28, MariaDB 10.4.18, MariaDB 10.5.9

--move-back

Restores the backup to the data directory.

Using this command, Mariabackup moves the backup from the target directory to the data directory, as defined by the `--datadir` option. You must stop the MariaDB Server before running this command. The data directory must be empty. If you want to overwrite the data directory with the backup, use the `--force-non-empty-directories` option.

Bear in mind, before you can restore a backup, you first need to run Mariabackup with the `--prepare` option. In the case of full backups, this makes the files point-in-time consistent. With incremental backups, this applies the deltas to the base backup. Once the backup is prepared, you can run `--move-back` to apply it to MariaDB Server.

```
$ mariabackup --move-back \
      --datadir=/var/mysql
```

Running the `--move-back` command moves the backup files to the data directory. Use this command if you don't want to save the backup for later. If you do want to save the backup for later, use the `--copy-back` command.

--mysqld

Used internally to prepare a backup.

--no-backup-locks

Mariabackup locks the database by default when it runs. This option disables support for Percona Server's backup locks.

When backing up Percona Server, Mariabackup would use backup locks by default. To be specific, backup locks refers to the `LOCK TABLES FOR BACKUP` and `LOCK BINLOG FOR BACKUP` statements. This option can be used to disable support for Percona Server's backup locks. This option has no effect when the server does not support Percona's backup locks.

This option may eventually be removed. See MDEV-19753 for more information.

```
$ mariabackup --backup --no-backup-locks
```

--no-lock

Disables table locks with the `FLUSH TABLE WITH READ LOCK` statement.

Using this option causes Mariabackup to disable table locks with the `FLUSH TABLE WITH READ LOCK` statement. Only use this option if:

- You are not executing DML statements on non-InnoDB tables during the backup. This includes the `mysql` database system tables (which are MyISAM).
- You are not executing any DDL statements during the backup.
- You are _not_ using the file "xtrabackup_binlog_info", which is not consistent with the data when --no-lock is used.

Use the file "xtrabackup_binlog_pos_innodb" [link] instead.
- **All** tables you're backing up use the InnoDB storage engine.

```
$ mariabackup --backup --no-lock
```

If you're considering `--no-lock` due to backups failing to acquire locks, this may be due to incoming replication events preventing the lock. Consider using the --safe-slave-backup option to momentarily stop the replica thread. This alternative may help the backup to succeed without resorting to `--no-lock`.

The --no-lock option only provides a consistent backup if the user ensures that no DDL or non-transactional table updates occur during the backup. The --no-lock option is not supported by MariaDB plc.

--no-timestamp

This option prevents creation of a time-stamped subdirectory of the BACKUP-ROOT-DIR given on the command line. When it is specified, the backup is done in BACKUP-ROOT-DIR instead. This is only valid in `innobackupex` mode, which can be enabled with the `--innobackupex` option.

--no-version-check

Disables version check.

Using this option, you can disable Mariabackup version check.

```
$ mariabackup --backup --no-version-check
```

--open-files-limit

Defines the maximum number of file descriptors.

```
--open-files-limit=#
```

Using this option, you can define the maximum number of file descriptors Mariabackup reserves with `setrlimit()`.

```
$ mariabackup --backup \
      --open-files-limit=
```

--parallel

Defines the number of threads to use for parallel data file transfer.

```
--parallel=#
```

Using this option, you can set the number of threads Mariabackup uses for parallel data file transfers. By default, it is set to 1.

-p, --password

Defines the password to use to connect to MariaDB Server.

```
--password=passwd
```

When you run Mariabackup, it connects to MariaDB Server in order to access and back up the databases and tables. Using this option, you can set the password Mariabackup uses to access the server. To set the user, use the `--user` option.

```
$ mariabackup --backup \
      --user=root \
      --password=root_password
```

--plugin-dir

Defines the directory for server plugins.

```
--plugin-dir=PATH
```

Using this option, you can define the path Mariabackup reads for MariaDB Server plugins. It only uses it during the `--prepare` phase to load the encryption plugin. It defaults to the `plugin_dir` server system variable.

```
$ mariabackup --backup \
      --plugin-dir=/var/mysql/lib/plugin
```

--plugin-load

Defines the encryption plugins to load.

```
--plugin-load=name
```

Using this option, you can define the encryption plugin you want to load. It is only used during the `--prepare` phase to load the encryption plugin. It defaults to the server `--plugin-load` option.

The option was removed starting from MariaDB 10.2.18

-P, --port

Defines the server port to connect to.

```
--port=#
```

When you run Mariabackup, it connects to MariaDB Server in order to access and back up your databases and tables. Using this option, you can set the port the utility uses to access the server over TCP/IP. To set the host, see the `--host` option. Use `mysql --help` for more details.

```
$ mariabackup --backup \
      --host=192.168.11.1 \
      --port=3306
```

--prepare

Prepares an existing backup to restore to the MariaDB Server.

Files that Mariabackup generates during `--backup` operations in the target directory are not ready for use on the Server. Before you can restore the data to MariaDB, you first need to prepare the backup.

In the case of full backups, the files are not point in time consistent, since they were taken at different times. If you try to restore the database without first preparing the data, InnoDB rejects the new data as corrupt. Running Mariabackup with the `--prepare` command readies the data so you can restore it to MariaDB Server. When working with incremental backups, you need to use the `--prepare` command and the `--incremental-dir` option to update the base backup with the deltas from an incremental backup.

```
$ mariabackup --prepare
```

Once the backup is ready, you can use the `--copy-back` or the `--move-back` commands to restore the backup to the server.

--print-defaults

Prints the utility argument list, then exits.

Using this argument, MariaDB prints the argument list to stdout and then exits. You may find this useful in debugging to see how the options are set for the utility.

```
$ mariabackup --print-defaults
```

--print-param

Prints the MariaDB Server options needed for copyback.

Using this option, Mariabackup prints to stdout the MariaDB Server options that the utility requires to run the `--copy-back` command option.

```
$ mariabackup --print-param
```

--rollback-xa

By default, Mariabackup will not commit or rollback uncommitted XA transactions, and when the backup is restored, any uncommitted XA transactions must be manually committed using `XA COMMIT` or manually rolled back using `XA ROLLBACK`.

In MariaDB 10.2, MariaDB 10.3, and MariaDB 10.4, Mariabackup's `--rollback-xa` option can be used to rollback uncommitted XA transactions while performing a `--prepare` operation, so that they do not need to be manually committed or rolled back when the backup is restored.

This option is not present from MariaDB 10.5, because the server has more robust ways of handling uncommitted XA transactions in later versions.

> This is an experimental option. Do not use this option in versions older than MariaDB 10.2.33, MariaDB 10.3.24, and MariaDB 10.4.14. Older implementation can cause corruption of InnoDB data.

--rsync

Defines whether to use rsync.

During normal operation, Mariabackup transfers local non-InnoDB files using a separate call to `cp` for each file. Using this option, you can optimize this process by performing this transfer with rsync, instead.

```
$ mariabackup --backup --rsync
```

This option is not compatible with the `--stream` option.

--safe-slave-backup

Stops replica SQL threads for backups.

When running Mariabackup on a server that uses replication, you may occasionally encounter locks that block backups. Using this option, it stops replica SQL threads and waits until the `Slave_open_temp_tables` in the `SHOW STATUS` statement is zero. If there are no open temporary tables, the backup runs, otherwise the SQL thread starts and stops until there are no open temporary tables.

```
$ mariabackup --backup \
      --safe-slave-backup \
      --safe-slave-backup-timeout=500
```

The backup fails if the `Slave_open_temp_tables` doesn't reach zero after the timeout period set by the --safe-slave-backup-timeout option.

--safe-slave-backup-timeout

Defines the timeout for replica backups.

```
--safe-slave-backup-timeout=#
```

When running Mariabackup on a server that uses replication, you may occasionally encounter locks that block backups. With the --safe-slave-backup option, it waits until the `Slave_open_temp_tables` in the `SHOW STATUS` statement reaches zero. Using this option, you set how long it waits. It defaults to 300.

```
$ mariabackup --backup \
      --safe-slave-backup \
      --safe-slave-backup-timeout=500
```

--secure-auth

Refuses client connections to servers using the older protocol.

Using this option, you can set it explicitly to refuse client connections to the server when using the older protocol, from before 4.1.1. This feature is enabled by default. Use the --skip-secure-auth option to disable it.

```
$ mariabackup --backup --secure-auth
```

--skip-innodb-adaptive-hash-index

Disables InnoDB Adaptive Hash Index.

Mariabackup initializes its own embedded instance of InnoDB using the same configuration as defined in the configuration file. Using this option you can explicitly disable the InnoDB Adaptive Hash Index. This feature is enabled by default for Mariabackup. If you want to explicitly enable it, use `--innodb-adaptive-hash-index`.

```
$ mariabackup --backup \
      --skip-innodb-adaptive-hash-index
```

--skip-innodb-doublewrite

Disables doublewrites for InnoDB tables.

Mariabackup initializes its own embedded instance of InnoDB using the same configuration as defined in the configuration file. When doublewrites are enabled, InnoDB improves fault tolerance with a doublewrite buffer. By default this feature is turned on. Using this option you can disable it for Mariabackup. To explicitly enable doublewrites, use the `--innodb-doublewrite` option.

```
$ mariabackup --backup \
     --skip-innodb-doublewrite
```

--skip-innodb-log-checksums

Defines whether to exclude checksums in the InnoDB logs.

Mariabackup initializes its own embedded instance of InnoDB using the same configuration as defined in the configuration file. Using this option, you can set Mariabackup to exclude checksums in the InnoDB logs. The feature is enabled by default. To explicitly enable it, use the `--innodb-log-checksums` option.

--skip-secure-auth

Refuses client connections to servers using the older protocol.

Using this option, you can set it accept client connections to the server when using the older protocol, from before 4.1.1. By default, it refuses these connections. Use the `--secure-auth` option to explicitly enable it.

```
$ mariabackup --backup --skip-secure-auth
```

--slave-info

Prints the binary log position and the name of the primary server.

If the server is a replica, then this option causes Mariabackup to print the hostname of the replica's replication primary and the binary log file and position of the replica's SQL thread to `stdout`.

This option also causes Mariabackup to record this information as a CHANGE MASTER command that can be used to set up a new server as a replica of the original server's primary after the backup has been restored. This information will be written to to the xtrabackup_slave_info file.

Mariabackup does **not** check if GTIDs are being used in replication. It takes a shortcut and assumes that if the gtid_slave_pos system variable is non-empty, then it writes the CHANGE MASTER command with the MASTER_USE_GTID option set to `slave_pos`. Otherwise, it writes the CHANGE MASTER command with the MASTER_LOG_FILE and MASTER_LOG_POS options using the primary's binary log file and position. See MDEV-19264 for more information.

```
$ mariabackup --slave-info
```

-S, --socket

Defines the socket for connecting to local database.

```
--socket=name
```

Using this option, you can define the UNIX domain socket you want to use when connecting to a local database server. The option accepts a string argument. For more information, see the `mysql --help` command.

```
$ mariabackup --backup \
      --socket=/var/mysql/mysql.sock
```

--ssl

Enables TLS. By using this option, you can explicitly configure Mariabackup to to encrypt its connection with TLS when communicating with the server. You may find this useful when performing backups in environments where security is extra important or when operating over an insecure network.

TLS is also enabled even without setting this option when certain other TLS options are set. For example, see the descriptions of the following options:

- `--ssl-ca`
- `--ssl-capath`
- `--ssl-cert`
- `--ssl-cipher`
- `--ssl-key`

--ssl-ca

Defines a path to a PEM file that should contain one or more X509 certificates for trusted Certificate Authorities (CAs) to use for TLS. This option requires that you use the absolute path, not a relative path. For example:

```
--ssl-ca=/etc/my.cnf.d/certificates/ca.pem
```

This option is usually used with other TLS options. For example:

```
$ mariabackup --backup \
   --ssl-cert=/etc/my.cnf.d/certificates/client-cert.pem \
   --ssl-key=/etc/my.cnf.d/certificates/client-key.pem \
   --ssl-ca=/etc/my.cnf.d/certificates/ca.pem
```

See Secure Connections Overview: Certificate Authorities (CAs) for more information.

This option implies the --ssl option.

--ssl-capath

Defines a path to a directory that contains one or more PEM files that should each contain one X509 certificate for a trusted Certificate Authority (CA) to use for TLS. This option requires that you use the absolute path, not a relative path. For example:

```
--ssl-capath=/etc/my.cnf.d/certificates/ca/
```

This option is usually used with other TLS options. For example:

```
$ mariabackup --backup \
   --ssl-cert=/etc/my.cnf.d/certificates/client-cert.pem \
   --ssl-key=/etc/my.cnf.d/certificates/client-key.pem \
   --ssl-ca=/etc/my.cnf.d/certificates/ca.pem \
   --ssl-capath=/etc/my.cnf.d/certificates/ca/
```

The directory specified by this option needs to be run through the openssl rehash command.

See Secure Connections Overview: Certificate Authorities (CAs) for more information

This option implies the --ssl option.

--ssl-cert

Defines a path to the X509 certificate file to use for TLS. This option requires that you use the absolute path, not a relative path. For example:

```
--ssl-cert=/etc/my.cnf.d/certificates/client-cert.pem
```

This option is usually used with other TLS options. For example:

```
$ mariabackup --backup \
   --ssl-cert=/etc/my.cnf.d/certificates/client-cert.pem \
   --ssl-key=/etc/my.cnf.d/certificates/client-key.pem \
   --ssl-ca=/etc/my.cnf.d/certificates/ca.pem
```

This option implies the --ssl option.

--ssl-cipher

Defines the list of permitted ciphers or cipher suites to use for TLS. For example:

```
--ssl-cipher=name
```

This option is usually used with other TLS options. For example:

```
$ mariabackup --backup \
   --ssl-cert=/etc/my.cnf.d/certificates/client-cert.pem \
   --ssl-key=/etc/my.cnf.d/certificates/client-key.pem \
   --ssl-ca=/etc/my.cnf.d/certificates/ca.pem
   --ssl-cipher=TLSv1.2
```

To determine if the server restricts clients to specific ciphers, check the ssl_cipher system variable.

This option implies the --ssl option.

--ssl-crl

Defines a path to a PEM file that should contain one or more revoked X509 certificates to use for TLS. This option requires that you use the absolute path, not a relative path. For example:

```
--ssl-crl=/etc/my.cnf.d/certificates/crl.pem
```

This option is usually used with other TLS options. For example:

```
$ mariabackup --backup \
   --ssl-cert=/etc/my.cnf.d/certificates/client-cert.pem \
   --ssl-key=/etc/my.cnf.d/certificates/client-key.pem \
   --ssl-ca=/etc/my.cnf.d/certificates/ca.pem \
   --ssl-crl=/etc/my.cnf.d/certificates/crl.pem
```

See Secure Connections Overview: Certificate Revocation Lists (CRLs) for more information.

This option is only supported if Mariabackup was built with OpenSSL. If Mariabackup was built with yaSSL, then this option is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.

--ssl-crlpath

Defines a path to a directory that contains one or more PEM files that should each contain one revoked X509 certificate to use for TLS. This option requires that you use the absolute path, not a relative path. For example:

```
--ssl-crlpath=/etc/my.cnf.d/certificates/crl/
```

This option is usually used with other TLS options. For example:

```
$ mariabackup --backup \
   --ssl-cert=/etc/my.cnf.d/certificates/client-cert.pem \
   --ssl-key=/etc/my.cnf.d/certificates/client-key.pem \
   --ssl-ca=/etc/my.cnf.d/certificates/ca.pem \
   --ssl-crlpath=/etc/my.cnf.d/certificates/crl/
```

The directory specified by this option needs to be run through the openssl rehash command.

See Secure Connections Overview: Certificate Revocation Lists (CRLs) for more information.

This option is only supported if Mariabackup was built with OpenSSL. If Mariabackup was built with yaSSL, then this option is not supported. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.

--ssl-key

Defines a path to a private key file to use for TLS. This option requires that you use the absolute path, not a relative path. For example:

```
--ssl-key=/etc/my.cnf.d/certificates/client-key.pem
```

This option is usually used with other TLS options. For example:

```
$ mariabackup --backup \
   --ssl-cert=/etc/my.cnf.d/certificates/client-cert.pem \
   --ssl-key=/etc/my.cnf.d/certificates/client-key.pem \
   --ssl-ca=/etc/my.cnf.d/certificates/ca.pem
```

This option implies the --ssl option.

--ssl-verify-server-cert

Enables server certificate verification. This option is disabled by default.

This option is usually used with other TLS options. For example:

```
$ mariabackup --backup \
   --ssl-cert=/etc/my.cnf.d/certificates/client-cert.pem \
   --ssl-key=/etc/my.cnf.d/certificates/client-key.pem \
   --ssl-ca=/etc/my.cnf.d/certificates/ca.pem \
   --ssl-verify-server-cert
```

--stream

Streams backup files to stdout.

```
--stream=xbstream
```

Using this command option, you can set Mariabackup to stream the backup files to stdout in the given format. Currently, the supported format is `xbstream`.

```
$ mariabackup --stream=xbstream > backup.xb
```

To extract all files from the xbstream archive into a directory use the `mbstream` utility

```
$ mbstream  -x < backup.xb
```

If a backup is streamed, then Mariabackup will record the format in the xtrabackup_info file.

--tables

Defines the tables you want to include in the backup.

```
--tables=REGEX
```

Using this option, you can define what tables you want Mariabackup to back up from the database. The table values are defined using Regular Expressions. To define the tables you want to exclude from the backup, see the --tables-exclude option.

```
$ mariabackup --backup \
     --databases=example
     --tables=nodes_* \
     --tables-exclude=nodes_tmp
```

If a backup is a partial backup, then Mariabackup will record that detail in the xtrabackup_info file.

--tables-exclude

Defines the tables you want to exclude from the backup.

```
--tables-exclude=REGEX
```

Using this option, you can define what tables you want Mariabackup to exclude from the backup. The table values are defined using Regular Expressions. To define the tables you want to include from the backup, see the --tables option.

```
$ mariabackup --backup \
     --databases=example
     --tables=nodes_* \
     --tables-exclude=nodes_tmp
```

If a backup is a partial backup, then Mariabackup will record that detail in the xtrabackup_info file.

--tables-file

Defines path to file with tables for backups.

```
--tables-file=/path/to/file
```

Using this option, you can set a path to a file listing the tables you want to back up. Mariabackup iterates over each line in the file. The format is `database.table`.

```
$ mariabackup --backup \
     --databases=example \
     --tables-file=/etc/mysql/backup-file
```

If a backup is a partial backup, then Mariabackup will record that detail in the xtrabackup_info file.

--target-dir

Defines the destination directory.

```
--target-dir=/path/to/target
```

Using this option you can define the destination directory for the backup. Mariabackup writes all backup files to this directory. Mariabackup will create the directory, if it does not exist (but it will not create the full path recursively, i.e. at least parent directory if the --target-dir must exist=

```
$ mariabackup --backup \
       --target-dir=/data/backups
```

--throttle

Defines the limit for I/O operations per second in IOS values.

```
--throttle=#
```

Using this option, you can set a limit on the I/O operations Mariabackup performs per second in IOS values. It is only used during the --backup command option.

--tls-version

This option accepts a comma-separated list of TLS protocol versions. A TLS protocol version will only be enabled if it is present in this list. All other TLS protocol versions will not be permitted. For example:

```
--tls-version="TLSv1.2,TLSv1.3"
```

This option is usually used with other TLS options. For example:

```
$ mariabackup --backup \
   --ssl-cert=/etc/my.cnf.d/certificates/client-cert.pem \
   --ssl-key=/etc/my.cnf.d/certificates/client-key.pem \
   --ssl-ca=/etc/my.cnf.d/certificates/ca.pem \
   --tls-version="TLSv1.2,TLSv1.3"
```

This option was added in MariaDB 10.4.6.

See Secure Connections Overview: TLS Protocol Versions for more information.

-t, --tmpdir

Defines path for temporary files.

```
--tmpdir=/path/tmp[;/path/tmp...]
```

Using this option, you can define the path to a directory Mariabackup uses in writing temporary files. If you want to use more than one, separate the values by a semicolon (that is, `;`). When passing multiple temporary directories, it cycles through them using round-robin.

```
$ mariabackup --backup \
     --tmpdir=/data/tmp;/tmp
```

--use-memory

Defines the buffer pool size that is used during the prepare stage.

```
--use-memory=124M
```

Using this option, you can define the buffer pool size for Mariabackup. Use it instead of `buffer_pool_size`.

```
$ mariabackup --prepare \
      --use-memory=124M
```

--user

Defines the username for connecting to the MariaDB Server.

```
--user=name
-u name
```

When Mariabackup runs it connects to the specified MariaDB Server to get its backups. Using this option, you can define the database user uses for authentication.

```
$ mariabackup --backup \
      --user=root \
      --password=root_passwd
```

--version

Prints version information.

Using this option, you can print the Mariabackup version information to stdout.

```
$ mariabackup --version
```

2.3.4.3 Full Backup and Restore with Mariabackup

Contents

When using Mariabackup, you have the option of performing a full or an incremental backup. Full backups create a complete backup of the database server in an empty directory while incremental backups update a previous backup with whatever changes to the data have occurred since the backup. This page documents how to perform full backups.

Backing up the Database Server

In order to back up the database, you need to run Mariabackup with the `--backup` option to tell it to perform a backup and with the `--target-dir` option to tell it where to place the backup files. When taking a full backup, the target directory must be empty or it must not exist.

To take a backup, run the following command:

```
$ mariabackup --backup \
   --target-dir=/var/mariadb/backup/ \
   --user=mariabackup --password=mypassword
```

The time the backup takes depends on the size of the databases or tables you're backing up. You can cancel the backup if you need to, as the backup process does not modify the database.

Mariabackup writes the backup files the target directory. If the target directory doesn't exist, then it creates it. If the target directory exists and contains files, then it raises an error and aborts.

Here is an example backup directory:

```
$ ls /var/mariadb/backup/

aria_log.0000001  mysql                  xtrabackup_checkpoints
aria_log_control  performance_schema     xtrabackup_info
backup-my.cnf     test                   xtrabackup_logfile
ibdata1           xtrabackup_binlog_info
```

Preparing the Backup for Restoration

The data files that Mariabackup creates in the target directory are not point-in-time consistent, given that the data files are copied at different times during the backup operation. If you try to restore from these files, InnoDB notices the inconsistencies and crashes to protect you from corruption

Before you can restore from a backup, you first need to **prepare** it to make the data files consistent. You can do so with the `--prepare` option.

```
$ mariabackup --prepare \
   --target-dir=/var/mariadb/backup/
```

Restoring the Backup

Once the backup is complete and you have prepared the backup for restoration (previous step), you can restore the backup using either the `--copy-back` or the `--move-back` options. The `--copy-back` option allows you to keep the original backup files. The `--move-back` option actually moves the backup files to the `datadir`, so the original backup files are lost.

- First, stop the MariaDB Server process.
- Then, ensure that the `datadir` is empty.
- Then, run Mariabackup with one of the options mentioned above:

```
$ mariabackup --copy-back \
   --target-dir=/var/mariadb/backup/
```

- Then, you may need to fix the file permissions.

When Mariabackup restores a database, it preserves the file and directory privileges of the backup. However, it writes the files to disk as the user and group restoring the database. As such, after restoring a backup, you may need to adjust the owner of the data directory to match the user and group for the MariaDB Server, typically `mysql` for both. For example, to recursively change ownership of the files to the `mysql` user and group, you could execute:

```
$ chown -R mysql:mysql /var/lib/mysql/
```

- Finally, start the MariaDB Server process.

Restoring with Other Tools

Once a full backup is prepared, it is a fully functional MariaDB data directory. Therefore, as long as the MariaDB Server process is stopped on the target server, you can technically restore the backup using any file copying tool, such as `cp` or `rysnc`. For example, you could also execute the following to restore the backup:

```
$ rsync -avrP /var/mariadb/backup /var/lib/mysql/
$ chown -R mysql:mysql /var/lib/mysql/
```

2.3.4.4 Incremental Backup and Restore with Mariabackup

Contents

When using Mariabackup, you have the option of performing a full or incremental backup. Full backups create a complete copy in an empty directory while incremental backups update a previous backup with new data. This page documents incremental backups.

InnoDB pages contain log sequence numbers, or LSN's. Whenever you modify a row on any InnoDB table on the database, the storage engine increments this number. When performing an incremental backup, Mariabackup checks the most recent LSN for the backup against the LSN's contained in the database. It then updates any of the backup files that have fallen behind.

Backing up the Database Server

In order to take an incremental backup, you first need to take a full backup. In order to back up the database, you need to run Mariabackup with the `--backup` option to tell it to perform a backup and with the `--target-dir` option to tell it where to place the backup files. When taking a full backup, the target directory must be empty or it must not exist.

To take a backup, run the following command:

```
$ mariabackup --backup \
   --target-dir=/var/mariadb/backup/ \
   --user=mariabackup --password=mypassword
```

This backs up all databases into the target directory `/var/mariadb/backup`. If you look in that directory at the `xtrabackup_checkpoints` file, you can see the LSN data provided by InnoDB.

For example:

```
backup_type = full-backuped
from_lsn = 0
to_lsn = 1635102
last_lsn = 1635102
recover_binlog_info = 0
```

Backing up the Incremental Changes

Once you have created a full backup on your system, you can also back up the incremental changes as often as you would like.

In order to perform an incremental backup, you need to run Mariabackup with the `--backup` option to tell it to perform a backup and with the `--target-dir` option to tell it where to place the incremental changes. The target directory must be empty. You also need to run it with the `--incremental-basedir` option to tell it the path to the full backup taken above. For example:

```
$ mariabackup --backup \
   --target-dir=/var/mariadb/inc1/ \
   --incremental-basedir=/var/mariadb/backup/ \
   --user=mariabackup --password=mypassword
```

This command creates a series of delta files that store the incremental changes in `/var/mariadb/inc1` . You can find a similar `xtrabackup_checkpoints` file in this directory, with the updated LSN values.

For example:

```
backup_type = incremental
from_lsn = 1635102
to_lsn = 1635114
last_lsn = 1635114
recover_binlog_info = 0
```

To perform additional incremental backups, you can then use the target directory of the previous incremental backup as the incremental base directory of the next incremental backup. For example:

```
$ mariabackup --backup \
   --target-dir=/var/mariadb/inc2/ \
   --incremental-basedir=/var/mariadb/inc1/ \
   --user=mariabackup --password=mypassword
```

Combining with `--stream` output

When using `--stream` , e.g for compression or encryption using external tools, the `xtrabackup_checkpoints` file containing the information where to continue from on the next incremental backup will also be part of the compressed/encrypted backup file, and so not directly accessible by default.

A directory containing an extra copy of the file can be created using the `--extra-lsndir=...` option though, and this directory can then be passed to the next incremental backup `--incremental-basedir=...` , for example:

```
# initial full backup
$ mariabackup --backup --stream=mbstream \
  --user=mariabackup --password=mypassword \
  --extra-lsndir=backup_base | gzip > backup_base.gz

# incremental backup
$ mariabackup --backup --stream=mbstream \
  --incremental-basedir=backup_base \
  --user=mariabackup --password=mypassword \
  --extra-lsndir=backup_inc1 | gzip > backup-inc1.gz
```

Preparing the Backup

Following the above steps, you have three backups in `/var/mariadb` : The first is a full backup, the others are increments on this first backup. In order to restore a backup to the database, you first need to apply the incremental backups to the base full backup. This is done using the `--prepare` command option. In MariaDB 10.1, you would also have to use the the

`--apply-log-only` option.

In MariaDB 10.2 and later, perform the following process:

First, prepare the base backup:

```
$ mariabackup --prepare \
   --target-dir=/var/mariadb/backup
```

Running this command brings the base full backup, that is, `/var/mariadb/backup`, into sync with the changes contained in the InnoDB redo log collected while the backup was taken.

Then, apply the incremental changes to the base full backup:

```
$ mariabackup --prepare \
   --target-dir=/var/mariadb/backup \
   --incremental-dir=/var/mariadb/inc1
```

Running this command brings the base full backup, that is, `/var/mariadb/backup`, into sync with the changes contained in the first incremental backup.

For each remaining incremental backup, repeat the last step to bring the base full backup into sync with the changes contained in that incremental backup.

Restoring the Backup

Once you've applied all incremental backups to the base, you can restore the backup using either the `--copy-back` or the `--move-back` options. The `--copy-back` option allows you to keep the original backup files. The `--move-back` option actually moves the backup files to the `datadir`, so the original backup files are lost.

- First, stop the MariaDB Server process.
- Then, ensure that the `datadir` is empty.
- Then, run Mariabackup with one of the options mentioned above:

```
$ mariabackup --copy-back \
   --target-dir=/var/mariadb/backup/
```

- Then, you may need to fix the file permissions.

When Mariabackup restores a database, it preserves the file and directory privileges of the backup. However, it writes the files to disk as the user and group restoring the database. As such, after restoring a backup, you may need to adjust the owner of the data directory to match the user and group for the MariaDB Server, typically `mysql` for both. For example, to recursively change ownership of the files to the `mysql` user and group, you could execute:

```
$ chown -R mysql:mysql /var/lib/mysql/
```

- Finally, start the MariaDB Server process.

2.3.4.5 Partial Backup and Restore with Mariabackup

Contents

When using Mariabackup, you have the option of performing partial backups. Partial backups allow you to choose which databases or tables to backup, as long as the table or partition involved is in an InnoDB file-per-table tablespace.This page documents how to perform partial backups.

Backing up the Database Server

Just like with full backups, in order to back up the database, you need to run Mariabackup with the `--backup` option to tell it to perform a backup and with the `--target-dir` option to tell it where to place the backup files. The target directory must be empty or not exist.

For a partial backup, there are a few other arguments that you can provide as well:

- To tell it which databases to backup, you can provide the `--databases` option.
- To tell it which databases to exclude from the backup, you can provide the `--databases-exclude` option.
- To tell it to check a file for the databases to backup, you can provide the `--databases-file` option.
- To tell it which tables to backup, you can use the `--tables` option.
- To tell it which tables to exclude from the backup, you can provide the `--tables-exclude` option.
- To tell it to check a file for specific tables to backup, you can provide the `--tables-file` option.

The non-file partial backup options support regex in the database and table names.

For example, to take a backup of any database that starts with the string `app1_` and any table in those databases that start with the string `tab_`, run the following command:

```
$ mariabackup --backup \
   --target-dir=/var/mariadb/backup/ \
   --databases='app1_*' --tables='tab_*' \
   --user=mariabackup --password=mypassword
```

Mariabackup cannot currently backup a subset of partitions from a partitioned table. Backing up a partitioned table is currently an all-or-nothing selection. See MDEV-17132 about that. If you need to backup a subset of partitions, then one possibility is that instead of using Mariabackup, you can export the file-per-table tablespaces of the partitions.

The time the backup takes depends on the size of the databases or tables you're backing up. You can cancel the backup if you need to, as the backup process does not modify the database.

Mariabackup writes the backup files the target directory. If the target directory doesn't exist, then it creates it. If the target directory exists and contains files, then it raises an error and aborts.

Preparing the Backup

Just like with full backups, the data files that Mariabackup creates in the target directory are not point-in-time consistent, given that the data files are copied at different times during the backup operation. If you try to restore from these files, InnoDB notices the inconsistencies and crashes to protect you from corruption. In fact, for partial backups, the backup is not even a completely functional MariaDB data directory, so InnoDB would raise more errors than it would for full backups. This point will also be very important to keep in mind during the restore process.

Before you can restore from a backup, you first need to **prepare** it to make the data files consistent. You can do so with the `--prepare` command option.

Partial backups rely on InnoDB's transportable tablespaces. For MariaDB to import tablespaces like these, InnoDB looks for a file with a `.cfg` extension. For Mariabackup to create these files, you also need to add the `--export` option during the prepare step.

For example, you might execute the following command:

```
$ mariabackup --prepare --export \
   --target-dir=/var/mariadb/backup/
```

If this operation completes without error, then the backup is ready to be restored.

MariaDB until 10.2.8

In MariaDB 10.2.8 and before, Mariabackup did not support the `--export` option. See MDEV-13466 about that. In these versions of MariaDB, this means that Mariabackup could not create `.cfg` files for InnoDB file-per-table tablespaces during the `--prepare` stage. You can still import file-per-table tablespaces without the `.cfg` files in many cases, so it may still be possible in those versions to restore partial backups or to restore individual tables and partitions with just the `.ibd` files. If you have a full backup and you need to create `.cfg` files for InnoDB file-per-table tablespaces, then you can do so by preparing the backup as usual without the `--export` option, and then restoring the backup, and then starting the server. At that point, you can use the server's built-in features to copy the transportable tablespaces.

Restoring the Backup

The restore process for partial backups is quite different than the process for full backups. A partial backup is not a completely functional data directory. The data dictionary in the InnoDB system tablespace will still contain entries for the databases and tables that were not included in the backup.

Rather than using the `--copy-back` or the `--move-back`, each individual InnoDB file-per-table tablespace file will have to be manually imported into the target server. The process that is used to import the file will depend on whether partitioning is involved.

Restoring Individual Non-Partitioned Tables

To restore individual non-partitioned tables from a backup, find the `.ibd` and `.cfg` files for the table in the backup, and then import them using the Importing Transportable Tablespaces for Non-partitioned Tables process.

Restoring Individual Partitions and Partitioned Tables

To restore individual partitions or partitioned tables from a backup, find the `.ibd` and `.cfg` files for the partition(s) in the backup, and then import them using the Importing Transportable Tablespaces for Partitioned Tables process.

2.3.4.6 Restoring Individual Tables and Partitions with Mariabackup

Contents

When using Mariabackup, you don't necessarily need to restore every table and/or partition that was backed up. Even if you're starting from a full backup, it is certainly possible to restore only certain tables and/or partitions from the backup, as long as the table or partition involved is in an InnoDB file-per-table tablespace. This page documents how to restore individual tables and partitions.

Preparing the Backup

Before you can restore from a backup, you first need to **prepare** it to make the data files consistent. You can do so with the `--prepare` command option.

The ability to restore individual tables and partitions relies on InnoDB's transportable tablespaces. For MariaDB to import tablespaces like these, InnoDB looks for a file with a `.cfg` extension. For Mariabackup to create these files, you also need to add the `--export` option during the prepare step.

For example, you might execute the following command:

```
$ mariabackup --prepare --export \
   --target-dir=/var/mariadb/backup/ \
   --user=mariabackup --password=mypassword
```

If this operation completes without error, then the backup is ready to be restored.

MariaDB until 10.2.8

Before MariaDB 10.2.9, Mariabackup did not support the `--export` option. See MDEV-13466 about that. In earlier versions of MariaDB, this means that Mariabackup could not create `.cfg` files for InnoDB file-per-table tablespaces during the `--prepare` stage. You can still import file-per-table tablespaces without the `.cfg` files in many cases, so it may still be possible in those versions to restore partial backups or to restore individual tables and partitions with just the `.ibd` files. If you have a full backup and you need to create `.cfg` files for InnoDB file-per-table tablespaces, then you can do so by preparing the backup as usual without the `--export` option, and then restoring the backup, and then starting the server. At that point, you can use the server's built-in features to copy the transportable tablespaces.

Restoring the Backup

The restore process for restoring individual tables and/or partitions is quite different than the process for full backups.

Rather than using the `--copy-back` or the `--move-back`, each individual InnoDB file-per-table tablespace file will have to be manually imported into the target server. The process that is used to restore the backup will depend on whether partitioning is involved.

Restoring Individual Non-Partitioned Tables

To restore individual non-partitioned tables from a backup, find the `.ibd` and `.cfg` files for the table in the backup, and then import them using the Importing Transportable Tablespaces for Non-partitioned Tables process.

Restoring Individual Partitions and Partitioned Tables

To restore individual partitions or partitioned tables from a backup, find the `.ibd` and `.cfg` files for the partition(s) in the backup, and then import them using the Importing Transportable Tablespaces for Partitioned Tables process.

2.3.4.7 Setting up a Replica with Mariabackup

The terms *master* and *slave* have historically been used in replication, but the terms terms *primary* and *replica* are now preferred. The old terms are used still used in parts of the documentation, and in MariaDB commands, although MariaDB 10.5 has begun the process of renaming. The documentation process is ongoing. See MDEV-18777 to follow progress on this effort.

Contents

Mariabackup makes it very easy to set up a replica using a full backup. This page documents how to set up a replica from a backup.

If you are using MariaDB Galera Cluster, then you may want to try one of the following pages instead:

- Configuring MariaDB Replication between MariaDB Galera Cluster and MariaDB Server
- Configuring MariaDB Replication between Two MariaDB Galera Clusters

Backup the Database and Prepare It

The first step is to simply take and prepare a fresh full backup of a database server in the replication topology. If the source database server is the desired replication primary, then we do not need to add any additional options when taking the full backup. For example:

```
$ mariabackup --backup \
   --target-dir=/var/mariadb/backup/ \
   --user=mariabackup --password=mypassword
```

If the source database server is a replica of the desired primary, then we should add the --slave-info option, and possibly the --safe-slave-backup option. For example:

```
$ mariabackup --backup \
   --slave-info --safe-slave-backup \
   --target-dir=/var/mariadb/backup/ \
   --user=mariabackup --password=mypassword
```

And then we would prepare the backup as you normally would. For example:

```
$ mariabackup --prepare \
   --target-dir=/var/mariadb/backup/
```

Copy the Backup to the New Replica

Once the backup is done and prepared, we can copy it to the new replica. For example:

```
$ rsync -avP /var/mariadb/backup dbserver2:/var/mariadb/backup
```

Restore the Backup on the New Replica

At this point, we can restore the backup to the datadir, as you normally would. For example:

```
$ mariabackup --copy-back \
   --target-dir=/var/mariadb/backup/
```

And adjusting file permissions, if necessary:

```
$ chown -R mysql:mysql /var/lib/mysql/
```

Create a Replication User on the Primary

Before the new replica can begin replicating from the primary, we need to create a user account on the primary that the replica can use to connect, and we need to grant the user account the REPLICATION SLAVE privilege. For example:

```
CREATE USER 'repl'@'dbserver2' IDENTIFIED BY 'password';
GRANT REPLICATION SLAVE ON *.*  TO 'repl'@'dbserver2';
```

Configure the New Replica

Before we start the server on the new replica, we need to configure it. At the very least, we need to ensure that it has a unique server_id value. We also need to make sure other replication settings are what we want them to be, such as the various GTID system variables, if those apply in the specific environment.

Once configuration is done, we can start the MariaDB Server process on the new replica.

Start Replication on the New Replica

At this point, we need to get the replication coordinates of the primary from the original backup directory.

If we took the backup on the primary, then the coordinates will be in the xtrabackup_binlog_info file. If we took the backup on another replica and if we provided the --slave-info option, then the coordinates will be in the file xtrabackup_slave_info file.

Mariabackup dumps replication coordinates in two forms: GTID coordinates and binary log file and position coordinates, like the ones you would normally see from SHOW MASTER STATUS output. We can choose which set of coordinates we would like to use to set up replication.

For example:

```
mariadb-bin.000096 568 0-1-2
```

Regardless of the coordinates we use, we will have to set up the primary connection using CHANGE MASTER TO and then start the replication threads with START SLAVE.

GTIDs

If we want to use GTIDs, then we will have to first set gtid_slave_pos to the GTID coordinates that we pulled from either the xtrabackup_binlog_info file or the xtrabackup_slave_info file in the backup directory. For example:

```
$ cat xtrabackup_binlog_info
mariadb-bin.000096 568 0-1-2
```

And then we would set `MASTER_USE_GTID=slave_pos` in the CHANGE MASTER TO command. For example:

```
SET GLOBAL gtid_slave_pos = "0-1-2";
CHANGE MASTER TO
   MASTER_HOST="dbserver1",
   MASTER_PORT=3306,
   MASTER_USER="repl",
   MASTER_PASSWORD="password",
   MASTER_USE_GTID=slave_pos;
START SLAVE;
```

File and Position

If we want to use the binary log file and position coordinates, then we would set `MASTER_LOG_FILE` and `MASTER_LOG_POS` in the CHANGE MASTER TO command to the file and position coordinates that we pulled; either the xtrabackup_binlog_info file or the xtrabackup_slave_info file in the backup directory, depending on whether the backup was taken from the primary or from a replica of the primary. For example:

```
CHANGE MASTER TO
   MASTER_HOST="dbserver1",
   MASTER_PORT=3306,
   MASTER_USER="repl",
   MASTER_PASSWORD="password",
   MASTER_LOG_FILE='mariadb-bin.000096',
   MASTER_LOG_POS=568;
START SLAVE;
```

Check the Status of the New Replica

We should be done setting up the replica now, so we should check its status with SHOW SLAVE STATUS. For example:

```
SHOW SLAVE STATUS\G
```

2.3.4.8 Files Backed Up By Mariabackup

Contents

Files Included in Backup

Mariabackup backs up the files listed below.

InnoDB Data Files

Mariabackup backs up the following InnoDB data files:

- InnoDB system tablespace
- InnoDB file-per-table tablespaces

MyRocks Data Files

Starting with MariaDB 10.2.16 and MariaDB 10.3.8, Mariabackup will back up tables that use the MyRocks storage engine. This data data is located in the directory defined by the `rocksdb_datadir` system variable. Mariabackup backs this data up by performing a checkpoint using the `rocksdb_create_checkpoint` system variable.

Mariabackup does not currently support partial backups for MyRocks.

Other Data Files

Mariabackup also backs up files with the following extensions:

- frm
- isl
- MYD
- MYI
- MAD
- MAI
- MRG
- TRG
- TRN
- ARM
- ARZ
- CSM
- CSV
- opt
- par

Files Excluded From Backup

Mariabackup does **not** back up the files listed below.

- InnoDB Temporary Tablespaces
- Binary logs
- Relay logs

2.3.4.9 Files Created by Mariabackup

Contents

Mariabackup creates the following files:

backup-my.cnf

During the backup, any server options relevant to Mariabackup are written to the `backup-my.cnf` option file, so that they can be re-read later during the `--prepare` stage.

ib_logfile0

In MariaDB 10.2.10 and later, Mariabackup creates an empty InnoDB redo log file called `ib_logfile0` as part of the `--prepare` stage. This file has 3 roles:

1. In the source server, `ib_logfile0` is the first (and possibly the only) InnoDB redo log file.
2. In the non-prepared backup, `ib_logfile0` contains all of the InnoDB redo log copied during the backup. Previous versions of Mariabackup would use a file called `xtrabackup_logfile` for this.
3. During the `--prepare` stage, `ib_logfile0` would previously be deleted. Now during the `--prepare` stage, `ib_logfile0` is initialized as an empty InnoDB redo log file. That way, if the backup is manually restored, any pre-existing InnoDB redo log files would get overwritten by the empty one. This helps to prevent certain kinds of known issues. For example, see Mariabackup Overview: Manual Restore with Pre-existing InnoDB Redo Log files.

xtrabackup_logfile

In MariaDB 10.2.9 and before, Mariabackup creates `xtrabackup_logfile` to store the InnoDB redo log, In later versions, `ib_logfile0` is created instead.

xtrabackup_binlog_info

This file stores the binary log file name and position that corresponds to the backup.

This file also stores the value of the `gtid_current_pos` system variable that correspond to the backup.

For example:

```
mariadb-bin.000096 568 0-1-2
```

The values in this file are only guaranteed to be consistent with the backup if the `--no-lock` option was **not** provided when the backup was taken.

xtrabackup_binlog_pos_innodb

This file is created by mariabackup to provide the binary log file name and position when the --no-lock option is used. It can be used instead of the file "xtrabackup_binlog_info" to obtain transactionally consistent binlog coordinates from the backup of a master server with the --no-lock option to minimize the impact on a running server.

Whenever a transaction is committed inside InnoDB when the binary log is enabled, the corresponding binlog coordinates are written to the InnoDB redo log along with the transaction commit. This allows one to restore the binlog coordinates corresponding to the last commit done by InnoDB along with a backup.

The limitation of using "xtrabackup_binlog_pos_innodb" with the "--no-lock" option is that no DDL or modification of non-transactional tables should be done during the backup. If the last event in the binlog is a DDL/non-transactional update, the coordinates in the file "xtrabackup_binlog_pos_innodb" will be too old. But as long as only InnoDB updates are done during the backup, the coordinates will be correct.

xtrabackup_checkpoints

The `xtrabackup_checkpoints` file contains metadata about the backup.

For example:

```
backup_type = full-backuped
from_lsn = 0
to_lsn = 1635102
last_lsn = 1635102
recover_binlog_info = 0
```

See below for a description of the fields.

If the `--extra-lsndir` option is provided, then an extra copy of this file will be saved in that directory.

backup_type

If the backup is a non-prepared full backup or a non-prepared partial backup, then `backup_type` is set to `full-backuped`.

If the backup is a non-prepared incremental backup, then `backup_type` is set to `incremental`.

If the backup has already been prepared, then `backup_type` is set to `log-applied`.

from_lsn

If `backup_type` is `full-backuped`, then `from_lsn` has the value of `0`.

If `backup_type` is `incremental`, then `from_lsn` has the value of the log sequence number (LSN) at which the backup started reading from the InnoDB redo log. This is internally used by Mariabackup when preparing incremental backups.

This value can be manually set during an incremental backup with the `--incremental-lsn` option. However, it is generally better to let Mariabackup figure out the `from_lsn` automatically by specifying a parent backup with the `--incremental-basedir` option.

to_lsn

`to_lsn` has the value of the log sequence number (LSN) of the last checkpoint in the InnoDB redo log. This is internally used by Mariabackup when preparing incremental backups.

last_lsn

`last_lsn` has the value of the last log sequence number (LSN) read from the InnoDB redo log. This is internally used by Mariabackup when preparing incremental backups.

xtrabackup_info

The `xtrabackup_info` file contains information about the backup. The fields in this file are listed below.

If the `--extra-lsndir` option is provided, then an extra copy of this file will be saved in that directory.

uuid

If a UUID was provided by the `--incremental-history-uuid` option, then it will be saved here. Otherwise, this will be the empty string.

name

If a name was provided by the `--history` or the `---incremental-history-name` options, then it will be saved here. Otherwise, this will be the empty string.

tool_name

The name of the Mariabackup executable that performed the backup. This is generally `mariabackup`.

tool_command

The arguments that were provided to Mariabackup when it performed the backup.

tool_version

The version of Mariabackup that performed the backup.

ibbackup_version

The version of Mariabackup that performed the backup.

server_version

The version of MariaDB Server that was backed up.

start_time

The time that the backup started.

end_time

The time that the backup ended.

lock_time

The amount of time that Mariabackup held its locks.

binlog_pos

This field stores the binary log file name and position that corresponds to the backup.

This field also stores the value of the `gtid_current_pos` system variable that correspond to the backup.

The values in this field are only guaranteed to be consistent with the backup if the `--no-lock` option was **not** provided when the backup was taken.

innodb_from_lsn

This is identical to `from_lsn` in `xtrabackup_checkpoints`.

If the backup is a full backup, then `innodb_from_lsn` has the value of `0`.

If the backup is an incremental backup, then `innodb_from_lsn` has the value of the log sequence number (LSN) at which the backup started reading from the InnoDB redo log.

innodb_to_lsn

This is identical to `to_lsn` in `xtrabackup_checkpoints`.

`innodb_to_lsn` has the value of the log sequence number (LSN) of the last checkpoint in the InnoDB redo log.

partial

If the backup is a partial backup, then this value will be `Y`.

Otherwise, this value will be `N`.

incremental

If the backup is an incremental backup, then this value will be `Y`.

Otherwise, this value will be `N`.

format

This field's value is the format of the backup.

If the `--stream` option was set to `xbstream`, then this value will be `xbstream`.

If the `--stream` option was **not** provided, then this value will be `file`.

compressed

If the `--compress` option was provided, then this value will be `compressed`.

Otherwise, this value will be `N`.

xtrabackup_slave_info

If the `--slave-info` option is provided, then this file contains the `CHANGE MASTER` command that can be used to set up a new server as a slave of the original server's master after the backup has been restored.

Mariabackup does **not** check if GTIDs are being used in replication. It takes a shortcut and assumes that if the `gtid_slave_pos` system variable is non-empty, then it writes the `CHANGE MASTER` command with the `MASTER_USE_GTID` option set to `slave_pos`. Otherwise, it writes the `CHANGE MASTER` command with the `MASTER_LOG_FILE` and `MASTER_LOG_POS` options using the master's binary log file and position. See MDEV-19264 for more information.

xtrabackup_galera_info

If the `--galera-info` option is provided, then this file contains information about a Galera Cluster node's state.

The file contains the values of the `wsrep_local_state_uuid` and `wsrep_last_committed` status variables.

The values are written in the following format:

```
wsrep_local_state_uuid:wsrep_last_committed
```

For example:

```
d38587ce-246c-11e5-bcce-6bbd0831cc0f:1352215
```

<table>.delta

If the backup is an incremental backup, then this file contains changed pages for the table.

<table>.delta.meta

If the backup is an incremental backup, then this file contains metadata about `<table>.delta` files. The fields in this file are listed below.

page_size

This field contains either the value of `innodb_page_size` or the value of the `KEY_BLOCK_SIZE` table option for the table if the `ROW_FORMAT` table option for the table is set to `COMPRESSED`.

zip_size

If the `ROW_FORMAT` table option for this table is set to `COMPRESSED`, then this field contains the value of the compressed page size.

space_id

This field contains the value of the table's `space_id`.

2.3.4.10 Using Encryption and Compression Tools With Mariabackup

Contents

Mariabackup supports streaming to stdout with the `--stream=xbstream` option. This option allows easy integration with popular encryption and compression tools. Below are several examples.

Encrypting and Decrypting Backup With openssl

The following example creates an AES-encrypted backup, protected with the password "mypass" and stores it in a file "backup.xb.enc":

```
mariabackup --user=root --backup --stream=xbstream  | openssl  enc -aes-256-cbc -k mypass >
backup.xb.enc
```

To decrypt and unpack this backup into the current directory, the following command can be used:

```
openssl  enc -d -aes-256-cbc -k mypass -in backup.xb.enc | mbstream -x
```

Compressing and Decompressing Backup With gzip

This example compresses the backup without encrypting:

```
mariabackup --user=root --backup --stream=xbstream | gzip > backupstream.gz
```

We can decompress and unpack the backup as follows:

```
gunzip -c backupstream.gz | mbstream -x
```

Compressing and Encrypting Backup, Using gzip and openssl

This example adds a compression step before the encryption, otherwise looks almost identical to the previous example:

```
mariabackup --user=root --backup --stream=xbstream | gzip | openssl  enc -aes-256-cbc -k mypass
> backup.xb.gz.enc
```

We can decrypt, decompress and unpack the backup as follow (note `gzip -d` in the pipeline):

```
openssl  enc -d -aes-256-cbc -k mypass -in backup.xb.gz.enc |gzip -d| mbstream -x
```

Compressing and Encrypting with 7Zip

7zip archiver is a popular utility (especially on Windows) that supports reading from standard output, with the - `-si` option, and writing to stdout with the `-so` option, and can thus be used together with Mariabackup.

Compressing backup with the 7z command line utility works as follows:

```
mariabackup --user=root --backup --stream=xbstream | 7z a -si backup.xb.7z
```

Uncompress and unpack the archive with

```
7z e backup.xb.7z -so |mbstream -x
```

7z also has builtin AES-256 encryption. To encrypt the backup from the previous example using password SECRET, add `-pSECRET` to the 7z command line.

Encrypting With GPG

Encryption

```
mariabackup --user=root --backup --stream=xbstream | gpg -c --passphrase SECRET --batch --yes -o backup.xb.gpg
```

Decrypt, unpack

```
gpg --decrypt --passphrase SECRET --batch --yes  backup.xb.gpg | mbstream -x
```

Interactive Input for Passphrases

Most of the described tools also provide a way to enter a passphrase interactively (although 7zip does not seem to work well when reading input from stdin). Please consult documentation of the tools for more info.

Writing extra status files

By default files like `xtrabackup_checkpoints` are also written to the output stream only, and so would not be available for taking further incremental backups without prior extraction from the compressed or encrypted stream output file.

To avoid this these files can additionally be written to a directory that can then be used as input for further incremental backups using the `--extra-lsndir=...` option.

See also e.g: Combining incremental backups with streaming output

2.3.4.11 How Mariabackup Works

Contents

This is a description of the different stages in Mariabackup, what they do and why they are needed.

Note that a few items are marked with `TODO`; these are things we are working on and will be in next version of Mariabackup.

Execution Stages

Initialization Phase

- Connect to mysqld instance, find out important variables (datadir ,InnoDB pagesize, encryption keys, encryption plugin etc)
- Scan the database directory, `datadir`, looking for InnoDB tablespaces, load the tablespaces (basically, it is an “open” in InnoDB sense)
- If --lock-ddl-per-table is used:
  - Do MDL locks, for InnoDB tablespaces that we want to copy. This is to ensure that there are no ALTER, RENAME , TRUNCATE or DROP TABLE on any of the tables that we want to copy.
  - This is implemented with:

```
BEGIN
For each affected table
SELECT 1 from <table> LIMIT 0
```

- If lock-ddl-per-table is not done, then Mariabackup would have to know all tables that were created or altered during the backup. See MDEV-16791.

Redo Log Handling

Start a dedicated thread in Mariabackup to copy InnoDB redo log (`ib_logfile*`).

- This is needed to record all changes done while the backup is running. (The redo log logically is a single circular file, split into innodb_log_files_in_group files.)
- The log is also used to see detect if any truncate or online alter tables are used.
- The assumption is that the copy thread will be able to keep up with server. It should always be able keep up, if the redo log is big enough.

Copy-phase for InnoDB Tablespaces

- Copy all selected tablespaces, file by file, in dedicated threads in Mariabackup without involving the mysqld server.
- This is special “careful” copy, it looks for page-level consistency by checking the checksum.
- The files are not point-in-time consistent as data may change during copy.
- The idea is that InnoDB recovery would make it point-in-time consistent.
- Copy Aria log files (TODO)

Create a Consistent Backup Point

- Execute FLUSH TABLE WITH READ LOCK. This is default, but may be omitted with the `--no-lock` parameter. The reason why `FLUSH` is needed is to ensure that all tables are in a consistent state at the exact same point in time, independent of storage engine.
- If `--lock-ddl-per-table` is used and there is a user query waiting for MDL, the user query will be killed to resolve a deadlock. Note that these are only queries of type ALTER, DROP, TRUNCATE or RENAME TABLE. (MDEV-15636)

Last Copy Phase

- Copy `.frm`, `MyISAM`, `Aria` and other storage engine files
- If `MyRocks` is used, create rocksdb checkpoint via "set rocksdb_create_checkpoint=$rocksdb_data_dir/mariabackup_rocksdb_checkpoint " command. The result of it is a directory with hardlinks to MyRocks files. Copy the checkpoint directory to the backup (or create hardlinks in backup directory is on the same partition as data directory). Remove the checkpoint directory.
- Copy tables that were created while the backup was running and do rename files that were changed during backup (since MDEV-16791)
- Copy the rest of InnoDB redo log, stop redo-log-copy thread
- Copy changes to Aria log files (They are append only, so this is easy to do) (TODO)
- Write some metadata info (binlog position)

Release Locks

- If FLUSH TABLE WITH READ LOCK was done:
  - execute: `UNLOCK TABLES`
- If `--lock-ddl-per-table` was done:
  - execute `COMMIT`

Handle Log Tables (TODO)

- If log tables exists:
  - Take MDL lock for log tables
  - Copy part of log tables that wasn't copied before
  - Unlock log tables

Notes

- If FLUSH TABLE WITH READ LOCK is not used, then only InnoDB tables will be consistent (not the privilege tables in the mysql database or the binary log). The backup point depends on the content of the redo log within the backup itself.

2.3.4.12 Mariabackup and BACKUP STAGE Commands

MariaDB starting with 10.4.1

The `BACKUP STAGE` commands were introduced in MariaDB 10.4.1.

Contents

The `BACKUP STAGE` commands are a set of commands to make it possible to make an efficient external backup tool. How Mariabackup uses these commands depends on whether you are using the version that is bundled with MariaDB Community Server or the version that is bundled with MariaDB Enterprise Server.

Mariabackup and `BACKUP STAGE` Commands in MariaDB Community Server

MariaDB starting with 10.4.1

In MariaDB Community Server, Mariabackup first supported `BACKUP STAGE` commands in MariaDB 10.4.1.

In MariaDB 10.3 and before, the `BACKUP STAGE` commands are **not** supported, so Mariabackup executes the `FLUSH TABLES WITH READ LOCK` command to lock the database. When the backup is complete, it executes the `UNLOCK TABLES` command to unlock the database.

In MariaDB 10.4 and later, the `BACKUP STAGE` commands are supported. However, the version of Mariabackup that is bundled with MariaDB Community Server does not yet use the `BACKUP STAGE` commands in the most efficient way. Mariabackup simply executes the following `BACKUP STAGE` commands to lock the database:

```
BACKUP STAGE START;
BACKUP STAGE BLOCK_COMMIT;
```

When the backup is complete, it executes the following `BACKUP STAGE` command to unlock the database:

```
BACKUP STAGE END;
```

If you would like to use a version of Mariabackup that uses the `BACKUP STAGE` commands in the most efficient way, then your best option is to use MariaDB Enterprise Backup that is bundled with MariaDB Enterprise Server.

Tasks Performed Prior to `BACKUP STAGE` in MariaDB Community Server

- Copy some transactional tables.
  - InnoDB (i.e. `ibdataN` and file extensions `.ibd` and `.isl`)

- Copy the tail of some transaction logs.
  - The tail of the InnoDB redo log (i.e. `ib_logfileN` files) will be copied for InnoDB tables.

`BACKUP STAGE START` in MariaDB Community Server

Mariabackup from MariaDB Community Server does not currently perform any tasks in the `START` stage.

`BACKUP STAGE FLUSH` in MariaDB Community Server

Mariabackup from MariaDB Community Server does not currently perform any tasks in the `FLUSH` stage.

`BACKUP STAGE BLOCK_DDL` in MariaDB Community Server

Mariabackup from MariaDB Community Server does not currently perform any tasks in the `BLOCK_DDL` stage.

`BACKUP STAGE BLOCK_COMMIT` in MariaDB Community Server

Mariabackup from MariaDB Community Server performs the following tasks in the `BLOCK_COMMIT` stage:

- Copy other files.
  - i.e. file extensions `.frm`, `.isl`, `.TRG`, `.TRN`, `.opt`, `.par`
- Copy some transactional tables.
  - Aria (i.e. `aria_log_control` and file extensions `.MAD` and `.MAI`)
- Copy the non-transactional tables.
  - MyISAM (i.e. file extensions `.MYD` and `.MYI`)
  - MERGE (i.e. file extensions `.MRG`)
  - ARCHIVE (i.e. file extensions `.ARM` and `.ARZ`)
  - CSV (i.e. file extensions `.CSM` and `.CSV`)
- Create a MyRocks checkpoint using the `rocksdb_create_checkpoint` system variable.
- Copy the tail of some transaction logs.
  - The tail of the InnoDB redo log (i.e. `ib_logfileN` files) will be copied for InnoDB tables.
- Save the binary log position to `xtrabackup_binlog_info`.
- Save the Galera Cluster state information to `xtrabackup_galera_info`.

`BACKUP STAGE END` in MariaDB Community Server

Mariabackup from MariaDB Community Server performs the following tasks in the `END` stage:

- Copy the MyRocks checkpoint into the backup.

Mariabackup and `BACKUP STAGE` Commands in MariaDB Enterprise Server

MariaDB starting with 10.2.25

MariaDB Enterprise Backup first supported `BACKUP STAGE` commands in MariaDB Enterprise Server 10.4.6-1, MariaDB Enterprise Server 10.3.16-1, and MariaDB Enterprise Server 10.2.25-1.

The following sections describe how the MariaDB Enterprise Backup version of Mariabackup that is bundled with MariaDB Enterprise Server uses each `BACKUP STAGE` command in an efficient way.

`BACKUP STAGE START` in MariaDB Enterprise Server

Mariabackup from MariaDB Enterprise Server performs the following tasks in the `START` stage:

- Copy all transactional tables.
  - InnoDB (i.e. `ibdataN` and file extensions `.ibd` and `.isl`)
  - Aria (i.e. `aria_log_control` and file extensions `.MAD` and `.MAI`)
- Copy the tail of all transaction logs.
  - The tail of the InnoDB redo log (i.e. `ib_logfileN` files) will be copied for InnoDB tables.
  - The tail of the Aria redo log (i.e. `aria_log.N` files) will be copied for Aria tables.

`BACKUP STAGE FLUSH` in MariaDB Enterprise Server

Mariabackup from MariaDB Enterprise Server performs the following tasks in the `FLUSH` stage:

- Copy all non-transactional tables that are not in use. This list of used tables is found with `SHOW OPEN TABLES`.
  - MyISAM (i.e. file extensions `.MYD` and `.MYI`)
  - MERGE (i.e. file extensions `.MRG`)
  - ARCHIVE (i.e. file extensions `.ARM` and `.ARZ`)
  - CSV (i.e. file extensions `.CSM` and `.CSV`)
- Copy the tail of all transaction logs.
  - The tail of the InnoDB redo log (i.e. `ib_logfileN` files) will be copied for InnoDB tables.
  - The tail of the Aria redo log (i.e. `aria_log.N` files) will be copied for Aria tables.

`BACKUP STAGE BLOCK_DDL` in MariaDB Enterprise Server

Mariabackup from MariaDB Enterprise Server performs the following tasks in the `BLOCK_DDL` stage:

- Copy other files.
  - i.e. file extensions `.frm`, `.isl`, `.TRG`, `.TRN`, `.opt`, `.par`
- Copy the non-transactional tables that were in use during `BACKUP STAGE FLUSH`.
  - MyISAM (i.e. file extensions `.MYD` and `.MYI`)
  - MERGE (i.e. file extensions `.MRG`)
  - ARCHIVE (i.e. file extensions `.ARM` and `.ARZ`)
  - CSV (i.e. file extensions `.CSM` and `.CSV`)
- Check `ddl.log` for DDL executed before the `BLOCK DDL` stage.
  - The file names of newly created tables can be read from `ddl.log`.
  - The file names of dropped tables can also be read from `ddl.log`.
  - The file names of renamed tables can also be read from `ddl.log`, so the files can be renamed instead of re-copying them.
- Copy changes to system log tables.
  - `mysql.general_log`
  - `mysql.slow_log`
  - This is easy as these are append only.
- Copy the tail of all transaction logs.
  - The tail of the InnoDB redo log (i.e. `ib_logfileN` files) will be copied for InnoDB tables.
  - The tail of the Aria redo log (i.e. `aria_log.N` files) will be copied for Aria tables.

`BACKUP STAGE BLOCK_COMMIT` in MariaDB Enterprise Server

Mariabackup from MariaDB Enterprise Server performs the following tasks in the `BLOCK_COMMIT` stage:

- Create a MyRocks checkpoint using the `rocksdb_create_checkpoint` system variable.
- Copy changes to system log tables.
  - `mysql.general_log`
  - `mysql.slow_log`
  - This is easy as these are append only.
- Copy changes to statistics tables.
  - `mysql.table_stats`
  - `mysql.column_stats`
  - `mysql.index_stats`
- Copy the tail of all transaction logs.
  - The tail of the InnoDB redo log (i.e. `ib_logfileN` files) will be copied for InnoDB tables.
  - The tail of the Aria redo log (i.e. `aria_log.N` files) will be copied for Aria tables.
- Save the binary log position to `xtrabackup_binlog_info`.
- Save the Galera Cluster state information to `xtrabackup_galera_info`.

`BACKUP STAGE END` in MariaDB Enterprise Server

Mariabackup from MariaDB Enterprise Server performs the following tasks in the `END` stage:

- Copy the MyRocks checkpoint into the backup.

2.3.4.13 mariabackup SST Method

The `mariabackup` SST method uses the Mariabackup utility for performing SSTs. It is one of the methods that does not block the donor node. Mariabackup was originally forked from Percona XtraBackup, and similarly, the `mariabackup` SST method was originally forked from the `xtrabackup-v2` SST method.

Note that if you use the `mariabackup` SST method, then you also need to have `socat` installed on the server. This is needed to stream the backup from the donor node to the joiner node. This is a limitation that was inherited from the `xtrabackup-v2` SST method.

Contents

Choosing Mariabackup for SSTs

To use the `mariabackup` SST method, you must set the `wsrep_sst_method=mariabackup` on both the donor and joiner node. It can be changed dynamically with `SET GLOBAL` on the node that you intend to be a SST donor. For example:

```
SET GLOBAL wsrep_sst_method='mariabackup';
```

It can be set in a server option group in an option file prior to starting up a node:

```
[mariadb]
...
wsrep_sst_method = mariabackup
```

For an SST to work properly, the donor and joiner node must use the same SST method. Therefore, it is recommended to set `wsrep_sst_method` to the same value on all nodes, since any node will usually be a donor or joiner node at some point.

Major version upgrades

The InnoDB redo log format has been changed in MariaDB 10.5 and MariaDB 10.8 in a way that will not allow the crash recovery or the preparation of a backup from an older major version. Because of this, the `mariabackup` SST method cannot be used for some major version upgrades, unless you temporarily edit the `wsrep_sst_mariabackup` script so that the `--prepare` step on the newer-major-version joiner will be executed using the older-major-version `mariabackup` tool.

The default method `wsrep_sst_method=rsync` will work for major version upgrades; see MDEV-27437.

Authentication and Privileges

To use the `mariabackup` SST method, Mariabackup needs to be able to authenticate locally on the donor node, so that it can create a backup to stream to the joiner. You can tell the donor node what username and password to use by setting the `wsrep_sst_auth` system variable. It can be changed dynamically with `SET GLOBAL` on the node that you intend to be a SST donor. For example:

```
SET GLOBAL wsrep_sst_auth = 'mariabackup:mypassword';
```

It can also be set in a server option group in an option file prior to starting up a node:

```
[mariadb]
...
wsrep_sst_auth = mariabackup:mypassword
```

Some authentication plugins do not require a password. For example, the `unix_socket` and `gssapi` authentication plugins do not require a password. If you are using a user account that does not require a password in order to log in, then you can just leave the password component of `wsrep_sst_auth` empty. For example:

```
[mariadb]
...
wsrep_sst_auth = mariabackup:
```

The user account that performs the backup for the SST needs to have the same privileges as Mariabackup, which are the `RELOAD`, `PROCESS`, `LOCK TABLES` and `REPLICATION CLIENT` global privileges. To be safe, you should ensure that these privileges are set on each node in your cluster. Mariabackup connects locally on the donor node to perform the backup, so the following user should be sufficient:

```
CREATE USER 'mariabackup'@'localhost' IDENTIFIED BY 'mypassword';
GRANT RELOAD, PROCESS, LOCK TABLES, REPLICATION CLIENT ON *.* TO 'mariabackup'@'localhost';
```

Passwordless Authentication - Unix Socket

It is possible to use the `unix_socket` authentication plugin for the user account that performs SSTs. This would provide the benefit of not needing to configure a plain-text password in `wsrep_sst_auth`.

The user account would have to have the same name as the operating system user account that is running the `mysqld` process. On many systems, this is the user account configured as the `user` option, and it tends to default to `mysql`.

For example, if the `unix_socket` authentication plugin is already installed, then you could execute the following to create the user account:

```
CREATE USER 'mysql'@'localhost' IDENTIFIED VIA unix_socket;
GRANT RELOAD, PROCESS, LOCK TABLES, REPLICATION CLIENT ON *.* TO 'mysql'@'localhost';
```

And then to configure `wsrep_sst_auth`, you could set the following in a server option group in an option file prior to starting up a node:

```
[mariadb]
...
wsrep_sst_auth = mysql:
```

Passwordless Authentication - GSSAPI

It is possible to use the `gssapi` authentication plugin for the user account that performs SSTs. This would provide the benefit of not needing to configure a plain-text password in `wsrep_sst_auth`.

The following steps would need to be done beforehand:

- You need a KDC running MIT Kerberos or Microsoft Active Directory.
- You will need to create a keytab file for the MariaDB server.
- You will need to install the package containing the `gssapi` authentication plugin.
- You will need to install the plugin in MariaDB, so that the `gssapi` authentication plugin is available to use.
- You will need to configure the plugin.
- You will need to create a user account that authenticates with the `gssapi` authentication plugin, so that the user account can be used for SSTs. This user account will need to correspond with a user account that exists on the backend KDC.

For example, you could execute the following to create the user account in MariaDB:

```
CREATE USER 'mariabackup'@'localhost' IDENTIFIED VIA gssapi;
GRANT RELOAD, PROCESS, LOCK TABLES, REPLICATION CLIENT ON *.* TO 'mariabackup'@'localhost';
```

And then to configure `wsrep_sst_auth`, you could set the following in a server option group in an option file prior to starting up a node:

```
[mariadb]
...
wsrep_sst_auth = mariabackup:
```

Choosing a Donor Node

When Mariabackup is used to create the backup for the SST on the donor node, Mariabackup briefly requires a system-wide lock at the end of the backup. In MariaDB 10.3 and before, this is done with `FLUSH TABLES WITH READ LOCK`. In MariaDB 10.4 and later, this is done with `BACKUP STAGE BLOCK_COMMIT`.

If a specific node in your cluster is acting as the *primary* node by receiving all of the application's write traffic, then this node should not usually be used as the donor node, because the system-wide lock could interfere with the application. In this case, you can define one or more preferred donor nodes by setting the `wsrep_sst_donor` system variable.

For example, let's say that we have a 5-node cluster with the nodes `node1`, `node2`, `node3`, `node4`, and `node5`, and let's say that `node1` is acting as the *primary* node. The preferred donor nodes for `node2` could be configured by setting the following in a server option group in an option file prior to starting up a node:

```
[mariadb]
...
wsrep_sst_donor=node3,node4,node5,
```

The trailing comma tells the server to allow any other node as donor when the preferred donors are not available. Therefore, if `node1` is the only node left in the cluster, the trailing comma allows it to be used as the donor node.

Socat Dependency

During the SST process, the donor node uses socat to stream the backup to the joiner node. Then the joiner node prepares the backup before restoring it. The socat utility must be installed on both the donor node and the joiner node in order for this to work. Otherwise, the MariaDB error log will contain an error like:

```
WSREP_SST: [ERROR] socat not found in path:
/usr/sbin:/sbin:/usr//bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin (20180122
14:55:32.993)
```

Installing Socat on RHEL/CentOS

On RHEL/CentOS, `socat` can be installed from the Extra Packages for Enterprise Linux (EPEL) repository.

TLS

This SST method supports two different TLS methods. The specific method can be selected by setting the `encrypt` option in the `[sst]` section of the MariaDB configuration file. The options are:

- TLS using OpenSSL encryption built into `socat` (`encrypt=2`)
- TLS using OpenSSL encryption with Galera-compatible certificates and keys (`encrypt=3`)

Note that `encrypt=1` refers to a TLS encryption method that has been deprecated and removed. `encrypt=4` refers to a TLS encryption method in `xtrabackup-v2` that has not yet been ported to `mariabackup`. See MDEV-18050 about that.

TLS Using OpenSSL Encryption Built into Socat

To generate keys compatible with this encryption method, you can follow these directions.

For example:

- First, generate the keys and certificates:

```
FILENAME=sst
openssl genrsa -out $FILENAME.key 1024
openssl req -new -key $FILENAME.key -x509 -days 3653 -out $FILENAME.crt
cat $FILENAME.key $FILENAME.crt >$FILENAME.pem
chmod 600 $FILENAME.key $FILENAME.pem
```

- On some systems, you may also have to add dhparams to the certificate:

```
openssl dhparam -out dhparams.pem 2048
cat dhparams.pem >> sst.pem
```

- Then, copy the certificate and keys to all nodes in the cluster.
- Then, configure the following on all nodes in the cluster:

```
[sst]
encrypt=2
tca=/etc/my.cnf.d/certificates/sst.crt
tcert=/etc/my.cnf.d/certificates/sst.pem
```

But replace the paths with whatever is relevant on your system.

This should allow your SSTs to be encrypted.

TLS Using OpenSSL Encryption with Galera-compatible Certificates and Keys

To generate keys compatible with this encryption method, you can follow these directions.

For example:

- First, generate the keys and certificates:

```
# CA
openssl genrsa 2048 > ca-key.pem
openssl req -new -x509 -nodes -days 365000 \
-key ca-key.pem -out ca-cert.pem

# server1
openssl req -newkey rsa:2048 -days 365000 \
-nodes -keyout server1-key.pem -out server1-req.pem
openssl rsa -in server1-key.pem -out server1-key.pem
openssl x509 -req -in server1-req.pem -days 365000 \
-CA ca-cert.pem -CAkey ca-key.pem -set_serial 01 \
-out server1-cert.pem
```

- Then, copy the certificate and keys to all nodes in the cluster.
- Then, configure the following on all nodes in the cluster:

```
[sst]
encrypt=3
tkey=/etc/my.cnf.d/certificates/server1-key.pem
tcert=/etc/my.cnf.d/certificates/server1-cert.pem
```

But replace the paths with whatever is relevant on your system.

This should allow your SSTs to be encrypted.

Logs

The `mariabackup` SST method has its own logging outside of the MariaDB Server logging.

Logging to SST Logs

MariaDB starting with 10.3.13

Starting with MariaDB 10.1.38, MariaDB 10.2.22, and MariaDB 10.3.13, logging for `mariabackup` SSTs works the following way.

By default, on the donor node, it logs to `mariabackup.backup.log`. This log file is located in the `datadir`.

By default, on the joiner node, it logs to `mariabackup.prepare.log` and `mariabackup.move.log` These log files are also located in the `datadir`.

By default, before a new SST is started, existing `mariabackup` SST log files are compressed and moved to `/tmp/sst_log_archive`. This behavior can be disabled by setting `sst-log-archive=0` in the `[sst]` option group

in an option file. Similarly, the archive directory can be changed by setting `sst-log-archive-dir`. For example:

```
[sst]
sst-log-archive=1
sst-log-archive-dir=/var/log/mysql/sst/
```

See MDEV-17973 for more information.

MariaDB until 10.3.13 MariaDB 10.2.22, and MariaDB 10.3.13, logging for `mariabackup` SSTs works the following way.

By default, on the donor node, it logs to `innobackup.backup.log`. This log file is located in the `datadir`.

By default, on the joiner node, it logs to `innobackup.prepare.log` and `innobackup.move.log`. These log files are located in the `.sst` directory, which is a hidden directory inside the `datadir`.

These log files are overwritten by each subsequent SST, so if an SST fails, it is best to copy them somewhere safe before starting another SST, so that the log files can be analyzed.

Logging to Syslog

You can redirect the SST logs to the syslog instead by setting the following in the `[sst]` option group in an option file:

```
[sst]
sst-syslog=1
```

You can also redirect the SST logs to the syslog by setting the following in the `[mysqld_safe]` option group in an option file:

```
[mysqld_safe]
syslog
```

Performing SSTs with IPv6 Addresses

If you are performing Mariabackup SSTs with IPv6 addresses, then the `socat` utility needs to be passed the `pf=ip6` option. This can be done by setting the `sockopt` option in the `[sst]` option group in an option file. For example:

```
[sst]
sockopt=",pf=ip6"
```

See MDEV-18797 for more information.

Manual SST with Mariabackup

In some cases, if Galera Cluster's automatic SSTs repeatedly fail, then it can be helpful to perform a "manual SST". See the following page on how to do that:

- Manual SST of Galera Cluster node with Mariabackup

3.2.8.3 Manual SST of Galera Cluster Node with Mariabackup

2.3.4.15 Individual Database Restores with MariaBackup from Full Backup

This method is to solve a flaw with Mariabackup; it cannot do single database restores from a full backup easily. There is a blog post that details a way to do this, but it's a manual process which is fine for a few tables but if you have hundreds or even thousands of tables then it would be impossible to do quickly.

Contents

We can't just move the data files to the datadir as the tables are not registered in the engines, so the database will error. Currently, the only effective method is to a do full restore in a test database and then dump the database that requires restoring or running a partial backup. **This has only been tested with InnoDB. Also, if you have stored procedures or triggers then these will need to be deleted and recreated.**

Some of the issues that this method overcomes:

- Tables not registered in the InnoDB engine so will error when you try to select from a table if you move the data files into the datadir
- Tables with foreign keys need to be created without keys, otherwise it will error when you discard the tablespace

Single Node

Below is the process to perform a single database restore.

Firstly, we will need the table structure from a mariadb-dump backup with the --no-data option. I recommend this is done at least once per day or every six hours via a cronjob. As it is just the structure, it will be very fast.

```
mariadb-dump -u root -p --all-databases --no-data > nodata.sql
```

Using SED to return only the table structure we require, then use vim or another text editor to make sure nothing is left.

```
sed -n '/Current Database: `DATABASENAME`/, /Current Database:/p' nodata.sql > trimednodata.sql
vim trimednodata.sql
```

I won't go over the backup process, as this is done earlier in other documents, such as full-backup-and-restore-with-mariabackup. Prepare the backup with any incremental-backup-and-restores that you have, and then run the following on the full backup folder using the --export option to generate files with .cfg extensions which InnoDB will look for.

```
Mariabackup --prepare --export --target-dir=/media/backups/fullbackupfolder
```

Once we have done these steps, we can then import the table structure. If you have used the --all-databases option, then you will need to either use SED or open it in a text editor and export out tables that you require. You will also need to log in to the database and create the database if the dump file doesn't. Run the following command below:

```
Mysql -u root -p schema_name < nodata.sql
```

Once the structure is in the database, we have now registered the tables to the engine. Next, we will run the following statements in the information_schema database, to export statements to import/discard table spaces and drop and create foreign keys which we will use later. (edit the CONSTRAINT_SCHEMA and TABLE_SCHEMA WHERE clause to the database you are restoring. Also, add the following lines after your SELECT and before the FROM to have MariaDB export the files to the OS)

```
SELECT ...
into outfile '/tmp/filename.sql'
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
FROM ...
```

The following are the statements that we will need later.

```
USE information_schema;
select concat("ALTER TABLE ",table_name," DISCARD TABLESPACE;")  AS discard_tablespace
from information_schema.tables
where TABLE_SCHEMA="DATABASENAME";

select concat("ALTER TABLE ",table_name," IMPORT TABLESPACE;") AS import_tablespace
from information_schema.tables
where TABLE_SCHEMA="DATABASENAME";

SELECT
concat ("ALTER TABLE ", rc.CONSTRAINT_SCHEMA, ".",rc.TABLE_NAME," DROP FOREIGN KEY ", rc.CONSTRAI
FROM REFERENTIAL_CONSTRAINTS AS rc
where CONSTRAINT_SCHEMA = 'DATABASENAME';

SELECT
CONCAT ("ALTER TABLE ",
KCU.CONSTRAINT_SCHEMA, ".",
KCU.TABLE_NAME,"
ADD CONSTRAINT ",
KCU.CONSTRAINT_NAME, "
FOREIGN KEY ", "
(`",KCU.COLUMN_NAME,"`)", "
REFERENCES `",REFERENCED_TABLE_NAME,"`
(`",REFERENCED_COLUMN_NAME,"`)" ,"
ON UPDATE " ,(SELECT UPDATE_RULE FROM REFERENTIAL_CONSTRAINTS WHERE CONSTRAINT_NAME = KCU.CONSTRA
ON DELETE ",(SELECT DELETE_RULE FROM REFERENTIAL_CONSTRAINTS WHERE CONSTRAINT_NAME = KCU.CONSTRAI
FROM KEY_COLUMN_USAGE AS KCU
WHERE KCU.CONSTRAINT_SCHEMA = 'DATABASENAME'
AND KCU.POSITION_IN_UNIQUE_CONSTRAINT >= 0
AND KCU.CONSTRAINT_NAME NOT LIKE 'PRIMARY';
```

Once we have run those statements, and they have been exported to a Linux directory or copied from a GUI interface.

Run the ALTER DROP KEYS statements in the database

```
ALTER TABLE schemaname.tablename DROP FOREIGN KEY key_name;
...
```

Once completed, run the DROP TABLE SPACE statements in the database

```
ALTER TABLE test DISCARD TABLESPACE;
...
```

Exit out the database and change into the directory of the full backup location. Run the following commands to copy all the .cfg and .ibd files to the datadir such as /var/lib/mysql/testdatabase (Change the datadir location if needed). Learn more about files that Mariabackup generates with files-created-by-mariabackup

```
cp *.cfg /var/lib/mysql
cp *.ibd /var/lib/mysql
```

After moving the files, it is very important that MySQL is the owner of the files, otherwise it won't have access to them and will error when we import the tablespaces.

```
sudo chown -R mysql:mysql /var/lib/mysql
```

Run the import table spaces statements in the database.

```
ALTER TABLE test IMPORT TABLESPACE;
...
```

Run the add key statements in the database

```
ALTER TABLE schmeaname.tablename ADD CONSTRAINT key_name FOREIGN KEY (`column_name`) REFERENCES `
...
```

We have successfully restored a single database. To test that this has worked, we can do a basic check on some tables.

```
use database
SELECT * from test limit 10;
```

Replica nodes

If you have a primary-replica set up, it would be best to follow the sets above for the primary node and then either take a full mariadb-dump or take a new full mariabackup and restore this to the replica. You can find more information about restoring a replica with mariabackup in Setting up a Replica with Mariabackup

After running the below command, copy to the replica and use the LESS linux command to grab the change master statement. Remember to follow this process: Stop replica > restore data > run CHANGE MASTER statement > start replica again.

```
mariadb-dump -u user -p --single-transaction --master-data=2 > fullbackup.sql
```

Please follow Setting up a Replica with Mariabackup on restoring a replica with Mariabackup

```
$ mariabackup --backup \
   --slave-info --safe-slave-backup \
   --target-dir=/var/mariadb/backup/ \
   --user=mariabackup --password=mypassword
```

Galera cluster

For this process to work with Galera cluster, we first need to understand that some statements are not replicated across Galera nodes. One of which is the DISCARD and IMPORT for ALTER TABLES statements, and these statements will need to be ran on all nodes. We also need to run the OS level steps on each server as seen below.

Run the ALTER DROP KEYS statements on ONE NODE as these are replicated.

```
ALTER TABLE schemaname.tablename DROP FOREIGN KEY key_name;
...
```

Once completed, run the DROP TABLE SPACE statements on EVERY NODE, as these are not replicated.

```
ALTER TABLE test DISCARD TABLESPACE;
...
```

Exit out the database and change into the directory of the full backup location. Run the following commands to copy all the .cfg and .ibd files to the datadir such as /var/lib/mysql/testdatabase (Change the datadir location if needed). Learn more about files that Mariabackup generates with files-created-by-mariabackup. This step needs to be done on all nodes. You will need to copy the backup files to each node, we can use the same backup on all nodes.

```
cp *.cfg /var/lib/mysql
cp *.ibd /var/lib/mysql
```

After moving the files, it is very important that MySQL is the owner of the files, otherwise it won't have access to them and will error when we import the tablespaces.

```
sudo chown -R mysql:mysql /var/lib/mysql
```

Run the import table spaces statements on EVERY NODE.

```
ALTER TABLE test IMPORT TABLESPACE;
...
```

Run the add key statements on ONE NODE

```
ALTER TABLE schmeaname.tablename ADD CONSTRAINT key_name FOREIGN KEY (`column_name`) REFERENCES `
...
```

1.3.6.3 mariadb-hotcopy

2.4 Server Monitoring & Logs

MariaDB can keep a number of log files, including the error log, the binary log, the general query log and the slow query log.

Overview of MariaDB Logs

What to log and what not to log.

Error Log

Record of critical errors that occurred during the server's operation.

Setting the Language for Error Messages

Specifying the language for the server error messages.

General Query Log

Log of every SQL query received from a client, as well as connects/disconnects.

Slow Query Log

Logging slow queries

Rotating Logs on Unix and Linux

Rotating logs on Unix and Linux with logrotate.

Binary Log

Contains a record of all changes to the databases, both data and structure

InnoDB Redo Log

The redo log is used by InnoDB during crash recovery.

InnoDB Undo Log

InnoDB Undo log.

MyISAM Log

Records all changes to MyISAM tables

Transaction Coordinator Log

The transaction coordinator log (tc_log) is used to coordinate transactions...

SQL Error Log Plugin

Records SQL-level errors to a log file.

Writing Logs Into Tables

The general query log and the slow query log can be written into system tables

Performance Schema

Monitoring server performance.

MariaDB Audit Plugin

Logging user activity with the MariaDB Audit Plugin.

There are 4 related questions.

2.4.1 Overview of MariaDB Logs

There are many variables in MariaDB that you can use to define what to log and when to log.

This article will give you an overview of the different logs and how to enable/disable logging to these.

Note that storage engines can have their logs too: for example, InnoDB keeps an Undo Log and a Redo Log which are used for rollback and crash recovery. However, this page only lists MariaDB server logs.

Error Log

- Always enabled
- Usually a file in the data directory, but some distributions may move this to other locations.
- All critical errors are logged here.
- One can get warnings to be logged by setting log_warnings.
- With the mysqld_safe `--syslog` option one can duplicate the messages to the system's syslog.

General Query Log

- Enabled with --general-log
- Logs all queries to a file or table.
- Useful for debugging or auditing queries.
- The super user can disable logging to it for a connection by setting SQL_LOG_OFF to 1.

Slow Query Log

- Enabled by starting mysqld with --slow-query-log
- Logs all queries to a file or table.
- Useful to find queries that causes performance problems.
- Logs all queries that takes more than long_query_time to run.
- One can decide what to log with the options --log-slow-admin-statements, --log-slow-slave-statements, log_slow_filter or log_slow_rate_limit.
- One can change what is logged by setting log_slow_verbosity.
- One can disable it globally by setting global.slow_query_log to 0
- In 10.1 one can disable it for a connection by setting local.slow_query_log to 0.

Binary Log

- Enabled by starting mysqld with --log-bin
- Used on machines that are, or may become, replication masters.
- Required for point-in-time recovery.
- Binary log files are mainly used by replication and can also be used with mariadb-binlog to apply on a backup to get the database up to date.
- One can decide what to log with --binlog-ignore-db=database_name or --binlog-do-db=database_name.
- The super user can disable logging for a connection by setting SQL_LOG_BIN to 0. However while this is 0, no changes done in this connection will be replicated to the slaves!
- For examples, see Using and Maintaining the Binary Log.

Examples

If you know that your next query will be slow and you don't want to log it in the slow query log, do:

```
SET LOCAL SLOW_QUERY_LOG=0;
```

If you are a super user running a log batch job that you don't want to have logged (for example mariadb-dump), do:

```
SET LOCAL SQL_LOG_OFF=1, LOCAL SLOW_QUERY_LOG=0;
```

mariadb-dump (previously mysqldump) since MariaDB 10.1 will add this automatically to your dump file if you run it with the `--skip-log-queries` option.

2.4.2 Error Log

Contents

The error log contains a record of critical errors that occurred during the server's operation, table corruption, start and stop information.

SQL errors can also be logged in a separate file using the SQL_ERROR_LOG plugin.

Configuring the Error Log Output Destination

MariaDB always writes its error log, but the destination is configurable.

Writing the Error Log to a File

To configure the error log to be written to a file, you can set the log_error system variable. You can configure a specific file name. However, if a specific file name is not configured, then the log will be written to the `${hostname}.err` file in the datadir directory by default.

The log_error system variable can be set in a server option group in an option file prior to starting up the server. For example, to write the error log to the default `${hostname}.err` file, you could configure the following:

```
[mariadb]
...
log_error
```

If you configure a specific file name as the log_error system variable, and if it is not an absolute path, then it will be relative to the datadir directory. For example, if you configured the following, then the error log would be written to `mariadb.err` in the datadir directory:

```
[mariadb]
...
log_error=mariadb.err
```

If it is a relative path, then the log_error is relative to the datadir directory.

However, the log_error system variable can also be an absolute path. For example:

```
[mariadb]
...
log_error=/var/log/mysql/mariadb.err
```

Another way to configure the error log file name is to set the log-basename option, which configures MariaDB to use a common prefix for all log files (e.g. general query log, slow query log, error log, binary logs, etc.). The error log file name will be built by adding a `.err` extension to this prefix. For example, if you configured the following, then the error log would still be written to `mariadb.err` in the datadir directory:

```
[mariadb]
...
log-basename=mariadb
log_error
```

The log-basename cannot be an absolute path. The log file name is relative to the datadir directory.

Writing the Error Log to Stderr on Unix

On Unix, if the log_error system variable is not set, then errors are written to `stderr`, which usually means that the log messages are output to the terminal that started `mysqld`.

If the log_error system variable was set in an option file or on the command-line, then it can still be unset by specifying `--skip-log-error`.

Writing the Error Log to Syslog on Unix

On Unix, the error log can also be redirected to the syslog. How this is done depends on how you start MariaDB.

Syslog with mysqld_safe

If you start MariaDB with mysqld_safe, then the error log can be redirected to the syslog. See mysqld_safe: Configuring MariaDB to Write the Error Log to Syslog for more information.

Syslog with Systemd

If you start MariaDB with systemd, then the error log can also be redirected to the syslog. See Systemd: Configuring MariaDB to Write the Error Log to Syslog for more information.

systemd also has its own logging system called the `journal`, and some errors may get logged there instead. See Systemd:Systemd Journal for more information.

Writing the Error Log to Console on Windows

On Windows, if the console option is specified, and if the log_error system variable is not used, then errors are written to the console. If both options are specified, then the last option takes precedence.

Writing the Error Log to the Windows Event Viewer

On Windows, error log messages are also written to the Windows Event Viewer. You can find MariaDB's error log messages by browsing **Windows Logs**, and then selecting **Application** or **Application Log**, depending on the Windows version.

In MariaDB 10.3 and before, you can find MariaDB's error log messages by searching for the **Source** `MySQL`.

In MariaDB 10.4 and later, you can find MariaDB's error log messages by searching for the **Source** `MariaDB`.

Finding the Error Log

To find where the error log is stored, one can find the options used for the error log with:

```
mariadbd --print-defaults
```

or

```
my_print_defaults --mysqld | grep log-error
```

If the above don't help, check also if your system is set to write to syslog, in which case you need to use journalctl to access it.

Configuring the Error Log Verbosity

The default value of the log_warnings system variable is `2`.

The log_warnings system variable can be used to configure the verbosity of the error log. It can be changed dynamically with SET GLOBAL. For example:

```
SET GLOBAL log_warnings=3;
```

It can also be set either on the command-line or in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
log_warnings=3
```

Some of the warnings included in each verbosity level are described below.

> The log_warnings system variable only has an effect on some log messages. Some log messages are **always** written to the error log, regardless of the error log verbosity. For example, most warnings from the InnoDB storage engine are not affected by log_warnings. For a complete list of log messages affected by log_warnings, see the description of the log_warnings system variable.

Verbosity Level 0

If log_warnings is `0`, then many optional warnings will not be logged. However, this does not prevent all warnings from being logged, because there are certain core warnings that will always be written to the error log. For example:

- If InnoDB strict mode is disabled, and if DDL is performed on a table that triggers a "Row size too large" error, then InnoDB will log a warning:

```
[Warning] InnoDB: Cannot add field col25 in table db1.tab because after
  adding it, the row size is 8477 which is greater than maximum allowed
  size (8126) for a record on index leaf page.
```

However, if InnoDB strict mode is enabled, then the same message will be logged as an error.

Verbosity Level 1

Default until MariaDB 10.2.3. If log_warnings is `1`, then many types of warnings are logged. Some useful warnings are:

- Replication-related messages:

```
[Note] Error reading relay log event: slave SQL thread was killed
[Note] Slave SQL thread exiting, replication stopped in log
  'dbserver-2-bin.000033'   at position 181420;
  GTID position '0-263316466-368886'
[Note] Slave I/O thread exiting, read up to log
  'dbserver-2-bin.000034', position 642;
  GTID position 0-263316466-368887
```

- Messages related to DNS lookup failures:

```
[Warning] IP address '192.168.1.193'
  could not be resolved: Name or service not known
```

- Messages related to the event scheduler:

```
[Note] Event Scheduler: Loaded 0 events
```

- Messages related to unsafe statements for statement-based replication:

```
[Warning] Unsafe statement written to the binary log using statement format since
  BINLOG_FORMAT = STATEMENT. The statement is unsafe because
  it uses a LIMIT clause. This
  is unsafe because the set of rows included cannot be predicted.
```

Frequent warnings about unsafe statements for statement-based replication can cause the error log to grow very large.

MariaDB will automatically detect frequent duplicate warnings about unsafe statements for statement-based replication. After 10 identical warnings are detected, MariaDB will prevent that same warning from being written to the error log again for the next 5 minutes.

Verbosity Level 2

Default from MariaDB 10.2.4. If log_warnings is `2`, then a couple other different kinds of warnings are printed. For example:

- Messages related to access denied errors:

```
[Warning] Access denied for user 'root'@'localhost' (using password: YES)
```

- Messages related to connections that are aborted due to errors or timeouts:

```
[Warning] Aborted connection 35 to db: 'unconnected' user:
  'user1@host1' host: '192.168.1.40' (Got an error writing communication packets)
[Warning] Aborted connection 36 to db: 'unconnected' user:
  'user1@host2' host: '192.168.1.230' (Got an error writing communication packets)
[Warning] Aborted connection 38 to db: 'db1' user:
  'user2' host: '192.168.1.60' (Unknown error)
[Warning] Aborted connection 51 to db: 'db1' user:
  'user2' host: '192.168.1.50' (Got an error reading communication packets)
[Warning] Aborted connection 52 to db: 'db1' user:
  'user3' host: '192.168.1.53' (Got timeout reading communication packets)
```

- Messages related to table handler errors:

```
[Warning] Can't find record in 'tab1'.
[Warning] Can't write; duplicate key in table 'tab1'.
[Warning] Lock wait timeout exceeded; try restarting transaction.
[Warning] The number of locks exceeds the lock table size.
[Warning] Update locks cannot be acquired during a READ UNCOMMITTED transaction.
```

- Messages related to the files used to persist replication state:
  - Either the default `master.info` file or the file that is configured by the master_info_file option.
  - Either the default `relay-log.info` file or the file that is configured by the relay_log_info_file system variable.

```
[Note] Reading Master_info: '/mariadb/data/master.info'
  Relay_info:'/mariadb/data/relay-log.info'
[Note] Initialized Master_info from '/mariadb/data/master.info'
[Note] Reading of all Master_info entries succeded
[Note] Deleted Master_info file '/mariadb/data/master.info'.
[Note] Deleted Master_info file '/mariadb/data/relay-log.info'.
```

- Messages about a master's binary log dump thread:

```
[Note] Start binlog_dump to slave_server(263316466), pos(, 4)
```

Verbosity Level 3

If log_warnings is `3`, then a couple other different kinds of warnings are printed. For example:

- Messages related to old-style language options:

```
[Warning] An old style --language value with language specific
  part detected: /usr/local/mysql/data/
[Warning] Use --lc-messages-dir without language specific part instead.
```

- Messages related to progress of InnoDB online DDL:

```
[Note] InnoDB: Online DDL : Start
[Note] InnoDB: Online DDL : Start reading clustered index of the table and
  create temporary files
[Note] InnoDB: Online DDL : End of reading clustered index of the table and
  create temporary files
[Note] InnoDB: Online DDL : Start merge-sorting index PRIMARY (1 / 3),
  estimated cost : 18.0263
[Note] InnoDB: Online DDL : merge-sorting has estimated 33 runs
[Note] InnoDB: Online DDL : merge-sorting current run 1 estimated 33 runs
[Note] InnoDB: Online DDL : merge-sorting current run 2 estimated 17 runs
[Note] InnoDB: Online DDL : merge-sorting current run 3 estimated 9 runs
[Note] InnoDB: Online DDL : merge-sorting current run 4 estimated 5 runs
[Note] InnoDB: Online DDL : merge-sorting current run 5 estimated 3 runs
[Note] InnoDB: Online DDL : merge-sorting current run 6 estimated 2 runs
[Note] InnoDB: Online DDL : End of  merge-sorting index PRIMARY (1 / 3)
[Note] InnoDB: Online DDL : Start building index PRIMARY (1 / 3),
  estimated cost : 27.0395
[Note] InnoDB: Online DDL : End of building index PRIMARY (1 / 3)
[Note] InnoDB: Online DDL : Completed
[Note] InnoDB: Online DDL : Start merge-sorting index ux1 (2 / 3),
  estimated cost : 5.7895
[Note] InnoDB: Online DDL : merge-sorting has estimated 2 runs
[Note] InnoDB: Online DDL : merge-sorting current run 1 estimated 2 runs
[Note] InnoDB: Online DDL : End of  merge-sorting index ux1 (2 / 3)
[Note] InnoDB: Online DDL : Start building index ux1 (2 / 3),
  estimated cost : 8.6842
[Note] InnoDB: Online DDL : End of building index ux1 (2 / 3)
[Note] InnoDB: Online DDL : Completed
[Note] InnoDB: Online DDL : Start merge-sorting index ix1 (3 / 3),
  estimated cost : 6.1842
[Note] InnoDB: Online DDL : merge-sorting has estimated 3 runs
[Note] InnoDB: Online DDL : merge-sorting current run 1 estimated 3 runs
[Note] InnoDB: Online DDL : merge-sorting current run 2 estimated 2 runs
[Note] InnoDB: Online DDL : End of  merge-sorting index ix1 (3 / 3)
[Note] InnoDB: Online DDL : Start building index ix1 (3 / 3),
  estimated cost : 9.2763
[Note] InnoDB: Online DDL : End of building index ix1 (3 / 3)
[Note] InnoDB: Online DDL : Completed
```

Verbosity Level 4

If log_warnings is `4`, then a couple other different kinds of warnings are printed. For example:

- Messages related to killed connections:

```
[Warning] Aborted connection 53 to db: 'db1' user:
  'user2' host: '192.168.1.50' (KILLED)
```

- Messages related to **all** closed connections:

```
[Warning] Aborted connection 56 to db: 'db1' user:
  'user2' host: '192.168.1.50' (CLOSE_CONNECTION)
```

- Messages related to released connections, such as when a transaction is committed and completion_type is set to `RELEASE`:

```
[Warning] Aborted connection 58 to db: 'db1' user:
  'user2' host: '192.168.1.50' (RELEASE)
```

Verbosity Level 9

If log_warnings is `9`, then some **very** verbose warnings are printed. For example:

- Messages about initializing plugins:

```
[Note] Initializing built-in plugins
[Note] Initializing plugins specified on the command line
[Note] Initializing installed plugins
```

MySQL's log_error_verbosity

MariaDB does not support the log_error_verbosity system variable added in MySQL 5.7.

Format

The format consists of the date (yyyy-mm-dd) and time, the thread ID, followed by the type of error (Note, Warning or Error) and the error message, for example:

```
2016-06-15 16:53:33 139651251140544 [Note] InnoDB:
  The InnoDB memory heap is disabled
```

Until MariaDB 10.1.4, the format only consisted of the date (yymmdd) and time, followed by the type of error (Note, Warning or Error) and the error message, for example:

```
160615 16:53:08 [Note] InnoDB: The InnoDB memory heap is disabled
```

Rotating the Error Log on Unix and Linux

Unix and Linux distributions offer the logrotate utility, which makes it very easy to rotate log files. See Rotating Logs on Unix and Linux for more information on how to use this utility to rotate the error log.

Error Messages File

Many error messages are ready from an error messages file that contains localized error messages. If the server can't find this file when it starts up, then you might see errors like the following:

```
[ERROR] Can't find messagefile '/usr/share/errmsg.sys'
```

If this error is occurring because the file is in a custom location, then you can configure this location by setting the lc_messages_dir system variable either on the command-line or in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
lc_messages_dir=/usr/share/mysql/
```

If you want to use a different locale for error messages, then you can also set the lc_messages system variable. For example:

```
[mariadb]
...
lc_messages_dir=/usr/share/mysql/
lc_messages=en_US
```

See Setting the Language for Error Messages for more information.

2.4.3 Setting the Language for Error Messages

Contents

MariaDB server error messages are by default in English. However, MariaDB server also supports error message localization in many different languages. Each supported language has its own version of the error message file called `errmsg.sys` in a dedicated directory for that language.

Supported Languages for Error Messages

Error message localization is supported for the following languages:

- Bulgarian
- Chinese (from MariaDB 10.4.25, 10.5.16, 10.6.8, 10.7.4, 10.8.3)
- Czech
- Danish
- Dutch
- English
- Estonian
- French
- Georgian (from MariaDB 10.11.3)
- German
- Greek
- Hindi
- Hungarian
- Italian
- Japanese
- Korean
- Norwegian
- Norwegian-ny (Nynorsk)
- Polish
- Portuguese
- Romanian
- Russian
- Serbian
- Slovak
- Spanish
- Swahili (from MariaDB 11.1.2)
- Swedish
- Ukrainian

Setting the `lc_messages` and `lc_messages_dir` System Variables

The lc_messages and lc_messages_dir system variables can be used to set the server locale used for error messages.

The lc_messages system variable can be specified as a locale name. The language of the associated locale will be used for error messages. See Server Locales for a list of supported locales and their associated languages.

The lc_messages system variable is set to `en_US` by default, which means that error messages are in English by default.

If the `lc_messages` system variable is set to a valid locale name, but the server can't find an error message file for the language associated with the locale, then the default language will be used instead.

This system variable can be specified as command-line arguments to mariadbd or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
lc_messages=fr_CA
```

The lc_messages system variable can also be changed dynamically with SET GLOBAL. For example:

```
SET GLOBAL lc_messages='fr_CA';
```

If a server has the lc_messages system variable set to the `fr_CA` locale like the above example, then error messages would be in French. For example:

```
SELECT blah;
ERROR 1054 (42S22): Champ 'blah' inconnu dans field list
```

The lc_messages_dir system variable can be specified either as the path to the directory storing the server's error message files or as the path to the directory storing the specific language's error message file.

The server initially tries to interpret the value of the `lc_messages_dir` system variable as a path to the directory storing the server's error message files. Therefore, it constructs the path to the language's error message file by concatenating the value of the lc_messages_dir system variable with the language name of the locale specified by the lc_messages system

variable .

If the server does not find the error message file for the language, then it tries to interpret the value of the lc_messages_dir system variable as a direct path to the directory storing the specific language's error message file.

This system variable can be specified as command-line arguments to mariadbd or it can be specified in a relevant server option group in an option file.

For example, to specify the path to the directory storing the server's error message files:

```
[mariadb]
...
lc_messages_dir=/usr/share/mysql/
```

Or to specify the path to the directory storing the specific language's error message file:

```
[mariadb]
...
lc_messages_dir=/usr/share/mysql/french/
```

The `lc_messages_dir` system variable can not be changed dynamically.

Setting the --language Option

The --language option can also be used to set the server's language for error messages, but it is deprecated. It is recommended to set the lc_messages system variable instead.

The --language option can be specified either as a language name or as the path to the directory storing the language's error message file. See Server Locales for a list of supported locales and their associated languages.

This option can be specified as command-line arguments to mariadbd or it can be specified in a relevant server option group in an option file.

For example, to specify a language name:

```
[mariadb]
...
language=french
```

Or to specify the path to the directory storing the language's error message file:

```
[mariadb]
...
language=/usr/share/mysql/french/
```

Character Set

The character set that the error messages are returned in is determined by the character_set_results variable, which defaults to UTF8.

2.4.4 General Query Log

Contents

The general query log is a log of every SQL query received from a client, as well as each client connect and disconnect. Since it's a record of every query received by the server, it can grow large quite quickly.

However, if you only want a record of queries that change data, it might be better to use the binary log instead. One important difference is that the binary log only logs a query when the transaction is committed by the server, but the general

query log logs a query immediately when it is received by the server.

Enabling the General Query Log

The general query log is disabled by default.

To enable the general query log, set the `general_log` system variable to `1`. It can be changed dynamically with `SET GLOBAL`. For example:

```
SET GLOBAL general_log=1;
```

It can also be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
general_log
```

Configuring the General Query Log Filename

By default, the general query log is written to `${hostname}.log` in the `datadir` directory. However, this can be changed.

One way to configure the general query log filename is to set the `general_log_file` system variable. It can be changed dynamically with `SET GLOBAL`. For example:

```
SET GLOBAL general_log_file='mariadb.log';
```

It can also be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
general_log
general_log_file=mariadb.log
```

If it is a relative path, then the `general_log_file` is relative to the `datadir` directory.

However, the `general_log_file` system variable can also be an absolute path. For example:

```
[mariadb]
...
general_log
general_log_file=/var/log/mysql/mariadb.log
```

Another way to configure the general query log filename is to set the `log-basename` option, which configures MariaDB to use a common prefix for all log files (e.g. general query log, slow query log, error log, binary logs, etc.). The general query log filename will be built by adding a `.log` extension to this prefix. This option cannot be set dynamically. It can be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
log-basename=mariadb
general_log
```

The `log-basename` cannot be an absolute path. The log file name is relative to the `datadir` directory.

Choosing the General Query Log Output Destination

The general query log can either be written to a file on disk, or it can be written to the `general_log` table in the `mysql` database. To choose the general query log output destination, set the `log_output` system variable.

Writing the General Query Log to a File

The general query log is output to a file by default. However, it can be explicitly chosen by setting the `log_output` system variable to `FILE`. It can be changed dynamically with `SET GLOBAL`. For example:

```
SET GLOBAL log_output='FILE';
```

It can also be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
log_output=FILE
general_log
general_log_file=queries.log
```

Writing the General Query Log to a Table

The general query log can either be written to the `general_log` table in the `mysql` database by setting the `log_output` system variable to `TABLE`. It can be changed dynamically with `SET GLOBAL`. For example:

```
SET GLOBAL log_output='TABLE';
```

It can also be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
log_output=TABLE
general_log
```

Some rows in this table might look like this:

```
SELECT * FROM mysql.general_log\G
*************************** 1. row ***************************
  event_time: 2014-11-11 08:40:04.117177
   user_host: root[root] @ localhost []
   thread_id: 74
   server_id: 1
command_type: Query
    argument: SELECT * FROM test.s
*************************** 2. row ***************************
  event_time: 2014-11-11 08:40:10.501131
   user_host: root[root] @ localhost []
   thread_id: 74
   server_id: 1
command_type: Query
    argument: SELECT * FROM mysql.general_log
...
```

See Writing logs into tables for more information.

Disabling the General Query Log for a Session

A user with the SUPER privilege can disable logging to the general query log for a connection by setting the SQL_LOG_OFF system variable to `1`. For example:

```
SET SESSION SQL_LOG_OFF=1;
```

Disabling the General Query Log for Specific Statements

In MariaDB 10.3.1 and later, it is possible to disable logging to the general query log for specific types of statements by setting the `log_disabled_statements` system variable. This option cannot be set dynamically. It can be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
log_output=FILE
general_log
general_log_file=queries.log
log_disabled_statements='slave,sp'
```

Rotating the General Query Log on Unix and Linux

Unix and Linux distributions offer the logrotate utility, which makes it very easy to rotate log files. See Rotating Logs on Unix and Linux for more information on how to use this utility to rotate the general query log.

2.4.5 Slow Query Log

The slow query log is a record of SQL queries that took a long time to perform.

Slow Query Log Overview

A record of SQL queries that took a long time to perform.

Slow Query Log Extended Statistics

The slow query log makes extended statistics available.

mysqldumpslow

Symlink or old name for mariadb-dumpslow.

EXPLAIN in the Slow Query Log

EXPLAIN output in the slow query log.

mysql.slow_log Table

Contents of the slow query log if written to table.

There are 1 related questions.

2.4.5.1 Slow Query Log Overview

Contents

The slow query log is a record of SQL queries that took a long time to perform.

Note that, if your queries contain user's passwords, the slow query log may contain passwords too. Thus, it should be protected.

The number of rows affected by the slow query are also recorded in the slow query log.

Enabling the Slow Query Log

The slow query log is disabled by default.

To enable the slow query log, set the slow_query_log system variable to `1`. It can be changed dynamically with SET GLOBAL. For example:

```
SET GLOBAL slow_query_log=1;
```

It can also be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
slow_query_log
```

Configuring the Slow Query Log Filename

By default, the slow query log is written to `${hostname}-slow.log` in the datadir directory. However, this can be changed.

One way to configure the slow query log filename is to set the slow_query_log_file system variable. It can be changed dynamically with SET GLOBAL. For example:

```
SET GLOBAL slow_query_log_file='mariadb-slow.log';
```

It can also be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
slow_query_log
slow_query_log_file=mariadb-slow.log
```

If it is a relative path, then the slow_query_log_file is relative to the datadir directory.

However, the slow_query_log_file system variable can also be an absolute path. For example:

```
[mariadb]
...
slow_query_log
slow_query_log_file=/var/log/mysql/mariadb-slow.log
```

Another way to configure the slow query log filename is to set the log-basename option, which configures MariaDB to use a common prefix for all log files (e.g. slow query log, general query log, error log, binary logs, etc.). The slow query log filename will be built by adding `-slow.log` to this prefix. This option cannot be set dynamically. It can be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
log-basename=mariadb
slow_query_log
```

The log-basename cannot be an absolute path. The log file name is relative to the datadir directory.

Choosing the Slow Query Log Output Destination

The slow query log can either be written to a file on disk, or it can be written to the slow_log table in the mysql database. To choose the slow query log output destination, set the log_output system variable.

Writing the Slow Query Log to a File

The slow query log is output to a file by default. However, it can be explicitly chosen by setting the log_output system variable to `FILE`. It can be changed dynamically with SET GLOBAL. For example:

```
SET GLOBAL log_output='FILE';
```

It can also be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
log_output=FILE
slow_query_log
slow_query_log_file=slow-queries.log
```

Writing the Slow Query Log to a Table

The slow query log can either be written to the slow_log table in the mysql database by setting the log_output system variable to `TABLE`. It can be changed dynamically with SET GLOBAL. For example:

```
SET GLOBAL log_output='TABLE';
```

It can also be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
log_output=TABLE
slow_query_log
```

Some rows in this table might look like this:

```
SELECT * FROM mysql.slow_log\G
...
*************************** 2. row ***************************
    start_time: 2014-11-11 07:56:28.721519
     user_host: root[root] @ localhost []
    query_time: 00:00:12.000215
     lock_time: 00:00:00.000000
     rows_sent: 1
 rows_examined: 0
            db: test
last_insert_id: 0
     insert_id: 0
     server_id: 1
      sql_text: SELECT SLEEP(12)
     thread_id: 74
...
```

See Writing logs into tables for more information.

Disabling the Slow Query Log for a Session

A user can disable logging to the slow query log for a connection by setting the slow_query_log system variable to `0`. For example:

```
SET SESSION slow_query_log=0;
```

Disabling the Slow Query Log for Specific Statements

In MariaDB 10.3.1 and later, it is possible to disable logging to the slow query log for specific types of statements by setting the log_slow_disabled_statements system variable. This option cannot be set dynamically. It can be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
log_output=FILE
general_log
general_log_file=queries.log
log_slow_disabled_statements='admin,call,slave,sp'
```

Configuring the Slow Query Log Time

The time that defines a slow query can be configured by setting the long_query_time system variable. It uses a units of seconds, with an optional milliseconds component. The default value is `10`. It can be changed dynamically with SET GLOBAL. For example:

```
SET GLOBAL long_query_time=5.0;
```

It can also be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
log_output=FILE
slow_query_log
slow_query_log_file=slow-queries.log
long_query_time=5.0
```

Logging Queries That Don't Use Indexes

It can be beneficial to log queries that don't use indexes to the slow query log, since queries that don't use indexes can usually be optimized either by adding an index or by doing a slight rewrite. The slow query log can be configured to log queries that don't use indexes regardless of their execution time by setting the log_queries_not_using_indexes system variable. It can be changed dynamically with SET GLOBAL. For example:

```
SET GLOBAL log_queries_not_using_indexes=ON;
```

It can also be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
log_output=FILE
slow_query_log
slow_query_log_file=slow-queries.log
long_query_time=5.0
log_queries_not_using_indexes=ON
```

As a significant number of queries can run quickly even without indexes, you can use the min_examined_row_limit system variable with log_queries_not_using_indexes to limit the logged queries to those having a material impact on the server.

Logging Queries That Examine a Minimum Row Limit

It can be beneficial to log queries that examine a minimum number of rows. The slow query log can be configured to log queries that examine a minimum number of rows regardless of their execution time by setting the min_examined_row_limit system variable. It can be changed dynamically with SET GLOBAL. For example:

```
SET GLOBAL min_examined_row_limit=100000;
```

It can also be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
log_output=FILE
slow_query_log
slow_query_log_file=slow-queries.log
long_query_time=5.0
min_examined_row_limit=100000
```

Logging Slow Administrative Statements

By default, the Slow Query Log only logs slow non-administrative statements. To log administrative statements, set the log_slow_admin_statements system variable. The Slow Query Log considers the following statements administrative: ALTER TABLE, ANALYZE TABLE, CHECK TABLE, CREATE INDEX, DROP INDEX, OPTIMIZE TABLE, and REPAIR TABLE. In MariaDB 10.3 and later, this also includes ALTER SEQUENCE statements.

You can dynamically enable this feature using a SET GLOBAL statement. For example:

```
SET GLOBAL log_slow_admin_statements=ON;
```

It can also be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
log_output=FILE
slow_query_log
slow_query_log_file=slow-queries.log
long_query_time=5.0
log_slow_admin_statements=ON
```

Enabling the Slow Query Log for Specific Criteria

It is possible to enable logging to the slow query log for queries that meet specific criteria by configuring the log_slow_filter system variable. It can be changed dynamically with SET GLOBAL. For example:

```
SET GLOBAL log_slow_filter='filesort,filesort_on_disk,tmp_table,tmp_table_on_disk';
```

It can also be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
log_output=FILE
slow_query_log
slow_query_log_file=slow-queries.log
long_query_time=5.0
log_slow_filter=filesort,filesort_on_disk,tmp_table,tmp_table_on_disk
```

Throttling the Slow Query Log

The slow query log can create a lot of I/O, so it can be beneficial to throttle it in some cases. The slow query log can be throttled by configuring the log_slow_rate_limit system variable. It can be changed dynamically with SET GLOBAL. For example:

```
SET GLOBAL log_slow_rate_limit=5;
```

It can also be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
log_output=FILE
slow_query_log
slow_query_log_file=slow-queries.log
long_query_time=5.0
log_slow_rate_limit=5
```

Configuring the Slow Query Log Verbosity

There are a few optional pieces of information that can be included in the slow query log for each query. This optional information can be included by configuring the log_slow_verbosity system variable. It can be changed dynamically with SET GLOBAL. For example:

```
SET GLOBAL log_slow_verbosity='full';
```

It can also be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
log_output=FILE
slow_query_log
slow_query_log_file=slow-queries.log
long_query_time=5.0
log_slow_verbosity=query_plan,explain,engine
```

It is possible to have EXPLAIN output printed in the slow query log.

Viewing the Slow Query Log

Slow query logs written to file can be viewed with any text editor, or you can use the mariadb-dumpslow tool to ease the process by summarizing the information.

Queries that you find in the log are key queries to try to optimize by constructing a more efficient query or by making better use of indexes.

For queries that appear in the log that cannot be optimized in the above ways, perhaps because they are simply very large selects, due to slow hardware, or very high lock/cpu/io contention, using shard/clustering/load balancing solutions, better hardware, or stats tables may help to improve these queries.

Slow query logs written to table can be viewed by querying the slow_log table.

Variables Related to the Slow Query Log

- slow_query_log - enable/disable the slow query log. Renamed to log_slow_query from MariaDB 10.11.0.
- log_output - how the output will be written
- slow_query_log_file - name of the slow query log file. Renamed to log_slow_query_file_name from MariaDB 10.11.0.
- long_query_time - time in seconds/microseconds defining a slow query. Renamed to log_slow_query_time from MariaDB 10.11.0.
- log_queries_not_using_indexes - whether to log queries that don't use indexes
- log_slow_admin_statements - whether to log certain admin statements
- log_slow_disabled_statements - types of statements that should not be logged in the slow query log
- min_examined_row_limit - minimum rows a query must examine to be slow. Renamed to log_slow_min_examined_row_limit from MariaDB 10.11.0.
- log_slow_rate_limit - permits a fraction of slow queries to be logged
- log_slow_verbosity - amount of detail in the log
- log_slow_filter - limit which queries to log
- log_slow_slave_statements - log slow statements executed by replica thread to the slow log if it is open.

Rotating the Slow Query Log on Unix and Linux

Unix and Linux distributions offer the logrotate utility, which makes it very easy to rotate log files. See Rotating Logs on Unix and Linux for more information on how to use this utility to rotate the slow query log.

3.3.4.6.5 Slow Query Log Extended Statistics

1.3.43.3 mysqldumpslow

2.4.5.4 EXPLAIN in the Slow Query Log

Switching it On

EXPLAIN output can be switched on by specifying the " `explain` " keyword in the log_slow_verbosity system variable. Alternatively, you can set with the `log-slow-verbosity` command line argument.

```
[mysqld]
log-slow-verbosity=query_plan,explain
```

EXPLAIN output will only be recorded if the slow query log is written to a file (and not to a table - see Writing logs into tables). This limitation also applies to other extended statistics that are written into the slow query log.

What it Looks Like

When explain recording is on, slow query log entries look like this:

```
# Time: 131112 17:03:32
# User@Host: root[root] @ localhost []
# Thread_id: 2  Schema: dbt3sf1  QC_hit: No
# Query_time: 5.524103  Lock_time: 0.000337  Rows_sent: 1  Rows_examined: 65633
#
# explain: id   select_type     table   type    possible_keys   key     key_len ref     rows
Extra
# explain: 1    SIMPLE  nation  ref     PRIMARY,n_name  n_name  26      const   1       Using
where; Using index
# explain: 1    SIMPLE  customer        ref     PRIMARY,i_c_nationkey   i_c_nationkey   5
dbt3sf1.nation.n_nationkey      3145    Using index
# explain: 1    SIMPLE  orders  ref     i_o_custkey     i_o_custkey     5
dbt3sf1.customer.c_custkey      7       Using index
#
SET timestamp=1384261412;
select count(*) from customer, orders, nation where c_custkey=o_custkey and
c_nationkey=n_nationkey and n_name='GERMANY';
```

EXPLAIN lines start with `# explain:` .

1.1.1.2.9.3.22 mysqlslow_log Table

2.4.6 Rotating Logs on Unix and Linux

Contents

Unix and Linux distributions offer the `logrotate` utility, which makes it very easy to rotate log files. This page will describe how to configure log rotation for the error log, general query log, and the slow query log.

Configuring Locations and File Names of Logs

The first step is to configure the locations and file names of logs. To make the log rotation configuration easier, it can be best to put these logs in a dedicated log directory.

We will need to configure the following:

- The error log location and file name is configured with the `log_error` system variable.
- The general query log location and file name is configured with the `general_log_file` system variable.
- The slow query log location and file name is configured with the `slow_query_log_file` system variable.

If you want to enable the general query log and slow query log immediately, then you will also have to configure the following:

- The general query log is enabled with the `general_log` system variable.
- The slow query log is enabled with the `slow_query_log` system variable.

These options can be set in a server option group in an option file prior to starting up the server. For example, if we wanted to put our log files in `/var/log/mysql/` , then we could configure the following:

```
[mariadb]
...
log_error=/var/log/mysql/mariadb.err
general_log
general_log_file=/var/log/mysql/mariadb.log
slow_query_log
slow_query_log_file=/var/log/mysql/mariadb-slow.log
long_query_time=5
```

We will also need to create the relevant directory:

```
sudo mkdir /var/log/mysql/
sudo chown mysql:mysql /var/log/mysql/
sudo chmod 0770 /var/log/mysql/
```

If you are using SELinux, then you may also need to set the SELinux context for the directory. See SELinux: Setting the File Context for Log Files for more information. For example:

```
sudo semanage fcontext -a -t mysqld_log_t "/var/log/mysql(/.*)?"
sudo restorecon -Rv /var/log/mysql
```

After MariaDB is restarted, it will use the new log locations and file names.

Configuring Authentication for Logrotate

The `logrotate` utility needs to be able to authenticate with MariaDB in order to flush the log files.

The easiest way to allow the `logrotate` utility to authenticate with MariaDB is to configure the `root@localhost` user account to use `unix_socket` authentication.

MariaDB starting with 10.4

In MariaDB 10.4 and later, the `root@localhost` user account is configured to use `unix_socket` authentication by default, so this part can be skipped in those versions.

MariaDB until 10.3

In MariaDB 10.3 and before, a user account is only able to have one authentication method at a time. In these versions, this means that once you enable `unix_socket` authentication for the `root@localhost` user account, you will no longer be able to use a password to log in with that user account. The user account will only be able to use `unix_socket` authentication.

In MariaDB 10.3 and before, you need to install the unix_socket plugin before you can configure the `root@localhost` user account to use it. For example:

```
INSTALL SONAME 'auth_socket';
```

After the plugin is installed, the `root@localhost` user account can be configured to use `unix_socket` authentication. How this is done depends on the version of MariaDB.

The `root@localhost` user account can be altered to use unix_socket authentication with the ALTER USER statement. For example:

```
ALTER USER 'root'@'localhost' IDENTIFIED VIA unix_socket;
```

<</product>>

Configuring Logrotate

At this point, we can configure the `logrotate` utility to rotate the log files.

On many systems, the primary `logrotate` configuration file is located at the following path:

- `/etc/logrotate.conf`

And the `logrotate` configuration files for individual services are located in the following directory:

- `/etc/logrotate.d/`

We can create a `logrotate` configuration file for MariaDB by executing the following command in a shell:

```
$ sudo tee /etc/logrotate.d/mariadb <<EOF
/var/log/mysql/* {
        su mysql mysql
        missingok
        create 660 mysql mysql
        notifempty
        daily
        minsize 1M # only use with logrotate >= 3.7.4
        maxsize 100M # only use with logrotate >= 3.8.1
        rotate 30
        # dateext # only use if your logrotate version is compatible with below dateformat
        # dateformat .%Y-%m-%d-%H-%M-%S # only use with logrotate >= 3.9.2
        compress
        delaycompress
        sharedscripts
        olddir archive/
        createolddir 770 mysql mysql # only use with logrotate >= 3.8.9
    postrotate
        # just if mysqld is really running
        if test -x /usr/bin/mysqladmin && \
           /usr/bin/mysqladmin ping &>/dev/null
        then
           /usr/bin/mysqladmin --local flush-error-log \
              flush-engine-log flush-general-log flush-slow-log
        fi
    endscript
}
EOF
```

You may have to modify this configuration file to use it on your system, depending on the specific version of the `logrotate` utility that is installed. See the description of each configuration directive below to determine which `logrotate` versions support that configuration directive.

Each specific configuration directive does the following:

- **`missingok`** : This directive configures it to ignore missing files, rather than failing with an error.
- **`create 660 mysql mysql`** : This directive configures it to recreate the log files after log rotation with the specified permissions and owner.
- **`notifempty`** : This directive configures it to skip a log file during log rotation if it is empty.
- **`daily`** : This directive configures it to rotate each log file once per day.
- **`minsize 1M`** : This directive configures it to skip a log file during log rotation if it is smaller than 1 MB. This directive is only available with `logrotate` 3.7.4 and later.
- **`maxsize 100M`** : This directive configures it to rotate a log file more frequently than daily if it grows larger than 100 MB. This directive is only available with `logrotate` 3.8.1 and later.
- **`rotate 30`** : This directive configures it to keep 30 old copies of each log file.
- **`dateext`** : This directive configures it to use the date as an extension, rather than just a number. This directive is only available with `logrotate` 3.7.6 and later.
- **`dateformat .%Y-%m-%d-%H-%M-%S`** : This directive configures it to use this date format string (as defined by the format specification for `strftime`) for the date extension configured by the `dateext` directive. This directive is only available with `logrotate` 3.7.7 and later. Support for `%H` is only available with `logrotate` 3.9.0 and later. Support for `%M` and `%S` is only available with `logrotate` 3.9.2 and later.
- **`compress`** : This directive configures it to compress the log files with `gzip` .
- **`delaycompress`** : This directive configures it to delay compression of each log file until the next log rotation. If the log file is compressed at the same time that it is rotated, then there may be cases where a log file is being compressed while the MariaDB server is still writing to the log file. Delaying compression of a log file until the next log rotation can prevent race conditions such as these that can happen between the compression operation and the MariaDB server's log flush operation.
- **`olddir archive/`** : This directive configures it to archive the rotated log files in `/var/log/mysql/archive/` .
- **`createolddir 770 mysql mysql`** : This directive configures it to create the directory specified by the `olddir` directive with the specified permissions and owner, if the directory does not already exist. This directive is only available with `logrotate` 3.8.9 and later.
- **`sharedscripts`** : This directive configures it to run the `postrotate` script just once, rather than once for each rotated log file.
- **`postrotate`** : This directive configures it to execute a script after log rotation. This particular script executes the mariadb-admin utility, which executes the FLUSH statement, which tells the MariaDB server to flush its various log files. When MariaDB server flushes a log file, it closes its existing file handle and reopens a new one. This ensure

that MariaDB server does not continue writing to a log file after it has been rotated. This is an important component of the log rotation process.

If our system does not have `logrotate` 3.8.9 or later, which is needed to support the `createolddir` directive, then we will also need to create the relevant directory specified by the `olddir` directive:

```
sudo mkdir /var/log/mysql/archive/
sudo chown mysql:mysql /var/log/mysql/archive/
sudo chmod 0770 /var/log/mysql/archive/
```

Testing Log Rotation

We can test log rotation by executing the `logrotate` utility with the `--force` option. For example:

```
sudo logrotate --force /etc/logrotate.d/mariadb
```

Keep in mind that under normal operation, the `logrotate` utility may skip a log file during log rotation if the utility does not believe that the log file needs to be rotated yet. For example:

- If you set the `notifempty` directive mentioned above, then it will be configured to skip a log file during log rotation if the log file is empty.
- If you set the `daily` directive mentioned above, then it will be configured to only rotate each log file once per day.
- If you set the `minsize 1M` directive mentioned above, then it will be configured to skip a log file during log rotation if the log file size is smaller than 1 MB.

However, when running tests with the `--force` option, the `logrotate` utility does not take these options into consideration.

After a few tests, we can see that the log rotation is indeed working:

```
$ sudo ls -l /var/log/mysql/archive/
total 48
-rw-rw---- 1 mysql mysql  440 Mar 31 15:31 mariadb.err.1
-rw-rw---- 1 mysql mysql  138 Mar 31 15:30 mariadb.err.2.gz
-rw-rw---- 1 mysql mysql  145 Mar 31 15:28 mariadb.err.3.gz
-rw-rw---- 1 mysql mysql 1007 Mar 31 15:27 mariadb.err.4.gz
-rw-rw---- 1 mysql mysql 1437 Mar 31 15:32 mariadb.log.1
-rw-rw---- 1 mysql mysql  429 Mar 31 15:31 mariadb.log.2.gz
-rw-rw---- 1 mysql mysql  439 Mar 31 15:28 mariadb.log.3.gz
-rw-rw---- 1 mysql mysql  370 Mar 31 15:27 mariadb.log.4.gz
-rw-rw---- 1 mysql mysql 3915 Mar 31 15:32 mariadb-slow.log.1
-rw-rw---- 1 mysql mysql  554 Mar 31 15:31 mariadb-slow.log.2.gz
-rw-rw---- 1 mysql mysql  569 Mar 31 15:28 mariadb-slow.log.3.gz
-rw-rw---- 1 mysql mysql  487 Mar 31 15:27 mariadb-slow.log.4.gz
```

Logrotate in Ansible

Let's see an example of how to configure logrotate in Ansible.

First, we'll create a couple of tasks in our playbook:

```
- name: Create mariadb_logrotate_old_dir
  file:
    path: "{{ mariadb_logrotate_old_dir }}"
    owner: mysql
    group: mysql
    mode: '770'
    state: directory

- name: Configure logrotate
  template:
    src: "../templates/logrotate.j2"
    dest: "/etc/logrotate.d/mysql"
```

The first task creates a directory to store the old, compressed logs, and set proper permissions.

The second task uploads logrotate configuration file into the proper directory, and calls it `mysql`. As you can see the original name is different, and it ends with the `.j2` extension, because it is a Jinja 2 template.

The file will look like the following:

```
{{ mariadb_log_dir }}/* {
        su mysql mysql
        missingok
        create 660 mysql mysql
        notifempty
        daily
        minsize 1M {{ mariadb_logrotate_min_size }}
        maxsize 100M {{ mariadb_logrotate_max_size }}
        rotate {{ mariadb_logrotate_old_dir }}
        dateformat .%Y-%m-%d-%H-%M-%S # only use with logrotate >= 3.9.2
        compress
        delaycompress
        sharedscripts
        olddir archive/
        createolddir 770 mysql mysql # only use with logrotate >= 3.8.9
    postrotate
        # just if mysqld is really running
        if test -x /usr/bin/mysqladmin && \
           /usr/bin/mysqladmin ping &>/dev/null
        then
           /usr/bin/mysqladmin --local flush-error-log \
              flush-engine-log flush-general-log flush-slow-log
        fi
    endscript
}
```

The file is very similar to the file shown above, which is obvious because we're still uploading a logrotate configuration file. Ansible is just a tool we've chosen to do this.

However, both in the tasks and in the template, we used some variables. This allows to use different paths and rotation parameters for different hosts, or host groups.

If we have a group host called `mariadb` that contains the default configuration for all our MariaDB servers, we can define these variables in a file called `group_vars/mariadb.yml`:

```
# MariaDB writes its logs here
mariadb_log_dir: /var/lib/mysql/logs

# logrotate configuration

mariadb_logrotate_min_size: 500M
mariadb_logrotate_max_size: 1G
mariadb_logrotate_old_files: 7
mariadb_logrotate_old_dir: /var/mysql/old-logs
```

After setting up logrotate in Ansible, you may want to deploy it to a non-production server and test it manually as explained above. Once you're sure that it works fine on one server, you can be confident in the new Ansible tasks and deploy them on all servers.

For more information on how to use Ansible to automate MariaDB configuration, see Ansible and MariaDB.

3.1.13 Binary Log

5.3.2.14 InnoDB Redo Log

5.3.2.15 InnoDB Undo Log

5.3.13.6 MyISAM Log

2.4.11 Transaction Coordinator Log

Transaction Coordinator Log Overview

The transaction coordinator log (tc_log) is used to coordinate transactions...

2.4.11.1 Transaction Coordinator Log Overview

Contents

The transaction coordinator log (tc_log) is used to coordinate transactions that affect multiple XA-capable storage engines. If you have two or more XA-capable storage engines enabled, then a transaction coordinator log must be available.

Types of Transaction Coordinator Logs

There are currently two implementations of the transaction coordinator log:

- Binary log-based transaction coordinator log
- Memory-mapped file-based transaction coordinator log

If the binary log is enabled on a server, then the server will use the binary log-based transaction coordinator log. Otherwise, it will use the memory-mapped file-based transaction coordinator log.

Binary Log-Based Transaction Coordinator Log

This transaction coordinator uses the binary log, which is enabled by the `log_bin` server option.

Memory-Mapped File-Based Transaction Coordinator Log

This transaction coordinator uses the memory-mapped file defined by the `--log-tc` server option. The size is defined by the `log_tc_size` system variable.

Some facts about this log:

- The log consists of a memory-mapped file that is divided into pages of 8KB size.
- The usable size of the first page is smaller because of the log header. There is a PAGE control structure for each page.
- Each page (or rather its PAGE control structure) can be in one of the three states - active, syncing, pool.
- There could be only one page in the active or syncing state, but many in the pool state - pool is a fifo queue.
- The usual lifecycle of a page is pool->active->syncing->pool.
- The "active" page is a page where new xid's are logged.
- The page stays active as long as the syncing slot is taken.
- The "syncing" page is being synced to disk. no new xid can be added to it.
- When the syncing is done the page is moved to a pool and an active page becomes "syncing".

The result of such an architecture is a natural "commit grouping" - If commits are coming faster than the system can sync, they do not stall. Instead, all commits that came since the last sync are logged to the same "active" page, and they all are synced with the next - one - sync. Thus, thought individual commits are delayed, throughput is not decreasing.

When an xid is added to an active page, the thread of this xid waits for a page's condition until the page is synced. When a syncing slot becomes vacant one of these waiters is awakened to take care of syncing. It syncs the page and signals all waiters that the page is synced. The waiters are counted, and a page may never become active again until waiters==0, which means that is all waiters from the previous sync have noticed that the sync was completed.

Note that a page becomes "dirty" and has to be synced only when a new xid is added into it. Removing a xid from a page does not make it dirty - we don't sync xid removals to disk.

Monitoring the Memory-Mapped File-Based Transaction Coordinator Log

The memory-mapped transaction coordinator log can be monitored with the following status variables:

- `Tc_log_max_pages_used`
- `Tc_log_page_size`
- `Tc_log_page_waits`

Heuristic Recovery with the Transaction Coordinator Log

One of the main purposes of the transaction coordinator log is in crash recovery. See Heuristic Recovery with the Transaction Coordinator Log for more information about that.

Known Issues

You must enable exactly N storage engines

Prior to MariaDB 10.1.10, if you were using the memory-mapped file-based transaction coordinator log, and then if the server crashed and you changed the number of XA-capable storage engines that it loaded, then you could see errors like the following:

```
2018-11-30 23:08:49 140046048638848 [Note] Recovering after a crash using tc.log
2018-11-30 23:08:49 140046048638848 [ERROR] Recovery failed! You must enable exactly 3 storage
engines that support two-phase commit protocol
2018-11-30 23:08:49 140046048638848 [ERROR] Crash recovery failed. Either correct the problem
(if it's, for example, out of memory error) and restart, or delete tc log and start mysqld with
--tc-heuristic-recover={commit|rollback}
2018-11-30 23:08:49 140046048638848 [ERROR] Can't init tc log
2018-11-30 23:08:49 140046048638848 [ERROR] Aborting
```

To recover from this error, delete the file defined by the `--log-tc` server option, and then restart the server with the `--tc-heuristic-recover` option set.

See MDEV-9214 for more information.

Bad magic header in tc log

If you are using the memory-mapped file-based transaction coordinator log, then it is possible to see errors like the following:

```
2018-09-19  4:29:31 0 [Note] Recovering after a crash using tc.log
2018-09-19  4:29:31 0 [ERROR] Bad magic header in tc log
2018-09-19  4:29:31 0 [ERROR] Crash recovery failed. Either correct the problem (if it's, for
example, out of memory error) and restart, or delete tc log and start mysqld with --tc-
heuristic-recover={commit|rollback}
2018-09-19  4:29:31 0 [ERROR] Can't init tc log
2018-09-19  4:29:31 0 [ERROR] Aborting
```

This means that the header of the memory-mapped file-based transaction coordinator log is corrupt. To recover from this error, delete the file defined by the `--log-tc` server option, and then restart the server with the `--tc-heuristic-recover` option set.

This issue is known to occur when using docker. In that case, the problem may be caused by using a MariaDB container version with a data directory from a different MariaDB or MySQL version. Therefore, some potential fixes are:

- Pinning the docker instance to a specific MariaDB version in the docker compose file, so that it consistently uses the same version.
- Running mariadb-upgrade to ensure that the data directory is upgraded to match the server version.

See this docker issue for more information.

MariaDB Galera Cluster

MariaDB Galera Cluster builds include a built-in plugin called `wsrep`. Prior to MariaDB 10.4.3, this plugin was internally considered an XA-capable storage engine. Consequently, these MariaDB Galera Cluster builds have multiple XA-capable storage engines by default, even if the only "real" storage engine that supports external XA transactions enabled on these builds by default is InnoDB. Therefore, when using one these builds MariaDB would be forced to use a transaction coordinator log by default, which could have performance implications.

For example, MDEV-16509 describes performance problems where MariaDB Galera Cluster actually performs better when the binary log is enabled. It is possible that this is caused by the fact that MariaDB is forced to use the memory-

mapped file-based transaction coordinator log in this case, which may not perform as well.

This became a bigger issue in MariaDB 10.1 when the MySQL-wsrep patch that powers MariaDB Galera Cluster was enabled on most MariaDB builds on Linux by default. Consequently, this built-in `wsrep` plugin would exist on those MariaDB builds on Linux by default. Therefore, MariaDB users might pay a performance penalty, even if they never actually intended to use the MariaDB Galera Cluster features included in MariaDB 10.1.

In MariaDB 10.4.3 and later, the built-in `wsrep` plugin has been changed to a replication plugin. Therefore, it is no longer considered an XA-capable storage engine, so it no longer forces MariaDB to use a transaction coordinator log by default.

See MDEV-16442 for more information.

2.4.11.2 Heuristic Recovery with the Transaction Coordinator Log

Contents

The transaction coordinator log (tc_log) is used to coordinate transactions that affect multiple XA-capable storage engines. One of the main purposes of this log is in crash recovery.

Modes of Crash Recovery

There are two modes of crash recovery:

- Automatic crash recovery.
- Manual heuristic recovery when `--tc-heuristic-recover` is set to some value other than `OFF`.

Automatic Crash Recovery

Automatic crash recovery occurs during startup when MariaDB needs to recover from a crash and `--tc-heuristic-recover` is set to `OFF`, which is the default value.

Automatic Crash Recovery with the Binary Log-Based Transaction Coordinator Log

If MariaDB needs to perform automatic crash recovery and if the binary log is enabled, then the error log will contain messages like this:

```
[Note] Recovering after a crash using cmdb-mariadb-0-bin
[Note] InnoDB: Buffer pool(s) load completed at 190313 11:24:29
[Note] Starting crash recovery...
[Note] Crash recovery finished.
```

Automatic Crash Recovery with the Memory-Mapped File-Based Transaction Coordinator Log

If MariaDB needs to perform automatic crash recovery and if the binary log is **not** enabled, then the error log will contain messages like this:

```
[Note] Recovering after a crash using tc.log
[Note] InnoDB: Buffer pool(s) load completed at 190313 11:26:32
[Note] Starting crash recovery...
[Note] Crash recovery finished.
```

Manual Heuristic Recovery

Manual heuristic recovery occurs when `--tc-heuristic-recover` is set to some value other than `OFF`. This might be needed if the server finds prepared transactions during crash recovery that are not in the transaction coordinator log. For example, the error log might contain an error like this:

```
[ERROR] Found 1 prepared transactions! It means that mysqld was not shut down properly last
time and critical recovery information (last binlog or tc.log file) was manually deleted after
a crash. You have to start mysqld with --tc-heuristic-recover switch to commit or rollback
pending transactions.
```

When manual heuristic recovery is initiated, MariaDB will ignore information about transactions in the transaction coordinator log during the recovery process. Prepared transactions that are encountered during the recovery process will either be rolled back or committed, depending on the value of `--tc-heuristic-recover`.

When manual heuristic recovery is initiated, the error log will contain a message like this:

```
[Note] Heuristic crash recovery mode
```

Manual Heuristic Recovery with the Binary Log-Based Transaction Coordinator Log

If `--tc-heuristic-recover` is set to some value other than `OFF` and if the binary log is enabled, then MariaDB will ignore information about transactions in the binary log during the recovery process. Prepared transactions that are encountered during the recovery process will either be rolled back or committed, depending on the value of `--tc-heuristic-recover`.

After the recovery process is complete, MariaDB will create a new empty binary log file, so that the old corrupt ones can be ignored.

Manual Heuristic Recovery with the Memory-Mapped File-Based Transaction Coordinator Log

If `--tc-heuristic-recover` is set to some value other than `OFF` and if the binary log is **not** enabled, then MariaDB will ignore information about transactions in the the memory-mapped file defined by the `--log-tc` option during the recovery process. Prepared transactions that are encountered during the recovery process will either be rolled back or committed, depending on the value of `--tc-heuristic-recover`.

5.4.11.7 SQL Error Log Plugin

1.1.1.2.9.6 Writing Logs Into Tables

1.1.1.2.9.2 Performance Schema

5.4.5 MariaDB Audit Plugin

2.5 Partitioning Tables

A huge table can be split into smaller subsets. Both data and indexes are partitioned.

Partitioning Overview

A table partitioning overview

Partitioning Types

A partitioning type determines how a table rows are distributed across partitions.

Partition Pruning and Selection

Partition pruning is when the optimizer knows which partitions are relevant for the query.

Partition Maintenance

For time series (includes list of PARTITION uses)

Partitioning Limitations

Limitations applying to partitioning in MariaDB.

Partitions Files

A partitioned table is stored in multiple files

Partitions Metadata

How to obtain information about partitions definition

Information Schema PARTITIONS Table

Table partition information.

There are 2 related questions.

2.5.1 Partitioning Overview

Contents

In MariaDB, a table can be split in smaller subsets. Both data and indexes are partitioned.

Uses for Partitioning

There can be several reasons to use this feature:

- Very big tables and indexes can be slow even with optimized queries. But if the target table is partitioned, queries that read a small number of partitions can be much faster.
- Partitioning allows one to distribute files over multiple storage devices. For example, we can have historical data on slower, larger disks (historical data are not supposed to be frequently read); and current data can be on faster disks, or SSD devices.
- In case we separate historical data from recent data, we will probably need to take regular backups of one partition, not the whole table.

Partitioning for Specific Storage Engines

Some MariaDB storage engines allow more interesting uses for partitioning.

SPIDER allows one to:

- Move partitions of the same table on different servers. In this way, the workload can be distributed on more physical or virtual machines (*data sharding*).
- All partitions of a SPIDER table can also live on the same machine. In this case there will be a small overhead (SPIDER will use connections to localhost), but queries that read multiple partitions will use parallel threads.

CONNECT allows one to:

- Build a table whose partitions are tables using different storage engines (like InnoDB, MyISAM, or even engines that do not support partitioning).
- Build an indexable, writeable table on several data files. These files can be in different formats.

See also: Using CONNECT - Partitioning and Sharding

Partitioning Types

When partitioning a table, the use should decide:

- a *partitioning type*;
- a *partitioning expression*.

A partitioning type is the method used by MariaDB to decide how rows are distributed over existing partitions. Choosing the proper partitioning type is important to distribute rows over partitions in an efficient way.

With some partitioning types, a partitioning expression is also required. A partitioning function is an SQL expression returning an integer or temporal value, used to determine which row will contain a given row. The partitioning expression is used for all reads and writes on involving the partitioned table, thus it should be fast.

See Partioning Types for a detailed description.

Enabling Partitioning

By default, MariaDB permits partitioning. You can determine this by using the SHOW PLUGINS statement, for example:

```
SHOW PLUGINS;
...
| Aria                          | ACTIVE   | STORAGE ENGINE     | NULL    | GPL     |
| FEEDBACK                      | DISABLED | INFORMATION SCHEMA | NULL    | GPL     |
| partition                     | ACTIVE   | STORAGE ENGINE     | NULL    | GPL     |
+-------------------------------+----------+--------------------+---------+---------+
```

If partition is listed as DISABLED:

```
| partition                     | DISABLED | STORAGE ENGINE     | NULL    | GPL     |
+-------------------------------+----------+--------------------+---------+---------+
```

MariaDB has either been built without partitioning support, or has been started with the the --skip-partition option, or one of its variants:

```
--skip-partition
--disable-partition
--partition=OFF
```

and you will not be able to create partitions.

Using Partitions

It is possible to create a new partitioned table using CREATE TABLE.

ALTER TABLE allows one to:

- Partition an existing table;
- Remove partitions from a partitioned table (**with all data in the partition**);
- Add/remove partitions, or reorganize them, as long as the partitioning function allows these operations (see below);
- Exchange a partition with a table;
- Perform administrative operations on some or all partitions (analyze, optimize, check, repair).

Adding Partitions

```
ADD PARTITION [IF NOT EXISTS] (partition_definition)
```

`ALTER TABLE ... ADD PARTITION` can be used to add partitions to an existing table:

```
CREATE OR REPLACE TABLE t1 (
  timestamp DATETIME NOT NULL
)
  ENGINE = InnoDB
  PARTITION BY RANGE (YEAR(timestamp))
  (
  PARTITION p0 VALUES LESS THAN (2013),
  PARTITION p1 VALUES LESS THAN (2014),
  PARTITION p2 VALUES LESS THAN (2015),
  PARTITION p3 VALUES LESS THAN (2016)
);

ALTER TABLE t1 ADD PARTITION (
  PARTITION p4 VALUES LESS THAN (2017),
  p5 VALUES LESS THAN (2018)
);
```

Converting Partitions to Tables

MariaDB starting with 10.7

```
CONVERT PARTITION partition_name TO TABLE tbl_name
```

`ALTER TABLE ... CONVERT PARTITION` can, from MariaDB 10.7, be used to convert partitions in an existing table to a standalone table:

```
CREATE OR REPLACE TABLE t1 (
   timestamp DATETIME NOT NULL
 )
   ENGINE = InnoDB
   PARTITION BY RANGE (YEAR(timestamp))
   (
   PARTITION p0 VALUES LESS THAN (2013),
   PARTITION p1 VALUES LESS THAN (2014),
   PARTITION p2 VALUES LESS THAN (2015),
   PARTITION p3 VALUES LESS THAN (2016)
 );

INSERT INTO t1 VALUES ('2012-11-11'),('2013-11-11'),('2014-11-11');

SELECT * FROM t1;
+---------------------+
| timestamp           |
+---------------------+
| 2012-11-11 00:00:00 |
| 2013-11-11 00:00:00 |
| 2014-11-11 00:00:00 |
+---------------------+

ALTER TABLE t1 CONVERT PARTITION p0 TO TABLE t2;

SELECT * FROM t1;
+---------------------+
| timestamp           |
+---------------------+
| 2013-11-11 00:00:00 |
| 2014-11-11 00:00:00 |
+---------------------+

SELECT * FROM t2;
+---------------------+
| timestamp           |
+---------------------+
| 2012-11-11 00:00:00 |
+---------------------+

SHOW CREATE TABLE t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE `t1` (
  `timestamp` datetime NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_swedish_ci
 PARTITION BY RANGE (year(`timestamp`))
(PARTITION `p1` VALUES LESS THAN (2014) ENGINE = InnoDB,
 PARTITION `p2` VALUES LESS THAN (2015) ENGINE = InnoDB,
 PARTITION `p3` VALUES LESS THAN (2016) ENGINE = InnoDB)

SHOW CREATE TABLE t2\G
*************************** 1. row ***************************
       Table: t2
Create Table: CREATE TABLE `t2` (
  `timestamp` datetime NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_swedish_ci
```

An alternative (and the only method prior to MariaDB 10.7) to convert partitions to tables is to use `ALTER TABLE ... EXCHANGE PARTITION`. This requires having to manually do the following steps:

- create an empty table with the same structure as the partition
- exchange the table with the partition
- drop the empty partition

For example:

```
CREATE OR REPLACE TABLE t1 (
   timestamp DATETIME NOT NULL
 )
   ENGINE = InnoDB
   PARTITION BY RANGE (YEAR(timestamp))
   (
   PARTITION p0 VALUES LESS THAN (2013),
   PARTITION p1 VALUES LESS THAN (2014),
   PARTITION p2 VALUES LESS THAN (2015),
   PARTITION p3 VALUES LESS THAN (2016)
 );

INSERT INTO t1 VALUES ('2012-11-11'),('2013-11-11'),('2014-11-11');

SELECT * FROM t1;
+---------------------+
| timestamp           |
+---------------------+
| 2012-11-11 00:00:00 |
| 2013-11-11 00:00:00 |
| 2014-11-11 00:00:00 |
+---------------------+

CREATE OR REPLACE TABLE t2 LIKE t1;

ALTER TABLE t2 REMOVE PARTITIONING;

ALTER TABLE t1 EXCHANGE PARTITION p0 WITH TABLE t2;

ALTER TABLE t1 DROP PARTITION p0;

SELECT * FROM t1;
+---------------------+
| timestamp           |
+---------------------+
| 2013-11-11 00:00:00 |
| 2014-11-11 00:00:00 |
+---------------------+

SELECT * FROM t2;
+---------------------+
| timestamp           |
+---------------------+
| 2012-11-11 00:00:00 |
+---------------------+

SHOW CREATE TABLE t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE `t1` (
  `timestamp` datetime NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_swedish_ci
 PARTITION BY RANGE (year(`timestamp`))
(PARTITION `p1` VALUES LESS THAN (2014) ENGINE = InnoDB,
 PARTITION `p2` VALUES LESS THAN (2015) ENGINE = InnoDB,
 PARTITION `p3` VALUES LESS THAN (2016) ENGINE = InnoDB)
1 row in set (0.001 sec)

SHOW CREATE TABLE t2\G
*************************** 1. row ***************************
       Table: t2
Create Table: CREATE TABLE `t2` (
  `timestamp` datetime NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_swedish_ci
```

Dropping Partitions

```
DROP PARTITION [IF EXISTS] partition_names
```

`ALTER TABLE ... DROP PARTITION` can be used to drop specific partitions (and discard all data within the specified partitions) for RANGE and LIST partitions. It cannot be used on `HASH` or `KEY` partitions. To rather remove all partitioning,

while leaving the data unaffected, see Removing Partitioning.

```
CREATE OR REPLACE TABLE t1 (
  timestamp DATETIME NOT NULL
)
  ENGINE = InnoDB
  PARTITION BY RANGE (YEAR(timestamp))
  (
  PARTITION p0 VALUES LESS THAN (2013),
  PARTITION p1 VALUES LESS THAN (2014),
  PARTITION p2 VALUES LESS THAN (2015),
  PARTITION p3 VALUES LESS THAN (2016)
);

INSERT INTO t1 VALUES ('2012-11-15');
SELECT * FROM t1;
+---------------------+
| timestamp           |
+---------------------+
| 2012-11-15 00:00:00 |
+---------------------+

ALTER TABLE t1 DROP PARTITION p0;

SELECT * FROM t1;
Empty set (0.002 sec)
```

Removing Partitioning

```
REMOVE PARTITIONING
```

`ALTER TABLE ... REMOVE PARTITIONING` will remove all partitioning from the table, while leaving the data unaffected. To rather drop a particular partition (and discard all of its data), see Dropping Partitions.

```
ALTER TABLE t1 REMOVE PARTITIONING;
```

Truncating Partitions

```
TRUNCATE PARTITION partition_names
```

`ALTER TABLE ... TRUNCATE PARTITION` will remove all data from the specified partition/s, leaving the table and partition structure unchanged. Partitions don't need to be contiguous.

```
CREATE OR REPLACE TABLE t1 (
  timestamp DATETIME NOT NULL
)
  ENGINE = InnoDB
  PARTITION BY RANGE (YEAR(timestamp))
  (
  PARTITION p0 VALUES LESS THAN (2013),
  PARTITION p1 VALUES LESS THAN (2014),
  PARTITION p2 VALUES LESS THAN (2015),
  PARTITION p3 VALUES LESS THAN (2016)
);

INSERT INTO t1 VALUES ('2012-11-01'),('2013-11-02'),('2014-11-03'),('2015-11-04');

SELECT * FROM t1;
+---------------------+
| timestamp           |
+---------------------+
| 2012-11-01 00:00:00 |
| 2013-11-02 00:00:00 |
| 2014-11-03 00:00:00 |
| 2015-11-04 00:00:00 |
+---------------------+

ALTER TABLE t1 TRUNCATE PARTITION p0,p2;

SELECT * FROM t1;
+---------------------+
| timestamp           |
+---------------------+
| 2013-11-02 00:00:00 |
| 2015-11-04 00:00:00 |
+---------------------+
```

2.5.2 Partitioning Types

Partitioning Types Overview

A partition type determines how a partitioned table rows are distributed across partitions

LIST Partitioning Type

LIST partitioning is used to assign each partition a list of values

RANGE Partitioning Type

The RANGE partitioning type is used to assign each partition a range of values.

HASH Partitioning Type

Form of partitioning in which the server takes care of the partition in which to place the data.

KEY Partitioning Type

Used to have the server assign the distribution of rows across partitions.

LINEAR HASH Partitioning Type

Form of partitioning, similar to HASH, in which the server takes care of th...

LINEAR KEY Partitioning Type

Form of partitioning similar to KEY partitioning.

RANGE COLUMNS and LIST COLUMNS Partitioning Types

Used to assign each partition a range or a list of values

2.5.2.1 Partitioning Types Overview

A partitioning type determines how a partitioned table's rows are distributed across partitions. Some partition types require the user to specify a partitioning expression that determines in which partition a row will be stored.

The size of individual partitions depends on the partitioning type. Read and write performance are affected by the

partitioning expression. Therefore, these choices should be made carefully.

MariaDB supports the following partitioning types:

- RANGE
- LIST
- RANGE COLUMNS and LIST COLUMNS
- HASH
- LINEAR HASH
- KEY
- LINEAR KEY
- SYSTEM_TIME

2.5.2.2 LIST Partitioning Type

LIST partitioning is conceptually similar to RANGE partitioning. In both cases you decide a partitioning expression (a column, or a slightly more complex calculation) and use it to determine which partitions will contain each row. However, with the RANGE type, partitioning is done by assigning a range of values to each partition. With the LIST type, we assign a set of values to each partition. This is usually preferred if the partitioning expression can return a limited set of values.

A variant of this partitioning method, LIST COLUMNS, allows us to use multiple columns and more datatypes.

Syntax

The last part of a CREATE TABLE statement can be the definition of the new table's partitions. In the case of LIST partitioning, the syntax is the following:

```
PARTITION BY LIST (partitioning_expression)
(
 PARTITION partition_name VALUES IN (value_list),
 [ PARTITION partition_name VALUES IN (value_list), ... ]
        [ PARTITION partition_name DEFAULT ]
)
```

PARTITION BY LIST indicates that the partitioning type is LIST.

The `partitioning_expression` is an SQL expression that returns a value from each row. In the simplest cases, it is a column name. This value is used to determine which partition should contain a row.

`partition_name` is the name of a partition.

`value_list` is a list of values. If `partitioning_expression` returns one of these values, the row will be stored in this partition. If we try to insert something that does not belong to any of these value lists, the row will be rejected with an error.

The `DEFAULT` partition catches all records which do not fit into other partitions. Only one `DEFAULT` partition is permitted. This option was added in MariaDB 10.2.

Use cases

LIST partitioning can be useful when we have a column that can only contain a limited set of values. Even in that case, RANGE partitioning could be used instead; but LIST partitioning allows us to equally distribute the rows by assigning a proper set of values to each partition.

2.5.2.3 RANGE Partitioning Type

Contents

The RANGE partitioning type is used to assign each partition a range of values generated by the partitioning expression. Ranges must be ordered, contiguous and non-overlapping. The minimum value is always included in the first range. The highest value may or may not be included in the last range.

A variant of this partitioning method, RANGE COLUMNS, allows us to use multiple columns and more datatypes.

Syntax

The last part of a CREATE TABLE statement can be definition of the new table's partitions. In the case of RANGE partitioning, the syntax is the following:

```
PARTITION BY RANGE (partitioning_expression)
(
 PARTITION partition_name VALUES LESS THAN (value),
 [ PARTITION partition_name VALUES LESS THAN (value), ... ]
)
```

PARTITION BY RANGE indicates that the partitioning type is RANGE.

The `partitioning_expression` is an SQL expression that returns a value from each row. In the simplest cases, it is a column name. This value is used to determine which partition should contain a row.

`partition_name` is the name of a partition.

`value` indicates the upper bound for that partition. The values must be ascending. For the first partition, the lower limit is NULL. When trying to insert a row, if its value is higher than the upper limit of the last partition, the row will be rejected (with an error, if the IGNORE keyword is not used).

If this is a problem, MAXVALUE can be specified as a value for the last partition. Note however that it is not possible to split partitions of an existing RANGE partitioned table. New partitions can be appended, but this will not be possible if the last partition's higher bound is MAXVALUE.

Use Cases

A typical use case is when we want to partition a table whose rows refer to a moment or period in time; for example commercial transactions, blog posts, or events of some kind. We can partition the table by year, to keep all recent data in one partition and distribute historical data in big partitions that are stored on slower disks. Or, if our queries always read rows which refer to the same month or week, we can partition the table by month or year week (in this case, historical data and recent data will be stored together).

AUTO_INCREMENT values also represent a chronological order. So, these values can be used to store old data in separate partitions. However, partitioning by id is not the best choice if we usually query a table by date.

Examples

In the following example, we will partition a log table by year.

```
CREATE TABLE log
(
 id INT UNSIGNED NOT NULL AUTO_INCREMENT,
 timestamp DATETIME NOT NULL,
 user INT UNSIGNED,
 ip BINARY(16) NOT NULL,
 action VARCHAR(20) NOT NULL,
 PRIMARY KEY (id, timestamp)
)
 ENGINE = InnoDB
PARTITION BY RANGE (YEAR(timestamp))
(
 PARTITION p0 VALUES LESS THAN (2013),
 PARTITION p1 VALUES LESS THAN (2014),
 PARTITION p2 VALUES LESS THAN (2015),
 PARTITION p3 VALUES LESS THAN (2016)
);
```

As an alternative, we can partition the table by both year and month:

```
CREATE TABLE log
(
 id INT UNSIGNED NOT NULL AUTO_INCREMENT,
 timestamp TIMESTAMP NOT NULL,
 user INT UNSIGNED,
 ip BINARY(16) NOT NULL,
 action VARCHAR(20) NOT NULL,
 PRIMARY KEY (id, timestamp)
)
 ENGINE = InnoDB
PARTITION BY RANGE (UNIX_TIMESTAMP(timestamp))
(
 PARTITION p0 VALUES LESS THAN (UNIX_TIMESTAMP('2014-08-01 00:00:00')),
 PARTITION p1 VALUES LESS THAN (UNIX_TIMESTAMP('2014-11-01 00:00:00')),
 PARTITION p2 VALUES LESS THAN (UNIX_TIMESTAMP('2015-01-01 00:00:00')),
 PARTITION p3 VALUES LESS THAN (UNIX_TIMESTAMP('2015-02-01 00:00:00'))
);
```

As you can see, we used the UNIX_TIMESTAMP function to accomplish the purpose. Also, the first two partitions cover longer periods of time (probably because the logged activities were less intensive).

In both cases, when our tables become huge and we don't need to store all historical data any more, we can drop the oldest partitions in this way:

```
 ALTER TABLE log DROP PARTITION p0;
```

We will still be able to drop a partition that does not contain the oldest data, but all rows stored in it will disappear.

Example of an error when inserting outside a defined partition range:

```
INSERT INTO log(id,timestamp) VALUES
  (1, '2016-01-01 01:01:01'),
  (2, '2015-01-01 01:01:01');
ERROR 1526 (HY000): Table has no partition for value 2016
```

Unless the IGNORE keyword is used:

```
INSERT IGNORE INTO log(id,timestamp) VALUES
  (1, '2016-01-01 01:01:01'),
  (2, '2015-01-01 01:01:01');

SELECT * FROM log;
+----+---------------------+------+------------------+--------+
| id | timestamp           | user | ip               | action |
+----+---------------------+------+------------------+--------+
|  2 | 2015-01-01 01:01:01 | NULL |                  |        |
+----+---------------------+------+------------------+--------+
```

An alternative definition with MAXVALUE as a catchall:

```
CREATE TABLE log
(
 id INT UNSIGNED NOT NULL AUTO_INCREMENT,
 timestamp DATETIME NOT NULL,
 user INT UNSIGNED,
 ip BINARY(16) NOT NULL,
 action VARCHAR(20) NOT NULL,
 PRIMARY KEY (id, timestamp)
)
 ENGINE = InnoDB
PARTITION BY RANGE (YEAR(timestamp))
(
 PARTITION p0 VALUES LESS THAN (2013),
 PARTITION p1 VALUES LESS THAN (2014),
 PARTITION p2 VALUES LESS THAN (2015),
 PARTITION p3 VALUES LESS THAN (2016),
 PARTITION p4 VALUES LESS THAN MAXVALUE
);
```

2.5.2.4 HASH Partitioning Type

Syntax

```
PARTITION BY HASH (partitioning_expression)
[PARTITIONS(number_of_partitions)]
```

Description

HASH partitioning is a form of partitioning in which the server takes care of the partition in which to place the data, ensuring an even distribution among the partitions.

It requires a column value, or an expression based on a column value, which is hashed, as well as the number of partitions into which to divide the table.

partitioning_expression needs to return a non-constant, deterministic integer. It is evaluated for each insert and update, so overly complex expressions can lead to performance issues. A hashing function operating on a single column, and where the value changes consistently with the column value, allows for easy pruning on ranges of partitions, and is usually a better choice. For this reason, using multiple columns in a hashing expression is not usually recommended.

number_of_partitions is a positive integer specifying the number of partitions into which to divide the table. If the `PARTITIONS` clause is omitted, the default number of partitions is one.

Determining the Partition

To determine which partition to use, the following calculation is performed: MOD(partitioning_expression, number_of_partitions)

For example, if the expression is TO_DAYS(datetime_column) and the number of partitions is 5, inserting a datetime value of '2023-11-15' would determine the partition as follows:

- TO_DAYS('2023-11-15') gives a value of 739204
- MOD(739204,5) returns 4 so the 4th partition is used.

HASH partitioning making use of the modulus of the hashing function's value. The LINEAR HASH partitioning type is similar, using a powers-of-two algorithm. Data is more likely to be evenly distributed over the partitions than with the LINEAR HASH partitioning type, however, adding, dropping, merging and splitting partitions is much slower.

Examples

```
CREATE OR REPLACE TABLE t1 (c1 INT, c2 DATETIME)
  PARTITION BY HASH(TO_DAYS(c2))
  PARTITIONS 5;
```

Using the Information Schema PARTITIONS Table for more information:

```
INSERT INTO t1 VALUES (1,'2023-11-15');

SELECT PARTITION_NAME,TABLE_ROWS FROM INFORMATION_SCHEMA.PARTITIONS
  WHERE TABLE_SCHEMA='test' AND TABLE_NAME='t1';
+----------------+------------+
| PARTITION_NAME | TABLE_ROWS |
+----------------+------------+
| p0             |          0 |
| p1             |          0 |
| p2             |          0 |
| p3             |          0 |
| p4             |          1 |
+----------------+------------+
```

2.5.2.5 KEY Partitioning Type

Syntax

```
PARTITION BY KEY ([column_names])
[PARTITIONS (number_of_partitions)]
```

Description

Partitioning by key is a type of partitioning that is similar to and can be used in a similar way as partitioning by hash.

KEY takes an optional list of *column_names*, and the hashing function is given by the server.

Just like HASH partitioning, in KEY partitioning the server takes care of the partition and ensures an even distribution among the partitions. However, the largest difference is that KEY partitioning makes use of *column_names*, and cannot accept a *partitioning_expression* which is based on *column_names*, in contrast to HASH partitioning, which can.

If no *column_names* are specified, the table's primary key is used if present, or not null unique key if no primary key is present. If neither of these keys are present, not specifying any *column_names* will result in `ERROR 1488 (HY000): Field in list of fields for partition function not found in table`

Unlike other partitioning types, columns used for partitioning by KEY are not limited to integer or NULL values.

KEY partitions do not support column index prefixes. Any columns in the partitioning key that make use of column prefixes are not used (see also MDEV-32727).

Example

```
CREATE OR REPLACE TABLE t1 (v1 INT)
  PARTITION BY KEY (v1)
  PARTITIONS 2;
```

```
CREATE OR REPLACE TABLE t1 (v1 INT, v2 INT)
  PARTITION BY KEY (v1,v2)
  PARTITIONS 2;
```

```
CREATE OR REPLACE TABLE t1 (
    id INT NOT NULL PRIMARY KEY,
    name VARCHAR(5)
)
PARTITION BY KEY()
PARTITIONS 2;
```

```
CREATE OR REPLACE TABLE t1 (
    id INT NOT NULL UNIQUE KEY,
    name VARCHAR(5)
)
PARTITION BY KEY()
PARTITIONS 2;
```

The unique key must be NOT NULL:

```
CREATE OR REPLACE TABLE t1 (
    id INT NULL UNIQUE KEY,
    name VARCHAR(5)
)
PARTITION BY KEY()
PARTITIONS 2;
ERROR 1488 (HY000): Field in list of fields for partition function not found in table
```

KEY requires *column_values* if no primary key or not null unique key is present:

```
CREATE OR REPLACE TABLE t1 (
    id INT NULL UNIQUE KEY,
    name VARCHAR(5)
)
PARTITION BY KEY()
PARTITIONS 2;
ERROR 1488 (HY000): Field in list of fields for partition function not found in table
```

```
CREATE OR REPLACE TABLE t1 (
    id INT NULL UNIQUE KEY,
    name VARCHAR(5)
)
PARTITION BY KEY(name)
PARTITIONS 2;
```

Primary key columns with index prefixes are silently ignored, so the following two queries are equivalent:

```
CREATE OR REPLACE TABLE t1 (
    a VARCHAR(10),
    b VARCHAR(10),
    c VARCHAR(10),
    PRIMARY KEY (a(5), b, c(5))
) PARTITION BY KEY() PARTITIONS 2;

CREATE OR REPLACE TABLE t1 (
    a VARCHAR(10),
    b VARCHAR(10),
    c VARCHAR(10),
    PRIMARY KEY (b)
) PARTITION BY KEY() PARTITIONS 2;
```

`a(5)` and `c(5)` are silently ignored in the former.

If all columns use index prefixes, the statement fails with a slightly misleading error:

```
CREATE OR REPLACE TABLE t1 (
    a VARCHAR(10),
    b VARCHAR(10),
    c VARCHAR(10),
    PRIMARY KEY (a(5), b(5), c(5))
) PARTITION BY KEY() PARTITIONS 2;
ERROR 1503 (HY000): A PRIMARY KEY must include all columns in the table's partitioning function
```

2.5.2.6 LINEAR HASH Partitioning Type

Syntax

```
PARTITION BY LINEAR HASH (partitioning_expression)
[PARTITIONS(number_of_partitions)]
```

Description

LINEAR HASH partitioning is a form of partitioning, similar to HASH partitioning, in which the server takes care of the partition in which to place the data, ensuring a relatively even distribution among the partitions.

LINEAR HASH partitioning makes use of a powers-of-two algorithm, while HASH partitioning uses the modulus of the hashing function's value. Adding, dropping, merging and splitting partitions is much faster than with the HASH partitioning type, however, data is less likely to be evenly distributed over the partitions.

Example

```
CREATE OR REPLACE TABLE t1 (c1 INT, c2 DATETIME)
  PARTITION BY LINEAR HASH(TO_DAYS(c2))
  PARTITIONS 5;
```

2.5.2.7 LINEAR KEY Partitioning Type

Syntax

```
LINEAR PARTITION BY KEY ([column_names])
[PARTITIONS (number_of_partitions)]
```

Description

LINEAR KEY partitioning is a form of partitioning, similar to KEY partitioning.

LINEAR KEY partitioning makes use of a powers-of-two algorithm, while KEY partitioning uses modulo arithmetic, to determine the partition number.

Adding, dropping, merging and splitting partitions is much faster than with the KEY partitioning type, however, data is less likely to be evenly distributed over the partitions.

Example

```
CREATE OR REPLACE TABLE t1 (v1 INT)
  PARTITION BY LINEAR KEY (v1)
  PARTITIONS 2;
```

2.5.2.8 RANGE COLUMNS and LIST COLUMNS Partitioning Types

RANGE COLUMNS and LIST COLUMNS are variants of, respectively, RANGE and LIST. With these partitioning types there is not a single partitioning expression; instead, a list of one or more columns is accepted. The following rules apply:

- The list can contain one or more columns.
- Columns can be of any integer, string, DATE, and DATETIME types.
- Only bare columns are permitted; no expressions.

All the specified columns are compared to the specified values to determine which partition should contain a specific row. See below for details.

Syntax

The last part of a CREATE TABLE statement can be definition of the new table's partitions. In the case of RANGE COLUMNS partitioning, the syntax is the following:

```
PARTITION BY RANGE COLUMNS (col1, col2, ...)
(
 PARTITION partition_name VALUES LESS THAN (value1, value2, ...),
 [ PARTITION partition_name VALUES LESS THAN (value1, value2, ...), ... ]
)
```

The syntax for LIST COLUMNS is the following:

```
PARTITION BY LIST COLUMNS (partitioning_expression)
(
 PARTITION partition_name VALUES IN (value1, value2, ...),
 [ PARTITION partition_name VALUES IN (value1, value2, ...), ... ]
        [ PARTITION partititon_name DEFAULT ]
)
```

`partition_name` is the name of a partition.

Comparisons

To determine which partition should contain a row, all specified columns will be compared to each partition definition.

With LIST COLUMNS, a row matches a partition if all row values are identical to the specified values. At most one partition can match the row.

With RANGE COLUMNS, a row matches a partition if all row values are less than the specified values. The first partition that matches the row values will be used.

The `DEFAULT` partition catches all records which do not fit in other partitions. Only one `DEFAULT` partition is allowed.

2.5.3 Partition Pruning and Selection

When a WHERE clause is related to the partitioning expression, the optimizer knows which partitions are relevant for the query. Other partitions will not be read. This optimization is called *partition pruning*.

EXPLAIN PARTITIONS can be used to know which partitions will be read for a given query. A column called `partitions` will contain a comma-separated list of the accessed partitions. For example:

```
EXPLAIN PARTITIONS SELECT * FROM orders WHERE id < 15000000;
+------+-------------+--------+------------+-------+---------------+---------+---------+-----
-+------+-------------+
| id   | select_type | table  | partitions | type  | possible_keys | key     | key_len | ref  |
rows | Extra       |
+------+-------------+--------+------------+-------+---------------+---------+---------+-----
-+------+-------------+
|    1 | SIMPLE      | orders | p0,p1      | range | PRIMARY       | PRIMARY | 4       | NULL
|    2 | Using where |
+------+-------------+--------+------------+-------+---------------+---------+---------+-----
-+------+-------------+
```

Sometimes the WHERE clause does not contain the necessary information to use partition pruning, or the optimizer cannot infer this information. However, we may know which partitions are relevant for the query. Since MariaDB 10.0, we can force MariaDB to only access the specified partitions by adding a PARTITION clause. This feature is called *partition selection*. For example:

```
SELECT * FROM orders PARTITION (p3) WHERE user_id = 50;
SELECT * FROM orders PARTITION (p2,p3) WHERE user_id >= 40;
```

The PARTITION clause is supported for all DML statements:

- SELECT
- INSERT
- UPDATE
- DELETE
- REPLACE

Partition Pruning and Triggers

In general, partition pruning is applied to statements contained in triggers.

However, note that if a `BEFORE INSERT` or `BEFORE UPDATE` trigger is defined on a table, MariaDB doesn't know in advance if the columns used in the partitioning expression will be changed. For this reason, it is forced to lock all partitions.

2.5.4 Partition Maintenance

Contents

Preface

This article covers

- PARTITIONing uses and non-uses
- How to Maintain a time-series PARTITIONed table
- AUTO_INCREMENT secrets

First, my Opinions on PARTITIONing

Taken from Rick's RoTs - Rules of Thumb

- #1: Don't use PARTITIONing until you know how and why it will help.
- Don't use PARTITION unless you will have >1M rows
- No more than 50 PARTITIONs on a table (open, show table status, etc, are impacted) (fixed in MySQL 5.6.6?; a better fix coming eventually in 5.7)
- PARTITION BY RANGE is the only useful method.
- SUBPARTITIONs are not useful.
- The partition field should not be the field first in any key.
- It is OK to have an AUTO_INCREMENT as the first part of a compound key, or in a non-UNIQUE index.

It is so tempting to believe that PARTITIONing will solve performance problems. But it is so often wrong.

PARTITIONing splits up one table into several smaller tables. But table size is rarely a performance issue. Instead, I/O time and indexes are the issues.

A common fallacy: "Partitioning will make my queries run faster". It won't. Ponder what it takes for a 'point query'. Without partitioning, but with an appropriate index, there is a BTree (the index) to drill down to find the desired row. For a billion rows, this might be 5 levels deep. With partitioning, first the partition is chosen and "opened", then a smaller BTree (of say 4 levels) is drilled down. Well, the savings of the shallower BTree is consumed by having to open the partition. Similarly, if you look at the disk blocks that need to be touched, and which of those are likely to be cached, you come to the conclusion that about the same number of disk hits is likely. Since disk hits are the main cost in a query, Partitioning does not gain any performance (at least for this typical case). The 2D case (below) gives the main contradiction to this discussion.

Use Cases for PARTITIONing

Use case #1 -- time series. Perhaps the most common use case where PARTITIONing shines is in a dataset where "old" data is periodically deleted from the table. RANGE PARTITIONing by day (or other unit of time) lets you do a nearly instantaneous DROP PARTITION plus REORGANIZE PARTITION instead of a much slower DELETE. Much of this blog is focused on this use case. This use case is also discussed in Big DELETEs

The big win for Case #1: DROP PARTITION is a lot faster than DELETEing a lot of rows.

Use case #2 -- 2-D index. INDEXes are inherently one-dimensional. If you need two "ranges" in the WHERE clause, try to migrate one of them to PARTITIONing.

Finding the nearest 10 pizza parlors on a map needs a 2D index. Partition pruning sort of gives a second dimension. See Latitude/Longitude Indexing That uses PARTITION BY RANGE(latitude) together with PRIMARY KEY(longitude, ...)

The big win for Case #2: Scanning fewer rows.

Use case #3 -- hot spot. This is a bit complicated to explain. Given this combination:

- A table's index is too big to be cached, but the index for one partition is cacheable, and
- The index is randomly accessed, and
- Data ingestion would normally be I/O bound due to updating the index Partitioning can keep all the index "hot" in RAM, thereby avoiding a lot of I/O.

The big win for Case #3: Improving caching to decrease I/O to speed up operations.

AUTO_INCREMENT in PARTITION

- For AUTO_INCREMENT to work (in any table), it must be the first field in some index. Period. There are no other requirements on indexing it.
- Being the first field in some index lets the engine find the 'next' value when opening the table.
- AUTO_INCREMENT need not be UNIQUE. What you lose: prevention of explicitly inserting a duplicate id. (This is rarely needed, anyway.)

Examples (where id is AUTO_INCREMENT):

- PRIMARY KEY (...), INDEX(id)
- PRIMARY KEY (...), UNIQUE(id, partition_key) -- not useful
- INDEX(id), INDEX(...) (but no UNIQUE keys)
- PRIMARY KEY(id), ... -- works only if id is the partition key (not very useful)

PARTITION maintenance for the time-series case

Let's focus on the maintenance task involved in Case #1, as described above.

You have a large table that is growing on one end and being pruned on the other. Examples include news, logs, and other

transient information. PARTITION BY RANGE is an excellent vehicle for such a table.

- DROP PARTITION is much faster than DELETE. (This is the big reason for doing this flavor of partitioning.)
- Queries often limit themselves to 'recent' data, thereby taking advantage of "partition pruning".

Depending on the type of data, and how long before it expires, you might have daily or weekly or hourly (etc) partitions.

There is no simple SQL statement to "drop partitions older than 30 days" or "add a new partition for tomorrow". It would be tedious to do this by hand every day.

High level view of the code

```
ALTER TABLE tbl
    DROP PARTITION from20120314;
ALTER TABLE tbl
    REORGANIZE PARTITION future INTO (
        PARTITION from20120415 VALUES LESS THAN (TO_DAYS('2012-04-16')),
        PARTITION future     VALUES LESS THAN MAXVALUE);
```

After which you have...

```
    CREATE TABLE tbl (
        dt DATETIME NOT NULL,  -- or DATE
        ...
        PRIMARY KEY (..., dt),
        UNIQUE KEY (..., dt),
        ...
    )
    PARTITION BY RANGE (TO_DAYS(dt)) (
        PARTITION start        VALUES LESS THAN (0),
        PARTITION from20120315 VALUES LESS THAN (TO_DAYS('2012-03-16')),
        PARTITION from20120316 VALUES LESS THAN (TO_DAYS('2012-03-17')),
        ...
        PARTITION from20120414 VALUES LESS THAN (TO_DAYS('2012-04-15')),
        PARTITION from20120415 VALUES LESS THAN (TO_DAYS('2012-04-16')),
        PARTITION future       VALUES LESS THAN MAXVALUE
    );
```

Why?

Perhaps you noticed some odd things in the example. Let me explain them.

- Partition naming: Make them useful.
- from20120415 ... 04-16: Note that the LESS THAN is the next day's date
- The "start" partition: See paragraph below.
- The "future" partition: This is normally empty, but it can catch overflows; more later.
- The range key (dt) must be included in any PRIMARY or UNIQUE key.
- The range key (dt) should be last in any keys it is in -- You have already "pruned" with it; it is almost useless in the index, especially at the beginning.
- DATETIME, etc -- I picked this datatype because it is typical for a time series. Newer MySQL versions allow TIMESTAMP. INT could be used; etc.
- There is an extra day (03-16 thru 04-16): The latest day is only partially full.

Why the bogus "start" partition? If an invalid datetime (Feb 31) were to be used, the datetime would turn into NULL. NULLs are put into the first partition. Since any SELECT could have an invalid date (yeah, this stretching things), the partition pruner always includes the first partition in the resulting set of partitions to search. So, if the SELECT must scan the first partition, it would be slightly more efficient if that partition were empty. Hence the bogus "start" partition. Longer discussion, by The Data Charmer 5.5 eliminates the bogus check, but only if you switch to a new syntax:

```
    PARTITION BY RANGE COLUMNS(dt) (
    PARTITION day_20100226 VALUES LESS THAN ('2010-02-27'), ...
```

More on the "future" partition. Sooner or later the cron/EVENT to add tomorrow's partition will fail to run. The worst that could happen is for tomorrow's data to be lost. The easiest way to prevent that is to have a partition ready to catch it, even if this partition is normally always empty.

Having the "future" partition makes the ADD PARTITION script a little more complex. Instead, it needs to take tomorrow's data from "future" and put it into a new partition. This is done with the REORGANIZE command shown. Normally nothing need be moved, and the ALTER takes virtually zero time.

When to do the ALTERs?

- DROP if the oldest partition is "too old".
- Add 'tomorrow' near the end of today, but don't try to add it twice.
- Do not count partitions -- there are two extra ones. Use the partition names or information_schema.PARTITIONS.PARTITION_DESCRIPTION.
- DROP/Add only once in the script. Rerun the script if you need more.
- Run the script more often than necessary. For daily partitions, run the script twice a day, or even hourly. Why? Automatic repair.

Variants

As I have said many times, in many places, BY RANGE is perhaps the only useful variant. And a time series is the most common use for PARTITIONing.

- (as discussed here) DATETIME/DATE with TO_DAYS()
- DATETIME/DATE with TO_DAYS(), but with 7-day intervals
- TIMESTAMP with TO_DAYS(). (version 5.1.43 or later)
- PARTITION BY RANGE COLUMNS(DATETIME) (5.5.0)
- PARTITION BY RANGE(TIMESTAMP) (version 5.5.15 / 5.6.3)
- PARTITION BY RANGE(TO_SECONDS()) (5.6.0)
- INT UNSIGNED with constants computed as unix timestamps.
- INT UNSIGNED with constants for some non-time-based series.
- MEDIUMINT UNSIGNED containing an "hour id": FLOOR(FROM_UNIXTIME(timestamp) / 3600)
- Months, Quarters, etc: Concoct a notation that works.

How many partitions?

- Under, say, 5 partitions -- you get very little of the benefits.
- Over, say, 50 partitions, and you hit inefficiencies elsewhere.
- Certain operations (SHOW TABLE STATUS, opening the table, etc) open every partition.
- MyISAM, before version 5.6.6, would lock all partitions before pruning!
- Partition pruning does not happen on INSERTs (until Version 5.6.7), so INSERT needs to open all the partitions.
- A possible 2-partition use case: http://forums.mysql.com/read.php?24,633179,633179
- 8192 partitions is a hard limit (1024 before MariaDB 10.0.4).
- Before "native partitions" (5.7.6), each partition consumed a chunk of memory.

Detailed code

Reference implementation, in Perl, with demo of daily partitions

The complexity of the code is in the discovery of the PARTITION names, especially of the oldest and the 'next'.

To run the demo,

- Install Perl and DBIx::DWIW (from CPAN).
- copy the txt file (link above) to demo_part_maint.pl
- execute perl demo_part_maint.pl to get the rest of the instructions

The program will generate and execute (when needed) either of these:

```
ALTER TABLE tbl REORGANIZE PARTITION
      future
INTO (
      PARTITION from20150606 VALUES LESS THAN (736121),
      PARTITION future VALUES LESS THAN MAXVALUE
)

ALTER TABLE tbl
                  DROP PARTITION from20150603
```

Postlog

Original writing -- Oct, 2012; Use cases added: Oct, 2014; Refreshed: June, 2015; 8.0: Sep, 2016

Slides from Percona Amsterdam 2015

PARTITIONing requires at least MySQL 5.1

The tips in this document apply to MySQL, MariaDB, and Percona.

- More on PARTITIONing
- LinkedIn discussion
- Why NOT Partition
- Geoff Montee's Stored Proc

Future (as envisioned in 2016):

- MySQL 5.7.6 has "native partitioning for InnoDB".
- FOREIGN KEY support, perhaps in a later 8.0.xx.
- "GLOBAL INDEX" -- this would avoid the need for putting the partition key in every unique index, but make DROP PARTITION costly. This will be farther into the future.

MySQL 8.0, released Sep, 2016, not yet GA)

- Only InnoDB tables can be partitioned -- MariaDB is likely to continue maintaining Partitioning on non-InnoDB tables, but Oracle is clearly not.
- Some of the problems having lots of partitions are lessened by the Data-Dictionary-in-a-table.

Native partitioning will give:

- This will improve performance slightly by combining two "handlers" into one.
- Decreased memory usage, especially when using a large number of partitions.

2.5.5 Partitioning Limitations with MariaDB

The following limitations apply to partitioning in MariaDB:

- Each table can contain a maximum of 8192 partitions. Until MariaDB 10.0.3, the limit was 1024.
- Queries are never parallelized, even when they involve multiple partitions.
- A table can only be partitioned if the storage engine supports partitioning.
- All partitions must use the same storage engine. For a workaround, see Using CONNECT - Partitioning and Sharding.
- A partitioned table cannot contain, or be referenced by, foreign keys.
- The query cache is not aware of partitioning and partition pruning. Modifying a partition will invalidate the entries related to the whole table.
- Updates can run more slowly when binlog_format=ROW and a partitioned table is updated than an equivalent update of a non-partitioned table.
- All columns used in the partitioning expression for a partitioned table must be part of every unique key that the table may have.

2.5.6 Partitions Files

A partitioned table is stored in multiple files. By default, these files are stored in the MariaDB (or InnoDB) data directory. It is possible to keep them in different paths by specifying DATA_DIRECTORY and INDEX_DIRECTORY table options. This is useful to store different partitions on different devices.

> Note that, if the innodb_file_per_table server system variable is set to 0 at the time of the table creation, all partitions will be stored in the system tablespace.

The following files exist for each partitioned tables:

File name	Notes
table_name.frm	Contains the table definition. Non-partitioned tables have this file, too.
table_name.par	Contains the partitions definitions.
table_name#P#partition_name.ext	Normal files created by the storage engine use this pattern for names. The extension depends on the storage engine.

For example, an InnoDB table with 4 partitions will have the following files:

```
orders.frm
orders.par
orders#P#p0.ibd
orders#P#p1.ibd
orders#P#p2.ibd
orders#P#p3.ibd
```

If we convert the table to MyISAM, we will have these files:

```
orders.frm
orders.par
orders#P#p0.MYD
orders#P#p0.MYI
orders#P#p1.MYD
orders#P#p1.MYI
orders#P#p2.MYD
orders#P#p2.MYI
orders#P#p3.MYD
orders#P#p3.MYI
```

2.5.7 Partitions Metadata

The PARTITIONS table in the INFORMATION_SCHEMA database contains information about partitions.

The SHOW TABLE STATUS statement contains a `Create_options` column, that contains the string 'partitioned' for partitioned tables.

The SHOW CREATE TABLE statement returns the CREATE TABLE statement that can be used to re-create a table, including the partitions definition.

1.1.1.2.9.1.1.32 Information Schema PARTITIONS Table

5.4.5 MariaDB Audit Plugin

2.7 Variables and Modes

The different variables and modes to use to affect how MariaDB works.

Full List of MariaDB Options, System and Status Variables

Complete alphabetical list of all MariaDB options as well as system and status variables.

Server Status Variables

List and description of the Server Status Variables.

Server System Variables

List of system variables.

OLD_MODE

Used to emulate behavior from older MariaDB and MySQL versions.

SQL_MODE

Used to emulate behavior from other SQL servers.

SQL_MODE=MSSQL

Microsoft SQL Server compatibility mode.

SQL_MODE=ORACLE

MariaDB understands a subset of Oracle's PL/SQL language.

There are 3 related questions.

2.7.1 Full List of MariaDB Options, System and Status Variables

Alphabetical list of all mariadbd Options, Server System Variables and Server Status Variables. The convention used is that variable names are listed with '_' and options with '-'. If a variable and option both exist, both versions are listed for easy searching.

Name
-a (--ansii)
--abort-slave-event-count
Aborted_clients
Aborted_connects
Aborted_connects_preauth
Access_denied_errors
Acl_column_grants
Acl_database_grants
Acl_function_grants
Acl_package_body_grants
Acl_package_spec_grants
Acl_procedure_grants
Acl_proxy_users
Acl_role_grants
Acl_roles
Acl_table_grants
Acl_users
--allow-suspicious-udfs, allow_suspicious_udfs
--alter-algorithm, alter_algorithm
--analyze-sample-percentage, analyze_sample_percentage
--ansii
--aria-block-size, aria_block_size
--aria-checkpoint-interval, aria_checkpoint_interval
--aria-checkpoint-log-activity, aria_checkpoint_log_activity
--aria-encrypt-tables, aria_encrypt_tables
--aria-force-start-after-recovery-failures, aria_force_start_after_recovery_failures
--aria-group-commit, aria_group_commit
--aria-group-commit-interval, aria_group_commit_interval
--aria-log-dir-path
--aria-log-file-size, aria_log_file_size
--aria-log-purge-type, aria_log_purge_type
--aria-max-sort-file-size, aria_max_sort_file_size
--aria-page-checksum, aria_page_checksum
--aria-pagecache-age-threshold, aria_pagecache_age_threshold
Aria_pagecache_blocks_not_flushed
Aria_pagecache_blocks_unused
Aria_pagecache_blocks_used
--aria-pagecache-buffer-size, aria_pagecache_buffer_size
--aria-pagecache-division-limit, aria_pagecache_division_limit
--aria-pagecache-file-hash-size, aria_pagecache_file_hash_size
Aria_pagecache_read_requests
Aria_pagecache_reads
Aria_pagecache_write_requests
Aria_pagecache_writes
--aria-recover, aria_recover

--aria-recover-options, aria_recover_options
--aria-repair-threads, aria_repair_threads
--aria-sort-buffer-size, aria_sort_buffer_size
--aria-stats-method, aria_stats_method
--aria-sync-log-dir, aria_sync_log_dir
Aria_transaction_log_syncs
aria_used_for_temp_tables
--autocommit, autocommit
--auto-increment-increment, auto_increment_increment
--auto-increment-offset, auto_increment_offset
--automatic-sp-privileges, automatic_sp_privileges
aws_key_management_key_spec
aws_key_management_log_level
aws_key_management_master_key_id
aws_key_management_mock
aws_key_management_region
aws_key_management_request_timeout
aws_key_management_rotate_key
--back-log, back_log
-b, --basedir, basedir
--big-tables, big_tables
--bind-address, bind_address
--binlog-alter-two-phase, binlog_alter_two_phase
--binlog-annotate-row-events, binlog_annotate_row_events
Binlog_bytes_written
Binlog_cache_disk_use
--binlog-cache-size, binlog_cache_size
Binlog_cache_use
--binlog-checksum, binlog_checksum
--binlog-commit-wait-count, binlog_commit_wait_count
--binlog-commit-wait-count, binlog_commit_wait_usec
Binlog_commits
--binlog-direct-non-transactional-updates, binlog_direct_non_transactional_updates
--binlog-do-db, binlog_do_db
--binlog-expire-logs-seconds, binlog_expire_logs_seconds
--binlog-file-cache-size, binlog_file_cache_size
--binlog-format, binlog_format
Binlog_group_commits
Binlog_group_commit_trigger_count
Binlog_group_commit_trigger_lock_wait
Binlog_group_commit_trigger_timeout
--binlog-ignore-db, binlog_ignore_db
--binlog-optimize-thread-scheduling, binlog_optimize_thread_scheduling
--binlog-row-image, binlog_row_image
--binlog-row-event-max-size, binlog_row_event_max_size
--binlog-row-metadata, binlog_row_metadata

Binlog_snapshot_file
Binlog_snapshot_position
Binlog_stmt_cache_disk_use
Binlog_stmt_cache_use
--binlog-stmt-cache-size, binlog_stmt_cache_size
--block-encryption-mode, block_encryption_mode
--bootstrap
--bulk-insert-buffer-size, bulk_insert_buffer_size
Busy_time
Bytes_received
Bytes_sent
cassandra_default_thrift_host
cassandra_failure_retries
cassandra_insert_batch_size
cassandra_multiget_batch_size
Cassandra_multiget_keys_scanned
Cassandra_multiget_reads
Cassandra_multiget_rows_read
Cassandra_network_exceptions
cassandra_read_consistency
cassandra_rnd_batch_size
Cassandra_row_inserts
Cassandra_row_insert_batches
Cassandra_timeout_exceptions
Cassandra_unavailable_exceptions
cassandra_write_consistency
character_set_client
--character-set-client-handshake
character_set_collations
character_set_connection
character_set_database
--character-set-filesystem, character_set_filesystem
character_set_results
-C, --character-set-server, character_set_server
character_set_system
--character-sets-dir, character_sets_dir
--check-constraint-checks, check_constraint_checks
-r, --chroot
collation_connection
collation_database
--collation-server, collation_server
Column_compressions
--column-compression-threshold, column_compression_threshold
--column-compression-zlib-level, column_compression_zlib_level
--column-compression-zlib-strategy, column_compression_zlib_strategy
--column-compression-zlib-wrap, column_compression_zlib_wrap

Column_decompressions
Com_admin_commands
Com_alter_db
Com_alter_db_upgrade
Com_alter_event
Com_alter_function
Com_alter_procedure
Com_alter_sequence
Com_alter_server
Com_alter_table
Com_alter_tablespace
Com_alter_user
Com_analyze
Com_assign_to_keycache
Com_backup
Com_backup_lock
Com_backup_table
Com_begin
Com_binlog
Com_call_procedure
Com_change_db
Com_change_master
Com_check
Com_checksum
Com_commit
Com_compound_sql
Com_create_db
Com_create_event
Com_create_function
Com_create_index
Com_create_package
Com_create_package_body
Com_create_procedure
Com_create_role
Com_create_sequence
Com_create_server
Com_create_table
Com_create_temporary_table
Com_create_trigger
Com_create_udf
Com_create_user
Com_create_view
Com_dealloc_sql
Com_delete
Com_delete_multi
Com_do

Com_drop_db
Com_drop_event
Com_drop_function
Com_drop_index
Com_drop_package
Com_drop_package_body
Com_drop_procedure
Com_drop_role
Com_drop_sequence
Com_drop_server
Com_drop_table
Com_drop_temporary_table
Com_drop_trigger
Com_drop_user
Com_drop_user
Com_drop_view
Com_empty_query
Com_execute_immediate
Com_execute_sql
Com_flush
Com_get_diagnostics
Com_grant
Com_grant_role
Com_ha_close
Com_ha_open
Com_ha_read
Com_help
Com_insert
Com_insert_select
Com_install_plugin
Com_kill
Com_load
Com_load_master_data
Com_load_master_table
Com_lock_tables
Com_multi
Com_optimize
Com_preload_keys
Com_prepare_sql
Com_purge
Com_purge_before_date
Com_release_savepoint
Com_rename_table
Com_rename_user
Com_repair
Com_replace

Com_replace_select
Com_reset
Com_resignal
Com_restore_table
Com_revoke
Com_revoke_all
Com_revoke_grant
Com_rollback
Com_rollback_to_savepoint
Com_savepoint
Com_select
Com_set_option
Com_show_authors
Com_show_binlog_events
Com_show_binlogs
Com_show_charsets
Com_show_client_statistics
Com_show_collations
Com_show_column_types
Com_show_contributors
Com_show_create_db
Com_show_create_event
Com_show_create_func
Com_show_create_package
Com_show_create_package_body
Com_show_create_proc
Com_show_create_table
Com_show_create_trigger
Com_show_create_user
Com_show_databases
Com_show_engine_logs
Com_show_engine_mutex
Com_show_engine_status
Com_show_events
Com_show_errors
Com_show_explain
Com_show_fields
Com_show_function_status
Com_show_generic
Com_show_grants
Com_show_keys
Com_show_index_statistics
Com_show_binlog_status
Com_show_master_status
Com_show_new_master
Com_show_open_tables

Com_show_package_status
Com_show_package_body_status
Com_show_plugins
Com_show_privileges
Com_show_procedure_status
Com_show_processlist
Com_show_profile
Com_show_profiles
Com_show_relaylog_events
Com_show_slave_hosts
Com_show_slave_status
Com_show_status
Com_show_storage_engines
Com_show_table_statistics
Com_show_table_status
Com_show_tables
Com_show_triggers
Com_show_user_statistics
Com_show_variable
Com_show_warnings
Com_shutdown
Com_signal
Com_slave_start
Com_slave_stop
Com_start_all_slaves
Com_start_slave
Com_stop_all_slaves
Com_stop_slave
Com_stmt_close
Com_stmt_execute
Com_stmt_fetch
Com_stmt_prepare
Com_stmt_reprepare
Com_stmt_reset
Com_stmt_send_long_data
Com_truncate
Com_uninstall_plugin
Com_unlock_tables
Com_update
Com_update_multi
Com_xa_commit
Com_xa_end
Com_xa_prepare
Com_xa_recover
Com_xa_rollback
Com_xa_start

--completion-type, completion_type
Compression
--concurrent-insert, concurrent_insert
--connect-class-path, connect_class_path
--connect-cond-push, connect_cond_push
--connect-conv-size, connect_conv_size
--connect-default-depth, connect_default_depth
--connect-default-prec, connect_default_prec
--connect-enable-mongo, connect_enable_mongo
--connect-exact-info, connect_exact_info
--connect-force-bson, connect_force_bson
--connect-indx-map, connect_indx_map
--connect-java-wrapper, connect_java_wrapper
--connect-json-all-path, connect_json_all_path
--connect-json-grp-size, connect_json_grp_size
--connect-json-null, connect_json_null
--connect-jvm-path, connect_jvm_path
--connect-timeout, connect_timeout
--connect-type-conv, connect_type_conv
--connect-use-tempfile, connect_use_tempfile
--connect-work-size, connect_work_size
--connect-xtrace, connect_xtrace
Connection_errors_accept
Connection_errors_internal
Connection_errors_max_connections
Connection_errors_peer_address
Connection_errors_select
Connection_errors_tcpwrap
Connections
--console
--core-file, core_file
Cpu_time
--cracklib-password-check
--cracklib-password-check-dictionary, cracklib_password_check-dictionary
Created_tmp_disk_tables
Created_tmp_files
Created_tmp_tables
-h, --datadir, datadir
--date-format, date_format
--datetime-format, datetime_format
--deadlock-search-depth-long, deadlock_search_depth_long
--deadlock-search-depth-short, deadlock_search_depth_short
--deadlock-timeout-long, deadlock_timeout_long
--deadlock-timeout-short, deadlock_timeout_short
-#, --debug, debug
--debug-assert-if-crashed-table

--debug-binlog-fsync-sleep
--debug-crc-break
--debug-flush
--debug-no-sync
--debug-no-thread-alarm, debug_no_thread_alarm
debug_sync
--debug-sync-timeout
--default-character-set
--default-master-connection, default_master_connection
--default-password-lifetime, default_password_lifetime
--default-regex-flags, default_regex_flags
--default-storage-engine, default_storage_engine
--default-table-type, default_table_type
--default-tmp-storage-engine, default_tmp_storage_engine
--default-time-zone
--default-week-format, default_week_format
--defaults-extra-file
--defaults-file
--delay-key-write, delay_key_write
Delayed_errors
--delayed-insert-limit, delayed_insert_limit
Delayed_insert_threads
--delayed-insert-timeout, delayed_insert_timeout
--delayed-queue-size, delayed_queue_size
Delayed_writes
Delete_scan
--des-key-file
--disconnect-on-expired-password, disconnect_on_expired_password
--disconnect-slave-event-count
--disks
--div-precision-increment, div_precision_increment
Empty_queries
--encrypt-binlog, encrypt_binlog
--encrypt-tmp-disk-tables, encrypt_tmp_disk_tables
--encrypt-tmp-files, encrypt_tmp_files
--encryption-algorithm, encryption_algorithm
enforce_storage_engine
--engine-condition-pushdown, engine_condition_pushdown
--eq-range-index-dive-limit, eq_range_index_dive_limit
error_count
--event-scheduler, event_scheduler
Executed_events
Executed_triggers
-T, --exit-info
--expensive-subquery-limit, expensive_subquery_limit
--expire-logs-days, expire_logs_days

--explicit-defaults-for-timestamp, explicit_defaults_for_timestamp
--external-locking
external_user
--extra-max-connections, extra_max_connections
--extra-port, extra_port
Feature_application_time_periods
Feature_check_constraint
Feature_custom_aggregate_functions
Feature_delay_key_write
Feature_dynamic_columns
Feature_fulltext
Feature_gis
Feature_insert_returning
Feature_invisible_columns
Feature_json
Feature_locale
Feature_subquery
Feature_timezone
Feature_trigger
Feature_window_functions
Feature_xml
--feedback
--feedback-http-proxy, feedback_http_proxy
--feedback-send-retry-wait, feedback_send_retry_wait
--feedback-send-timeout, feedback_send_timeout
feedback_server_uid
--feedback-url, feedback_url
--feedback-user-info, feedback_user_info
--file-key-management-encryption-algorithm, file_key_management_encryption_algorithm
--file-key-management-filekey, file_key_management_filekey
--file-key-management-filename, file_key_management_filename
--flashback
--flush, flush
Flush_commands
--flush-time, flush_time
foreign_key_checks
--ft-boolean-syntax, ft_boolean_syntax
--ft-max-word-len, ft_max_word_len
--ft-min-word-len, ft_min_word_len
--ft-query-expansion-limit, ft_query_expansion_limit
--ft-stopword-file, ft_stopword_file
--gdb
--general-log, general_log
--general-log-file, general_log_file
--getopt-prefix-matching
--group-concat-max-len, group_concat_max_len

--gssapi-keytab-path, gssapi_keytab_path
--gssapi-principal-name, gssapi_principal_name
--gssapi-mech-name, gssapi_mech_name
gtid_binlog_pos
gtid_binlog_state
--gtid-cleanup-batch-size, gtid_cleanup_batch_size
gtid_current_pos
--gtid-domain-id, gtid_domain_id
--gtid-ignore-duplicates, gtid_ignore_duplicates
gtid_pos_auto_engines
gtid_seq_no
gtid_slave_pos
--gtid-strict-mode, gtid_strict_mode
-h, --datadir, datadir
Handler_commit
Handler_delete
Handler_discover
Handler_external_lock
Handler_icp_attempts
Handler_icp_match
Handler_mrr_init
Handler_mrr_key_refills
Handler_mrr_rowid_refills
Handler_prepare
Handler_read_first
Handler_read_key
Handler_read_last
Handler_read_next
Handler_read_prev
Handler_read_retry
Handler_read_rnd
Handler_read_rnd_deleted
Handler_read_rnd_next
Handler_rollback
Handler_savepoint
Handler_savepoint_rollback
Handler_tmp_delete
Handler_tmp_update
Handler_tmp_write
Handler_update
Handler_write
--handlersocket-accept-balance, handlersocket_accept_balance
--handlersocket-address, handlersocket_address
--handlersocket-backlog, handlersocket_backlog
--handlersocket-epoll, handlersocket_epoll

--handlersocket-plain-secret, handlersocket_plain_secret
--handlersocket-plain-secret-wr, handlersocket_plain_secret_wr
--handlersocket-port, handlersocket_port
--handlersocket-port-wr, handlersocket_port_wr
--handlersocket-rcvbuf, handlersocket_rcvbuf
--handlersocket-readsize, handlersocket_readsize
--handlersocket-sndbuf, handlersocket_sndbuf
--handlersocket-threads, handlersocket_threads
--handlersocket-threads_wr, handlersocket_threads_wr
--handlersocket-timeout, handlersocket_timeout
--handlersocket-verbose, handlersocket_verbose
--handlersocket-wrlock-timeout, handlersocket_wrlock_timeout
hashicorp-key-management-cache-timeout
hashicorp-key-management-cache-version-timeout
hashicorp-key-management-caching-enabled
hashicorp-key-management-check-kv-version
hashicorp-key-management-retries
hashicorp-key-management-timeout
hashicorp-key-management-token
hashicorp-key-management-use-cache-on-timeout
hashicorp-key-management-vault-ca
hashicorp-key-management-vault-url
have_compress
have_crypt
have_csv
have_dynamic_loading
have_geometry
have_innodb
have_ndbcluster
have_openssl
have_partitioning
have_profiling
have_query_cache
have_rtree_keys
have_ssl
have_symlink
--help
--histogram-size, histogram_size
--histogram-type, histogram_type
--host-cache-size, host_cache_size
hostname
identity
--idle-readonly-transaction-timeout, idle_readonly_transaction_timeout
--idle-transaction-timeout, idle_transaction_timeout
--idle-write-transaction-timeout, idle_write_transaction_timeout
--ignore-db-dirs, ignore_db_dirs

--ignore-builtin-innodb, ignore_builtin_innodb
--in-predicate-conversion-threshold, in_predicate_conversion_threshold
in_transaction
--init-connect, init_connect
--init-file, init_file
--init-rpl-role
--init-slave, init_slave
--innodb
--innodb-adaptive-checkpoint, innodb_adaptive_checkpoint
--innodb-adaptive-flushing, innodb_adaptive_flushing
--innodb-adaptive-flushing-lwm, innodb_adaptive_flushing_lwm
--innodb_adaptive-flushing-method, innodb_adaptive_flushing_method
Innodb_adaptive_hash_cells
Innodb_adaptive_hash_hash_searches
Innodb_adaptive_hash_heap_buffers
--innodb-adaptive-hash-index, innodb_adaptive_hash_index
--innodb-adaptive-hash-index-partitions, innodb_adaptive_hash_index_partitions
--innodb-adaptive-hash-index-parts, innodb_adaptive_hash_index_parts
Innodb_adaptive_hash_non_hash_searches
--innodb-adaptive-max-sleep-delay, innodb_adaptive_max_sleep_delay
--innodb-additional-mem-pool-size, innodb_additional_mem_pool_size
--innodb-api-bk-commit_interval, innodb_api_bk_commit_interval
--innodb-api-disable-rowlock, innodb_api_disable_rowlock
--innodb_api_enable_binlog, innodb_api_enable_binlog
--innodb-api-enable-mdl, innodb_api_enable_mdl
--innodb-api-trx-level, innodb_api_trx_level
--innodb-auto-lru-dump, innodb-auto-lru-dump
--innodb-autoextend-increment, innodb_autoextend_increment
--innodb-autoinc-lock-mode, innodb_autoinc_lock_mode
Innodb_available_undo_logs
Innodb_background_log_sync
--innodb-background-scrub-data-check-interval, innodb_background_scrub_data_check_interval
--innodb-background-scrub-data-compressed, innodb_background_scrub_data_compressed
--innodb-background-scrub-data-interval, innodb_background_scrub_data_interval
--innodb-background-scrub-data-uncompressed, innodb_background_scrub_data_uncompressed
--innodb-blocking-buffer-pool-restore, innodb_blocking_buffer_pool_restore
--innodb-buf-dump-status-frequency, innodb_buf_dump_status_frequency
Innodb_buffer_pool_bytes_data
Innodb_buffer_pool_bytes_dirty
--innodb-buffer-pool-chunk-size, innodb_buffer_pool_chunk_size
--innodb-buffer-pool-dump-at-shutdown, innodb_buffer_pool_dump_at_shutdown
--innodb-buffer-pool-dump-now, innodb_buffer_pool_dump_now
--innodb-buffer-pool-dump-pct, innodb_buffer_pool_dump_pct
Innodb_buffer_pool_dump_status
--innodb-buffer-pool-evict, innodb_buffer_pool_evict

--innodb-buffer-pool-filename, innodb_buffer_pool_filename
--innodb-buffer-pool-instances, innodb_buffer_pool_instances
--innodb-buffer-pool-load-abort, innodb_buffer_pool_load_abort
--innodb-buffer-pool-load-at-startup, innodb_buffer_pool_load_at_startup
--innodb-buffer-pool-load-now, innodb_buffer_pool_load_now
Innodb_buffer_pool_load_incomplete
--innodb-buffer-pool-load-pages-abort, innodb_buffer_pool_load_pages_abort
Innodb_buffer_pool_load_status
Innodb_buffer_pool_pages_data
Innodb_buffer_pool_pages_dirty
Innodb_buffer_pool_pages_flushed
Innodb_buffer_pool_pages_LRU_flushed
Innodb_buffer_pool_pages_LRU_freed
Innodb_buffer_pool_pages_free
Innodb_buffer_pool_pages_made_not_young
Innodb_buffer_pool_pages_made_young
Innodb_buffer_pool_pages_misc
Innodb_buffer_pool_pages_old
Innodb_buffer_pool_pages_total
--innodb-buffer-pool-populate, innodb_buffer_pool_populate
Innodb_buffer_pool_read_ahead
Innodb_buffer_pool_read_ahead_evicted
Innodb_buffer_pool_read_ahead_rnd
Innodb_buffer_pool_read_requests
Innodb_buffer_pool_reads
Innodb_buffer_pool_resize_status
--innodb-buffer-pool-restore-at-startup, innodb_buffer_pool_restore_at_startup
--innodb-buffer-pool-shm-checksum, innodb_buffer_pool_shm_checksum
--innodb-buffer-pool-shm-key, innodb_buffer_pool_shm_key
--innodb-buffer-pool-size, innodb_buffer_pool_size
Innodb_buffer_pool_wait_free
Innodb_buffer_pool_write_requests
Innodb_buffered_aio_submitted
--innodb-change-buffer-dump, innodb_change_buffer_dump
--innodb-change-buffer-max-size, innodb_change_buffer_max_size
--innodb-change-buffering, innodb_change_buffering
--innodb-change-buffering-debug, innodb_change_buffering_debug
Innodb_checkpoint_age
--innodb-checkpoint-age-target, innodb_checkpoint_age_target
Innodb_checkpoint_max_age
Innodb_checkpoint_target_age
--innodb-checksum-algorithm, innodb_checksum_algorithm
--innodb-checksums, innodb_checksums
--innodb-cleaner-lsn-age-factor, innodb_cleaner_lsn_age_factor
--innodb-cmp
--innodb-cmp-per-index-enabled, innodb_cmp_per_index_enabled

--innodb-cmp-reset
--innodb-cmpmem
--innodb-cmpmem-reset
--innodb-commit-concurrency, innodb_commit_concurrency
--innodb-compression-algorithm, innodb_compression_algorithm
--innodb-compression-default, innodb_compression_default
--innodb-compression-failure-threshold-pct, innodb_compression_failure_threshold_pct
--innodb-compression-level, innodb_compression_level
--innodb-compression-pad-pct-max, innodb_compression_pad_pct_max
--innodb-concurrency-tickets, innodb_concurrency_tickets
--innodb-corrupt-table-action, innodb_corrupt_table_action
Innodb_current_row_locks
--innodb-data-file-buffering, innodb_data_file_buffering
--innodb-data-file-path, innodb_data_file_path
--innodb-data-file-write_through, innodb_data_file_write_through
Innodb_data_fsyncs
--innodb-data-home-dir, innodb_data_home_dir
Innodb_data_pending_fsyncs
Innodb_data_pending_reads
Innodb_data_pending_writes
Innodb_data_read
Innodb_data_reads
Innodb_data_writes
Innodb_data_written
Innodb_dblwr_pages_written
Innodb_dblwr_writes
--innodb-deadlock-detect, innodb_deadlock_detect
--innodb-deadlock-report, innodb_deadlock_report
Innodb_deadlocks
--innodb-default-encryption-key-id, innodb_default_encryption_key_id
--innodb-default-page-encryption-key, innodb_default_page_encryption_key
--innodb-default-row-format, innodb_default_row_format
--innodb-defragment, innodb_defragment
Innodb_defragment_compression_failures
Innodb_defragment_count
Innodb_defragment_failures
--innodb-defragment-fill-factor, innodb_defragment_fill_factor
--innodb-defragment-fill-factor-n-recs, innodb_defragment_fill_factor_n_recs
--innodb-defragment-frequency, innodb_defragment_frequency
--innodb-defragment-n-pages, innodb_defragment_n_pages
--innodb-defragment-stats-accuracy, innodb_defragment_stats_accuracy
--innodb-dict-size-limit, innodb_dict_size_limit
Innodb_dict_tables
--innodb-disable-sort-file-cache, innodb_disable_sort_file_cache
innodb_disallow_writes
--innodb-doublewrite, innodb_doublewrite

--innodb-doublewrite-file, innodb_doublewrite_file
--innodb_empty-free-list-algorithm, innodb_empty_free_list_algorithm
--innodb-enable-unsafe-group-commit, innodb_enable_unsafe_group_commit
--innodb-encrypt-log, innodb_encrypt_log
--innodb-encrypt-tables, innodb_encrypt_tables
--innodb-encrypt-temporary-tables, innodb_encrypt_temporary_tables
Innodb_encryption_n_merge_blocks_decrypted
Innodb_encryption_n_merge_blocks_encrypted
Innodb_encryption_n_rowlog_blocks_decrypted
Innodb_encryption_n_rowlog_blocks_encrypted
Innodb_encryption_n_temp_blocks_decrypted
Innodb_encryption_n_temp_blocks_encrypted
Innodb_encryption_num_key_requests
--innodb-encryption-rotate-key-age, innodb_encryption_rotate_key_age
Innodb_encryption_rotation_estimated_iops
--innodb-encryption-rotation-iops, innodb_encryption_rotation_iops
Innodb_encryption_rotation_pages_flushed
Innodb_encryption_rotation_pages_modified
Innodb_encryption_rotation_pages_read_from_cache
Innodb_encryption_rotation_pages_read_from_disk
--innodb-encryption-threads, innodb_encryption_threads
--innodb-extra-rsegments, innodb_extra_rsegments
--innodb-extra-undoslots, innodb_extra_undoslots
--innodb-fake-changes, innodb_fake_changes
--innodb-fast-checksum, innodb_fast_checksum
--innodb-fast-shutdown, innodb_fast_shutdown
--innodb-fatal-semaphore-wait-threshold, innodb_fatal_semaphore_wait_threshold
--innodb-file-format, innodb_file_format
--innodb-file-format-check, innodb_file_format_check
--innodb-file-format-max, innodb_file_format_max
--innodb-file-io-threads
--innodb-file-per-table, innodb_file_per_table
--innodb-fill-factor, innodb_fill_factor
--innodb-flush-log-at-timeout, innodb_flush_log_at_timeout
--innodb-flush-log-at-trx-commit, innodb_flush_log_at_trx_commit
--innodb-flush-method, innodb_flush_method
--innodb-flush-neighbor-pages, innodb_flush_neighbor_pages
--innodb-flush-neighbors, innodb_flush_neighbors
--innodb-flush-sync, innodb_flush_sync
--innodb-flushing-avg-loops, innodb_flushing_avg_loops
--innodb-force-load-corrupted, innodb_force_load_corrupted
--innodb-force-primary-key, innodb_force_primary_key
--innodb-force-recovery, innodb_force_recovery
--innodb-foreground-preflush, innodb_foreground_preflush
--innodb-ft-aux-table, innodb_ft_aux_table
--innodb-ft-cache-size, innodb_ft_cache_size

--innodb-ft-enable-diag-print, innodb_ft_enable_diag_print
--innodb-ft-enable-stopword, innodb_ft_enable_stopword
--innodb-ft-max-token-size, innodb_ft_max_token_size
--innodb-ft-min-token-size, innodb_ft_min_token_size
--innodb-ft-num-word-optimize, innodb_ft_num_word_optimize
--innodb-ft-result-cache-limit, innodb_ft_result_cache_limit
--innodb-ft-server-stopword-table, innodb_ft_server_stopword_table
--innodb-ft-sort-pll-degree, innodb_ft_sort_pll_degree
--innodb-ft-total-cache-size, innodb_ft_total_cache_size
--innodb-ft-user-stopword-table, innodb_ft_user_stopword_table
Innodb_have_atomic_builtins
Innodb_have_bzip2
Innodb_have_lz4
Innodb_have_lzma
Innodb_have_lzo
Innodb_have_punch_hole
Innodb_have_snappy
Innodb_history_list_length
--innodb-ibuf-accel-rate, innodb_ibuf_accel_rate
--innodb-ibuf-active-contract, innodb_ibuf_active_contract
Innodb_ibuf_discarded_delete_marks
Innodb_ibuf_discarded_deletes
Innodb_ibuf_discarded_inserts
Innodb_ibuf_free_list
--innodb-ibuf-max-size, innodb_ibuf_max_size
Innodb_ibuf_merged_delete_marks
Innodb_ibuf_merged_deletes
Innodb_ibuf_merged_inserts
Innodb_ibuf_merges
Innodb_ibuf_segment_size
Innodb_ibuf_size
--innodb-idle-flush-pct, innodb_idle_flush_pct
--innodb-immediate-scrub-data-uncompressed, innodb_immediate_scrub_data_uncompressed
--innodb-import-table-from-xtrabackup, innodb_import_table_from_xtrabackup
--innodb-index-stats
Innodb_instant_alter_column
--innodb-instant-alter-column-allowed, innodb_instant_alter_column_allowed
--innodb-instrument-semaphores, innodb_instrument_semaphores
--innodb-io-capacity, innodb_io_capacity
--innodb-io-capacity-max, innodb_io_capacity_max
innodb_kill_idle_transaction
--innodb-large-prefix, innodb_large_prefix
--innodb-lazy-drop-table, innodb_lazy_drop_table
--innodb-lock-schedule-algorithm, innodb_lock_schedule_algorithm
--innodb-lock-wait-timeout, innodb_lock_wait_timeout
--innodb-lock-waits

--innodb-locking-fake-changes, innodb_locking_fake_changes
--innodb-locks
--innodb-locks-unsafe-for-binlog, innodb_locks_unsafe_for_binlog
--innodb-log-arch-dir, innodb_log_arch_dir
--innodb-log-arch-expire-sec, innodb_log_arch_expire_sec
--innodb-log-archive, innodb_log_archive
--innodb-log-block-size, innodb_log_block_size
--innodb-log-buffer-size, innodb_log_buffer_size
-- innodb-log-checksum-algorithm, innodb_log_checksum_algorithm
-- innodb-log-checksums, innodb_log_checksums
-- innodb-log-compressed-pages, innodb_log_compressed_pages
--innodb-log-file-buffering, innodb_log_file_buffering
--innodb-log-file-size, innodb_log_file_size
--innodb-log-file-write_through, innodb_log_file_write_through
--innodb-log-files-in-group, innodb_log_files_in_group
--innodb-log-group-home-dir, innodb_log_group_home_dir
--innodb-log-optimize-ddl, innodb_log_optimize_ddl
Innodb_log_waits
--innodb-log-write-ahead-size, innodb_log_write_ahead_size
Innodb_log_write_requests
Innodb_log_writes
--innodb-lru-flush-size, innodb_lru_flush_size
--innodb-lru-scan-depth, innodb_lru_scan_depth
Innodb_lsn_current
Innodb_lsn_flushed
Innodb_lsn_last_checkpoint
Innodb_master_thread_1_second_loops
Innodb_master_thread_10_second_loops
Innodb_master_thread_active_loops
Innodb_master_thread_background_loops
Innodb_master_thread_idle_loops
Innodb_master_thread_main_flush_loops
Innodb_master_thread_sleeps
--innodb-max-bitmap-file-size, innodb_max_bitmap_file_size
--innodb-max-changed-pages, innodb_max_changed_pages
--innodb-max-dirty-pages-pct, innodb_max_dirty_pages_pct
--innodb-max-dirty-pages-pct-lwm, innodb_max_dirty_pages_pct_lwm
--innodb-max-purge-lag, innodb_max_purge_lag
--innodb-max-purge-lag-delay, innodb_max_purge_lag_delay
--innodb-max-purge-lag-wait, innodb_max_purge_lag_wait
Innodb_max_trx_id
--innodb-max-undo-log-size, innodb_max_undo_log_size
Innodb_mem_adaptive_hash
Innodb_mem_dictionary
Innodb_mem_total
--innodb-merge-sort-block-size, innodb_merge_sort_block_size

--innodb-mirrored-log-groups, innodb_mirrored_log_groups
--innodb-monitor-disable, innodb_monitor_disable
--innodb-monitor-enable, innodb_monitor_enable
--innodb-monitor-reset, innodb_monitor_reset
--innodb-monitor-reset-all, innodb_monitor_reset_all
--innodb-mtflush-threads, innodb_mtflush_threads
Innodb_mutex_os_waits
Innodb_mutex_spin_rounds
Innodb_mutex_spin_waits
Innodb_num_index_pages_written
Innodb_num_non_index_pages_written
Innodb_num_open_files
Innodb_num_page_compressed_trim_op
Innodb_num_page_compressed_trim_op_saved
Innodb_num_pages_encrypted
Innodb_num_pages_page_compressed
Innodb_num_pages_page_compression_error
Innodb_num_pages_page_decompressed
Innodb_num_pages_page_decrypted
Innodb_num_pages_page_encryption_error
--innodb-numa-interleave, innodb_numa_interleave
--innodb-old-blocks-pct, innodb_old_blocks_pct
--innodb-old-blocks-time, innodb_old_blocks_time
Innodb_oldest_view_low_limit_trx_id
--innodb-online-alter-log-max-size, innodb_online_alter_log_max_size
Innodb_onlineddl_pct_progress
Innodb_onlineddl_rowlog_pct_used
Innodb_onlineddl_rowlog_rows
--innodb-open-files, innodb_open_files
--innodb-optimize-fulltext-only, innodb_optimize_fulltext_only
Innodb_os_log_fsyncs
Innodb_os_log_pending_fsyncs
Innodb_os_log_pending_writes
Innodb_os_log_written
--innodb-page-cleaners, innodb_page_cleaners
Innodb_page_compression_saved
Innodb_page_compression_trim_sect512
Innodb_page_compression_trim_sect1024
Innodb_page_compression_trim_sect2048
Innodb_page_compression_trim_sect4096
Innodb_page_compression_trim_sect8192
Innodb_page_compression_trim_sect16384
Innodb_page_compression_trim_sect32768
--innodb-page-size, innodb_page_size
Innodb_page_size
Innodb_pages_created

Innodb_pages_read
Innodb_pages0_read
Innodb_pages_written
--innodb-pass-corrupt-table, innodb-pass-corrupt-table
--innodb-prefix-index-cluster-optimization, innodb_prefix_index_cluster_optimization
--innodb-print-all-deadlocks, innodb_print_all_deadlocks
--innodb-purge-batch-size, innodb_purge_batch_size
--innodb-purge-rseg-truncate-frequency, innodb_purge_rseg_truncate_frequency
--innodb-purge-threads, innodb_purge_threads
Innodb_purge_trx_id
Innodb_purge_undo_no
--innodb-random-read-ahead, innodb_random_read_ahead
--innodb-read-ahead, innodb_read_ahead
--innodb-read-ahead-threshold, innodb_read_ahead_threshold
--innodb-read-io-threads, innodb_read_io_threads
--innodb-read-only, innodb_read_only
Innodb_read_views_memory
--innodb-recovery-stats, innodb_recovery_stats
--innodb-recovery-update-relay-log, innodb-recovery-update-relay-log
--innodb-replication-delay, innodb_replication_delay
--innodb-rollback-on-timeout, innodb_rollback_on_timeout
--innodb-rollback-segments, innodb_rollback_segments
Innodb_row_lock_current_waits
Innodb_row_lock_numbers
Innodb_row_lock_time
Innodb_row_lock_time_avg
Innodb_row_lock_time_max
Innodb_row_lock_waits
Innodb_rows_deleted
Innodb_rows_inserted
Innodb_rows_read
Innodb_rows_updated
--innodb-rseg
Innodb_s_lock_os_waits
Innodb_s_lock_spin_rounds
Innodb_s_lock_spin_waits
--innodb_safe_truncate, innodb_safe_truncate
--innodb-sched-priority-cleaner, innodb_sched_priority_cleaner
Innodb_scrub_background_page_reorganizations
Innodb_scrub_background_page_split_failures_missing_index
Innodb_scrub_background_page_split_failures_out_of_filespace
Innodb_scrub_background_page_split_failures_underflow
Innodb_scrub_background_page_split_failures_unknown
Innodb_scrub_background_page_splits
--innodb-scrub-log, innodb_scrub_log
Innodb_scrub_log

--innodb-scrub-log-interval, innodb_scrub_log_interval
--innodb-scrub-log-speed, innodb_scrub_log_speed
Innodb_secondary_index_triggered_cluster_reads
Innodb_secondary_index_triggered_cluster_reads_avoided
--innodb-show-locks-held, innodb-show-locks-held
--innodb-show-verbose-locks, innodb_show_verbose_locks
innodb_simulate_comp_failures
--innodb-sort-buffer-size, innodb_sort_buffer_size
--innodb-spin-wait-delay, innodb_spin_wait_delay
--innodb-stats-auto-recalc, innodb_stats_auto_recalc
--innodb-stats-auto-update, innodb_stats_auto_update
--innodb-stats-include-delete-marked, innodb_stats_include_delete_marked
--innodb-stats-method, innodb_stats_method
--innodb-stats-modified-counter, innodb_stats_modified_counter
--innodb_stats_on_metadata, innodb-stats-on-metadata
--innodb-stats-persistent, innodb_stats_persistent
--innodb-stats-persistent-sample-pages, innodb_stats_persistent_sample_pages
--innodb-stats-sample-pages, innodb_stats_sample_pages
--innodb-stats-traditional, innodb_stats_traditional
--innodb-stats-transient-sample-pages, innodb_stats_transient_sample_pages
--innodb-stats-update-need-lock, innodb_stats_update_need_lock
--innodb-status-file
--innodb-status-output, innodb_status_output
--innodb-status-output-locks, innodb_status_output_locks
--innodb-strict-mode, innodb_strict_mode
--innodb-support-xa, innodb_support_xa
--innodb-sync-array-size, innodb_sync_array_size
--innodb-sync-spin-loops, innodb_sync_spin_loops
--innodb-sys-indexes
--innodb-sys-stats
--innodb-sys-tables
Innodb_system_rows_deleted
Innodb_system_rows_inserted
Innodb_system_rows_read
Innodb_system_rows_updated
--innodb-table-locks, innodb_table_locks
--innodb-table-stats
--innodb_temp_data_file_path, innodb_temp_data_file_path
--innodb-thread-concurrency, innodb_thread_concurrency
--innodb-thread-concurrency-timer-based, innodb_thread_concurrency_timer_based
--innodb-thread-sleep-delay, innodb_thread_sleep_delay
--innodb-tmpdir, innodb_tmpdir
--innodb-track-changed-pages, innodb_track_changed_pages
--innodb-track-redo-log-now, innodb_track_redo_log_now
--innodb-trx

--innodb-truncate-temporary-tablespace-now, innodb_truncate_temporary_tablespace_now
Innodb_truncated_status_writes
--innodb-undo-directory, innodb_undo_directory
--innodb-undo-log-truncate, innodb_undo_log_truncate
--innodb-undo-logs, innodb_undo_logs
--innodb-undo-tablespaces, innodb_undo_tablespaces
Innodb_undo_truncations
--innodb-use-atomic-writes, innodb_use_atomic_writes
--innodb-use-fallocate, innodb_use_fallocate
--innodb-use-global-flush-log-at-trx-commit, innodb_use_global_flush_log_at_trx_commit
--innodb-use-mtflush, innodb_use_mtflush
--innodb-use-native_aio, innodb_use_native_aio
--innodb-use-purge-thread, innodb_use_purge_thread
--innodb-use-stacktrace, innodb_use_stacktrace
--innodb-use-sys-malloc, innodb_use_sys_malloc
--innodb-use-sys-stats-table, innodb_use_sys_stats_table
--innodb-use-trim, innodb_use_trim
innodb_version
--innodb-write-io-threads, innodb_write_io_threads
Innodb_x_lock_os_waits
Innodb_x_lock_spin_rounds
Innodb_x_lock_spin_waits
--insert_id
--install
--install-manual
--interactive-timeout, interactive_timeout
--join-buffer-size, join_buffer_size
--join-buffer-space-limit, join_buffer_space_limit
--join-cache-level, join_cache_level
--keep-files-on-create, keep_files_on_create
Key_blocks_not_flushed
Key_blocks_unused
Key_blocks_used
Key_blocks_warm
--key-buffer-size, key_buffer_size
--key-cache-age-threshold, key_cache_age_threshold
--key-cache-block-size, key_cache_block_size
--key-cache-division-limit, key_cache_division_limit
--key-cache-file-hash-size, key_cache_file_hash_size
--key-cache-segments, key_cache_segments
Key_read_requests
Key_reads
Key_write_requests
Key_writes
-L, --language, language
large_files_support

large_page_size
--large-pages, large_pages
last_gtid
last_insert_id
Last_query_cost
--lc-messages, lc_messages
--lc-messages-dir, lc_messages_dir
--lc-time-names, lc_time_names
license
--local-infile, local_infile
--lock-wait-timeout, lock_wait_timeout
locked_in_memory
-l, --log, log
--log-basename
--log-bin, log_bin
log_bin_basename
--log-bin-compress, log_bin_compress
--log-bin-compress-min-len, log_bin_compress_min_len
--log-bin-index, log_bin_index
--log-bin-trust-function-creators, log_bin_trust_function_creators
--log-bin-trust-routine-creators
--log-ddl-recovery
--log-disabled-statements, log_disabled_statements
--log-error, log_error
-0, --log-long-format
--log-output, log_output
--log-queries-not-using-indexes, log_queries_not_using_indexes
--log-short-format
--log-slave-updates, log_slave_updates
--log-slow-admin-statements, log_slow_admin_statements
--log-slow-disabled-statements, log_slow_disabled_statements
--log-slow-file
--log-slow-filter, log_slow_filter
--log-slow-min-examined-row-limit, log_slow_min_examined_row_limit
--log-slow-queries, log_slow_queries
--log-slow-query, log_slow_query
--log-slow-query-file, log_slow_query_file
--log-slow-query-time, log_slow_query_time
--log-slow-rate-limit, log_slow_rate_limit
--log-slow-slave-statements, log_slow_slave_statements
--log-slow-time
--log-slow-verbosity, log_slow_verbosity
--log-tc
--log-tc-size, log_tc_size
-W, --log-warnings, log_warnings
--long-query-time, long_query_time

--log-isam
--low-priority-updates, low_priority_updates
lower_case_file_system
--lower-case-table-names, lower_case_table_names
--master-connect-retry
Master_gtid_wait_count
Master_gtid_wait_time
Master_gtid_wait_timeouts
--master-host
--master-info-file
--master-password
--master-port
--master-retry-count
--master-ssl
--master-ssl-ca
--master-ssl-capath
--master-ssl-cert
--master-ssl-cipher
--master-ssl-key
--master-user
--master-verify-checksum, master_verify_checksum
--max-allowed-packet, max_allowed_packet
--max-binlog-dump-events
--max-binlog-cache-size, max_binlog_cache_size
--max-binlog-size, max_binlog_size
--max-binlog-stmt-cache-size, max_binlog_stmt_cache_size
--max-connect-errors, max_connect_errors
--max-connections, max_connections
--max-delayed-threads, max_delayed_threads
--max-digest-length, max_digest_length
--max-error-count, max_error_count
--max-heap-table-size, max_heap_table_size
max_insert_delayed_threads
--max-join-size, max_join_size
--max-length-for-sort-data, max_length_for_sort_data
--max-long-data-size, max_long_data_size
--max-prepared-stmt-count, max_prepared_stmt_count
--max-password-errors, max_password_errors
--max-recursive-iterations, max_recursive_iterations
--max-relay-log-size, max_relay_log_size
--max-rowid-filter-size, max_rowid_filter_size
--max-seeks-for-key, max_seeks_for_key
--max-session-mem-used, max_session_mem_used
--max-sort-length, max_sort_length
--max-sp-recursion-depth, max_sp_recursion_depth
--max-statement-time, max_statement_time

Max_statement_time_exceeded
--max-tmp-tables, max_tmp_tables
Max_used_connections_time
Max_used_connections
--max-user-connections, max_user_connections
--max-write-lock-count, max_write_lock_count
--memlock
Memory_used
Memory_used_initial
--metadata-locks-cache-size, metadata_locks_cache_size
--metadata-locks-hash-instances, metadata_locks_hash_instances
--min-examined-row-limit, min-examined-row-limit
mroonga_action_on_fulltext_query_error
mroonga_boolean_mode_syntax_flags
Mroonga_count_skip
mroonga_database_path_prefix
mroonga_default_parser
mroonga_default_tokenizer
mroonga_default_wrapper_engine
mroonga_dry_write
mroonga_enable_operations_recording
mroonga_enable_optimization
Mroonga_fast_order_limit
mroonga_libgroonga_embedded
mroonga_libgroonga_support_zlib
mroonga_libgroonga_support_zstd
mroonga_libgroonga_version
mroonga_log_file
mroonga_log_level
mroonga_match_escalation_threshold
mroonga_max_n_records_for_estimate
mroonga_query_log_file
mroonga_vector_column_delimiter
mroonga_version
--mrr-buffer-size, mrr_buffer_size
--multi-range-count, multi_range_count
--myisam-block-size, myisam_block_size
--myisam-data-pointer-size, myisam_data_pointer_size
--myisam-max-extra-sort-file-size, myisam_max_extra_sort_file_size
--myisam-max-sort-file-size, myisam_max_sort_file_size
--myisam-mmap-size, myisam_mmap_size
--myisam-recover-options, myisam_recover_options
--myisam-repair-threads, myisam_repair_threads
--myisam-sort-buffer-size, myisam_sort_buffer_size
--myisam-stats-method, myisam_stats_method
--myisam-use-mmap, myisam_use_mmap

--mysql56-temporal-format, mysql56_temporal_format
--named-pipe, named_pipe
--ndb-use-copying-alter-table
--net-buffer-length, net_buffer_length
--net-read-timeout, net_read_timeout
--net-retry-count, net_retry_count
--net-write-timeout, net_write_timeout
Not_flushed_delayed_rows
--new
--old, old
--old-alter-table, old_alter_table
--old-mode, old_mode
--old-passwords, old_passwords
--old-style-user-limits
--one-thread
Open_files
--open-files-limit, open_files_limit
Open_streams
Open_table_definitions
Open_tables
Opened_files
Opened_plugin_libraries
Opened_table_definitions
Opened_tables
Opened_views
--optimizer-extra-pruning-depth , optimizer_extra_pruning_depth
--optimizer-max-sel-arg-weight , optimizer_max_sel_arg_weight
--optimizer-prune-level , optimizer_prune_level
--optimizer-search-depth, optimizer_search_depth
--optimizer-selectivity-sampling-limit, optimizer_selectivity_sampling_limit
--optimizer-switch, optimizer_switch
--optimizer-trace, optimizer_trace
--optimizer-trace-max-mem-size, optimizer_trace_max_mem_size
--optimizer-use-condition-selectivity, optimizer_use_condition_selectivity
oqgraph_allow_create_integer_latch
Oqgraph_boost_version
Oqgraph_compat_mode
Oqgraph_verbose_debug
--pam-debug, pam_debug
-P, --port, port
--pam-use-cleartext-plugin, pam_use_cleartext_plugin
--pam-windbind-workaround, pam_windbind_workaround
--password-reuse-check-interval, password_reuse_check_interval
--pbxt
--pbxt-auto-increment-mode
--pbxt-checkpoint-frequency

--pbxt-data-file-grow-size
--pbxt-data-log-threshold
--pbxt-flush-log-at-trx-commit
--pbxt-garbage-threshold
--pbxt-index-cache-size
--pbxt-log-buffer-size
--pbxt-log-cache-size
--pbxt-log-file-count
--pbxt-log-file-threshold
--pbxt-max-threads
--pbxt-offline-log-function
--pbxt-record-cache-size
--pbxt-row-file-grow-size
--pbxt-statistics
--pbxt-sweeper-priority
--pbxt-support-xa
--pbxt-transaction-buffer-size
--performance-schema, performance_schema
Performance_schema_accounts_lost
--performance-schema-accounts-size, performance_schema_accounts_size
Performance_schema_cond_classes_lost
Performance_schema_cond_instances_lost
--performance-schema-consumer-events-stages-current
--performance-schema-consumer-events-stages-history
--performance-schema-consumer-events-stages-history-long
--performance-schema-consumer-events-statements-current
--performance-schema-consumer-events-statements-history
--performance-schema-consumer-events-statements-history-long
--performance-schema-consumer-events-waits-current
--performance-schema-consumer-events-waits-history
--performance-schema-consumer-events-waits-history-long
--performance-schema-consumer-global-instrumentation
--performance-schema-consumer-statements-digest
--performance-schema-consumer-thread-instrumentation
Performance_schema_digest_lost
--performance-schema-digests-size, performance_schema_digests_size
--performance-schema-events-stages-history-long-size, performance_schema_events_stages_history_long_size
--performance-schema-events-stages-history-size, performance_schema_events_stages_history_size
--performance-schema-events-statements-history-long-size, performance_schema_events_statements_history_long_size
--performance-schema-events-statements-history-size, performance_schema_events_statements_history_size
--performance-schema-events-transactions-history-long-size, performance_schema_events_transactions_history_long_size
--performance-schema-events-transactions-history-size, performance_schema_events_transactions_history_size
--performance-schema-events-waits-history-long-size, performance_schema_events_waits_history_long_size
--performance-schema-events-waits-history-size, performance_schema_events_waits_history_size
Performance_schema_file_classes_lost
Performance_schema_file_handles_lost

Performance_schema_file_instances_lost
Performance_schema_hosts_lost
--performance-schema-hosts-size, performance_schema_hosts_size
Performance_schema_locker_lost
Performance_schema_index_stat_lost
--performance-schema-max-cond-classes, performance_schema_max_cond_classes
--performance-schema-max-cond-instances, performance_schema_max_cond_instances
--performance-schema-max-digest-length, performance_schema_max_digest_length
--performance-schema-max-file-classes, performance_schema_max_file_classes
--performance-schema-max-file-handles, performance_schema_max_file_handles
--performance-schema-max-file-instances, performance_schema_max_file_instances
--performance-schema-max-index-stat, performance_schema_max_index_stat
--performance-schema-max-memory-classes, performance_schema_max_memory_classes
--performance-schema-max-metadata-locks, performance_schema_max_metadata_locks
--performance-schema-max-mutex-classes, performance_schema_max_mutex_classes
--performance-schema-max-mutex-instances, performance_schema_max_mutex_instances
--performance-schema-max-prepared-statement-instances, performance_schema_max_prepared_statement_instances
--performance-schema-max-program-instances, performance_schema_max_program_instances
--performance-schema-max-sql-text-length, performance_schema_max_sql_text_length
--performance-schema-max-rwlock-classes, performance_schema_max_rwlock_classes
--performance-schema-max-rwlock-instances, performance_schema_max_rwlock_instances
--performance-schema-max-socket-classes, performance_schema_max_socket_classes
--performance-schema-max-socket-instances, performance_schema_max_socket_instances
--performance-schema-max-stage-classes, performance_schema_max_stage_classes
--performance-schema-max-statement-classes, performance_schema_max_statement_classes
--performance-schema-max-statement-stack, performance_schema_max_statement_stack
--performance-schema-max-table-handles, performance_schema_max_table_handles
--performance-schema-max-table-instances, performance_schema_max_table_instances
--performance-schema-max-table-lock-stat, performance_schema_max_table_lock_stat
--performance-schema-max-thread-classes, performance_schema_max_thread_classes
--performance-schema-max-thread-instances, performance_schema_max_thread_instances
Performance_schema_memory_classes_lost
Performance_schema_metadata_lock_lost
Performance_schema_mutex_classes_lost
Performance_schema_mutex_instances_lost
Performance_schema_nested_statement_lost
Performance_schema_prepared_statements_lost
Performance_schema_program_lost
Performance_schema_rwlock_classes_lost
Performance_schema_rwlock_instances_lost
Performance_schema_session_connect_attrs_lost
Performance_schema_socket_classes_lost
Performance_schema_socket_instances_lost
Performance_schema_stage_classes_lost
Performance_schema_stage_classes_lost
Performance_schema_statement_classes_lost

--performance-schema-session-connect-attrs-size, performance_schema_session_connect_attrs_size
--performance-schema-setup-actors-size, performance_schema_setup_actors_size
--performance-schema-setup-objects-size, performance_schema_setup_objects_size
Performance_schema_table_handles_lost
Performance_schema_table_instances_lost
Performance_schema_table_lock_stat_lost
Performance_schema_thread_classes_lost
Performance_schema_thread_instances_lost
Performance_schema_schema_users_lost
--performance_schema_users_size, performance_schema_users_size
--pid-file, pid_file
--plugin-load
--plugin-load-add
--plugin-dir, plugin_dir
--plugin-maturity, plugin_maturity
-P, --port, port
--port-open-timeout
--preload-buffer-size, preload_buffer_size
Prepared_stmt_count
profiling
--profiling-history-size, profiling_history_size
--progress-report-time, progress_report_time
protocol_version
--proxy-protocol-networks, proxy_protocol_networks
proxy_user
pseudo_slave_mode
pseudo_thread_id
Qcache_free_blocks
Qcache_free_memory
Qcache_hits
Qcache_inserts
Qcache_lowmem_prunes
Qcache_not_cached
Qcache_queries_in_cache
Qcache_total_blocks
Queries
--query-alloc-block-size, query_alloc_block_size
--query-cache-info
--query-cache-limit, query_cache_limit
--query-cache-min-res-unit, query_cache_min_res_unit
--query-cache-size, query_cache_size
--query-cache-strip-comments, query_cache_strip_comments
--query-cache-type, query_cache_type
--query-cache-wlock-invalidate, query_cache_wlock_invalidate
--query-prealloc-size, query_prealloc_size
--query-response-time

--query-response-time-audit
query_response_time_flush
--query-response-time-range-base , query_response_time_range_base
query_response_time_exec_time_debug
--query-response-time-stats, query_response_time_stats
Questions
-r, --chroot
rand_seed1
rand_seed2
--range-alloc-block-size, range_alloc_block_size
--read-buffer-size, read_buffer_size
--read-binlog-speed-limit, read_binlog_speed_limit
--read-only, read_only
--read-rnd-buffer-size, read_rnd_buffer_size
--record-buffer
--redirect-url, redirect_url
--relay-log, relay_log
relay_log_basename
--relay-log-index, relay_log_index
--relay-log-info-file, relay_log_info_file
--relay-log-purge, relay_log_purge
--relay-log-recovery, relay_log_recovery
--relay-log-space-limit, relay_log_space_limit
--remove
--replicate-annotate-row-events, replicate_annotate_row_events
--replicate-do-db, replicate_do_db
--replicate-do-table, replicate_do_table
--replicate-events-marked-for-skip, replicate_events_marked_for_skip
--replicate-ignore-db, replicate_ignore_db
--replicate-ignore-table, replicate_ignore_table
--replicate-rewrite-db, replicate_rewrite_db
--replicate-same-server-id
--replicate-wild-do-table, replicate_wild_do_table
--replicate-wild-ignore-table, replicate_wild_ignore_table
--report-host, report_host
--report-password, report_password
--report-port, report_port
--report-user, report_user
--require-secure-transport, require_secure_transport
--rocksdb-access-hint-on-compaction-start, rocksdb_access_hint_on_compaction_start
--rocksdb-advise-random-on-open, rocksdb_advise_random_on_open
--rocksdb-allow-concurrent-memtable-write, rocksdb_allow_concurrent_memtable_write
--rocksdb-allow-mmap-reads, rocksdb_allow_mmap_reads
--rocksdb-allow-mmap-writes, rocksdb_allow_mmap_writes
--rocksdb-allow-to-start-after-corruption, rocksdb_allow_to_start_after_corruption
--rocksdb-background-sync, rocksdb_background_sync

--rocksdb_base-background-compactions, rocksdb_base_background_compactions
--rocksdb-blind-delete-primary-key, rocksdb_blind_delete_primary_key
--rocksdb-block-cache-size, rocksdb_block_cache_size
--rocksdb_block_restart_interval, rocksdb_block_restart_interval
Rocksdb_block_cache_add
Rocksdb_block_cache_add_failures
Rocksdb_block_bytes_read
Rocksdb_block_bytes_write
Rocksdb_block_cache_data_add
Rocksdb_block_cache_data_bytes_insert
Rocksdb_block_cache_data_hit
Rocksdb_block_cache_data_miss
Rocksdb_block_cache_filter_add
Rocksdb_block_cache_filter_bytes_evict
Rocksdb_block_cache_filter_bytes_insert
Rocksdb_block_cache_filter_hit
Rocksdb_block_cache_filter_miss
Rocksdb_block_cache_hit
Rocksdb_block_cache_index_add
Rocksdb_block_cache_index_bytes_evict
Rocksdb_block_cache_index_bytes_insert
Rocksdb_block_cache_index_hit
Rocksdb_block_cache_index_miss
Rocksdb_block_cache_miss
Rocksdb_block_cachecompressed_hit
Rocksdb_block_cachecompressed_miss
--rocksdb-block-size, rocksdb_block_size
--rocksdb-block-size-deviation, rocksdb_block_size_deviation
Rocksdb_bloom_filter_full_positive
Rocksdb_bloom_filter_full_true_positive
Rocksdb_bloom_filter_prefix_checked
Rocksdb_bloom_filter_prefix_useful
Rocksdb_bloom_filter_useful
--rocksdb-bulk-load, rocksdb_bulk_load
--rocksdb-bulk-load_allow_sk, rocksdb_bulk_load_allow_sk
--rocksdb-bulk-load_allow_unsorted, rocksdb_bulk_load_allow_unsorted
--rocksdb-bulk-load-size, rocksdb_bulk_load_size
--rocksdb-bytes-per-sync, rocksdb_bytes_per_sync
Rocksdb_bytes_read
Rocksdb_bytes_written
--rocksdb-cache-dump, rocksdb_cache_dump
--rocksdb-cache-high-pri-pool-ratio, rocksdb_cache_high_pri_pool_ratio
--rocksdb-cache-index-and-filter-blocks, rocksdb_cache_index_and_filter_blocks
--rocksdb-cache-index-and-filter-with-high-priority, rocksdb_cache_index_and_filter_with_high_priority
--rocksdb-checksums-pct, rocksdb_checksums_pct
--rocksdb-collect-sst-properties, rocksdb_collect_sst_properties

--rocksdb-commit-in-the-middle, rocksdb_commit_in_the_middle
--rocksdb-commit-time-batch-for-recovery, rocksdb_commit_time_batch_for_recovery
--rocksdb-compact-cf, rocksdb_compact_cf
Rocksdb_compact_read_bytes
Rocksdb_compact_write_bytes
Rocksdb_compaction_key_drop_new
Rocksdb_compaction_key_drop_obsolete
Rocksdb_compaction_key_drop_user
--rocksdb-compaction-readahead-size, rocksdb_compaction_readahead_size
--rocksdb-compaction-sequential-deletes, rocksdb_compaction_sequential_deletes
--rocksdb-compaction-sequential-deletes-count-sd, rocksdb_compaction_sequential_deletes_count_sd
--rocksdb-concurrent-prepare, rocksdb_concurrent_prepare
--rocksdb-compaction-sequential-deletes-file-size, rocksdb_compaction_sequential_deletes_file_size
--rocksdb-compaction-sequential-deletes-window, rocksdb_compaction_sequential_deletes_window
Rocksdb_covered_secondary_key_lookups
--rocksdb-create-checkpoint, rocksdb_create_checkpoint
--rocksdb-create-if-missing, rocksdb_create_if_missing
--rocksdb-create-missing-column-families, rocksdb_create_missing_column_families
--rocksdb-datadir, rocksdb_datadir
--rocksdb-db-write-buffer-size, rocksdb_db_write_buffer_size
--rocksdb-deadlock-detect, rocksdb_deadlock_detect
--rocksdb-deadlock-detect-depth, rocksdb_deadlock_detect_depth
--rocksdb-debug-manual-compaction-delay, rocksdb_debug_manual_compaction_delay
--rocksdb-debug-optimizer-no-zero-cardinality, rocksdb_debug_optimizer_no_zero_cardinality
--rocksdb-debug-ttl-ignore-pk, rocksdb_debug_ttl_ignore_pk
--rocksdb-debug-ttl-read-filter-ts, rocksdb_debug_ttl_read_filter_ts
--rocksdb-debug-ttl-rec-ts, rocksdb_debug_ttl_rec_ts
--rocksdb-debug-ttl-snapshot-ts, rocksdb_debug_ttl_snapshot_ts
--rocksdb-default-cf-options, rocksdb_default_cf_options
--rocksdb-delayed-write-rate, rocksdb_delayed_write_rate
--rocksdb-delete-cf, rocksdb_delete_cf
--rocksdb-delete-obsolete-files-period-micros, rocksdb_delete_obsolete_files_period_micros
--rocksdb-enable-2pc, rocksdb_enable_2pc
--rocksdb-enable-bulk-load-api, rocksdb_enable_bulk_load_api
--rocksdb-enable-insert-with-update-caching, rocksdb_enable_insert_with_update_caching
--rocksdb-enable-thread-tracking, rocksdb_enable_thread_tracking
--rocksdb-enable-ttl, rocksdb_enable_ttl
--rocksdb-enable-ttl-read-filtering, rocksdb_enable_ttl_read_filtering
--rocksdb-enable-write-thread-adaptive-yield, rocksdb_enable_write_thread_adaptive_yield
--rocksdb-error-if-exists, rocksdb_error_if_exists
--rocksdb-error-on-suboptimal-collation, rocksdb_on_suboptimal_collation
--rocksdb-flush-log-at-trx-commit, rocksdb_flush_log_at_trx_commit
--rocksdb-flush-memtable-on-analyze, rocksdb_flush_memtable_on_analyze
Rocksdb_flush_write_bytes
--rocksdb-force-compute-memtable-stats, rocksdb_force_compute_memtable_stats
--rocksdb-force-compute-memtable-stats-cachetime, rocksdb_force_compute_memtable_stats_cachetime

--rocksdb-force-flush-memtable-and-lzero-now, rocksdb_force_flush_memtable_and_lzero_now
--rocksdb-force-flush-memtable-now, rocksdb_force_flush_memtable_now
--rocksdb-force-index-records-in-range, rocksdb_force_index_records_in_range
Rocksdb_get_hit_l0
Rocksdb_get_hit_l1
Rocksdb_get_hit_l2_and_up
Rocksdb_getupdatessince_calls
--rocksdb-git-hash, rocksdb_git_hash
--rocksdb-hash-index-allow-collision, rocksdb_hash_index_allow_collision
--rocksdb-ignore-unknown-options, rocksdb_ignore_unknown_options
--rocksdb-index-type, rocksdb_index_type
--rocksdb-info-log-level, rocksdb_info_log_level
--rocksdb-io-write-timeout, rocksdb_io_write_timeout
--rocksdb-is-fd-close-on-exec, rocksdb_is_fd_close_on_exec
Rocksdb_iter_bytes_read
--rocksdb-keep-log-file-num, rocksdb_keep_log_file_num
Rocksdb_l0_num_files_stall_micros
Rocksdb_l0_slowdown_micros
--rocksdb-large-prefix, rocksdb_large_prefix
--rocksdb-lock-scanned-rows, rocksdb_lock_scanned_rows
--rocksdb-lock-wait-timeout, rocksdb_lock_wait_timeout
--rocksdb-log-dir, rocksdb_log_dir
--rocksdb-log-file-time-to-roll, rocksdb_log_file_time_to_roll
--rocksdb-manifest-preallocation-size, rocksdb_manifest_preallocation_size
--rocksdb-manual-compaction-threads, rocksdb_manual_compaction_threads
Rocksdb_manual_compactions_processed
Rocksdb_manual_compactions_running
--rocksdb-manual-wal-flush, rocksdb_manual_wal_flush
--rocksdb-master-skip-tx-api, rocksdb_master_skip_tx_api
--rocksdb-max-background-compactions, rocksdb_max_background_compactions
--rocksdb-max-latest-deadlocks, rocksdb_max_latest_deadlocks
--rocksdb-max-background-flushes, rocksdb_max_background_flushes
--rocksdb-max-background-jobs, rocksdb_max_background_jobs
--rocksdb-max-log-file-size, rocksdb_max_log_file_size
--rocksdb-max-manifest-file-size, rocksdb_max_manifest_file_size
--rocksdb-max-manual-compactions, rocksdb_max_manual_compactions
--rocksdb-max-open-files, rocksdb_max_open_files
--rocksdb-max-row-locks, rocksdb_max_row_locks
--rocksdb-max-subcompactions, rocksdb_max_subcompactions
--rocksdb-max-total-wal-size, rocksdb_max_total_wal_size
Rocksdb_memtable_compaction_micros
Rocksdb_memtable_hit
Rocksdb_memtable_miss
Rocksdb_memtable_total
Rocksdb_memtable_unflushed
--rocksdb-merge-buf-size, rocksdb_merge_buf_size

--rocksdb-merge-combine-read-size, rocksdb_merge_combine_read_size
--rocksdb-merge-tmp-file-removal-delay-ms, rocksdb_merge_tmp_file_removal_delay_ms
--rocksdb-new-table-reader-for-compaction-inputs, rocksdb_new_table_reader_for_compaction_inputs
--rocksdb-no-block-cache, rocksdb_no_block_cache
Rocksdb_no_file_closes
Rocksdb_no_file_errors
Rocksdb_no_file_opens
Rocksdb_num_iterators
Rocksdb_number_block_not_compressed
Rocksdb_db_next
Rocksdb_db_next_found
Rocksdb_db_prev
Rocksdb_db_prev_found
Rocksdb_db_seek
Rocksdb_db_seek_found
Rocksdb_number_deletes_filtered
Rocksdb_number_keys_read
Rocksdb_number_keys_updated
Rocksdb_number_keys_written
Rocksdb_number_merge_failures
Rocksdb_number_multiget_bytes_read
Rocksdb_number_multiget_get
Rocksdb_number_multiget_keys_read
Rocksdb_number_reseeks_iteration
Rocksdb_number_sst_entry_delete
Rocksdb_number_sst_entry_merge
Rocksdb_number_sst_entry_other
Rocksdb_number_sst_entry_put
Rocksdb_number_sst_entry_singledelete
Rocksdb_number_superversion_acquires
Rocksdb_number_superversion_cleanups
Rocksdb_number_superversion_releases
--rocksdb-override-cf-options, rocksdb_override_cf_options
--rocksdb-paranoid-checks, rocksdb_paranoid_checks
--rocksdb-pause-background-work, rocksdb_pause_background_work
--rocksdb-perf-context-level, rocksdb_perf_context_level
--rocksdb-persistent-cache-path, rocksdb_persistent_cache_path
--rocksdb-persistent-cache-size-mb, rocksdb_persistent_cache_size_mb
--rocksdb-pin-l0-filter-and-index-blocks-in-cache, rocksdb_pin_l0_filter_and_index_blocks_in_cache
--rocksdb-print-snapshot-conflict-queries, rocksdb_print_snapshot_conflict_queries
Rocksdb_queries_point
Rocksdb_queries_range
--rocksdb-rate-limiter-bytes-per-sec, rocksdb_rate_limiter_bytes_per_sec
--rocksdb-read-free-rpl-tables, rocksdb_read_free_rpl_tables
--rocksdb-records-in-range, rocksdb_records_in_range
--rocksdb-remove-mariabackup-checkpoint, rocksdb_remove_mariabackup_checkpoint

--rocksdb-reset-stats, rocksdb_reset_stats
--rocksdb-rollback-on-timeout, rocksdb_rollback_on_timeout
Rocksdb_row_lock_deadlocks
Rocksdb_row_lock_wait_timeouts
Rocksdb_rows_deleted
Rocksdb_rows_deleted_blind
Rocksdb_rows_expired
Rocksdb_rows_filtered
Rocksdb_rows_inserted
Rocksdb_rows_read
Rocksdb_rows_updated
--rocksdb-seconds-between-stat-computes, rocksdb_seconds_between_stat_computes
--rocksdb-signal-drop-index-thread, rocksdb_signal_drop_index_thread
--rocksdb-sim-cache-size, rocksdb_sim_cache_size
--rocksdb-skip-bloom-filter-on-read, rocksdb_skip_bloom_filter_on_read
--rocksdb-skip-fill-cache, rocksdb_skip_fill_cache
--rocksdb-skip-unique-check-tables, rocksdb_skip_unique_check_tables
Rocksdb_snapshot_conflict_errors
--rocksdb-sst-mgr-rate-bytes-per-sec, rocksdb_sst_mgr_rate_bytes_per_sec
Rocksdb_stall_l0_file_count_limit_slowdowns
Rocksdb_stall_l0_file_count_limit_stops
Rocksdb_stall_locked_l0_file_count_limit_slowdowns
Rocksdb_stall_locked_l0_file_count_limit_stops
Rocksdb_stall_memtable_limit_slowdowns
Rocksdb_stall_memtable_limit_stops
Rocksdb_stall_micros
Rocksdb_stall_pending_compaction_limit_slowdowns
Rocksdb_stall_pending_compaction_limit_stops
Rocksdb_stall_total_slowdowns
Rocksdb_stall_total_stops
--rocksdb-stats-dump-period-sec, rocksdb_stats_dump_period_sec
--rocksdb-stats-level, rocksdb_stats_level
--rocksdb-stats-recalc-rate, rocksdb_stats_recalc_rate
--rocksdb-store-row-debug-checksums, rocksdb_store_row_debug_checksums
--rocksdb-strict-collation-check, rocksdb_strict_collation_check
--rocksdb-strict-collation-exceptions, rocksdb_strict_collation_exceptions
--rocksdb-supported-compression-types, rocksdb_supported_compression_types
Rocksdb_system_rows_deleted
Rocksdb_system_rows_inserted
Rocksdb_system_rows_read
Rocksdb_system_rows_updated
--rocksdb-table-cache-numshardbits, rocksdb_table_cache_numshardbits
--rocksdb-table-stats-sampling-pct, rocksdb_table_stats_sampling_pct
--rocksdb-tmpdir, rocksdb_tmpdir
--rocksdb-trace-sst-api, rocksdb_trace_sst_api
--rocksdb-two-write-queues, rocksdb_two_write_queues

--rocksdb-unsafe-for-binlog, rocksdb_unsafe_for_binlog
--rocksdb-update-cf-options, rocksdb_update_cf_options
--rocksdb-use-adaptive-mutex, rocksdb_use_adaptive_mutex
--rocksdb-use-clock-cache, rocksdb_use_clock_cache
--rocksdb-use-direct-io-for-flush-and-compaction, rocksdb_use_direct_io_for_flush_and_compaction
--rocksdb-use-direct-reads, rocksdb_use_direct_reads
--rocksdb-use-fsync, rocksdb_use_fsync
--rocksdb-validate-tables, rocksdb_validate_tables
--rocksdb-verify-row-debug-checksums, rocksdb_verify_row_debug_checksums
Rocksdb_wal_bytes
--rocksdb-wal-bytes-per-sync, rocksdb_wal_bytes_per_sync
--rocksdb-wal-dir, rocksdb_wal_dir
Rocksdb_wal_group_syncs
--rocksdb-wal-recovery-mode, rocksdb_wal_recovery_mode
--rocksdb-wal-size-limit-mb, rocksdb_wal_size_limit_mb
Rocksdb_wal_synced
--rocksdb-wal-ttl-seconds, rocksdb_wal_ttl_seconds
--rocksdb-whole-key-filtering, rocksdb_whole_key_filtering
--rocksdb-write-batch-max-bytes, rocksdb_write_batch_max_bytes
--rocksdb-write-disable-wal, rocksdb_write_disable_wal
--rocksdb-write-ignore-missing-column-families, rocksdb_write_ignore_missing_column_families
--rocksdb-write-policy, rocksdb_write_policy
Rocksdb_write_other
Rocksdb_write_self
Rocksdb_write_timedout
Rocksdb_write_wal
--rowid-merge-buff-size, rowid_merge_buff_size
Resultset_metadata_skipped
Rows_read
Rows_sent
Rows_tmp_read
--rpl-recovery-rank, rpl_recovery_rank
Rpl_semi_sync_master_clients
rpl-semi-sync-master-enabled rpl_semi_sync_master_enabled
Rpl_semi_sync_master_net_avg_wait_time
Rpl_semi_sync_master_net_wait_time
Rpl_semi_sync_master_net_waits
Rpl_semi_sync_master_no_times
Rpl_semi_sync_master_no_tx
Rpl_semi_sync_master_status
Rpl_semi_sync_master_timefunc_failures
rpl-semi-sync-master-timeout, rpl_semi_sync_master_timeout
rpl-semi-sync-master-trace-level , rpl_semi_sync_master_trace_level
Rpl_semi_sync_master_tx_avg_wait_time
Rpl_semi_sync_master_tx_wait_time
Rpl_semi_sync_master_tx_waits

rpl-semi-sync-master-wait-no-slave, rpl_semi_sync_master_wait_no_slave
rpl-semi-sync-master-wait-point, rpl_semi_sync_master_wait_point
Rpl_semi_sync_master_wait_pos_backtraverse
Rpl_semi_sync_master_wait_sessions
Rpl_semi_sync_master_yes_tx
rpl-semi-sync-slave-delay-master, rpl_semi_sync_slave_delay_master
rpl-semi-sync-slave-enabled, rpl_semi_sync_slave_enabled
rpl-semi-sync-slave-kill-conn-timeout, rpl_semi_sync_slave_kill_conn_timeout
Rpl_semi_sync_slave_status
rpl-semi-sync-slave-trace-level, rpl_semi_sync_slave_trace_level
Rpl_status
Rpl_transactions_multi_engine
-s, --symbolic-links
--s3-access-key, s3_access_key
--s3-block-size, s3_block_size
--s3-bucket, s3_bucket
--s3-debug, s3_debug
--s3-host-name, s3_host_name
--s3-pagecache-age-threshold, s3_pagecache_age_threshold
--s3-pagecache-buffer-size, s3_pagecache_buffer_size
--s3-pagecache-division-limit, s3_pagecache_division_limit
--s3-pagecache-file-hash-size, s3_pagecache_file_hash_size
--s3-port, s3_port
--s3-protocol-version, s3_protocol_version
--s3-region, s3_region
--s3-secret-key, s3_secret_key
--s3-slave-ignore-updates, s3_slave_ignore_updates
--s3-use_http, s3_use_http
--safe-mode
--safe-show-database, safe_show_database
--safe-user-create
--safemalloc-mem-limit
--secure-auth, secure_auth
--secure-file-priv, secure_file_priv
--secure-timestamp, secure_timestamp
Select_full_join
Select_full_range_join
Select_range
Select_range_check
Select_scan
--server-audit
Server_audit_active
Server_audit_current_log
--server-audit-events, server_audit_events
--server-audit-excl-users, server_audit_excl_users
--server-audit-file-path, server_audit_file_path

Server_audit_last_error
--server-audit-file-rotate-now, server_audit_file_rotate_now
--server-audit-file-rotate-size, server_audit_file_rotate_size
--server-audit-file-rotations, server_audit_file_rotations
--server-audit-incl-users, server_audit_incl_users
--server-audit-loc-info, server_audit_loc_info
--server-audit-logging, server_audit_logging
--server-audit-mode, server_audit_mode
--server-audit-output-type, server_audit_output_type
--server-audit-query-limit, server_audit_query_limit
--server-audit-syslog-facility, server_audit_syslog_facility
--server-audit-syslog-ident, server_audit_syslog_ident
--server-audit-syslog-info, server_audit_syslog_info
--server-audit-syslog-priority, server_audit_syslog_priority
Server_audit_writes_failed
--server-id, server_id
--session-track-schema, session_track_schema
--session-track-state-change, session_track_state_change
--session-track-system-variables, session_track_system_variables
--session-track-transaction-info, session_track_transaction_info
-O, --set-variable
shared_memory
shared_memory_base_name
--show_old_temporals, show_old_temporals
--show-slave-auth-info
--silent-startup
--simple-password-check-digits, simple_password_check_digits
--simple_password-check-letters-same-case, simple_password_check_letters_same_case
--simple-password-check-minimal_length, simple_password_check_minimal_length
--simple_password-check-other-characters, simple_password_check_other_characters
--skip-automatic-sp-privileges
--skip-bdb
--skip-external-locking, skip_external_locking
--skip-grant-tables, skip_grant_tables
--skip-host-cache
--skip-innodb
--skip-innodb-checksums
--skip-innodb-doublewrite
--skip-large-pages
--skip-log-error
--skip-name-resolve, skip_name_resolve
--skip-new
--skip-networking, skip_networking
skip_parallel_replication
--skip-partition
skip_replication

--skip-show-database, skip_show_database
--skip-slave-start
--skip-ssl
--skip-stack-trace
--skip-symbolic-links
--skip-symlink
--skip-thread-priority
--slave-compressed-protocol, slave_compressed_protocol
Slave_connections
--slave-ddl-exec-mode, slave_ddl_exec_mode
--slave-domain-parallel-threads, slave_domain_parallel_threads
--slave-exec-mode, slave_exec_mode
Slave_heartbeat_period
--slave-load-tmpdir, slave_load_tmpdir
--slave-max-allowed-packet, slave_max_allowed_packet
--slave-max-statement-time, slave_max_statement_time
--slave-net-timeout, slave_net_timeout
Slave_open_temp_tables
--slave-parallel-max-queued, slave_parallel_max_queued
slave_parallel_mode
--slave-parallel-threads, slave_parallel_threads
--slave-parallel-workers, slave_parallel_workers
Slave_received_heartbeats
Slave_retried_transactions
slave_run_triggers_for_rbr, slave-run-triggers-for-rbr
Slave_running
--slave-skip-errors, slave_skip_errors
Slave_skipped_errors
--slave-sql-verify-checksum, slave_sql_verify_checksum
--slave-transaction-retries, slave_transaction_retries
--slave-transaction-retry-errors, slave_transaction_retry_errors
--slave-transaction-retry-interval, slave_transaction_retry_interval
--slave-type-conversions, slave_type_conversions
Slaves_connected
Slaves_running
Slow_launch_threads
--slow-launch-time, slow_launch_time
Slow_queries
--slow-query-log, slow_query_log
--slow-query-log-file, slow_query_log_file
--slow-start-timeout
--socket, socket
--sort-buffer-size, sort_buffer_size
Sort_merge_passes
Sort_priority_queue_sorts
Sort_range

Sort_rows
Sort_scan
Sphinx_error
Sphinx_time
Sphinx_total
Sphinx_total_found
Sphinx_word_count
Sphinx_words
spider_auto_increment_mode
spider_bgs_first_read
spider_bgs_mode
spider_bgs_second_read
spider_bka_engine
spider_bka_mode
spider_block_size
spider_buffer_size
spider_bulk_size
spider_bulk_update_mode
spider_bulk_update_size
spider_casual_read
spider_conn_recycle_mode
spider_conn_recycle_strict
spider_conn_wait_timeout
spider_connect_error_interval
spider_connect_mutex
spider_connect_retry_count
spider_connect_retry_interval
spider_connect_timeout
spider_crd_bg_mode
spider_crd_interval
spider_crd_mode
spider_crd_sync
spider_crd_type
spider_crd_weight
spider_delete_all_rows_type
Spider_direct_aggregate
Spider_direct_delete
spider_direct_dup_insert
spider_direct_order_limit
Spider_direct_order_limit
Spider_direct_update
spider_dry_access
spider_error_read_mode
spider_error_write_mode
spider_first_read
spider_force_commit

spider_general_log
spider_ignore_comments
spider_index_hint_pushdown
spider_init_sql_alloc_size
spider_internal_limit
spider_internal_offset
spider_internal_optimize
spider_internal_optimize_local
spider_internal_sql_log_off
spider_internal_unlock
spider_internal_xa
spider_internal_xa_id_type
spider_internal_xa_snapshot
spider_load_crd_at_startup
spider_load_sts_at_startup
spider_local_lock_table
spider_lock_exchange
spider_log_result_error_with_sql
spider_log_result_errors
spider_low_mem_read
spider_max_connections
spider_max_order
Spider_mon_table_cache_version
Spider_mon_table_cache_version_req
spider_multi_split_read
spider_net_read_timeout
spider_net_write_timeout
Spider_parallel_search
spider_ping_interval_at_trx_start
spider_quick_mode
spider_quick_page_byte
spider_quick_page_size
spider_read_only_mode
spider_remote_access_charset
spider_remote_autocommit
spider_remote_default_database
spider_remote_sql_log_off
spider_remote_time_zone
spider_remote_trx_isolation
spider_remote_wait_timeout
spider_reset_sql_alloc
spider_same_server_link
spider_second_read
spider_select_column_mode
spider_selupd_lock_mode
spider_semi_split_read

spider_semi_split_read_limit
spider_semi_table_lock
spider_semi_table_lock_connection
spider_semi_trx
spider_semi_trx_isolation
spider_skip_default_condition
spider_skip_parallel_search
spider_slave_trx_isolation
spider_split_read
spider_store_last_crd
spider_store_last_sts
spider_strict_group_by
spider_sts_bg_mode
spider_sts_interval
spider_sts_mode
spider_sts_sync
spider_support_xa
spider_sync_autocommit
spider_sync_sql_mode
spider_sync_time_zone
spider_sync_trx_isolation
spider_table_crd_thread_count
spider_table_init_error_interval
spider_table_sts_thread_count
spider_udf_ct_bulk_insert_interval
spider_udf_ct_bulk_insert_rows
spider_udf_ds_bulk_insert_rows
spider_udf_ds_table_loop_mode
spider_udf_ds_use_real_table
spider_udf_table_lock_mutex_count
spider_udf_table_mon_mutex_count
spider_use_all_conns_snapshot
spider_use_cond_other_than_pk_for_update
spider_use_consistent_snapshot
spider_use_default_database
spider_use_flash_logs
spider_use_handler
spider_use_pushdown_udf
spider_use_table_charset
spider_version
spider_wait_timeout
spider_xa_register_mode
--sporadic-binlog-dump-fail
sql_auto_is_null
sql_big_selects
sql_big_tables

sql_buffer_result
--sql-bin-update-same
--sql-error-log-filename, sql_error_log_filename
--sql-error-log-rate, sql_error_log_rate
--sql-error-log-rotate, sql_error_log_rotate
--sql-error-log-rotations, sql_error_log_rotations
--sql-error-log-size-limit, sql_error_log_size_limit
--sql-error-log-warnings, sql_error_log_warnings
--sql-if-exists, sql_if_exists
sql_log_bin
sql_log_off
sql_log_update
sql_low_priority_updates
sql_max_join_size
--sql-mode, sql_mode
sql_notes
sql_quote_show_create
--sql-safe-updates, sql_safe_updates
sql_select_limit
sql_slave_skip_counter
sql_warnings
--ssl
Ssl_accept_renegotiates
Ssl_accepts
--ssl-ca, ssl_ca
Ssl_callback_cache_hits
--ssl-capath, ssl_capath
--ssl-cert, ssl_cert
--ssl-cipher, ssl_cipher
Ssl_cipher
Ssl_cipher_list
Ssl_client_connects
Ssl_connect_renegotiates
--ssl-crl, ssl_crl
--ssl-crlpath, ssl_crlpath
Ssl_ctx_verify_depth
Ssl_ctx_verify_mode
Ssl_default_timeout
Ssl_finished_accepts
Ssl_finished_connects
--ssl-key, ssl_key
Ssl_server_not_after
Ssl_server_not_before
Ssl_session_cache_hits
Ssl_session_cache_misses
Ssl_session_cache_mode

Ssl_session_cache_overflows
Ssl_session_cache_size
Ssl_session_cache_timeouts
Ssl_sessions_reused
Ssl_used_session_cache_entries
Ssl_verify_depth
Ssl_verify_mode
--standard-compliant-cte, standard_compliant_cte
--stack-trace
--standalone
storage_engine
--stored-program-cache, stored_program_cache
--strict-password-validation, strict_password_validation
Subquery_cache_hit
Subquery_cache_miss
-s, --symbolic-links
--sync-binlog, sync_binlog
--sync-frm, sync_frm
--sync-master-info, sync_master_info
--sync-relay-log, sync_relay_log
--sync-relay-log-info, sync_relay_log_info
--sync-sys
Syncs
--sysdate-is-now
-T, --exit-info
--system_time_zone
--system-versioning-alter-history, system_versioning_alter_history
system_versioning_asof
--system-versioning-innodb-algorithm-simple, system_versioning_innodb_algorithm_simple
--system-versioning-insert-history, system_versioning_insert_history
--table-cache
--table-definition-cache, table_definition_cache
--table-lock-wait-timeout, table_lock_wait_timeout
Table_locks_immediate
Table_locks_waited
--table-open-cache, table_open_cache
Table_open_cache_active_instances
Table_open_cache_hits
--table-open-cache-instances, table_open_cache_instances
Table_open_cache_misses
Table_open_cache_overflows
table_type
--tc-heuristic-recover
Tc_log_max_pages_used
Tc_log_page_size
Tc_log_page_waits

--tcp-keepalive-interval, tcp_keepalive_interval
--tcp-keepalive-probes, tcp_keepalive_probes
--tcp-keepalive-interval, tcp_keepalive_time
--tcp-nodelay, tcp_nodelay
--temp-pool
--test-expect-abort
--test-ignore-wrong-options
--thread-alarm
--thread-cache-size, thread_cache_size
--thread-concurrency, thread_concurrency
--thread-handling, thread_handling
--thread-pool-dedicated-listener, thread_pool_dedicated_listener
--thread-pool-exact-stats, thread_pool_exact_stats
--thread-pool-idle-timeout, thread_pool_idle_timeout
--thread-pool-max-threads, thread_pool_max_threads
--thread-pool-min-threads, thread_pool_min_threads
thread_pool_oversubscribe
--thread-pool-prio-kickup-timer, thread_pool_prio_kickup_timer
--thread-pool-priority, thread_pool_priority
--thread-pool-size, thread_pool_size
--thread-pool-stall-limit, thread_pool_stall_limit
Threadpool_idle_threads
Threadpool_threads
--thread-stack, thread_stack
Threads_cached
Threads_connected
Threads_created
Threads_running
--timed-mutexes, timed_mutexes
timestamp
--time-format, time-format
time_zone
--tls-version, tls_version
--tmp-disk-table-size, tmp_disk_table_size
--tmp-memory-table-size, tmp_memory_table_size
--tmp-table-size, tmp_table_size
-t, --tmpdir, tmpdir
tokudb_alter_print_error
tokudb_analyze_time
Tokudb_basement_deserialization_fixed_key
Tokudb_basement_deserialization_variable_key
Tokudb_basements_decompressed_for_write
Tokudb_basements_decompressed_prefetch
Tokudb_basements_decompressed_prelocked_range
Tokudb_basements_decompressed_target_query
Tokudb_basements_fetched_for_write

Tokudb_basements_fetched_for_write_bytes
Tokudb_basements_fetched_for_write_seconds
Tokudb_basements_fetched_prefetch
Tokudb_basements_fetched_prefetch_bytes
Tokudb_basements_fetched_prefetch_seconds
Tokudb_basements_fetched_prelocked_range
Tokudb_basements_fetched_prelocked_range_bytes
Tokudb_basements_fetched_prelocked_range_seconds
Tokudb_basements_fetched_target_query
Tokudb_basements_fetched_target_query_bytes
Tokudb_basements_fetched_target_query_seconds
Tokudb_broadcase_messages_injected_at_root
Tokudb_buffers_decompressed_prefetch
Tokudb_buffers_decompressed_for_write
Tokudb_buffers_decompressed_prelocked_range
Tokudb_buffers_decompressed_target_query
Tokudb_buffers_fetched_for_write
Tokudb_buffers_fetched_for_write_bytes
Tokudb_buffers_fetched_for_write_seconds
Tokudb_buffers_fetched_prefetch
Tokudb_buffers_fetched_prefetch_bytes
Tokudb_buffers_fetched_prefetch_seconds
Tokudb_buffers_fetched_prelocked_range
Tokudb_buffers_fetched_prelocked_range_bytes
Tokudb_buffers_fetched_prelocked_range_seconds
Tokudb_buffers_fetched_target_query
Tokudb_buffers_fetched_target_query_bytes
Tokudb_buffers_fetched_target_query_seconds
tokudb_bulk_fetch
tokudb_cache_size
Tokudb_cachetable_cleaner_executions
Tokudb_cachetable_cleaner_iterations
Tokudb_cachetable_cleaner_period
Tokudb_cachetable_evictions
Tokudb_cachetable_long_wait_pressure_count
Tokudb_cachetable_long_wait_pressure_time
Tokudb_cachetable_miss
Tokudb_cachetable_miss_time
Tokudb_cachetable_prefetches
Tokudb_cachetable_size_cachepressure
Tokudb_cachetable_size_cloned
Tokudb_cachetable_size_current
Tokudb_cachetable_size_leaf
Tokudb_cachetable_size_limit
Tokudb_cachetable_size_nonleaf
Tokudb_cachetable_size_rollback

Tokudb_cachetable_size_writing
Tokudb_cachetable_wait_pressure_count
Tokudb_cachetable_wait_pressure_time
tokudb_check_jemalloc
Tokudb_checkpoint_begin_time
Tokudb_checkpoint_duration
Tokudb_checkpoint_duration_last
Tokudb_checkpoint_failed
Tokudb_checkpoint_last_began
Tokudb_checkpoint_last_complete_began
Tokudb_checkpoint_last_complete_ended
tokudb_checkpoint_lock
Tokudb_checkpoint_long_begin_count
Tokudb_checkpoint_long_begin_time
tokudb_checkpoint_on_flush_logs
Tokudb_checkpoint_period
Tokudb_checkpoint_taken
tokudb_checkpointing_period
tokudb_cleaner_iterations
tokudb_cleaner_period
tokudb_commit_sync
tokudb_create_index_online
Tokudb_cursor_skip_deleted_leaf_entry
tokudb_data_dir
Tokudb_db_closes
Tokudb_db_open_current
Tokudb_db_open_max
Tokudb_db_opens
tokudb_debug
Tokudb_descriptor_set
Tokudb_dictionary_broadcast_updates
Tokudb_dictionary_updates
tokudb_directio
tokudb_disable_hot_alter
tokudb_disable_prefetching
tokudb_disable_slow_alter
tokudb_empty_scan
Tokudb_filesystem_fsync_num
Tokudb_filesystem_fsync_time
Tokudb_filesystem_long_fsync_num
Tokudb_filesystem_long_fsync_time
Tokudb_filesystem_threads_blocked_by_full_disk
tokudb_fs_reserve_percent
tokudb_fsync_log_period
tokudb_hide_default_row_format
tokudb_killed_time

tokudb_last_lock_timeout
Tokudb_leaf_compression_to_memory_seconds
Tokudb_leaf_decompression_to_memory_seconds
Tokudb_leaf_deserialization_to_memory_seconds
Tokudb_leaf_node_compression_ratio
Tokudb_leaf_node_full_evictions
Tokudb_leaf_node_full_evictions_bytes
Tokudb_leaf_node_partial_evictions
Tokudb_leaf_node_partial_evictions_bytes
Tokudb_leaf_nodes_created
Tokudb_leaf_nodes_destroyed
Tokudb_leaf_nodes_flushed_checkpoint
Tokudb_leaf_nodes_flushed_checkpoint_bytes
Tokudb_leaf_nodes_flushed_checkpoint_seconds
Tokudb_leaf_nodes_flushed_checkpoint_uncompressed_bytes
Tokudb_leaf_nodes_flushed_not_checkpoint
Tokudb_leaf_nodes_flushed_not_checkpoint_bytes
Tokudb_leaf_nodes_flushed_not_checkpoint_secondss
Tokudb_leaf_nodes_flushed_not_checkpoint_uncompressed_bytes
Tokudb_leaf_serialization_to_memory_seconds
tokudb_load_save_space
tokudb_loader_memory_size
Tokudb_loader_num_created
Tokudb_loader_num_current
Tokudb_loader_num_max
tokudb_lock_timeout
tokudb_lock_timeout_debug
Tokudb_locktree_escalation_num
Tokudb_locktree_escalation_seconds
Tokudb_locktree_latest_post_escalation_memory_size
Tokudb_locktree_long_wait_count
Tokudb_locktree_long_wait_escalation_count
Tokudb_locktree_long_wait_escalation_time
Tokudb_locktree_long_wait_time
Tokudb_locktree_memory_size
Tokudb_locktree_memory_size_limit
Tokudb_locktree_open_current
Tokudb_locktree_pending_lock_requests
Tokudb_locktree_sto_eligible_num
Tokudb_locktree_sto_ended_num
Tokudb_locktree_sto_ended_seconds
Tokudb_locktree_timeout_count
Tokudb_locktree_wait_count
Tokudb_locktree_wait_escalation_count
Tokudb_locktree_wait_escalation_time
Tokudb_locktree_wait_time

tokudb_log_dir
Tokudb_logger_wait_long
Tokudb_logger_writes
Tokudb_logger_writes_bytes
Tokudb_logger_writes_seconds
Tokudb_logger_writes_uncompressed_bytes
tokudb_max_lock_memory
Tokudb_mem_estimated_maximum_memory_footprint
Tokudb_messages_flushed_from_h1_to_leaves_bytes
Tokudb_messages_ignored_by_leaf_due_to_msn
Tokudb_messages_in_trees_estimate_bytes
Tokudb_messages_injected_at_root
Tokudb_messages_injected_at_root_bytes
Tokudb_nonleaf_compression_to_memory_seconds
Tokudb_nonleaf_decompression_to_memory_seconds
Tokudb_nonleaf_deserialization_to_memory_seconds
Tokudb_nonleaf_node_compression_ratio
Tokudb_nonleaf_node_full_evictions
Tokudb_nonleaf_node_full_evictions_bytes
Tokudb_nonleaf_node_partial_evictions
Tokudb_nonleaf_node_partial_evictions_bytes
Tokudb_nonleaf_nodes_created
Tokudb_nonleaf_nodes_destroyed
Tokudb_nonleaf_nodes_flushed_to_disk_checkpoint
Tokudb_nonleaf_nodes_flushed_to_disk_checkpoint_bytes
Tokudb_nonleaf_nodes_flushed_to_disk_checkpoint_seconds
Tokudb_nonleaf_nodes_flushed_to_disk_checkpoint_uncompressed_bytes
Tokudb_nonleaf_nodes_flushed_to_disk_not_checkpoint
Tokudb_nonleaf_nodes_flushed_to_disk_not_checkpoint_bytes
Tokudb_nonleaf_nodes_flushed_to_disk_not_checkpoint_seconds
Tokudb_nonleaf_nodes_flushed_to_disk_not_checkpoint_uncompressed_bytes
Tokudb_nonleaf_serialization_to_memory_seconds
tokudb_optimize_index_fraction
tokudb_optimize_index_name
tokudb_optimize_throttle
Tokudb_overall_node_compression_ratio
Tokudb_pivots_fetched_for_query
Tokudb_pivots_fetched_for_query_bytes
Tokudb_pivots_fetched_for_query_seconds
Tokudb_pivots_fetched_for_prefetch
Tokudb_pivots_fetched_for_prefetch_bytes
Tokudb_pivots_fetched_for_prefetch_seconds
Tokudb_pivots_fetched_for_write
Tokudb_pivots_fetched_for_write_bytes
Tokudb_pivots_fetched_for_write_seconds
tokudb_pk_insert_mode

tokudb_prelock_empty
Tokudb_promotion_h1_roots_injected_into
Tokudb_promotion_injections_at_depth_0
Tokudb_promotion_injections_at_depth_1
Tokudb_promotion_injections_at_depth_2
Tokudb_promotion_injections_at_depth_3
Tokudb_promotion_injections_lower_than_depth_3
Tokudb_promotion_leaf_roots_injected_into
Tokudb_promotion_roots_split
Tokudb_promotion_stopped_after_locking_child
Tokudb_promotion_stopped_at_height_1
Tokudb_promotion_stopped_child_locked_or_not_in_memory
Tokudb_promotion_stopped_child_not_fully_in_memory
Tokudb_promotion_stopped_nonempty_buffer
tokudb_read_block_size
tokudb_read_buf_size
tokudb_read_status_frequency
tokudb_row_format
tokudb_rpl_check_readonly
tokudb_rpl_lookup_rows
tokudb_rpl_lookup_rows_delay
tokudb_rpl_unique_checks
tokudb_rpl_unique_checks_delay
tokudb_support_xa
tokudb_tmp_dir
Tokudb_txn_aborts
Tokudb_txn_begin
Tokudb_txn_begin_read_only
Tokudb_txn_commits
tokudb_version
tokudb_write_status_frequency
--transaction-alloc-block-size, transaction_alloc_block_size
--transaction-isolation, transaction_isolation
--transaction-prealloc-size, transaction_prealloc_size
--transaction-read-only
Transactions_gtid_foreign_engine
Transactions_multi_engine
tx_isolation
tx_read_only
-u, --user
unique_checks
--updatable-views-with-limit, updatable_views_with_limit
Update_scan
Uptime
Uptime_since_flush_status
-u, --user

--use-stat-tables, use_stat_tables
--userstat, userstat
-v, --verbose
-V, --version, version
version_comment
version_compile_machine
version_compile_os
version_malloc_library
version_source_revision
version_ssl_library
-W, --log-warnings, log_warnings
--wait-timeout, wait_timeout
warning_count
wsrep
wsrep_allowlist
wsrep_applier_thread_count
wsrep_apply_oooe
wsrep_apply_oool
wsrep_auto_increment_control
wsrep_causal_reads
wsrep_cert_deps_distance
wsrep_certification_rules
wsrep_certify_nonPK
wsrep_cluster_address
wsrep_cluster_capabilities
wsrep_cluster_conf_id
wsrep_cluster_name
wsrep_cluster_size
wsrep_cluster_state_uuid
wsrep_cluster_status
wsrep_connected
wsrep_convert_LOCK_to_trx
wsrep_data_home_dir
wsrep_dbug_option
wsrep_debug
wsrep_desync
wsrep_dirty_reads
wsrep_drupal_282555_workaround
wsrep_flow_control_paused
wsrep_flow_control_recv
wsrep_flow_control_sent
wsrep_gtid_domain_id
wsrep_gtid_mode
wsrep_gtid_seq_no
wsrep_forced_binlog_format
wsrep_ignore_apply_errors

wsrep_last_committed
wsrep_load_data_splitting
wsrep_local_bf_aborts
wsrep_local_cert_failures
wsrep_local_commits
wsrep_local_index
wsrep_local_recv_queue
wsrep_local_recv_queue_avg
wsrep_local_replays
wsrep_local_send_queue
wsrep_local_send_queue_avg
wsrep_local_state
wsrep_local_state_comment
wsrep_local_state_uuid
wsrep_log_conflicts
wsrep_max_ws_rows
wsrep_max_ws_size
wsrep_mode
wsrep_mysql_replication_bundle
--wsrep-new-cluster
wsrep_node_address
wsrep_node_incoming_address
wsrep_node_name
wsrep_notify_cmd
wsrep_on
wsrep_OSU_method
wsrep_protocol_version
wsrep_provider_name
wsrep_patch_version
wsrep_provider
wsrep_provider_options
wsrep_provider_vendor
wsrep_provider_version
wsrep_ready
wsrep_received
wsrep_received_bytes
wsrep_recover
wsrep_reject_queries
wsrep_replicate_myisam
wsrep_replicated
wsrep_replicated_bytes
wsrep_retry_autocommit
wsrep_rollbacker_thread_count
wsrep_slave_FK_checks
wsrep_slave_threads
wsrep_slave_UK_checks

wsrep_sr_store
wsrep_sst_auth
wsrep_sst_donor
wsrep_sst_donor_rejects_queries
wsrep_sst_method
wsrep_sst_receive_address
wsrep_start_position
wsrep_status_file
wsrep_strict_ddl
wsrep_sync_wait
wsrep-trx-fragment-size
wsrep-trx-fragment-unit
wsrep_thread_count

2.7.2 Server System Variables

Contents

40. binlog_alter_two_phase
41. binlog_annotate_row_events
42. binlog_cache_size
43. binlog_checksum
44. binlog_commit_wait_count
45. binlog_commit_wait_usec
46. binlog_direct_non_transactional_updates
47. binlog_do_db
48. binlog_expire_logs_seconds
49. binlog_file_cache_size
50. binlog_format
51. binlog_ignore_db
52. binlog_optimize_thread_scheduling
53. binlog_do_db
54. binlog_row_event_max_size
55. binlog_row_metadata
56. binlog_stmt_cache_size
57. block_encryption_mode
58. bulk_insert_buffer_size
59. cassandra_default_thrift_host
60. cassandra_failure_retries
61. cassandra_insert_batch_size
62. cassandra_multiget_batch_size
63. cassandra_read_consistency
64. cassandra_rnd_batch_size
65. cassandra_write_consistency
66. character_set_client
67. character_set_collations
68. character_set_connection
69. character_set_database
70. character_set_filesystem
71. character_set_results
72. character_set_server
73. character_set_system
74. character_sets_dir
75. check_constraint_checks
76. collation_connection
77. collation_database
78. collation_server
79. column_compression_threshold
80. column_compression_zlib_level
81. column_compression_zlib_strategy
82. column_compression_zlib_wrap
83. completion_type
84. concurrent_insert
85. connect_class_path
86. connect_cond_push
87. connect_conv_size
88. connect_default_depth
89. connect_default_prec
90. connect_enable_mongo
91. connect_exact_info
92. connect_force_bson
93. connect_indx_map
94. connect_java_wrapper
95. connect_json_all_path
96. connect_json_grp_size
97. connect_json_null
98. connect_jvm_path
99. connect_timeout
100. connect_type_conv
101. connect_use_tempfile
102. connect_work_size
103. connect_xtrace
104. core_file
105. cracklib_password_check
106. cracklib_password_check_dictionary

107. datadir
108. date_format
109. datetime_format
110. deadlock_search_depth_long
111. deadlock_search_depth_short
112. deadlock_timeout_long
113. deadlock_timeout_short
114. debug/debug_dbug
115. debug_no_thread_alarm
116. debug_sync
117. default_master_connection
118. default_password_lifetime
119. default_regex_flags
120. default_storage_engine
121. default_table_type
122. default_tmp_storage_engine
123. default_week_format
124. delay_key_write
125. delayed_insert_limit
126. delayed_insert_timeout
127. delayed_queue_size
128. disconnect_on_expired_password
129. div_precision_increment
130. encrypt_binlog
131. encrypt_tmp_disk_tables
132. encrypt_tmp_files
133. encryption_algorithm
134. enforce_storage_engine
135. engine_condition_pushdown
136. eq_range_index_dive_limit
137. error_count
138. event_scheduler
139. expensive_subquery_limit
140. expire_logs_days
141. explicit_defaults_for_timestamp
142. external_user
143. extra_max_connections
144. extra_port
145. feedback
146. feedback_http_proxy
147. feedback_send_retry_wait
148. feedback_send_timeout
149. feedback_server_uid
150. feedback_url
151. feedback_user_info
152. file_key_management_encryption_algorithm
153. file_key_management_filekey
154. file_key_management_filename
155. flush
156. flush_time
157. foreign_key_checks
158. ft_boolean_syntax
159. ft_max_word_len
160. ft_min_word_len
161. ft_query_expansion_limit
162. ft_stopword_file
163. general_log
164. general_log_file
165. group_concat_max_len
166. gssapi_keytab_path
167. gssapi_principal_name
168. gssapi_mech_name
169. gtid_binlog_pos
170. gtid_binlog_state
171. gtid_cleanup_batch_size
172. gtid_current_pos
173. gtid_domain_id

174. gtid_ignore_duplicates
175. gtid_seq_no
176. gtid_slave_pos
177. gtid_strict_mode
178. gtid_pos_auto_engines
179. handlersocket_accept_balance
180. handlersocket_address
181. handlersocket_backlog
182. handlersocket_epoll
183. handlersocket_port
184. handlersocket_port_wr
185. handlersocket_sndbuf
186. handlersocket_rcvbuf
187. handlersocket_readsize
188. handlersocket_threads
189. handlersocket_threads_wr
190. handlersocket_timeout
191. handlersocket_verbose
192. handlersocket_wrlock_timeout
193. have_compress
194. have_crypt
195. have_csv
196. have_dynamic_loading
197. have_geometry
198. have_innodb
199. have_ndbcluster
200. have_openssl
201. have_partitioning
202. have_profiling
203. have_query_cache
204. have_rtree_keys
205. have_ssl
206. have_symlink
207. histogram_size
208. histogram_type
209. host_cache_size
210. hostname
211. identity
212. ignore_builtin_innodb
213. idle_readonly_transaction_timeout
214. idle_transaction_timeout
215. idle_write_transaction_timeout
216. ignore_db_dirs
217. in_predicate_conversion_threshold
218. in_transaction
219. init_connect
220. init_file
221. init_slave
222. innodb_adaptive_checkpoint
223. innodb_adaptive_flushing
224. innodb_adaptive_flushing_lwm
225. innodb_adaptive_flushing_method
226. innodb_adaptive_hash_index
227. innodb_adaptive_hash_index_partitions
228. innodb_adaptive_hash_index_parts
229. innodb_adaptive_max_sleep_delay
230. innodb_additional_mem_pool_size
231. innodb_api_bk_commit_interval
232. innodb_api_disable_rowlock
233. innodb_api_enable_binlog
234. innodb_api_enable_mdl
235. innodb_api_trx_level
236. innodb_auto_lru_dump
237. innodb_autoextend_increment
238. innodb_autoinc_lock_mode
239. innodb_background_scrub_data_check_interval
240. innodb_background_scrub_data_compressed
241. innodb_background_scrub_data_interval

242. innodb_background_scrub_data-uncompressed
243. innodb_blocking_buffer_pool_restore
244. innodb_buf_dump_status_frequency
245. innodb_buffer_pool_chunk_size
246. innodb_buffer_pool_dump_at_shutdown
247. innodb_buffer_pool_dump_now
248. innodb_buffer_pool_evict
249. innodb_buffer_pool_filename
250. innodb_buffer_pool_instances
251. innodb_buffer_pool_load_abort
252. innodb_buffer_pool_load_at_startup
253. innodb_buffer_pool_load_now
254. innodb_buffer_pool_load_pages_abort
255. innodb_buffer_pool_populate
256. innodb_buffer_pool_restore_at_startup
257. innodb_buffer_pool_shm_checksum
258. innodb_buffer_pool_shm_key
259. innodb_buffer_pool_size
260. innodb_change_buffer_dump
261. innodb_change_buffer_max_size
262. innodb_change_buffering
263. innodb_change_buffering_debug
264. innodb_checkpoint_age_target
265. innodb_checksum_algorithm
266. innodb_checksums
267. innodb_cleaner_lsn_age_factor
268. innodb_cmp_per_index_enabled
269. innodb_commit_concurrency
270. innodb_compression_algorithm
271. innodb_compression_default
272. innodb_compression_failure_threshold_pct
273. innodb_compression_level
274. innodb_compression_pad_pct_max
275. innodb_concurrency_tickets
276. innodb_corrupt_table_action
277. innodb_data_file_buffering
278. innodb_data_file_path
279. innodb_data_file_write_through
280. innodb_data_home_dir
281. innodb_deadlock_detect
282. innodb_deadlock_report
283. innodb_default_encryption_key_id
284. innodb_default_page_encryption_key
285. innodb_default_row_format
286. innodb_defragment
287. innodb_defragment_fill_factor
288. innodb_defragment_fill_factor_n_recs
289. innodb_defragment_frequency
290. innodb_defragment_n_pages
291. innodb_defragment_stats_accuracy
292. innodb_dict_size_limit
293. innodb_disable_sort_file_cache
294. innodb_disallow_writes
295. innodb_doublewrite
296. innodb_doublewrite_file
297. innodb_empty_free_list_algorithm
298. innodb_enable_unsafe_group_commit
299. innodb_encrypt_log
300. innodb_encrypt_tables
301. innodb_encrypt_temporary_tables
302. innodb_encryption_rotate_key_age
303. innodb_encryption_rotation_iops
304. innodb_encryption_threads
305. innodb_extra_rsegments
306. innodb_extra_undoslots
307. innodb_fake_changes
308. innodb_fast_checksum

309. innodb_fast_shutdown
310. innodb_fatal_semaphore_wait_threshold
311. innodb_file_format
312. innodb_file_format_check
313. innodb_file_format_max
314. innodb_file_per_table
315. innodb_fill_factor
316. innodb_flush_log_at_timeout
317. innodb_flush_log_at_trx_commit
318. innodb_flush_method
319. innodb_flush_neighbor_pages
320. innodb_flush_neighbors
321. innodb_flush_sync
322. innodb_flushing_avg_loops
323. innodb_force_load_corrupted
324. innodb_force_primary_key
325. innodb_force_recovery
326. innodb_foreground_preflush
327. innodb_ft_aux_table
328. innodb_ft_cache_size
329. innodb_ft_enable_diag_print
330. innodb_ft_enable_stopword
331. innodb_ft_max_token_size
332. innodb_ft_min_token_size
333. innodb_ft_num_word_optimize
334. innodb_ft_result_cache_limit
335. innodb_ft_server_stopword_table
336. innodb_ft_sort_pll_degree
337. innodb_ft_total_cache_size
338. innodb_ft_user_stopword_table
339. innodb_ibuf_accel_rate
340. innodb_ibuf_active_contract
341. innodb_ibuf_max_size
342. innodb_idle_flush_pct
343. innodb_immediate_scrub_data-uncompressed
344. innodb_import_table_from_xtrabackup
345. innodb_instant_alter_column_allowed
346. innodb_instrument_semaphores
347. innodb_io_capacity
348. innodb_io_capacity_max
349. innodb_kill_idle_transaction
350. innodb_large_prefix
351. innodb_lazy_drop_table
352. innodb_lock_schedule_algorithm
353. innodb_lock_wait_timeout
354. innodb_locking_fake_changes
355. innodb_locks_unsafe_for_binlog
356. innodb_log_arch_dir
357. innodb_log_arch_expire_sec
358. innodb_log_archive
359. innodb_log_block_size
360. innodb_log_buffer_size
361. innodb_log_checksum_algorithm
362. innodb_log_checksums
363. innodb_log_compressed_pages
364. innodb_log_file_size
365. innodb_log_file_write_through
366. innodb_log_files_in_group
367. innodb_log_group_home_dir
368. innodb_log_optimize_ddl
369. innodb_log_write_ahead_size
370. innodb_lru_flush_size
371. innodb_lru_scan_depth
372. innodb_max_bitmap_file_size
373. innodb_max_changed_pages
374. innodb_max_dirty_pages_pct
375. innodb_max_dirty_pages_pct_lwm
376. innodb_max_purge_lag

376. innodb_max_purge_lag
377. innodb_max_purge_lag_delay
378. innodb_max_purge_lag_wait
379. innodb_max_undo_log_size
380. innodb_merge_sort_block_size
381. innodb_mirrored_log_groups
382. innodb_monitor_disable
383. innodb_monitor_enable
384. innodb_monitor_reset
385. innodb_monitor_reset_all
386. innodb_mtflush_threads
387. innodb_numa_interleave
388. innodb_old_blocks_pct
389. innodb_old_blocks_time
390. innodb_online_alter_log_max_size
391. innodb_open_files
392. innodb_optimize_fulltext_only
393. innodb_page_cleaners
394. innodb_page_size
395. innodb_prefix_index_cluster_optimization
396. innodb_print_all_deadlocks
397. innodb_purge_batch_size
398. innodb_purge_rseg_truncate_frequency
399. innodb_purge_threads
400. innodb_random_read_ahead
401. innodb_read_ahead
402. innodb_read_ahead_threshold
403. innodb_read_io_threads
404. innodb_read_only
405. innodb_recovery_stats
406. innodb_recovery_update_relay_log
407. innodb_replication_delay
408. innodb_rollback_on_timeout
409. innodb_rollback_segments
410. innodb_safe_truncate
411. innodb_sched_priority_cleaner
412. innodb_scrub_log
413. innodb_scrub_log_interval
414. innodb_scrub_log_speed
415. innodb_show_locks_held
416. innodb_show_verbose_locks
417. innodb_simulate_comp_failures
418. innodb_sort_buffer_size
419. innodb_spin_wait_delay
420. innodb_stats_auto_recalc
421. innodb_stats_auto_update
422. innodb_stats_include_delete_marked
423. innodb_stats_method
424. innodb_stats_modified_counter
425. innodb_stats_on_metadata
426. innodb_stats_persistent
427. innodb_stats_persistent_sample_pages
428. innodb_stats_sample_pages
429. innodb_stats_traditional
430. innodb_stats_transient_sample_pages
431. innodb_stats_update_need_lock
432. innodb_status_output
433. innodb_status_output_locks
434. innodb_strict_mode
435. innodb_support_xa
436. innodb_sync_array_size
437. innodb_sync_spin_loops
438. innodb_table_locks
439. innodb_temp_data_file_path
440. innodb_thread_concurrency
441. innodb_thread_concurrency_timer_based
442. innodb_thread_sleep_delay
443. innodb_tmpdir

444. innodb_track_changed_pages
445. innodb_track_redo_log_now
446. innodb_truncate_temporary_tablespace_now
447. innodb_undo_directory
448. innodb_undo_log_truncate
449. innodb_undo_logs
450. innodb_use_atomic_writes
451. innodb_use_fallocate
452. innodb_use_global_flush_log_at_trx_commit
453. innodb_use_native_aio
454. innodb_use_purge_thread
455. innodb_use_stacktrace
456. innodb_use_sys_malloc
457. innodb_use_sys_stats_table
458. innodb_version
459. innodb_write_io_threads
460. insert_id
461. interactive_timeout
462. join_buffer_size
463. join_buffer_space_limit
464. join_cache_level
465. keep_files_on_create
466. key_buffer_size
467. key_cache_age_threshold
468. key_cache_block_size
469. key_cache_division_limit
470. key_cache_file_hash_size
471. key_cache_segments
472. large_files_support
473. large_page_size
474. large_pages
475. last_gtid
476. last_insert_id
477. lc_messages
478. lc_messages_dir
479. lc_time_names
480. license
481. local_infile
482. lock_wait_timeout
483. locked_in_memory
484. log
485. log_bin
486. log_bin_basename
487. log_bin_compress
488. log_bin_compress_min_len
489. log_bin_index
490. log_bin_trust_function_creators
491. log_disabled_statements
492. log_error
493. log_output
494. log_queries_not_using_indexes
495. log_slave_updates
496. log_slow_admin_statements
497. log_slow_disabled_statements
498. log_slow_filter
499. log_slow_min_examined_row_limit
500. log_slow_queries
501. log_slow_query
502. log_slow_query_file
503. log_slow_query_time
504. log_slow_rate_limit
505. log_slow_slave_statements
506. log_slow_verbosity
507. log_slow_max_warnings
508. log_tc_size
509. log_warnings
510. long_query_time

511. low_priority_updates
512. lower_case_file_system
513. lower_case_table_names
514. master_verify_checksum
515. max_allowed_packet
516. max_binlog_cache_size
517. max_binlog_size
518. max_binlog_stmt_cache_size
519. max_connect_errors
520. max_connections
521. max_delayed_threads
522. max_digest_length
523. max_error_count
524. max_heap_table_size
525. max_insert_delayed_threads
526. max_join_size
527. max_length_for_sort_data
528. max_long_data_size
529. max_password_errors
530. max_prepared_stmt_count
531. max_recursive_iterations
532. max_relay_log_size
533. max_rowid_filter_size
534. max_seeks_for_key
535. max_session_mem_used
536. max_sort_length
537. max_sp_recursion_depth
538. max_statement_time
539. max_tmp_tables
540. max_user_connections
541. max_write_lock_count
542. metadata_locks_cache_size
543. metadata_locks_hash_instances
544. min_examined_row_limit
545. mroonga_action_on_fulltext_query_error
546. mroonga_boolean_mode_syntax_flags
547. mroonga_database_path_prefix
548. mroonga_default_parser
549. mroonga_default_tokenizer
550. mroonga_default_wrapper_engine
551. mroonga_dry_write
552. mroonga_enable_operations_recording
553. mroonga_enable_optimization
554. mroonga_libgroonga_embedded
555. mroonga_libgroonga_support_zlib
556. mroonga_libgroonga_support_zstd
557. mroonga_libgroonga_version
558. mroonga_log_file
559. mroonga_log_level
560. mroonga_match_escalation_threshold
561. mroonga_max_n_records_for_estimate
562. mroonga_query_log_file
563. mroonga_vector_column_delimiter
564. mroonga_version
565. mrr_buffer_size
566. multi_range_count
567. myisam_block_size
568. myisam_data_pointer_size
569. myisam_max_sort_file_size
570. myisam_mmap_size
571. myisam_recover_options
572. myisam_repair_threads
573. myisam_sort_buffer_size
574. myisam_stats_method
575. myisam_use_mmap
576. mysql56_temporal_format
577. named_pipe
578. net_buffer_length

578. net_buffer_length
579. net_read_timeout
580. net_retry_count
581. net_write_timeout
582. note_verbosity
583. old
584. old_alter_table
585. old_mode
586. old_passwords
587. open_files_limit
588. optimizer_extra_pruning_depth
589. optimizer_max_sel_args
590. optimizer_max_sel_arg_weight
591. optimizer_prune_level
592. optimizer_search_depth
593. optimizer_selectivity_sampling_limit
594. optimizer_switch
595. optimizer_trace
596. optimizer_trace_max_mem_size
597. optimizer_use_condition_selectivity
598. oqgraph_allow_create_integer_latch
599. pam_debug
600. pam_use_cleartext_plugin
601. pam_winbind_workaround
602. performance_schema
603. performance_schema_accounts_size
604. performance_schema_digests_size
605. performance_schema_events_stages_history_long_size
606. performance_schema_events_stages_history_size
607. performance_schema_events_statements_history_long_size
608. performance_schema_events_statements_history_size
609. performance_schema_events_transactions_history_long_size
610. performance_schema_events_transactions_history_size
611. performance_schema_events_waits_history_long_size
612. performance_schema_events_waits_history_size
613. performance_schema_hosts_size
614. performance_schema_max_cond_classes
615. performance_schema_max_cond_instances
616. performance_schema_max_digest_length
617. performance_schema_max_file_classes
618. performance_schema_max_file_handles
619. performance_schema_max_file_instances
620. performance_schema_max_index_stat
621. performance_schema_max_memory_classes
622. performance_schema_max_metadata_locks
623. performance_schema_max_mutex_classes
624. performance_schema_max_mutex_instances
625. performance_schema_max_prepared_statement_instances
626. performance_schema_max_program_instances
627. performance_schema_max_sql_text_length
628. performance_schema_max_rwlock_classes
629. performance_schema_max_rwlock_instances
630. performance_schema_max_socket_classes
631. performance_schema_max_socket_instances
632. performance_schema_max_stage_classes
633. performance_schema_max_statement_classes
634. performance_schema_max_statement_stack
635. performance_schema_max_table_handles
636. performance_schema_max_table_instances
637. performance_schema_max_table_lock_stat
638. performance_schema_max_thread_classes
639. performance_schema_max_thread_instances
640. performance_schema_session_connect_attrs_size
641. performance_schema_setup_actors_size
642. performance_schema_setup_objects_size
643. performance_schema_users_size
644. pid_file
645. plugin_dir

646. plugin_maturity
647. port
648. preload_buffer_size
649. profiling
650. profiling_history_size
651. progress_report_time
652. protocol_version
653. proxy_protocol_networks
654. proxy_user
655. pseudo_slave_mode
656. pseudo_thread_id
657. query_alloc_block_size
658. query_cache_limit
659. query_cache_min_res_unit
660. query_cache_size
661. query_cache_strip_comments
662. query_cache_type
663. query_cache_wlock_invalidate
664. query_prealloc_size
665. query_response_time_flush
666. query_response_time_range_base
667. query_response_time_range_exec_time_debug
668. query_response_time_stats
669. rand_seed1
670. rand_seed2
671. range_alloc_block_size
672. read_binlog_speed_limit
673. read_buffer_size
674. read_only
675. read_rnd_buffer_size
676. redirect_url
677. relay_log
678. relay_log_basename
679. relay_log_index
680. relay_log_info_file
681. relay_log_purge
682. relay_log_recovery
683. relay_log_space_limit
684. replicate_annotate_row_events
685. replicate_do_db
686. replicate_do_table
687. replicate_events_marked_for_skip
688. replicate_ignore_db
689. replicate_ignore_table
690. replicate_rewrite_db
691. replicate_wild_do_table
692. replicate_wild_ignore_table
693. report_host
694. report_password
695. report_port
696. report_user
697. require_secure_transport
698. rocksdb_access_hint_on_compaction_start
699. rocksdb_advise_random_on_open
700. rocksdb_allow_concurrent_memtable_write
701. rocksdb_allow_mmap_reads
702. rocksdb_allow_mmap_writes
703. rocksdb_allow_to_start_after_corruption
704. rocksdb_background_sync
705. rocksdb_base_background_compactions
706. rocksdb_blind_delete_primary_key
707. rocksdb_block_cache_size
708. rocksdb_block_restart_interval
709. rocksdb_block_size
710. rocksdb_block_size_deviation
711. rocksdb_bulk_load
712. rocksdb_bulk_load_allow_sk

713. rocksdb_bulk_load_allow_unsorted
714. rocksdb_bulk_load_size
715. rocksdb_bytes_per_sync
716. rocksdb_cache_dump
717. rocksdb_cache_high_pri_pool_ratio
718. rocksdb_cache_index_and_filter_blocks
719. rocksdb_cache_index_and_filter_with_high_priority
720. rocksdb_checksums_pct
721. rocksdb_collect_sst_properties
722. rocksdb_commit_in_the_middle
723. rocksdb_commit_time_batch_for_recovery
724. rocksdb_compact_cf
725. rocksdb_compaction_readahead_size
726. rocksdb_compaction_sequential_deletes
727. rocksdb_compaction_sequential_deletes_count_sd
728. rocksdb_compaction_sequential_deletes_file_size
729. rocksdb_compaction_sequential_deletes_window
730. rocksdb_concurrent_prepare
731. rocksdb_create_checkpoint
732. rocksdb_create_if_missing
733. rocksdb_create_missing_column_families
734. rocksdb_datadir
735. rocksdb_db_write_buffer_size
736. rocksdb_deadlock_detect
737. rocksdb_deadlock_detect_depth
738. rocksdb_debug_manual_compaction_delay
739. rocksdb_debug_optimizer_no_zero_cardinality
740. rocksdb_debug_ttl_ignore_pk
741. rocksdb_debug_ttl_read_filter_ts
742. rocksdb_debug_ttl_rec_ts
743. rocksdb_debug_ttl_snapshot_ts
744. rocksdb_default_cf_options
745. rocksdb_delayed_write_rate
746. rocksdb_delete_cf
747. rocksdb_delete_obsolete_files_period_micros
748. rocksdb_enable_2pc
749. rocksdb_enable_bulk_load_api
750. rocksdb_enable_insert_with_update_caching
751. rocksdb_enable_thread_tracking
752. rocksdb_enable_ttl
753. rocksdb_enable_ttl_read_filtering
754. rocksdb_enable_write_thread_adaptive_yield
755. rocksdb_error_if_exists
756. rocksdb_error_on_suboptimal_collation
757. rocksdb_flush_log_at_trx_commit
758. rocksdb_flush_memtable_on_analyze
759. rocksdb_force_compute_memtable_stats
760. rocksdb_force_compute_memtable_stats_cachetime
761. rocksdb_force_flush_memtable_and_lzero_now
762. rocksdb_force_flush_memtable_now
763. rocksdb_force_index_records_in_range
764. rocksdb_git_hash
765. rocksdb_hash_index_allow_collision
766. rocksdb_ignore_unknown_options
767. rocksdb_index_type
768. rocksdb_info_log_level
769. rocksdb_io_write_timeout
770. rocksdb_is_fd_close_on_exec
771. rocksdb_keep_log_file_num
772. rocksdb_large_prefix
773. rocksdb_lock_scanned_rows
774. rocksdb_lock_wait_timeout
775. rocksdb_log_dir
776. rocksdb_log_file_time_to_roll
777. rocksdb_manifest_preallocation_size
778. rocksdb_manual_compaction_threads
779. rocksdb_manual_wal_flush
780. rocksdb_master_skip_tx_api

780. rocksdb_master_skip_tx_api
781. rocksdb_max_background_compactions
782. rocksdb_max_background_flushes
783. rocksdb_max_background_jobs
784. rocksdb_max_latest_deadlocks
785. rocksdb_max_log_file_size
786. rocksdb_max_manifest_file_size
787. rocksdb_max_manual_compactions
788. rocksdb_max_open_files
789. rocksdb_max_row_locks
790. rocksdb_max_subcompactions
791. rocksdb_max_total_wal_size
792. rocksdb_merge_buf_size
793. rocksdb_merge_combine_read_size
794. rocksdb_merge_tmp_file_removal_delay_ms
795. rocksdb_new_table_reader_for_compaction_inputs
796. rocksdb_no_block_cache
797. rocksdb_override_cf_options
798. rocksdb_paranoid_checks
799. rocksdb_pause_background_work
800. rocksdb_perf_context_level
801. rocksdb_persistent_cache_path
802. rocksdb_persistent_cache_size_mb
803. rocksdb_pin_l0_filter_and_index_blocks_in_cache
804. rocksdb_print_snapshot_conflict_queries
805. rocksdb_rate_limiter_bytes_per_sec
806. rocksdb_read_free_rpl_tables
807. rocksdb_records_in_range
808. rocksdb_remove_mariabackup_checkpoint
809. rocksdb_reset_stats
810. rocksdb_rollback_on_timeout
811. rocksdb_seconds_between_stat_computes
812. rocksdb_signal_drop_index_thread
813. rocksdb_sim_cache_size
814. rocksdb_skip_bloom_filter_on_read
815. rocksdb_skip_fill_cache
816. rocksdb_skip_unique_check_tables
817. rocksdb-sst-mgr-rate-bytes-per-sec
818. rocksdb_stats_dump_period_sec
819. rocksdb_stats_level
820. rocksdb_stats_recalc_rate
821. rocksdb_store_row_debug_checksums
822. rocksdb_strict_collation_check
823. rocksdb_strict_collation_exceptions
824. rocksdb_supported_compression_types
825. rocksdb_table_cache_numshardbits
826. rocksdb_table_stats_sampling_pct
827. rocksdb_tmpdir
828. rocksdb_trace_sst_api
829. rocksdb_two_write_queues
830. rocksdb_unsafe_for_binlog
831. rocksdb_update_cf_options
832. rocksdb_use_adaptive_mutex
833. rocksdb_use_clock_cache
834. rocksdb_use_direct_io_for_flush_and_compaction
835. rocksdb_use_direct_reads
836. rocksdb_use_fsync
837. rocksdb_validate_tables
838. rocksdb_verify_row_debug_checksums
839. rocksdb_wal_bytes_per_sync
840. rocksdb_wal_dir
841. rocksdb_wal_recovery_mode
842. rocksdb_wal_size_limit_mb
843. rocksdb_wal_ttl_seconds
844. rocksdb_whole_key_filtering
845. rocksdb_write_batch_max_bytes
846. rocksdb_write_disable_wal
847. rocksdb_write_ignore_missing_column_families

848. rocksdb_write_policy
849. rowid_merge_buff_size
850. rpl_recovery_rank
851. rpl_semi_sync_master_enabled
852. rpl_semi_sync_master_timeout
853. rpl_semi_sync_master_trace_level
854. rpl_semi_sync_master_wait_no_slave
855. rpl_semi_sync_master_wait_point
856. rpl_semi_sync_slave_delay_master
857. rpl_semi_sync_slave_enabled
858. rpl_semi_sync_slave_kill_conn_timeout
859. rpl_semi_sync_slave_trace_level
860. s3_access_key
861. s3_block_size
862. s3_bucket
863. s3_debug
864. s3_host_name
865. s3_pagecache_age_threshold
866. s3_pagecache_buffer_size
867. s3_pagecache_division_limit
868. s3_pagecache_file_hash_size
869. s3_port
870. s3_protocol_version
871. s3_region
872. s3_secret_key
873. s3_slave_ignore_updates
874. s3_use_http
875. safe_show_database
876. secure_auth
877. secure_file_priv
878. secure_timestamp
879. server_audit_events
880. server_audit_excl_users
881. server_audit_file_path
882. server_audit_file_rotate_now
883. server_audit_file_rotate_size
884. server_audit_file_rotations
885. server_audit_incl_users
886. server_audit_loc_info
887. server_audit_logging
888. server_audit_mode
889. server_audit_output_type
890. server_audit_query_limit
891. server_audit_syslog_facility
892. server_audit_syslog_ident
893. server_audit_syslog_info
894. server_audit_syslog_priority
895. server_id
896. session_track_schema
897. session_track_state_change
898. session_track_system_variables
899. session_track_transaction_info
900. shared_memory
901. shared_memory_base_name
902. simple_password_check_digits
903. simple_password_check_letters_same_case
904. simple_password_check_minimal_length
905. simple_password_check_other_characters
906. skip_external_locking
907. skip_grant_tables
908. skip_name_resolve
909. skip_networking
910. skip_parallel_replication
911. skip_replication
912. skip_show_database
913. slave_compressed_protocol
914. slave_ddl_exec_mode

915. slave_domain_parallel_threads
916. slave_exec_mode
917. slave_load_tmpdir
918. slave_max_allowed_packet
919. slave_max_statement_time
920. slave_net_timeout
921. slave_parallel_max_queued
922. slave_parallel_mode
923. slave_parallel_threads
924. slave_parallel_workers
925. slave_run_triggers_for_rbr
926. slave_skip_errors
927. slave_sql_verify_checksum
928. slave_transaction_retries
929. slave_transaction_retry_errors
930. slave_transaction_retry_interval
931. slave_type_conversions
932. slow_launch_time
933. slow_query_log
934. slow_query_log_file
935. socket
936. sort_buffer_size
937. spider_auto_increment_mode
938. spider_auto_increment_mode
939. spider_auto_increment_mode
940. spider_bgs_second_read
941. spider_bka_engine
942. spider_bka_mode
943. spider_block_size
944. spider_buffer_size
945. spider_bulk_size
946. spider_bulk_update_mode
947. spider_bulk_update_size
948. spider_casual_read
949. spider_conn_recycle_mode
950. spider_conn_recycle_strict
951. spider_conn_wait_timeout
952. spider_connect_error_interval
953. spider_connect_mutex
954. spider_connect_retry_count
955. spider_connect_retry_interval
956. spider_connect_timeout
957. spider_crd_bg_mode
958. spider_crd_interval
959. spider_crd_mode
960. spider_crd_sync
961. spider_crd_type
962. spider_crd_weight
963. spider_delete_all_rows_type
964. spider_direct_dup_insert
965. spider_direct_order_limit
966. spider_dry_access
967. spider_error_read_mode
968. spider_error_write_mode
969. spider_first_read
970. spider_force_commit
971. spider_general_log
972. spider_ignore_comments
973. spider_index_hint_pushdown
974. spider_init_sql_alloc_size
975. spider_internal_limit
976. spider_internal_offset
977. spider_internal_optimize
978. spider_internal_optimize_local
979. spider_internal_sql_log_off
980. spider_internal_unlock
981. spider_internal_xa
982. spider_internal_xa_id_type

983. spider_internal_xa_snapshot
984. spider_load_crd_at_startup
985. spider_load_sts_at_startup
986. spider_local_lock_table
987. spider_lock_exchange
988. spider_log_result_error_with_sql
989. spider_log_result_errors
990. spider_low_mem_read
991. spider_max_connections
992. spider_max_order
993. spider_multi_split_read
994. spider_net_read_timeout
995. spider_net_write_timeout
996. spider_ping_interval_at_trx_start
997. spider_quick_mode
998. spider_quick_page_byte
999. spider_quick_page_size
1000. spider_read_only_mode
1001. spider_remote_access_charset
1002. spider_remote_autocommit
1003. spider_remote_default_database
1004. spider_remote_sql_log_off
1005. spider_remote_time_zone
1006. spider_remote_trx_isolation
1007. spider_remote_wait_timeout
1008. spider_reset_sql_alloc
1009. spider_same_server_link
1010. spider_second_read
1011. spider_select_column_mode
1012. spider_selupd_lock_mode
1013. spider_semi_split_read
1014. spider_semi_split_read_limit
1015. spider_semi_table_lock
1016. spider_semi_table_lock_connection
1017. spider_semi_trx
1018. spider_semi_trx_isolation
1019. spider_skip_default_condition
1020. spider_skip_parallel_search
1021. spider_slave_trx_isolation
1022. spider_split_read
1023. spider_store_last_crd
1024. spider_store_last_sts
1025. spider_strict_group_by
1026. spider_sts_bg_mode
1027. spider_sts_interval
1028. spider_sts_mode
1029. spider_sts_sync
1030. spider_support_xa
1031. spider_suppress_comment_ignored_warning
1032. spider_sync_autocommit
1033. spider_sync_sql_mode
1034. spider_sync_time_zone
1035. spider_sync_trx_isolation
1036. spider_table_crd_thread_count
1037. spider_table_init_error_interval
1038. spider_table_sts_thread_count
1039. spider_udf_ct_bulk_insert_interval
1040. spider_udf_ct_bulk_insert_rows
1041. spider_udf_ds_bulk_insert_rows
1042. spider_udf_ds_table_loop_mode
1043. spider_udf_ds_use_real_table
1044. spider_udf_table_lock_mutex_count
1045. spider_udf_table_mon_mutex_count
1046. spider_use_all_conns_snapshot
1047. spider_use_cond_other_than_pk_for_update
1048. spider_use_consistent_snapshot
1049. spider_use_default_database

1050. spider_use_flash_logs
1051. spider_use_handler
1052. spider_use_pushdown_udf
1053. spider_use_table_charset
1054. spider_version
1055. spider_wait_timeout
1056. spider_xa_register_mode
1057. sql_auto_is_null
1058. sql_big_selects
1059. sql_big_tables
1060. sql_buffer_result
1061. sql_error_log_filename
1062. sql_error_log_rate
1063. sql_error_log_rotate
1064. sql_error_log_rotations
1065. sql_error_log_size_limit
1066. sql_error_log_warnings
1067. sql_if_exists
1068. sql_log_bin
1069. sql_log_off
1070. sql_log_update
1071. sql_low_priority_updates
1072. sql_max_join_size
1073. sql_mode
1074. sql_notes
1075. sql_quote_show_create
1076. sql_safe_updates
1077. sql_select_limit
1078. sql_slave_skip_counter
1079. sql_warnings
1080. ssl_ca
1081. ssl_capath
1082. ssl_cert
1083. ssl_cipher
1084. ssl_crl
1085. ssl_crlpath
1086. ssl_key
1087. storage_engine
1088. standard_compliant_cte
1089. stored_program_cache
1090. strict_password_validation
1091. sync_binlog
1092. sync_frm
1093. sync_master_info
1094. sync_relay_log
1095. sync_relay_log_info
1096. system_time_zone
1097. system_versioning_alter_history
1098. system_versioning_asof
1099. system_versioning_innodb_algorithm_simple
1100. system_versioning_insert_history
1101. table_definition_cache
1102. table_lock_wait_timeout
1103. table_open_cache
1104. table_open_cache_instances
1105. table_type
1106. tcp_keepalive_interval
1107. tcp_keepalive_probes
1108. tcp_keepalive_time
1109. tcp_nodelay
1110. thread_cache_size
1111. thread_concurrency
1112. thread_handling
1113. thread_pool_dedicated_listener
1114. thread_pool_exact_stats
1115. thread_pool_idle_timeout
1116. thread_pool_max_threads

1117. thread_pool_min_threads
1118. thread_pool_oversubscribe
1119. thread_pool_prio_kickup_timer
1120. thread_pool_priority
1121. thread_pool_size
1122. thread_pool_stall_limit
1123. thread_stack
1124. time_format
1125. time_zone
1126. timed_mutexes
1127. timestamp
1128. tls_version
1129. tmp_disk_table_size
1130. tmp_memory_table_size
1131. tmp_table_size
1132. tmpdir
1133. tokudb_alter_print_error
1134. tokudb_analyze_time
1135. tokudb_block_size
1136. tokudb_bulk_fetch
1137. tokudb_cache_size
1138. tokudb_check_jemalloc
1139. tokudb_checkpoint_lock
1140. tokudb_checkpoint_on_flush_logs
1141. tokudb_checkpointing_period
1142. tokudb_cleaner_iterations
1143. tokudb_cleaner_period
1144. tokudb_commit_sync
1145. tokudb_create_index_online
1146. tokudb_data_dir
1147. tokudb_debug
1148. tokudb_directio
1149. tokudb_disable_hot_alter
1150. tokudb_disable_prefetching
1151. tokudb_disable_slow_alter
1152. tokudb_empty_scan
1153. tokudb_fs_reserve_percent
1154. tokudb_fsync_log_period
1155. tokudb_hide_default_row_format
1156. tokudb_killed_time
1157. tokudb_last_lock_timeout
1158. tokudb_load_save_space
1159. tokudb_loader_memory_size
1160. tokudb_lock_timeout
1161. tokudb_lock_timeout_debug
1162. tokudb_log_dir
1163. tokudb_max_lock_memory
1164. tokudb_optimize_index_fraction
1165. tokudb_optimize_index_name
1166. tokudb_optimize_throttle
1167. tokudb_pk_insert_mode
1168. tokudb_prelock_empty
1169. tokudb_read_block_size
1170. tokudb_read_buf_size
1171. tokudb_read_status_frequency
1172. tokudb_row_format
1173. tokudb_rpl_check_readonly
1174. tokudb_rpl_lookup_rows
1175. tokudb_rpl_lookup_rows_delay
1176. tokudb_rpl_unique_checks
1177. tokudb_rpl_unique_checks_delay
1178. tokudb_support_xa
1179. tokudb_tmp_dir
1180. tokudb_version
1181. tokudb_write_status_frequency
1182. transaction_alloc_block_size
1183. transaction_isolation
1184. transaction_prealloc_size

1185. transaction_read_only
1186. tx_isolation
1187. tx_read_only
1188. unique_checks
1189. updatable_views_with_limit
1190. use_stat_tables
1191. userstat
1192. version
1193. version_comment
1194. version_compile_machine
1195. version_compile_os
1196. version_malloc_library
1197. version_source_revision
1198. version_ssl_library
1199. wait_timeout
1200. warning_count
1201. wsrep_allowlist
1202. wsrep_auto_increment_control
1203. wsrep_causal_reads
1204. wsrep_certification_rules
1205. wsrep_certify_nonPK
1206. wsrep_cluster_address
1207. wsrep_cluster_name
1208. wsrep_convert_LOCK_to_trx
1209. wsrep_data_home_dir
1210. wsrep_dbug_option
1211. wsrep_debug
1212. wsrep_desync
1213. wsrep_dirty_reads
1214. wsrep_drupal_282555_workaround
1215. wsrep_forced_binlog_format
1216. wsrep_gtid_domain_id
1217. wsrep_gtid_mode
1218. wsrep_gtid_seq_no
1219. wsrep_ignore_apply_errors
1220. wsrep_load_data_splitting
1221. wsrep_log_conflicts
1222. wsrep_max_ws_rows
1223. wsrep_max_ws_size
1224. wsrep_mode
1225. wsrep_mysql_replication_bundle
1226. wsrep_node_address
1227. wsrep_node_incoming_address
1228. wsrep_node_name
1229. wsrep_notify_cmd
1230. wsrep_on
1231. wsrep_OSU_method
1232. wsrep_provider
1233. wsrep_provider_options
1234. wsrep_recover
1235. wsrep_reject_queries
1236. wsrep_replicate_myisam
1237. wsrep_restart_slave
1238. wsrep_retry_autocommit
1239. wsrep_slave_FK_checks
1240. wsrep_slave_threads
1241. wsrep_slave_UK_checks
1242. wsrep_sr_store
1243. wsrep_sst_auth
1244. wsrep_sst_donor
1245. wsrep_sst_donor_rejects_queries
1246. wsrep_sst_method
1247. wsrep_sst_receive_address
1248. wsrep_start_position
1249. wsrep_status_file
1250. wsrep_strict_ddl
1251. wsrep_sync_wait

1252. wsrep_trx_fragment_size
1253. wsrep_trx_fragment_unit

About the Server System Variables

MariaDB has many system variables that can be changed to suit your needs.

The full list of server variables are listed in the contents on this page, and most are described on this page, but some are described elsewhere:

- Aria System Variables
- CONNECT System Variables
- Galera System Variables
- Global Transaction ID System Variables
- HandlerSocket Plugin System Variables
- InnoDB System Variables
- Mroonga System Variables
- MyRocks System Variables
- MyISAM System Variables
- Performance Schema System Variables
- Replication and Binary Log System Variables
- S3 Storage Engine System Variables
- Server_Audit System Variables
- Spider System Variables
- SQL_ERROR_LOG Plugin System Variables
- SSL System Variables
- Threadpool System Variables
- TokuDB System Variables

See also the Full list of MariaDB options, system and status variables.

Most of these can be set with command line options and many of them can be changed at runtime. Variables that can be changed at runtime (and therefore are not read-only) are described as "Dynamic" below, and elsewhere in the documentation.

There are a few ways to see the full list of server system variables:

- While in the mariadb client, run:

  ```
  SHOW VARIABLES;
  ```

  See SHOW VARIABLES for instructions on using this command.

- From your shell, run mariadbd like so:

  ```
  mariadbd --verbose --help
  ```

- View the Information Schema GLOBAL_VARIABLES, SESSION_VARIABLES, and SYSTEM_VARIABLES tables.

Setting Server System Variables

There are several ways to set server system variables:

- Specify them on the command line:

  ```
  shell> ./mysqld_safe --aria_group_commit="hard"
  ```

- Specify them in your my.cnf file (see Configuring MariaDB with my.cnf for more information):

  ```
  aria_group_commit = "hard"
  ```

- Set them from the mariadb client using the SET command. Only variables that are dynamic can be set at runtime in this way. Note that variables set in this way will not persist after a restart.

  ```
  SET GLOBAL aria_group_commit="hard";
  ```

By convention, server variables have usually been specified with an underscore in the configuration files, and a dash on the command line. You can however specify underscores as dashes - they are interchangeable.

Variables that take a numeric size can either be specified in full, or with a suffix for easier readability. Valid suffixes are:

Suffix	Description	Value
K	kilobytes	1024
M	megabytes	1024^2
G	gigabytes	1024^3
T	terabytes	1024^4 (from MariaDB 10.3.3)
P	petabytes	1024^5 (from MariaDB 10.3.3)
E	exabytes	1024^6 (from MariaDB 10.3.3)

The suffix can be upper or lower-case.

List of Server System Variables

allow_suspicious_udfs

- **Description:** Allows use of user-defined functions consisting of only one symbol `x()` without corresponding `x_init()` or `x_deinit()`. That also means that one can load any function from any library, for example `exit()` from `libc.so`. Not recommended unless you require old UDFs with one symbol that cannot be recompiled. Before MariaDB 10.10, available as an option only.
- **Commandline:** `--allow-suspicious-udfs`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB 10.10

---

alter_algorithm

- **Description:** The implied `ALGORITHM` for ALTER TABLE if no `ALGORITHM` clause is specified. The deprecated variable old_alter_table is an alias for this.
  - `COPY` corresponds to the pre-MySQL 5.1 approach of creating an intermediate table, copying data one row at a time, and renaming and dropping tables.
  - `INPLACE` requests that the operation be refused if it cannot be done natively inside a the storage engine.
  - `DEFAULT` (the default) chooses `INPLACE` if available, and falls back to `COPY`.
  - `NOCOPY` refuses to copy a table.
  - `INSTANT` refuses an operation that would involve any other than metadata changes.
- **Commandline:** `--alter-algorithm=default`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `enumerated`
- **Default Value:** `DEFAULT`
- **Valid Values:** `DEFAULT`, `COPY`, `INPLACE`, `NOCOPY`, `INSTANT`
- **Introduced:** MariaDB 10.3.7

analyze_sample_percentage

- **Description:** Percentage of rows from the table ANALYZE TABLE will sample to collect table statistics. Set to 0 to let MariaDB decide what percentage of rows to sample.
- **Commandline:** `--analyze-sample-percentage=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `100.000000`
- **Range:** `0` to `100`
- **Introduced:** MariaDB 10.4.3

autocommit

- **Description:** If set to 1, the default, all queries are committed immediately. The LOCK IN SHARE MODE and FOR UPDATE clauses therefore have no effect. If set to 0, they are only committed upon a COMMIT statement, or rolled

back with a ROLLBACK statement. If autocommit is set to 0, and then changed to 1, all open transactions are immediately committed.

- **Commandline:** `--autocommit[=#]`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `1`

---

automatic_sp_privileges

- **Description:** When set to 1, the default, when a stored routine is created, the creator is automatically granted permission to ALTER (which includes dropping) and to EXECUTE the routine. If set to 0, the creator is not automatically granted these privileges.
- **Commandline:** `--automatic-sp-privileges` , `--skip-automatic-sp-privileges`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `1`

---

back_log

- **Description:** Connections take a small amount of time to start, and this setting determines the number of outstanding connection requests MariaDB can have, or the size of the listen queue for incoming TCP/IP requests. Requests beyond this will be refused. Increase if you expect short bursts of connections. Cannot be set higher than the operating system limit (see the Unix listen() man page). If not set, set to `0` , or the `--autoset-back-log` option is used, will be autoset to the lower of `900` and (50 + max_connections/5).
- **Commandline:** `--back-log=#`
- **Scope:** Global
- **Dynamic:** No
- **Type:** number
- **Default Value:**
  - The lower of `900` and (50 + max_connections/5)

---

basedir

- **Description:** Path to the MariaDB installation directory. Other paths are usually resolved relative to this base directory.
- **Commandline:** `--basedir=path` or `-b path`
- **Scope:** Global
- **Dynamic:** No
- **Type:** directory name

---

big_tables

- **Description:** If this system variable is set to 1, then temporary tables will be saved to disk intead of memory.
  - This system variable's original intention was to allow result sets that were too big for memory-based temporary tables and to avoid the resulting 'table full' errors.
  - This system variable is no longer needed, because the server can automatically convert large memory-based temporary tables into disk-based temporary tables when they exceed the value of the `tmp_memory_table_size` system variable.
  - To prevent memory-based temporary tables from being used at all, set the `tmp_memory_table_size` system variable to `0` .
  - In MariaDB 5.5 and earlier, `sql_big_tables` is a synonym.
  - In MariaDB 10.5, this system variable is deprecated.
- **Commandline:** `--big-tables`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `0`
- **Deprecated:** MariaDB 10.5.0

---

bind_address

- **Description:** By default, the MariaDB server listens for TCP/IP connections on all addresses. You can specify an alternative when the server starts using this option; either a host name, an IPv4 or an IPv6 address, "::" or "*" (all addresses). In some systems, such as Debian and Ubuntu, the bind_address is set to 127.0.0.1, which binds the server to listen on localhost only. `bind_address` has always been available as a mariadbd option; from MariaDB 10.3.3 its also available as a system variable. Before MariaDB 10.6.0 "::" implied listening additionally on IPv4 addresses like "*". From 10.6.0 onwards it refers to IPv6 stictly. Starting with MariaDB 10.11, a comma-separated list of addresses to bind to can be given. See also Configuring MariaDB for Remote Client Access.
- **Commandline:** `--bind-address=addr`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** (Empty string)
- **Valid Values:** Host name, IPv4, IPv6, ::, *
- **Introduced:** MariaDB 10.3.3 (as a system variable)

---

block_encryption_mode

- **Description:** Default block encryption mode for AES_ENCRYPT() and AES_DECRYPT() functions.
- **Commandline:** `--block-encryption-mode=val`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `aes-128-ecb`
- **Valid values:** `aes-128-ecb`, `aes-192-ecb`, `aes-256-ecb`, `aes-128-cbc`, `aes-192-cbc`, `aes-256-cbc`, `aes-128-ctr`, `aes-192-ctr`, `aes-256-ctr`
- **Introduced:** MariaDB 11.2.0

---

bulk_insert_buffer_size

- **Description:** Size in bytes of the per-thread cache tree used to speed up bulk inserts into MyISAM and Aria tables. A value of 0 disables the cache tree.
- **Commandline:** `--bulk-insert-buffer-size=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `8388608`
- **Range - 32 bit:** `0` to `4294967295`
- **Range - 64 bit:** `0` to `18446744073709547520`

---

character_set_client

- **Description:** Determines the character set for queries arriving from the client. It can be set per session by the client, although the server can be configured to ignore client requests with the `--skip-character-set-client-handshake` option. If the client does not request a character set, or requests a character set that the server does not support, the global value will be used. utf16, utf16le, utf32 and ucs2 cannot be used as client character sets. From MariaDB 10.6, the `utf8` character set (and related collations) is by default an alias for `utf8mb3` rather than the other way around. It can be set to imply `utf8mb4` by changing the value of the old_mode system variable.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `utf8mb3` (>= MariaDB 10.6), `utf8` (<= MariaDB 10.5)

---

character_set_collations

- **Description:** Overrides for character set default collations. Takes a comma-delimited list of character set and collation settings, for example `SET @@character_set_collations = 'utf8mb4=uca1400_ai_ci, latin2=latin2_hungarian_ci';` The new variable will take effect in all cases where a character set is explicitly or implicitly specified without an explicit COLLATE clause, including but not limited to:
  - Column collation

    - Table collation
    - Database collation
    - CHAR(expr USING csname)
    - CONVERT(expr USING csname)
    - CAST(expr AS CHAR CHARACTER SET csname)
    - '' - character string literal
    - _utf8mb3'text' - a character string literal with an introducer
    - _utf8mb3 X'61' - a character string literal with an introducer with hex notation
    - _utf8mb3 0x61 - a character string literal with an introducer with hex hybrid notation
    - @@collation_connection after a SET NAMES without COLLATE
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** Empty
- **Introduced:** MariaDB 11.2

`character_set_connection`

- **Description:** Character set used for number to string conversion, as well as for literals that don't have a character set introducer. From MariaDB 10.6, the `utf8` character set (and related collations) is by default an alias for `utf8mb3` rather than the other way around. It can be set to imply `utf8mb4` by changing the value of the old_mode system variable.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `utf8mb3` (>= MariaDB 10.6), `utf8` (<= MariaDB 10.5)

`character_set_database`

- **Description:** Character set used by the default database, and set by the server whenever the default database is changed. If there's no default database, character_set_database contains the same value as character_set_server. This variable is dynamic, but should not be set manually, only by the server.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `latin1`

`character_set_filesystem`

- **Description:** The character set for the filesystem. Used for converting file names specified as a string literal from character_set_client to character_set_filesystem before opening the file. By default set to `binary`, so no conversion takes place. This could be useful for statements such as LOAD_FILE() or LOAD DATA INFILE on system where multi-byte file names are use.
- **Commandline:** `--character-set-filesystem=name`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `binary`

`character_set_results`

- **Description:** Character set used for results and error messages returned to the client. From MariaDB 10.6, the `utf8` character set (and related collations) is by default an alias for `utf8mb3` rather than the other way around. It can be set to imply `utf8mb4` by changing the value of the old_mode system variable.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `utf8mb3` (>= MariaDB 10.6), `utf8` (<= MariaDB 10.5)

`character_set_server`

- **Description:** Default character set used by the server. See character_set_database for character sets used by the default database. Defaults may be different on some systems, see for example Differences in MariaDB in Debian.
- **Commandline:** `--character-set-server`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `latin1`

---

character_set_system

- **Description:** Character set used by the server to store identifiers, always set to utf8, or its synonym utf8mb3 starting with MariaDB 10.6. From MariaDB 10.6, the `utf8` character set (and related collations) is by default an alias for `utf8mb3` rather than the other way around. It can be set to imply `utf8mb4` by changing the value of the old_mode system variable.
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** `utf8mb3` (>= MariaDB 10.6), `utf8` (<= MariaDB 10.5)

---

character_sets_dir

- **Description:** Directory where the character sets are installed.
- **Commandline:** `--character-sets-dir=path`
- **Scope:** Global
- **Dynamic:** No
- **Type:** directory name

---

check_constraint_checks

- **Description:** If set to `0`, will disable constraint checks, for example when loading a table that violates some constraints that you plan to fix later.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Type:** boolean
- **Default:** ON

---

collation_connection

- **Description:** Collation used for the connection character set.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `string`

---

collation_database

- **Description:** Collation used for the default database. Set by the server if the default database changes, if there is no default database the value from the `collation_server` variable is used. This variable is dynamic, but should not be set manually, only by the server.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `string`

---

collation_server

- **Description:** Default collation used by the server. This is set to the default collation for a given character set automatically when character_set_server is changed, but it can also be set manually. Defaults may be different on some systems, see for example Differences in MariaDB in Debian.
- **Commandline:** `--collation-server=name`
- **Scope:** Global, Session

- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `latin1_swedish_ci`

---

completion_type

- **Description:** The transaction completion type. If set to `NO_CHAIN` or `0` (the default), there is no effect on commits and rollbacks. If set to `CHAIN` or `1`, a COMMIT statement is equivalent to COMMIT AND CHAIN, while a ROLLBACK is equivalent to ROLLBACK AND CHAIN, so a new transaction starts straight away with the same isolation level as transaction that's just finished. If set to `RELEASE` or `2`, a COMMIT statement is equivalent to COMMIT RELEASE, while a ROLLBACK is equivalent to ROLLBACK RELEASE, so the server will disconnect after the transaction completes. Note that the transaction completion type only applies to explicit commits, not implicit commits.
- **Commandline:** `--completion-type=name`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `enumerated`
- **Default Value:** `NO_CHAIN`
- **Valid Values:** `0`, `1`, `2`, `NO_CHAIN`, `CHAIN`, `RELEASE`

---

concurrent_insert

- **Description:** If set to `AUTO` or `1`, the default, MariaDB allows concurrent INSERTs and SELECTs for MyISAM tables with no free blocks in the data (deleted rows in the middle). If set to `NEVER` or `0`, concurrent inserts are disabled. If set to `ALWAYS` or `2`, concurrent inserts are permitted for all MyISAM tables, even those with holes, in which case new rows are added at the end of a table if the table is being used by another thread.

  If the --skip-new option is used when starting the server, concurrent_insert is set to `NEVER`.

  Changing the variable only affects new opened tables. Use FLUSH TABLES If you want it to also affect cached tables.

  See Concurrent Inserts for more.
- **Commandline:** `--concurrent-insert[=value]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `enumerated`
- **Default Value:** `AUTO`
- **Valid Values:** `0`, `1`, `2`, `AUTO`, `NEVER`, `ALWAYS`

---

connect_timeout

- **Description:** Time in seconds that the server waits for a connect packet before returning a 'Bad handshake'. Increasing may help if clients regularly encounter 'Lost connection to MySQL server at 'X', system error: error_number' type-errors.
- **Commandline:** `--connect-timeout=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Type:** numeric
- **Default Value:** `10`

---

core_file

- **Description:** Write a core-file on crashes. The file name and location are system dependent. On Linux it is usually called `core.${PID}`, and it is usually written to the data directory. However, this can be changed.
  - See Enabling Core Dumps for more information.
  - Previously this system variable existed only as an option, but it was also made into a read-only system variable starting with MariaDB 10.3.9, MariaDB 10.2.17 and MariaDB 10.1.35.
  - On Windows >= MariaDB 10.4.3, this option is set by default.
  - Note that the option accepts no arguments; specifying `--core-file` sets the value to `ON`. It cannot be disabled in the case of Windows >= MariaDB 10.4.3.

- **Commandline:** `--core-file`
- **Scope:** Global
- **Dynamic:** No
- **Type:** boolean
- **Default Value:**
  - Windows >= MariaDB 10.4.3: `ON`
  - All other systems: `OFF`

---

datadir

- **Description:** Directory where the data is stored.
- **Commandline:** `--datadir=path` or `-h path`
- **Scope:** Global
- **Dynamic:** No
- **Type:** directory name

---

date_format

- **Description:** Unused.
- **Removed:** MariaDB 11.3.0

---

datetime_format

- **Description:** Unused.
- **Removed:** MariaDB 11.3.0

---

debug/debug_dbug

- **Description:** Available in debug builds only (built with -DWITH_DEBUG=1). Used in debugging through the DBUG library to write to a trace file. Just using `--debug` will write a trace of what mariadbd is doing to the default trace file.
- **Commandline:** `-#` , `--debug[=debug_options]`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:**
  - <= MariaDB 10.4: `d:t:i:o,/tmp/mysqld.trace` (Unix) or `d:t:i:O,\mysqld.trace` (Windows)
  - >= MariaDB 10.5: `d:t:i:o,/tmp/mariadbd.trace` (Unix) or `d:t:i:O,\mariadbd.trace` (Windows)
- **Debug Options:** See the option flags on the mysql_debug page
- **Removed:** MariaDB 11.3.0

---

debug_no_thread_alarm

- **Description:** Disable system thread alarm calls. Disabling it may be useful in debugging or testing, never do it in production.
- **Commandline:** `--debug-no-thead-alarm=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB

---

debug_sync

- **Description:** Used in debugging to show the interface to the Debug Sync facility. MariaDB needs to be configured with -DENABLE_DEBUG_SYNC=1 for this variable to be available.
- **Scope:** Session
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `OFF` or `ON - current signal` *`signal name`*

default_password_lifetime

- **Description:** This defines the global password expiration policy. 0 means automatic password expiration is disabled. If the value is a positive integer N, the passwords must be changed every N days. This behavior can be overridden using the password expiration options in ALTER USER.
- **Commandline:** `--default-password-lifetime=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Type:** numeric
- **Default Value:** `0`
- **Range:** `0` to `4294967295`
- **Introduced:** MariaDB 10.4.3

default_regex_flags

- **Description:** Introduced to address remaining incompatibilities between PCRE and the old regex library. Accepts a comma-separated list of zero or more of the following values:

Value	Pattern equivalent	Meaning
DOTALL	(?s)	. matches anything including NL
DUPNAMES	(?J)	Allow duplicate names for subpatterns
EXTENDED	(?x)	Ignore white space and # comments
EXTRA	(?X)	extra features (e.g. error on unknown escape character)
MULTILINE	(?m)	^ and $ match newlines within data
UNGREEDY	(?U)	Invert greediness of quantifiers

- **Commandline:** `--default-regex-flags=value`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Type:** enumeration
- **Default Value:** empty
- **Valid Values:** `DOTALL`, `DUPNAMES`, `EXTENDED`, `EXTRA`, `MULTILINE`, `UNGREEDY`

default_storage_engine

- **Description:** The default storage engine. The default storage engine must be enabled at server startup or the server won't start.
- **Commandline:** `--default-storage-engine=name`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Type:** enumeration
- **Default Value:** `InnoDB`

default_table_type

- **Description:** A synonym for default_storage_engine. Removed in MariaDB 5.5.
- **Commandline:** `--default-table-type=name`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Removed:** MariaDB 5.5

default_tmp_storage_engine

- **Description:** Default storage engine that will be used for tables created with CREATE TEMPORARY TABLE where no engine is specified. For internal temporary tables see aria_used_for_temp_tables). The storage engine used must be active or the server will not start. See default_storage_engine for the default for non-temporary tables. Defaults to NULL, in which case the value from default_storage_engine is used. ROCKSDB temporary tables cannot be created. Before MariaDB 10.7, attempting to do so would silently fail, and a MyISAM table would instead be created. From MariaDB 10.7, an error is returned.

- **Commandline:** `--default-tmp-storage-engine=name`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `enumeration`
- **Default Value:** NULL

---

default_week_format

- **Description:** Default mode for the WEEK() function. See that page for details on the different modes
- **Commandline:** `--default-week-format=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `7`

---

delay_key_write

- **Description:** Specifies how MyISAM tables handles CREATE TABLE DELAY_KEY_WRITE. If set to `ON`, the default, any DELAY KEY WRITEs are honored. The key buffer is then flushed only when the table closes, speeding up writes. MyISAM tables should be automatically checked upon startup in this case, and --external locking should not be used, as it can lead to index corruption. If set to `OFF`, DELAY KEY WRITEs are ignored, while if set to `ALL`, all new opened tables are treated as if created with DELAY KEY WRITEs enabled.
- **Commandline:** `--delay-key-write[=name]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `enumeration`
- **Default Value:** `ON`
- **Valid Values:** `ON`, `OFF`, `ALL`

---

delayed_insert_limit

- **Description:** After this many rows have been inserted with INSERT DELAYED, the handler will check for and execute any waiting SELECT statements.
- **Commandline:** `--delayed-insert-limit=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `100`
- **Range:** `1` to `4294967295`

---

delayed_insert_timeout

- **Description:** Time in seconds that the INSERT DELAYED handler will wait for INSERTs before terminating.
- **Commandline:** `--delayed-insert-timeout=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `300`

---

delayed_queue_size

- **Description:** Number of rows, per table, that can be queued when performing INSERT DELAYED statements. If the queue becomes full, clients attempting to perform INSERT DELAYED's will wait until the queue has room available again.
- **Commandline:** `--delayed-queue-size=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Type:** numeric
- **Default Value:** `1000`

- **Range:** `1 to 4294967295`

---

disconnect_on_expired_password

- **Description:** When a user password has expired (see User Password Expiry), this variable controls how the server handles clients that are not aware of the sandbox mode. If enabled, the client is not permitted to connect, otherwise the server puts the client in a sandbox mode.
- **Commandline:** `--disconnect-on-expired-password[={0|1}]`
- **Scope:** Global
- **Dynamic:** Yes
- **Type:** boolean
- **Default Value:** `OFF`
- **Introduced:** MariaDB 10.4.3

---

div_precision_increment

- **Description:** The precision of the result of the decimal division will be the larger than the precision of the dividend by that number. By default it's `4`, so `SELECT 2/15` would return 0.1333 and `SELECT 2.0/15` would return 0.13333. After setting div_precision_increment to `6`, for example, the same operation would return 0.133333 and 0.1333333 respectively.

  From MariaDB 10.1.46, MariaDB 10.2.33, MariaDB 10.3.24, MariaDB 10.4.14 and MariaDB 10.5.5, `div_precision_increment` is taken into account in intermediate calculations. Previous versions did not, and the results were dependent on the optimizer, and therefore unpredictable.

  In MariaDB 10.1.46, MariaDB 10.1.47, MariaDB 10.2.33, MariaDB 10.2.34, MariaDB 10.2.35, MariaDB 10.3.24, MariaDB 10.3.25, MariaDB 10.4.14, MariaDB 10.4.15, MariaDB 10.5.5 and MariaDB 10.5.6 only, the fix truncated decimal values after every division, resulting in lower precision in some cases for those versions only.

  From MariaDB 10.1.48, MariaDB 10.2.35, MariaDB 10.3.26, MariaDB 10.4.16 and MariaDB 10.5.7, a different fix was implemented. Instead of truncating decimal values after every division, they are instead truncated for comparison purposes only.

  For example

  Versions other than MariaDB 10.1.46, MariaDB 10.1.47, MariaDB 10.2.33, MariaDB 10.2.34, MariaDB 10.2.35, MariaDB 10.3.24, MariaDB 10.3.25, MariaDB 10.4.14, MariaDB 10.4.15, MariaDB 10.5.5 and MariaDB 10.5.6:

```
SELECT (55/23244*1000);
+-----------------+
| (55/23244*1000) |
+-----------------+
|          2.3662 |
+-----------------
```

  MariaDB 10.1.46, MariaDB 10.1.47, MariaDB 10.2.33, MariaDB 10.2.34, MariaDB 10.2.35, MariaDB 10.3.24, MariaDB 10.3.25, MariaDB 10.4.14, MariaDB 10.4.15, MariaDB 10.5.5 and MariaDB 10.5.6 only:

```
SELECT (55/23244*1000);
+-----------------+
| (55/23244*1000) |
+-----------------+
|          2.4000 |
+-----------------+
```

  This is because the intermediate result, `SELECT 55/23244` takes into account `div_precision_increment` and results were truncated after every division in those versions only.

- **Commandline:** `--div-precision-increment=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `4`
- **Range:** `0` to `30`

---

encrypt_tmp_disk_tables

- **Description:** Enables automatic encryption of all internal on-disk temporary tables that are created during query execution if `aria_used_for_temp_tables=ON` is set. See Data at Rest Encryption and Enabling Encryption for Internal On-disk Temporary Tables.
- **Commandline:** `--encrypt-tmp-disk-tables[={0|1}]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

encrypt_tmp_files

- **Description:** Enables automatic encryption of temporary files, such as those created for filesort operations, binary log file caches, etc. See Data at Rest Encryption.
- **Commandline:** `--encrypt-tmp-files[={0|1}]`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

encryption_algorithm

- **Description:** Which encryption algorithm to use for table encryption. `aes_cbc` is the recommended one. See Table and Tablespace Encryption.
- **Commandline:** `--encryption-algorithm=value`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `enum`
- **Default Value:** `none`
- **Valid Values:** `none`, `aes_ecb`, `aes_cbc`, `aes_ctr`
- **Introduced:** MariaDB 10.1.3
- **Removed:** MariaDB 10.1.4

---

enforce_storage_engine

- **Description:** Force the use of a particular storage engine for new tables. Used to avoid unwanted creation of tables using another engine. For example, setting to InnoDB will prevent any MyISAM tables from being created. If another engine is specified in a CREATE TABLE statement, the outcome depends on whether the `NO_ENGINE_SUBSTITUTION` SQL_MODE has been set or not. If set, the query will fail, while if not set, a warning will be returned and the table created according to the engine specified by this variable. The variable has a session scope, but is only modifiable by a user with the SUPER privilege.
- **Commandline:** None
- **Scope:** Session
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `none`

---

engine_condition_pushdown

- **Description:** Deprecated in MariaDB 5.5 and removed and replaced by the optimizer_switch `engine_condition_pushdown={on|off}` flag in MariaDB 10.0.. Specifies whether the engine condition pushdown optimization is enabled. Since MariaDB 10.1.1, engine condition pushdown is enabled for all engines that support it.
- **Commandline:** `--engine-condition-pushdown`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Deprecated:** MariaDB 5.5
- **Removed:** MariaDB 10.0

eq_range_index_dive_limit

- **Description:** Limit used for speeding up queries listed by long nested INs. The optimizer will use existing index statistics instead of doing index dives for equality ranges if the number of equality ranges for the index is larger than or equal to this number. If set to `0` (unlimited, the default), index dives are always used.
- **Commandline:** `--eq-range-index-dive-limit=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `200` (>= MariaDB 10.4.3), `0` (<= MariaDB 10.4.2)
- **Range:** `0` to `4294967295`

error_count

- **Description:** Read-only variable denoting the number of errors from the most recent statement in the current session that generated errors. See SHOW_ERRORS().
- **Scope:** Session
- **Dynamic:** Yes
- **Data Type:** `numeric`

event_scheduler

- **Description:** Status of the Event Scheduler. Can be set to `ON` or `OFF`, while `DISABLED` means it cannot be set at runtime. Setting the variable will cause a load of events if they were not loaded at startup.
- **Commandline:** `--event-scheduler[=value]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `enumeration`
- **Default Value:** `OFF`
- **Valid Values:** `ON` (or `1`), `OFF` (or `0`), `DISABLED`

expensive_subquery_limit

- **Description:** Number of rows to be examined for a query to be considered expensive, that is, maximum number of rows a subquery may examine in order to be executed during optimization and used for constant optimization.
- **Commandline:** `--expensive-subquery-limit=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `100`
- **Range:** `0` upwards

explicit_defaults_for_timestamp

- **Description:** This option causes CREATE TABLE to create all TIMESTAMP columns as NULL with the DEFAULT NULL attribute, Without this option, TIMESTAMP columns are NOT NULL and have implicit DEFAULT clauses.
- **Commandline:** `--explicit-defaults-for-timestamp=[={0|1}]`
- **Scope:**
  - Global, Session (>= MariaDB 10.8.4, MariaDB 10.7.5, MariaDB 10.6.9, MariaDB 10.5.17)
  - Global (<= MariaDB 10.8.3, MariaDB 10.7.4, MariaDB 10.6.8, MariaDB 10.5.16)
- **Dynamic:**
  - Yes (>= MariaDB 10.8.4, MariaDB 10.7.5, MariaDB 10.6.9, MariaDB 10.5.17)
  - No (<= MariaDB 10.8.3, MariaDB 10.7.4, MariaDB 10.6.8, MariaDB 10.5.16)
- **Data Type:** `boolean`
- **Default Value:** `ON` (>= MariaDB 10.10), `OFF` (<= MariaDB 10.9)

external_user

- **Description:** External user name set by the plugin used to authenticate the client. `NULL` if native MariaDB

authentication is used.
- **Scope:** Session
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** `NULL`

---

flush

- **Description:** Usually, MariaDB writes changes to disk after each SQL statement, and the operating system handles synchronizing (flushing) it to disk. If set to `ON`, the server will synchronize all changes to disk after each statement.
- **Commandline:** `--flush`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

flush_time

- **Description:** Interval in seconds that tables are closed to synchronize (flush) data to disk and free up resources. If set to 0, the default, there is no automatic synchronizing tables and closing of tables. This option should not be necessary on systems with sufficient resources.
- **Commandline:** `--flush_time=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`

---

foreign_key_checks

- **Description:** If set to 1 (the default) foreign key constraints (including ON UPDATE and ON DELETE behavior) InnoDB tables are checked, while if set to 0, they are not checked. `0` is not recommended for normal use, though it can be useful in situations where you know the data is consistent, but want to reload data in a different order from that that specified by parent/child relationships. Setting this variable to 1 does not retrospectively check for inconsistencies introduced while set to 0.
- **Commandline:** None
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `1`

---

ft_boolean_syntax

- **Description:** List of operators supported by an IN BOOLEAN MODE full-text search. If you wish to change, note that each character must be ASCII and non-alphanumeric, the full string must be 14 characters and the first or second character must be a space. Positions 10, 13 and 14 are reserved for future extensions. Also, no duplicates are permitted except for the phrase quoting characters in positions 11 and 12, which may be the same.
- **Commandline:** `--ft-boolean-syntax=name`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `+ -><()*:""&|`

---

ft_max_word_len

- **Description:** Maximum length for a word to be included in the MyISAM full-text index. If this variable is changed, the full-text index must be rebuilt in order for the new value to take effect. The quickest way to do this is by issuing a `REPAIR TABLE table_name QUICK` statement. See innodb_ft_max_token_size for the InnoDB equivalent.
- **Commandline:** `--ft-max-word-len=#`
- **Scope:** Global
- **Dynamic:** No

- **Data Type:** `numeric`
- **Default Value:** `84`
- **Minimum Value:** `10`

---

ft_min_word_len

- **Description:** Minimum length for a word to be included in the MyISAM full-text index. If this variable is changed, the full-text index must be rebuilt in order for the new value to take effect. The quickest way to do this is by issuing a `REPAIR TABLE table_name QUICK` statement. See innodb_ft_min_token_size for the InnoDB equivalent.
- **Commandline:** `--ft-min-word-len=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `4`
- **Minimum Value:** `1`

---

ft_query_expansion_limit

- **Description:** For full-text searches, denotes the numer of top matches when using WITH QUERY EXPANSION.
- **Commandline:** `--ft-query-expansion-limit=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `20`
- **Range:** `0` to `1000`

---

ft_stopword_file

- **Description:** File containing a list of stopwords for use in MyISAM full-text searches. Unless an absolute path is specified the file will be looked for in the data directory. The file is not parsed for comments, so all words found become stopwords. By default, a built-in list of words (built from `storage/myisam/ft_static.c file`) is used. Stopwords can be disabled by setting this variable to `''` (an empty string). If this variable is changed, the full-text index must be rebuilt. The quickest way to do this is by issuing a `REPAIR TABLE table_name QUICK` statement. See innodb_ft_server_stopword_table for the InnoDB equivalent.
- **Commandline:** `--ft-stopword-file=file_name`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `file name`
- **Default Value:** `(built-in)`

---

general_log

- **Description:** If set to 0, the default unless the --general-log option is used, the general query log is disabled, while if set to 1, the general query log is enabled. See log_output for how log files are written. If that variable is set to `NONE`, no logs will be written even if general_query_log is set to `1`.
- **Commandline:** `--general-log`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `0`

---

general_log_file

- **Description:** Name of the general query log file. If this is not specified, the name is taken from the log-basename setting or from your system hostname with `.log` as a suffix.
- **Commandline:** `--general-log-file=file_name`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `file name`
- **Default Value:** *`host_name`*`.log`

group_concat_max_len

- **Description:** Maximum length in bytes of the returned result for the functions GROUP_CONCAT(), JSON_OBJECTAGG and JSON_ARRAYAGG.
- **Commandline:** `--group-concat-max-len=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:**
  - `1048576` (1M)
- **Range:** `4` to `4294967295`

.

have_compress

- **Description:** If the zlib compression library is accessible to the server, this will be set to `YES`, otherwise it will be `NO`. The COMPRESS() and UNCOMPRESS() functions will only be available if set to `YES`.
- **Scope:** Global
- **Dynamic:** No

have_crypt

- **Description:** If the crypt() system call is available this variable will be set to `YES`, otherwise it will be set to `NO`. If set to `NO`, the ENCRYPT() function cannot be used.
- **Scope:** Global
- **Dynamic:** No

have_csv

- **Description:** If the server supports CSV tables, will be set to `YES`, otherwise will be set to `NO`. Removed in MariaDB 10.0, use the Information Schema PLUGINS table or SHOW ENGINES instead.
- **Scope:** Global
- **Dynamic:** No
- **Removed:** MariaDB 10.0

have_dynamic_loading

- **Description:** If the server supports dynamic loading of plugins, will be set to `YES`, otherwise will be set to `NO`.
- **Scope:** Global
- **Dynamic:** No

have_geometry

- **Description:** If the server supports spatial data types, will be set to `YES`, otherwise will be set to `NO`.
- **Scope:** Global
- **Dynamic:** No

have_ndbcluster

- **Description:** If the server supports NDBCluster (disabled in MariaDB).
- **Scope:** Global
- **Dynamic:** No
- **Removed:** MariaDB 10.0

have_partitioning

- **Description:** If the server supports partitioning, will be set to `YES`, unless the `--skip-partition` option is used, in which case will be set to `DISABLED`. Will be set to `NO` otherwise. Removed in MariaDB 10.0 - SHOW PLUGINS should be used instead.
- **Scope:** Global
- **Dynamic:** No
- **Removed:** MariaDB 10.0

---

have_profiling

- **Description:** If statement profiling is available, will be set to `YES`, otherwise will be set to `NO`. See SHOW PROFILES() and SHOW PROFILE().
- **Scope:** Global
- **Dynamic:** No

---

have_query_cache

- **Description:** If the server supports the query cache, will be set to `YES`, otherwise will be set to `NO`.
- **Scope:** Global
- **Dynamic:** No

---

have_rtree_keys

- **Description:** If RTREE indexes (used for spatial indexes) are available, will be set to `YES`, otherwise will be set to `NO`.
- **Scope:** Global
- **Dynamic:** No

---

have_symlink

- **Description:** This system variable can be used to determine whether the server supports symbolic links (note that it has no meaning on Windows).
  - If symbolic links are supported, then the value will be `YES`.
  - If symbolic links are not supported, then the value will be `NO`.
  - If symbolic links are disabled with the --symbolic-links option and the `skip` option prefix (i.e. --skip-symbolic-links), then the value will be `DISABLED`.
  - Symbolic link support is required for the INDEX DIRECTORY and DATA DIRECTORY table options.
- **Scope:** Global
- **Dynamic:** No

---

histogram_size

- **Description:** Number of bytes used for a histogram, or, from MariaDB 10.7 when histogram_type is set to `JSON_HB`, number of buckets. If set to 0, no histograms are created by ANALYZE.
- **Commandline:** `--histogram-size=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `254` (>= MariaDB 10.4.3), `0` (<= MariaDB 10.4.2)
- **Range:** `0` to `255`

---

histogram_type

- **Description:** Specifies the type of histograms created by ANALYZE.
  - `SINGLE_PREC_HB` - single precision height-balanced.
  - `DOUBLE_PREC_HB` - double precision height-balanced.
  - `JSON_HB` - JSON histograms (from MariaDB 10.7)
- **Commandline:** `--histogram-type=value`
- **Scope:** Global, Session
- **Dynamic:** Yes

- **Data Type:** `enumeration`
- **Default Value:**
  - `JSON_HB` (>= MariaDB 11.0)
  - `DOUBLE_PREC_HB` (<= MariaDB 10.11, >= MariaDB 10.4.3)
  - `SINGLE_PREC_HB` (<= MariaDB 10.4.2)
- **Valid Values:**
  - `SINGLE_PREC_HB`, `DOUBLE_PREC_HB` (<= MariaDB 10.6)
  - `SINGLE_PREC_HB`, `DOUBLE_PREC_HB`, `JSON_HB` (>= MariaDB 10.7)

---

host_cache_size

- **Description:** Number of host names that will be cached to avoid resolving. Setting to `0` disables the cache. Changing the value while the server is running causes an implicit FLUSH HOSTS, clearing the host cache and truncating the performance_schema.host_cache table. If you are connecting from a lot of different machines you should consider increasing.
- **Commandline:** `--host-cache-size=#`.
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `128`
- **Range:** `0` to `65536`

---

hostname

- **Description:** When the server starts, this variable is set to the server host name.
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`

---

identity

- **Description:** A synonym for last_insert_id variable.

---

idle_readonly_transaction_timeout

- **Description:** Time in seconds that the server waits for idle read-only transactions before killing the connection. If set to `0`, the default, connections are never killed. See also idle_transaction_timeout, idle_write_transaction_timeout and Transaction Timeouts.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `31536000`
- **Introduced:** MariaDB 10.3.0

---

idle_transaction_timeout

- **Description:** Time in seconds that the server waits for idle transactions before killing the connection. If set to `0`, the default, connections are never killed. See also idle_readonly_transaction_timeout, idle_write_transaction_timeout and Transaction Timeouts.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `31536000`
- **Introduced:** MariaDB 10.3.0

---

idle_write_transaction_timeout

- **Description:** Time in seconds that the server waits for idle read-write transactions before killing the connection. If set to `0`, the default, connections are never killed. See also idle_transaction_timeout, idle_readonly_transaction_timeout and Transaction Timeouts. Called `idle_readwrite_transaction_timeout` until MariaDB 10.3.2.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `31536000`
- **Introduced:** MariaDB 10.3.0

---

ignore_db_dirs

- **Description:** Tells the server that this directory can never be a database. That means two things - firstly it is ignored by the SHOW DATABASES command and INFORMATION_SCHEMA tables. And secondly, USE, CREATE DATABASE and SELECT statements will return an error if the database from the ignored list specified. Use this option several times if you need to ignore more than one directory. To make the list empty set the void value to the option as --ignore-db-dir=. If the option or configuration is specified multiple times, viewing this value will list the ignore directories separated by a period.
- **Commandline:** `--ignore-db-dirs=dir`.
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`

---

in_predicate_conversion_threshold

- **Description:** The minimum number of scalar elements in the value list of an IN predicate that triggers its conversion to an IN subquery. Set to 0 to disable the conversion. See Conversion of Big IN Predicates Into Subqueries.
- **Commandline:** `--in-predicate-conversion-threshold=#`
- **Scope:** Global, Session
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `1000`
- **Range:** `0` to `4294967295`
- **Introduced:** MariaDB 10.3.18 (previously debug builds only)

---

in_transaction

- **Description:** Session-only and read-only variable that is set to `1` if a transaction is in progress, `0` if not.
- **Commandline:** No
- **Scope:** Session
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `0`

---

init_connect

- **Description:** String containing one or more SQL statements, separated by semicolons, that will be executed by the server for each client connecting. If there's a syntax error in the one of the statements, the client will fail to connect. For this reason, the statements are not executed for users with the SUPER privilege or, from MariaDB 10.5.2, the CONNECTION ADMIN privilege, who can then still connect and correct the error. See also init_file.
- **Commandline:** `--init-connect=name`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`

---

init_file

- **Description:** Name of a file containing SQL statements that will be executed by the server on startup. Each statement should be on a new line, and end with a semicolon. See also init_connect.
- **Commandline:** `init-file=file_name`

- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `file name`

---

insert_id

- **Description:** Value to be used for the next statement inserting a new AUTO_INCREMENT value.
- **Scope:** Session
- **Dynamic:** Yes
- **Data Type:** `numeric`

---

interactive_timeout

- **Description:** Time in seconds that the server waits for an interactive connection (one that connects with the mysql_real_connect() CLIENT_INTERACTIVE option) to become active before closing it. See also wait_timeout.
- **Commandline:** `--interactive-timeout=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `28800`
- **Range: (Windows):** `1` **to** `2147483`
- **Range: (Other):** `1` **to** `31536000`

---

join_buffer_size

- **Description:** Minimum size in bytes of the buffer used for queries that cannot use an index, and instead perform a full table scan. Increase to get faster full joins when adding indexes is not possible, although be aware of memory issues, since joins will always allocate the minimum size. Best left low globally and set high in sessions that require large full joins. In 64-bit platforms, Windows truncates values above 4GB to 4GB with a warning. See also Block-Based Join Algorithms - Size of Join Buffers.
- **Commandline:** `--join-buffer-size=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `262144` (256kB)
- **Range (non-Windows):** `128` to `18446744073709547520`
- **Range (Windows):** `8228` to `18446744073709547520`

---

join_buffer_space_limit

- **Description:** Maximum size in bytes of the query buffer, By default 1024*128*10. See Block-based join algorithms.
- **Commandline:** `--join-buffer-space-limit=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `2097152`
- **Range:** `2048` to `18446744073709551615`

---

join_cache_level

- **Description:** Controls which of the eight block-based algorithms can be used for join operations. See Block-based join algorithms for more information.
  - 1 – flat (Block Nested Loop) BNL
  - 2 – incremental BNL
  - 3 – flat Block Nested Loop Hash (BNLH)
  - 4 – incremental BNLH
  - 5 – flat Batch Key Access (BKA)
  - 6 – incremental BKA
  - 7 – flat Batch Key Access Hash (BKAH)
  - 8 – incremental BKAH

- **Commandline:** `--join-cache-level=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `2`
- **Range:** `0` to `8`

---

keep_files_on_create

- **Description:** If a MyISAM table is created with no DATA DIRECTORY option, the .MYD file is stored in the database directory. When set to `0`, the default, if MariaDB finds another .MYD file in the database directory it will overwrite it. Setting this variable to `1` means that MariaDB will return an error instead, just as it usually does in the same situation outside of the database directory. The same applies for .MYI files and no INDEX DIRECTORY option. Deprecated in MariaDB 10.8.0.
- **Commandline:** `--keep-files-on-create=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Deprecated:** MariaDB 10.8.0

---

large_files_support

- **Description:** ON if the server if was compiled with large file support or not, else OFF
- **Scope:** Global
- **Dynamic:** No

---

large_page_size

- **Description:** Indicates the size of memory page if large page support (Linux only) is enabled. The page size is determined from the Hugepagesize setting in `/proc/meminfo`. See large_pages. Deprecated and unused in MariaDB 10.5.3 since multiple page size support was added.
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** Autosized (see description)
- **Deprecated:** MariaDB 10.5.3

---

large_pages

- **Description:** Indicates whether large page support (prior to MariaDB 10.5, Linux only, by now supported Windows and BSD distros, also called huge pages) is used. This is set with `--large-pages` or disabled with `--skip-large-pages`. Large pages are used for the innodb buffer pool and for online DDL (of size 3* innodb_sort_buffer_size (or 6 when encryption is used)). To use large pages, the Linux `sysctl` variable `kernel.shmmax` must be large than the llocation. Also the `sysctl` variable `vm.nr_hugepages` multipled by large-page) must be larger than the usage. The ulimit for locked memory must be sufficient to cover the amount used (`ulimit -l` and equalivent in /etc/security/limits.conf / or in systemd LimitMEMLOCK). If these operating system controls or insufficient free huge pages are available, the allocation of large pages will fall back to conventional memory allocation and a warning will appear in the logs. Only allocations of the default `Hugepagesize` currently occur (see `/proc/meminfo`).
- **Commandline:** `--large-pages`, `--skip-large-pages`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

last_insert_id

- **Description:** Contains the same value as that returned by LAST_INSERT_ID(). Note that setting this variable doen't update the value returned by the underlying function.

- **Scope:** Session
- **Dynamic:** Yes
- **Data Type:** `numeric`

---

lc_messages

- **Description:** This system variable can be specified as a locale name. The language of the associated locale will be used for error messages. See Server Locales for a list of supported locales and their associated languages.
  - This system variable is set to `en_US` by default, which means that error messages are in English by default.
  - If this system variable is set to a valid locale name, but the server can't find an error message file for the language associated with the locale, then the default language will be used instead.
  - This system variable is used along with the `lc_messages_dir` system variable to construct the path to the error messages file.
  - See Setting the Language for Error Messages for more information.
- **Commandline:** `--lc-messages=name`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `en_us`

---

lc_messages_dir

- **Description:** This system variable can be specified either as the path to the directory storing the server's error message files or as the path to the directory storing the specific language's error message file. See Server Locales for a list of available locales and their related languages.
  - The server initially tries to interpret the value of this system variable as a path to the directory storing the server's error message files. Therefore, it constructs the path to the language's error message file by concatenating the value of this system variable with the language name of the locale specified by the `lc_messages` system variable .
  - If the server does not find the error message file for the language, then it tries to interpret the value of this system variable as a direct path to the directory storing the specific language's error message file.
  - See Setting the Language for Error Messages for more information.
- **Commandline:** `--lc-messages-dir=path`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `directory name`

---

lc_time_names

- **Description:** The locale that determines the language used for the date and time functions DAYNAME(), MONTHNAME() and DATE_FORMAT(). Locale names are language and region subtags, for example 'en_ZA' (English - South Africa) or 'es_US: Spanish - United States'. The default is always 'en-US' regardless of the system's locale setting. See server locale for a full list of supported locales.
- **Commandline:** `--lc-time-names=name`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `en_US`

---

license

- **Description:** Server license, for example `GPL` .
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`

---

local_infile

- **Description:** If set to `1` , LOCAL is supported for LOAD DATA INFILE statements. If set to `0` , usually for security reasons, attempts to perform a LOAD DATA LOCAL will fail with an error message.

- **Commandline:** `--local-infile=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `ON`

---

lock_wait_timeout

- **Description:** Timeout in seconds for attempts to acquire metadata locks. Statements using metadata locks include FLUSH TABLES WITH READ LOCK, LOCK TABLES, HANDLER and DML and DDL operations on tables, stored procedures and functions, and views. The timeout is separate for each attempt, of which there may be multiple in a single statement. `0` (from MariaDB 10.3.0) means no wait. See WAIT and NOWAIT.
- **Commandline:** `--lock-wait-timeout=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:**
  - `86400` (1 day)
- **Range:**
  - `0` to `31536000`

---

locked_in_memory

- **Description:** Indicates whether --memlock was used to lock mariadbd in memory.
- **Commandline:** `--memlock`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

log

- **Description:** Deprecated and removed in MariaDB 10.0, use general_log instead.
- **Commandline:** `-l [filename]` or `--log[=filename]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `OFF`
- **Removed:** MariaDB 10.0

---

log_disabled_statements

- **Description:** If set, the specified type of statements (slave and/or stored procedure statements) will not be logged to the general log. Multiple values are comma-separated, without spaces.
- **Commandline:** `--log-disabled_statements=value`
- **Scope:** Global, Session
- **Dynamic:** No
- **Data Type:** `set`
- **Default Value:** `sp`
- **Valid Values:** `slave` and/or `sp`, or empty string for none
- **Introduced:** MariaDB 10.3.1

---

log_error

- **Description:** Specifies the name of the error log. If --console is specified later in the configuration (Windows only) or this option isn't specified, errors will be logged to stderr. If no name is provided, errors will still be logged to `hostname.err` in the `datadir` directory by default. If a configuration file sets `--log-error`, one can reset it with `--skip-log-error` (useful to override a system wide configuration file). MariaDB always writes its error log, but the destination is configurable. See error log for details.
- **Commandline:** `--log-error[=name]`, `--skip-log-error`

- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `file name`
- **Default Value:** (empty string)

---

log_output

- **Description:** How the output for the general query log and the slow query log is stored. By default written to file (`FILE`), it can also be stored in the general_log and slow_log tables in the mysql database (`TABLE`), or not stored at all (`NONE`). More than one option can be chosen at the same time, with `NONE` taking precedence if present. Logs will not be written if logging is not enabled. See Writing logs into tables, and the slow_query_log and general_log server system variables.
- **Commandline:** `--log-output=name`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `set`
- **Default Value:** `FILE`
- **Valid Values:** `TABLE` , `FILE` or `NONE`

---

log_queries_not_using_indexes

- **Description:** Queries that don't use an index, or that perform a full index scan where the index doesn't limit the number of rows, will be logged to the slow query log (regardless of time taken). The slow query log needs to be enabled for this to have an effect. Mapped to `log_slow_filter='not_using_index'` from MariaDB 10.3.1.
- **Commandline:** `--log-queries-not-using-indexes`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

log_slow_admin_statements

- **Description:** Log slow OPTIMIZE, ANALYZE, ALTER and other administrative statements to the slow log if it is open. See also log_slow_disabled_statements and log_slow_filter. Deprecated, use log_slow_filter without `admin` .
- **Commandline:** `--log-slow-admin-statements`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:**
  - `ON`
- **Deprecated:** MariaDB 11.0.1

---

log_slow_disabled_statements

- **Description:** If set, the specified type of statements will not be logged to the slow query log. See also log_slow_admin_statements and log_slow_filter.
- **Commandline:** `--log-slow-disabled_statements=value`
- **Scope:** Global, Session
- **Dynamic:** No
- **Data Type:** `set`
- **Default Value:** `sp`
- **Valid Vales:** `admin` , `call` , `slave` and/or `sp`
- **Introduced:** MariaDB 10.3.1

---

log_slow_filter

- **Description:** Comma-delimited string (without spaces) containing one or more settings for filtering what is logged to the slow query log. If a query matches one of the types listed in the filter, and takes longer than long_query_time, it will be logged(except for 'not_using_index' which is always logged if enabled, regardless of the time). Sets log-slow-admin-statements to ON. See also log_slow_disabled_statements.

    - `admin` log administrative queries (create, optimize, drop etc...)
    - `filesort` logs queries that use a filesort.
    - `filesort_on_disk` logs queries that perform a a filesort on disk.
    - `filesort_priority_queue` (from MariaDB 10.3.2)
    - `full_join` logs queries that perform a join without indexes.
    - `full_scan` logs queries that perform full table scans.
    - `not_using_index` logs queries that don't use an index, or that perform a full index scan where the index doesn't limit the number of rows. Disregards long_query_time, unlike other options. log_queries_not_using_indexes maps to this option. From MariaDB 10.3.1.
    - `query_cache` log queries that are resolved by the query cache.
    - `query_cache_miss` logs queries that are not found in the query cache.
    - `tmp_table` logs queries that create an implicit temporary table.
    - `tmp_table_on_disk` logs queries that create a temporary table on disk.
- **Commandline:** `log-slow-filter=value1[,value2...]`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `enumeration`
- **Default Value:**
    - `admin`, `filesort`, `filesort_on_disk`, `full_join`, `full_scan`, `query_cache`, `query_cache_miss`, `tmp_table`, `tmp_table_on_disk` (<= MariaDB 10.3.0)
    - `admin`, `filesort`, `filesort_on_disk`, `filesort_priority_queue`, `full_join`, `full_scan`, `query_cache`, `query_cache_miss`, `tmp_table`, `tmp_table_on_disk` (>= MariaDB 10.3.1)
- **Valid Values:** `admin`, `filesort`, `filesort_on_disk`, `filesort_priority_queue`, `full_join`, `full_scan`, `query_cache`, `query_cache_miss`, `tmp_table`, `tmp_table_on_disk`

---

log_slow_min_examined_row_limit

- **Description:** If a query examines more than this number of rows, it is logged to the slow query log. If set to `0`, the default, no row limit is used. `min_examined_row_limit` is an alias.
- **Commandline:** `--log-slow-min-examined-row-limit=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0-4294967295`
- **Introduced:** MariaDB 10.11.0

---

log_slow_queries

- **Description:** Deprecated and removed in MariaDB 10.0, use slow_query_log instead.
- **Commandline:** `--log-slow-queries[=name]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Removed:** MariaDB 10.0

---

log_slow_query

- **Description:** If set to 0, the default unless the --slow-query-log option is used, the slow query log is disabled, while if set to 1 (both global and session variables), the slow query log is enabled. Named slow_query_log before MariaDB 10.11.0, which is now an alias.
- **Commandline:** `--slow-query-log`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `0`
- **Introduced:** MariaDB 10.11.0
- **See also:** See log_output to see how log files are written. If that variable is set to `NONE`, no logs will be written even if log_slow_query is set to `1`.

---

log_slow_query_file

- **Description:** Name of the slow query log file. Before MariaDB 10.11, was named slow_query_log_file. This was named `log_slow_query_file_name` in the MariaDB 10.11.0 preview release.
- **Commandline:** `--log-slow-query-file=file_name`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `file name`
- **Default Value:** *`host_name`*`-slow.log`
- **Introduced:** MariaDB 10.11.0

---

log_slow_query_time

- **Description:** If a query takes longer than this many seconds to execute (microseconds can be specified too), the Slow_queries status variable is incremented and, if enabled, the query is logged to the slow query log. Before MariaDB 10.11, was named long_query_time.
- **Commandline:** `--log-slow-query-time=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `10.000000`
- **Range:** `0` to `31536000`
- **Introduced:** MariaDB 10.11.0

---

log_slow_rate_limit

- **Description:** The slow query log will log every this many queries. The default is `1`, or every query, while setting it to `20` would log every 20 queries, or five percent. Aims to reduce I/O usage and excessively large slow query logs. See also Slow Query Log Extended Statistics.
- **Commandline:** `log-slow-rate-limit=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1`
- **Range:** `1` upwards

---

log_slow_verbosity

- **Description:** Controls information to be added to the slow query log. Options are added in a comma-delimited string. See also Slow Query Log Extended Statistics. log_slow_verbosity is not supported when log_output='TABLE'.
  - `query_plan` logs query execution plan information
  - `innodb` Alias to `engine` (from MariaDB 10.6.15 and MariaDB 10.11.5), previously ignored.
  - `explain` prints EXPLAIN output in the slow query log. See EXPLAIN in the Slow Query Log.
  - `engine` Logs engine statistics (from MariaDB 10.6.15 and MariaDB 10.11.5).
  - `warnings` Print all errors, warnings and notes for the statement to the slow query log. (from MariaDB 10.6.16).
  - `all` Enables all above options (From MariaDB 10.6.16)
  - `full` Enables all above options.
- **Commandline:** `log-slow-verbosity=value1[,value2...]`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `enumeration`
- **Default Value:** (Empty)
- **Valid Values:**
  - >= MariaDB 10.6.16, MariaDB 10.11.6: (Empty), `query_plan`, `innodb`, `explain`, `engine`, `warnings`, `all`, `full`
  - >= MariaDB 10.6.15, MariaDB 10.11.5: (Empty), `query_plan`, `innodb`, `explain`, `engine`, `full`
  - <= MariaDB 10.6.14, MariaDB 10.11.4: (Empty), `query_plan`, `innodb`, `explain`

---

log_slow_max_warnings

- **Description:** Max numbers of warnings printed to slow query log per statement
- **Commandline:** `log-slow-max-warnings=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `10`
- **Range:** `0` to `1000`
- **Introduced:** MariaDB 10.6.16

log_tc_size

- **Description:** Defines the size in bytes of the memory-mapped file-based transaction coordinator log, which is only used if the binary log is disabled. If you have two or more XA-capable storage engines enabled, then a transaction coordinator log must be available. This size is defined in multiples of 4096. See Transaction Coordinator Log for more information. Also see the `--log-tc` server option and the `--tc-heuristic-recover` option.
- **Commandline:** `log-tc-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `24576`
- **Range:** `12288` to `18446744073709551615`

---

log_warnings

- **Description:** Determines which additional warnings are logged. Setting to `0` disables additional warning logging. Note that this does not prevent all warnings, there is a core set of warnings that will always be written to the error log. The additional warnings are as follows:
  - log_warnings >= 1
    - Event scheduler information.
    - System signals
    - Wrong usage of `--user`
    - Failed setrlimit() and mlockall()
    - Changed limits
    - Wrong values of lower_case_table_names and stack_size
    - Wrong values for command line options
    - Start log position and some master information when starting slaves
    - Slave reconnects
    - Killed slaves
    - Error reading relay logs
    - Unsafe statements for statement-based replication. If this warning occurs frequently, it is throttled to prevent flooding the log.
    - Disabled plugins that one tried to enable or use.
    - UDF files that didn't include the required init functions.
    - DNS lookup failures.
  - log_warnings >= 2
    - Access denied errors.
    - Connections aborted or closed due to errors or timeouts.
    - Table handler errors
    - Messages related to the files used to persist replication state:
      - Either the default `master.info` file or the file that is configured by the `master_info_file` option.
      - Either the default `relay-log.info` file or the file that is configured by the `relay_log_info_file` system variable.
    - Information about a master's binary log dump thread.
  - log_warnings >= 3
    - All errors and warnings during MyISAM repair and auto recover.
    - Information about old-style language options.
    - Information about progress of InnoDB online DDL.
  - log_warnings >=4
    - Connections aborted due to "Too many connections" errors.
    - Connections closed normally.
    - Connections aborted due to `KILL`.
    - Connections closed due to released connections, such as when `completion_type` is set to `RELEASE`.
  - log_warnings >=9
    - Information about initializing plugins.

- **Commandline:** `-W [level]` or `--log-warnings[=level]`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:**
  - `2`
- **Range:** `0` to `4294967295`

---

long_query_time

- **Description:** If a query takes longer than this many seconds to execute (microseconds can be specified too), the Slow_queries status variable is incremented and, if enabled, the query is logged to the slow query log. From MariaDB 10.11.0, this is an alias for log_slow_query_time.
- **Commandline:** `--long-query-time=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `10.000000`
- **Range:** `0` upwards

---

low_priority_updates

- **Description:** If set to 1 (0 is the default), for storage engines that use only table-level locking (Aria, MyISAM, MEMORY and MERGE), all INSERTs, UPDATEs, DELETEs and LOCK TABLE WRITEs will wait until there are no more SELECTs or LOCK TABLE READs pending on the relevant tables. Set this to 1 if reads are prioritized over writes.
  - In MariaDB 5.5 and earlier, `sql_low_priority_updates` is a synonym.
- **Commandline:** `--low-priority-updates`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `0`

---

lower_case_file_system

- **Description:** Read-only variable describing whether the file system is case-sensitive. If set to `OFF`, file names are case-sensitive. If set to `ON`, they are not case-sensitive.
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `##`

---

lower_case_table_names

- **Description:** If set to `0` (the default on Unix-based systems), table names and aliases and database names are compared in a case-sensitive manner. If set to `1` (the default on Windows), names are stored in lowercase and not compared in a case-sensitive manner. If set to `2` (the default on Mac OS X), names are stored as declared, but compared in lowercase. This system variable's value cannot be changed after the datadir has been initialized. lower_case_table_names is set when a MariaDB instance starts, and it remains constant afterwards.
- **Commandline:** `--lower-case-table-names[=#]`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `0` (Unix), `1` (Windows), `2` (Mac OS X)
- **Range:** `0` to `2`

---

max_allowed_packet

- **Description:** Maximum size in bytes of a packet or a generated/intermediate string. The packet message buffer is

initialized with the value from net_buffer_length, but can grow up to max_allowed_packet bytes. Set as large as the largest BLOB, in multiples of 1024. If this value is changed, it should be changed on the client side as well. See slave_max_allowed_packet for a specific limit for replication purposes.

- **Commandline:** `--max-allowed-packet=#`
- **Scope:** Global, Session
- **Dynamic:** Yes (Global), No (Session)
- **Data Type:** `numeric`
- **Default Value:**
  - `16777216` (16M)
  - `1073741824` (1GB) (client-side)
- **Range:** `1024` to `1073741824`

---

max_connect_errors

- **Description:** Limit to the number of successive failed connects from a host before the host is blocked from making further connections. The count for a host is reset to zero if they successfully connect. To unblock, flush the host cache with a FLUSH HOSTS statement or mariadb-admin flush-hosts. The performance_schema.host_cache table contains the status of the current hosts.
- **Commandline:** `--max-connect-errors=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `100`
- **Range:** `1` to `4294967295`

---

max_connections

- **Description:** The maximum number of simultaneous client connections. See also Handling Too Many Connections. Note that this value affects the number of file descriptors required on the operating system. Minimum was changed from `1` to `10` to avoid possible unexpected results for the user (MDEV-18252). Note that MariaDB always has one reserved connection for a superuser. Additionally it can listen on a separate port, so will be available even when the max_connections limit is reached.
- **Commandline:** `--max-connections=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `151`
- **Range:** `10` to `100000`

---

max_delayed_threads

- **Description:** Limits to the number of INSERT DELAYED threads. Once this limit is reached, the insert is handled as if there was no DELAYED attribute. If set to `0`, DELAYED is ignored entirely. The session value can only be set to `0` or to the same as the global value.
- **Commandline:** `--max-delayed-threads=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `20`
- **Range:** `0` to `16384`

---

max_digest_length

- **Description:** Maximum length considered for computing a statement digest, such as used by the Performance Schema and query rewrite plugins. Statements that differ after this many bytes produce the same digest, and are aggregated for statistics purposes. The variable is allocated per session. Increasing will allow longer statements to be distinguished from each other, but increase memory use, while decreasing will reduce memory use, but more statements may become indistinguishable.
- **Commandline:** `--max-digest-length=#`
- **Scope:** Global,
- **Dynamic:** No

- **Data Type:** `numeric`
- **Default Value:** `1024`
- **Range:** `0` to `1048576`

---

max_error_count

- **Description:** Specifies the maximum number of messages stored for display by SHOW ERRORS and SHOW WARNINGS statements.
- **Commandline:** `--max-error-count=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `64`
- **Range:** `0` to `65535`

---

max_heap_table_size

- **Description:** Maximum size in bytes for user-created MEMORY tables. Setting the variable while the server is active has no effect on existing tables unless they are recreated or altered. The smaller of max_heap_table_size and tmp_table_size also limits internal in-memory tables. When the maximum size is reached, any further attempts to insert data will receive a "table ... is full" error. Temporary tables created with CREATE TEMPORARY will not be converted to Aria, as occurs with internal temporary tables, but will also receive a table full error.
- **Commandline:** `--max-heap-table-size=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `16777216`
- **Range :** `16384` to `4294966272`

---

max_insert_delayed_threads

- **Description:** Synonym for max_delayed_threads.

---

max_join_size

- **Description:** Statements will not be performed if they are likely to need to examine more than this number of rows, row combinations or do more disk seeks. Can prevent poorly-formatted queries from taking server resources. Changing this value to anything other the default will reset sql_big_selects to 0. If sql_big_selects is set again, max_join_size will be ignored. This limit is also ignored if the query result is sitting in the query cache. Previously named sql_max_join_size, which is still a synonym.
- **Commandline:** `--max-join-size=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `18446744073709551615`
- **Range:** `1` to `18446744073709551615`

---

max_length_for_sort_data

- **Description:** Used to decide which algorithm to choose when sorting rows. If the total size of the column data, not including columns that are part of the sort, is less than `max_length_for_sort_data` , then we add these to the sort key. This can speed up the sort as we don't have to re-read the same row again later. Setting the value too high can slow things down as there will be a higher disk activity for doing the sort.
- **Commandline:** `--max-length-for-sort-data=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1024`
- **Range:** `4` to `8388608`

---

max_long_data_size

- **Description:** Maximum size for parameter values sent with mysql_stmt_send_long_data(). If not set, will default to the value of max_allowed_packet. Deprecated in MariaDB 5.5 and removed in MariaDB 10.5.0; use max_allowed_packet instead.
- **Commandline:** `--max-long-data-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:**
  - `16777216` (16M)
- **Range:** `1024` to `4294967295`
- **Deprecated:** MariaDB 5.5
- **Removed:** MariaDB 10.5.0

---

max_password_errors

- **Description:** The maximum permitted number of failed connection attempts due to an invalid password before a user is blocked from further connections. FLUSH_PRIVILEGES will permit the user to connect again. This limit is ignored for users with the SUPER privilege or, from MariaDB 10.5.2, the CONNECTION ADMIN privilege. The maximum also doesn't apply to users with a hostname of localhost, 127.0.0.1 or ::1.
- **Commandline:** `--max-password-errors=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `4294967295`
- **Range:** `1` to `4294967295`
- **Introduced:** MariaDB 10.4.2

---

max_prepared_stmt_count

- **Description:** Maximum number of prepared statements on the server. Can help prevent certain forms of denial-of-service attacks. If set to `0`, no prepared statements are permitted on the server.
- **Commandline:** `--max-prepared-stmt-count=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `16382`
- **Range:** `0` to `4294967295` (>= MariaDB 10.3.6), `0` to `1048576` (<= MariaDB 10.3.5)

---

max_recursive_iterations

- **Description:** Maximum number of iterations when executing recursive queries, used to prevent infinite loops in recursive CTEs.
- **Commandline:** `--max-recursive-iterations=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1000` (>= MariaDB 10.6.0), `4294967295` (<= MariaDB 10.5)
- **Range:** `0` to `4294967295`

---

max_rowid_filter_size

- **Description:** The maximum size of the container of a rowid filter.
- **Commandline:** `--max-rowid-filter-size=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `131072`
- **Range:** `1024` to `18446744073709551615`
- **Introduced:** MariaDB 10.4.3

max_seeks_for_key

- **Description:** The optimizer assumes that the number specified here is the most key seeks required when searching with an index, regardless of the actual index cardinality. If this value is set lower than its default and maximum, indexes will tend to be preferred over table scans.
- **Commandline:** `--max-seeks-for-key=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `4294967295`
- **Range:** `1` to `4294967295`

max_session_mem_used

- **Description:** Amount of memory a single user session is allowed to allocate. This limits the value of the session variable Memory_used.
- **Commandline:** `--max-session-mem-used=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `9223372036854775807` (8192 PB)
- **Range:** `8192` to `18446744073709551615`

max_sort_length

- **Description:** Maximum size in bytes used for sorting data values - anything exceeding this is ignored. The server uses only the first `max_sort_length` bytes of each value and ignores the rest. Increasing this may require sort_buffer_size to be increased (especially if ER_OUT_OF_SORTMEMORY errors start appearing).
- **Commandline:** `--max-sort-length=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1024`
- **Range:**
  - `4` to `8388608` (<= MariaDB 10.4.13, MariaDB 10.5.3)
  - `8` to `8388608` (>= MariaDB 10.4.14, MariaDB 10.5.4)

max_sp_recursion_depth

- **Description:** Permitted number of recursive calls for a stored procedure. `0`, the default, no recursion is permitted. Increasing this value increases the thread stack requirements, so you may need to increase thread_stack as well. This limit doesn't apply to stored functions.
- **Commandline:** `--max-sp-recursion-depth[=#]`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `255`

max_statement_time

- **Description:** Maximum time in seconds that a query can execute before being aborted. This includes all queries, not just SELECT statements, but excludes statements in stored procedures. If set to 0, no limit is applied. See Aborting statements that take longer than a certain time to execute for details and limitations. Useful when combined with SET STATEMENT for limiting the execution times of individual queries. Replicas are not affected by this variable, however, from MariaDB 10.10, there's slave_max_statement_time that sets the limit to abort queries on a replica.
- **Commandline:** `--max-statement-time[=#]`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`

- **Default Value:** `0.000000`
- **Range:** `0` to `31536000`

---

max_tmp_tables

- **Description:** Unused.
- **Removed:** MariaDB 11.3.0

---

max_user_connections

- **Description:** Maximum simultaneous connections permitted for each user account. When set to `0`, there is no per user limit. Setting it to `-1` stops users without the SUPER privilege or, from MariaDB 10.5.2, the CONNECTION ADMIN privilege, from connecting to the server. The session variable is always read-only and only privileged users can modify user limits. The session variable defaults to the global `max_user_connections` variable, unless the user's specific `MAX_USER_CONNECTIONS` resource option is non-zero. When both global variable and the user resource option are set, the user's MAX_USER_CONNECTIONS is used. Note: This variable does not affect users with the SUPER privilege or, from MariaDB 10.5.2, the CONNECTION ADMIN privilege.
- **Commandline:** `--max-user-connections=#`
- **Scope:** Global, Session
- **Dynamic:** Yes, (except when globally set to `0` or `-1`)
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `-1` to `4294967295`

---

max_write_lock_count

- **Description:** Read lock requests will be permitted for processing after this many write locks. Applies only to storage engines that use table level locks (thr_lock), so no effect with InnoDB or Archive.
- **Commandline:** `--max-write-lock-count=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `4294967295`
- **Range:** `0-4294967295`

---

metadata_locks_cache_size

- **Description:** Size of the metadata locks cache, used for reducing the need to create and destroy synchronization objects. It is particularly helpful on systems where this process is inefficient, such as Windows XP.
- **Commandline:** `--metadata-locks-cache-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `1024`
- **Range:** `1` to `1048576`

---

metadata_locks_hash_instances

- **Description:** Number of hashes used by the set of metadata locks. The metadata locks are partitioned into separate hashes in order to reduce contention.
- **Commandline:** `--metadata-locks-hash-instances=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `8`
- **Range:** `1` to `1024`

---

min_examined_row_limit

- **Description:** If a query examines more than this number of rows, it is logged to the slow query log. If set to `0`, the default, no row limit is used. From MariaDB 10.11.0, this is an alias for log_slow_min_examined_row_limit.
- **Commandline:** `--min-examined-row-limit=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0-4294967295`

---

mrr_buffer_size

- **Description:** Size of buffer to use when using multi-range read with range access. See Multi Range Read optimization for more information.
- **Commandline:** `--mrr-buffer-size=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `262144`
- **Range** `8192` to `2147483648`

---

multi_range_count

- **Description:** Ignored. Use mrr_buffer_size instead.
- **Commandline:** `--multi-range-count=#`
- **Default Value:** `256`
- **Removed:** MariaDB 10.5.1

---

mysql56_temporal_format

- **Description:** If set (the default), MariaDB uses the MySQL 5.6 low level formats for TIME, DATETIME and TIMESTAMP instead of the MariaDB 5.3 version. The version MySQL introduced in 5.6 requires more storage, but potentially allows negative dates and has some advantages in replication. There should be no reason to revert to the old MariaDB 5.3 microsecond format. See also MDEV-10723.
- **Commandline:** `--mysql56-temporal-format`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `ON`

---

named_pipe

- **Description:** On Windows systems, determines whether connections over named pipes are permitted.
- **Commandline:** `--named-pipe`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

net_buffer_length

- **Description:** The starting size, in bytes, for the connection and thread buffers for each client thread. The size can grow to max_allowed_packet. This variable's session value is read-only. Can be set to the expected length of client statements if memory is a limitation.
- **Commandline:** `--net-buffer-length=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `16384`
- **Range:** `1024` to `1048576`

---

net_read_timeout

- **Description:** Time in seconds the server will wait for a client connection to send more data before aborting the read. See also net_write_timeout and slave_net_timeout
- **Commandline:** `--net-read-timeout=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `30`
- **Range:** `1` upwards

---

net_retry_count

- **Description:** Permit this many retries before aborting when attempting to read or write on a communication port. On FreeBSD systems should be set higher as threads are sent internal interrupts..
- **Commandline:** `--net-retry-count=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `10`
- **Range:** `1` to `4294967295`

---

net_write_timeout

- **Description:** Time in seconds to wait on writing a block to a connection before aborting the write. See also net_read_timeout and slave_net_timeout.
- **Commandline:** `--net-write-timeout=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `60`
- **Range:** `1` upwards

---

note_verbosity

- **Description:** Verbosity level for note-warnings given to the user. Options are added in a comma-delimited string, except for `all`, which sets all options. See also Notes when an index cannot be used. Be aware that if the old sql_notes variable is 0, one will not get any notes. Setting `note_verbosity` to "" is the recommended way to disable notes.
  - `basic` All old notes.
  - `unusable_keys` Give warnings for unusable keys for SELECT, DELETE and UPDATE.
  - `explain` Give warnings for unusable keys for EXPLAIN.
  - `all` Enables all above options. This has to be given alone.
- **Commandline:** `note-verbosity=value1[,value2...]`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `enumeration`
- **Default Value:** `basic,explain`
- **Valid Values:** `basic,explain,unusable_keys` or `all`.
- **Introduced:** MariaDB 10.6.16

old

- **Description:** Disabled by default, enabling it reverts index hints to those used before MySQL 5.1.17. Enabling may lead to replication errors. Deprecated and replaced by old_mode from MariaDB 10.9.
- **Commandline:** `--old`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Deprecated:** MariaDB 10.9

---

old_alter_table

- **Description:** From MariaDB 10.3.7, an alias for alter_algorithm. Prior to that, if set to `1` (`0` is default), MariaDB reverts to the non-optimized, pre-MySQL 5.1, method of processing ALTER TABLE statements. A temporary table is created, the data is copied over, and then the temporary table is renamed to the original.
- **Commandline:** `--old-alter-table`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `enumerated` (>=MariaDB 10.3.7), `boolean` (<= MariaDB 10.3.6)
- **Default Value:** See alter_algorithm (>= MariaDB 10.3.7), `0` (<= MariaDB 10.3.6)
- **Valid Values:** See alter_algorithm for the full list.
- **Deprecated:** MariaDB 10.3.7 (superceded by alter_algorithm)
- **Removed:** MariaDB 11.2.0

---

old_mode

- **Description:** Used for getting MariaDB to emulate behavior from an old version of MySQL or MariaDB. See OLD Mode. Fully replaces the old variable from MariaDB 10.9.
- **Commandline:** `--old-mode`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `UTF8_IS_UTF8MB3` (>= MariaDB 10.6) `(empty string)` (<= MariaDB 10.5)
- **Valid Values:** See OLD Mode for the full list.

---

old_passwords

- **Description:** If set to `1` (`0` is default), MariaDB reverts to using the `mysql_old_password` authentication plugin by default for newly created users and passwords, instead of the `mysql_native_password` authentication plugin.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

open_files_limit

- **Description:** The number of file descriptors available to MariaDB. If you are getting the `Too many open files` error, then you should increase this limit. If set to 0, then MariaDB will calculate a limit based on the following:

  MAX(max_connections*5, max_connections +table_open_cache*2)

  MariaDB sets the limit with `setrlimit`. MariaDB cannot set this to exceed the hard limit imposed by the operating system. Therefore, you may also need to change the hard limit. There are a few ways to do so.
  - If you are using `mysqld_safe` to start `mariadbd`, then see the instructions at mysqld_safe: Configuring the Open Files Limit.
  - If you are using `systemd` to start `mysqld`, then see the instructions at systemd: Configuring the Open Files Limit.
  - Otherwise, you can change the hard limit for the `mysql` user account by modifying `/etc/security/limits.conf`. See Configuring Linux for MariaDB: Configuring the Open Files Limit for more details.
- **Commandline:** `--open-files-limit=count`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** Autosized (see description)
- **Range:** `0` to `4294967295`

---

optimizer_extra_pruning_depth

- **Description:**If the optimizer needs to enumerate a join prefix of this size or larger, then it will try aggressively prune away the search space.

- **Commandline:** `--optimizer-extra-pruning-depth1[=#]`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `8`
- **Range:** `0` to `62`
- **Introduced:** MariaDB 10.10.1

---

optimizer_max_sel_args

- **Description:** The maximum number of SEL_ARG objects created when optimizing a range. If more objects would be needed, the range will not be used by the optimizer.
- **Commandline:** `--optimizer-max-sel-args=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `16000`
- **Range:** `0` to `4294967295`
- **Introduced:** MariaDB 10.6.16

---

optimizer_max_sel_arg_weight

- **Description:** The maximum weight of the SEL_ARG graph. Set to 0 for no limit.
- **Commandline:** `--optimizer-max-sel-arg-weight=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `32000`
- **Range:** `0` to `18446744073709551615`
- **Introduced:** MariaDB 10.5.9

---

optimizer_prune_level

- **Description:**Controls the heuristic(s) applied during query optimization to prune less-promising partial plans from the optimizer search space.
  - `0` : heuristics are disabled and an exhaustive search is performed
  - `1` : the optimizer will use heuristics to prune less-promising partial plans from the optimizer search space
  - `2` : tables using EQ_REF will be joined together as 'one entity' and the different combinations of these tables will not be considered (from MariaDB 10.10)
- **Commandline:** `--optimizer-prune-level[=#]`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `2` (>= MariaDB 10.10), `1` (<= MariaDB 10.9)

---

optimizer_search_depth

- **Description:** Maximum search depth by the query optimizer. Smaller values lead to less time spent on execution plans, but potentially less optimal results. If set to `0`, MariaDB will automatically choose a reasonable value. Since the better results from more optimal planning usually offset the longer time spent on planning, this is set as high as possible by default. `63` is a valid value, but its effects (switching to the original find_best search) are deprecated.
- **Commandline:** `--optimizer-search-depth[=#]`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `62`
- **Range:** `0` to `63`

---

optimizer_selectivity_sampling_limit

- **Description:** Controls number of record samples to check condition selectivity. Only used if `optimizer_use_condition_selectivity > 4`.
- **Commandline:** `optimizer-selectivity-sampling-limit[=#]`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `100`
- **Range:** `10` upwards

---

optimizer_switch

- **Description:** A series of flags for controlling the query optimizer. See Optimizer Switch for defaults, and a comparison to MySQL.
- **Commandline:** `--optimizer-switch=value`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `string`
- **Valid Values:**
  - `condition_pushdown_for_derived={on|off}`
  - `condition_pushdown_for_subquery={on|off}` (>=MariaDB 10.4.0)
  - `condition_pushdown_from_having={on|off}` (>=MariaDB 10.4.3)
  - `default` - set all optimizations to their default values.
  - `derived_merge={on|off}` - see Derived table merge optimization
  - `derived_with_keys={on|off}` - see Derived table with key optimization
  - `engine_condition_pushdown={on|off}` . Deprecated in MariaDB 10.1.1 as engine condition pushdown is now automatically enabled for all engines that support it.
  - `exists_to_in={on|off}` - see EXISTS-to-IN optimization
  - `extended_keys={on|off}` - see Extended Keys
  - `firstmatch={on|off}` - see First Match Strategy
  - `hash_join_cardinality={on|off}` - see hash_join_cardinality-optimizer_switch-flag (>= MariaDB 11.0.2, MariaDB 10.11.3, MariaDB 10.6.13)
  - `index_condition_pushdown={on|off}` - see Index Condition Pushdown
  - `index_merge={on|off}`
  - `index_merge_intersection={on|off}`
  - `index_merge_sort_intersection={on|off}` - more details
  - `index_merge_sort_union={on|off}`
  - `index_merge_union={on|off}`
  - `in_to_exists={on|off}` - see IN-TO-EXISTS transformation
  - `join_cache_bka={on|off}` - see Block-Based Join Algorithms
  - `join_cache_hashed={on|off}` - see Block-Based Join Algorithms
  - `join_cache_incremental={on|off}` - see Block-Based Join Algorithms
  - `loosescan={on|off}` - see LooseScan strategy
  - `materialization={on|off}` - Semi-join and non semi-join materialization.
  - `mrr={on|off}` - see Multi Range Read optimization
  - `mrr_cost_based={on|off}` - see Multi Range Read optimization
  - `mrr_sort_keys={on|off}` - see Multi Range Read optimization
  - `not_null_range_scan={on|off}` - see not_null_range_scan optimization (>= MariaDB 10.5.0)
  - `optimize_join_buffer_size={on|off}` - see Block-Based Join Algorithms
  - `orderby_uses_equalities={on|off}` - if not set, the optimizer ignores equality propagation. See MDEV-8989.
  - `outer_join_with_cache={on|off}` - see Block-Based Join Algorithms
  - `partial_match_rowid_merge={on|off}` - see Non-semi-join subquery optimizations
  - `partial_match_table_scan={on|off}` - see Non-semi-join subquery optimizations
  - `rowid_filter={on|off}` - see Rowid Filtering Optimization (>= MariaDB 10.4.3)
  - `sargable_casefold={on|off}` (>= MariaDB 11.3.0)
  - `semijoin={on|off}` - see Semi-join subquery optimizations
  - `semijoin_with_cache={on|off}` - see Block-Based Join Algorithms
  - `split_materialized={on|off}`
  - `subquery_cache={on|off}` - see subquery cache.
  - `table_elimination={on|off}` - see Table Elimination User Interface

---

optimizer_trace

- **Description:** Controls tracing of the optimizer: optimizer_trace=option=val[,option=val...], where option is one of {enabled} and val is one of {on, off, default}
- **Commandline:** `--optimizer-trace=value`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `enum`
- **Default Value:** `enabled=off`
- **Valid Values:** `enabled={on|off|default}`
- **Introduced:** MariaDB 10.4.3

---

optimizer_trace_max_mem_size

- **Description:** Limits the memory used while tracing a query by specifying the maximum allowed cumulated size, in bytes, of stored optimizer traces.
- **Commandline:** `--optimizer-trace-max-mem-size=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1048576`
- **Range:** `1` to `18446744073709551615`
- **Introduced:** MariaDB 10.4.3

---

optimizer_use_condition_selectivity

- **Description:** Controls which statistics can be used by the optimizer when looking for the best query execution plan.
  - `1` Use selectivity of predicates as in MariaDB 5.5.
  - `2` Use selectivity of all range predicates supported by indexes.
  - `3` Use selectivity of all range predicates estimated without histogram.
  - `4` Use selectivity of all range predicates estimated with histogram.
  - `5` Additionally use selectivity of certain non-range predicates calculated on record sample.
- **Commandline:** `--optimizer-use-condition-selectivity=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `4` (>= MariaDB 10.4.1), `1` (<= MariaDB 10.4.0)
- **Range:** `1` to `5`

---

pid_file

- **Description:** Full path of the process ID file.
- **Commandline:** `--pid-file=file_name`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `file name`

---

plugin_dir

- **Description:** Path to the plugin directory. For security reasons, either make sure this directory can only be read by the server, or set secure_file_priv.
- **Commandline:** `--plugin-dir=path`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `directory name`
- **Default Value:** `BASEDIR/lib/plugin`

---

plugin_maturity

- **Description:** The lowest acceptable plugin maturity. MariaDB will not load plugins less mature than the specified level.
- **Commandline:** `--plugin-maturity=level`

- **Scope:** Global
- **Dynamic:** No
- **Type:** enum
- **Default Value:** One less than the server maturity (>= MariaDB 10.3.3), `unknown` (<= MariaDB 10.3.2)
- **Valid Values:** `unknown`, `experimental`, `alpha`, `beta`, `gamma`, `stable`

---

port

- **Description:** Port to listen for TCP/IP connections. If set to `0`, will default to, in order of preference, my.cnf, the MYSQL_TCP_PORT environment variable, /etc/services, built-in default (3306).
- **Commandline:** `--port=#`, `-P`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `3306`
- **Range:** `0` to `65535`

---

preload_buffer_size

- **Description:** Size in bytes of the buffer allocated when indexes are preloaded.
- **Commandline:** `--preload-buffer-size=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `32768`
- **Range:** `1024` to `1073741824`

---

profiling

- **Description:** If set to `1` (`0` is default), statement profiling will be enabled. See SHOW PROFILES() and SHOW PROFILE().
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

profiling_history_size

- **Description:** Number of statements about which profiling information is maintained. If set to `0`, no profiles are stored. See SHOW PROFILES.
- **Commandline:** `--profiling-history-size=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `15`
- **Range:** `0` to `100`

---

progress_report_time

- **Description:** Time in seconds between sending progress reports to the client for time-consuming statements. If set to `0`, progress reporting will be disabled.
- **Commandline:** `--progress-report-time=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `5`
- **Range:** `0` to `4294967295`

---

protocol_version

- **Description:** The version of the client/server protocol used by the MariaDB server.
- **Commandline:** None
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `10`
- **Range:** `0` to `4294967295`

---

proxy_protocol_networks

- **Description:** Enable proxy protocol for these source networks. The syntax is a comma separated list of IPv4 and IPv6 networks. If the network doesn't contain a mask, it is considered to be a single host. "*" represents all networks and must be the only directive on the line. String "localhost" represents non-TCP local connections (Unix domain socket, Windows named pipe or shared memory). See Proxy Protocol Support.
- **Commandline:** `--proxy-protocol-networks=value`
- **Scope:** Global
- **Dynamic:** Yes (>= MariaDB 10.3.6), No (<= MariaDB 10.3.5)
- **Data Type:** `string`
- **Default Value:** (empty)
- **Introduced:** MariaDB 10.3.1

---

proxy_user

- **Description:** Set to the proxy user account name if the current client is a proxy, else `NULL`.
- **Scope:** Session
- **Dynamic:** No
- **Data Type:** `string`

---

pseudo_slave_mode

- **Description:** For internal use by the server.
- **Scope:** Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `OFF`

---

pseudo_thread_id

- **Description:** For internal use only.
- **Scope:** Session
- **Dynamic:** Yes
- **Data Type:** `numeric`

---

query_alloc_block_size

- **Description:** Size in bytes of the extra blocks allocated during query parsing and execution (after query_prealloc_size is used up).
- **Commandline:** `--query-alloc-block-size=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `16384`
- **Range - 32 bit:** `1024` to `4294967295`
- **Range - 64 bit:** `1024` to `18446744073709547520`

---

query_cache_limit

- **Description:** Size in bytes for which results larger than this are not stored in the query cache.
- **Commandline:** `--query-cache-limit=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1048576` (1MB)
- **Range:** `0` to `4294967295`

---

query_cache_min_res_unit

- **Description:** Minimum size in bytes of the blocks allocated for query cache results.
- **Commandline:** `--query-cache-min-res-unit=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `4096` (4KB)
- **Range - 32 bit:** `1024` to `4294967295`
- **Range - 64 bit:** `1024` to `18446744073709547520`

---

query_cache_size

- **Description:** Size in bytes available to the query cache. About 40KB is needed for query cache structures, so setting a size lower than this will result in a warning. `0`, the default before MariaDB 10.1.7, effectively disables the query cache.

Warning: Starting from MariaDB 10.1.7, query_cache_type is automatically set to ON if the server is started with the query_cache_size set to a non-zero (and non-default) value. This will happen even if query_cache_type is explicitly set to OFF in the configuration.

- **Commandline:** `--query-cache-size=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1M` (although frequently given a default value in some setups)
- **Valid Values:** `0` upwards in units of 1024.

---

query_cache_strip_comments

- **Description:** If set to `1` (`0` is default), the server will strip any comments from the query before searching to see if it exists in the query cache. Multiple space, line feeds, tab and other white space characters will also be removed.
- **Commandline:** `query-cache-strip-comments`
- **Scope:** Session, Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

query_cache_type

- **Description:** If set to `0`, the query cache is disabled (although a buffer of query_cache_size bytes is still allocated). If set to `1` all SELECT queries will be cached unless SQL_NO_CACHE is specified. If set to `2` (or `DEMAND`), only queries with the SQL CACHE clause will be cached. Note that if the server is started with the query cache disabled, it cannot be enabled at runtime.

Warning: Starting from MariaDB 10.1.7, query_cache_type is automatically set to ON if the server is started with the query_cache_size set to a non-zero (and non-default) value. This will happen even if query_cache_type is explicitly set to OFF in the configuration.

- **Commandline:** `--query-cache-type=#`
- **Scope:** Global, Session

- **Dynamic:** Yes
- **Data Type:** `enumeration`
- **Default Value:** `OFF`
- **Valid Values:** `0` or `OFF`, `1` or `ON`, `2` or `DEMAND`

---

query_cache_wlock_invalidate

- **Description:** If set to `0`, the default, results present in the query cache will be returned even if there's a write lock on the table. If set to `1`, the client will first have to wait for the lock to be released.
- **Commandline:** `--query-cache-wlock-invalidate`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

query_prealloc_size

- **Description:** Size in bytes of the persistent buffer for query parsing and execution, allocated on connect and freed on disconnect. Increasing may be useful if complex queries are being run, as this will reduce the need for more memory allocations during query operation. See also query_alloc_block_size.
- **Commandline:** `--query-prealloc-size=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `24576`
- **Range:** `1024` to `4294967295`

---

rand_seed1

- **Description:** `rand_seed1` and `rand_seed2` facilitate replication of the RAND() function. The master passes the value of these to the slaves so that the random number generator is seeded in the same way, and generates the same value, on the slave as on the master. Until MariaDB 10.1.4, the variable value could not be viewed, with the SHOW VARIABLES output always displaying zero.
- **Commandline:** None
- **Scope:** Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** Varies
- **Range:** `0` to `18446744073709551615`

---

rand_seed2

- **Description:** See rand_seed1.

---

range_alloc_block_size

- **Description:** Size in bytes of blocks allocated during range optimization. The unit size in 1024.
- **Commandline:** `--range-alloc-block-size=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `4096`
- **Range - 32 bit:** `4096` to `4294967295`
- **Range - 64 bit:** `4096` to `18446744073709547520`

---

read_buffer_size

- **Description:** Each thread performing a sequential scan (for MyISAM, Aria and MERGE tables) allocates a buffer of this size in bytes for each table scanned. Increase if you perform many sequential scans. If not in a multiple of 4KB,

will be rounded down to the nearest multiple. Also used in ORDER BY's for caching indexes in a temporary file (not temporary table), for caching results of nested queries, for bulk inserts into partitions, and to determine the memory block size of MEMORY tables.

- **Commandline:** `--read-buffer-size=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `131072`
- **Range:** `8200 to 2147479552`

---

read_only

- **Description:** When set to `1` (`0` is default), no updates are permitted except from users with the SUPER privilege or, from MariaDB 10.5.2, the READ ONLY ADMIN privilege, or replica servers updating from a primary. The `read_only` variable is useful for replica servers to ensure no updates are accidentally made outside of what are performed on the primary. Inserting rows to log tables, updates to temporary tables and OPTIMIZE TABLE or ANALYZE TABLE statements are excluded from this limitation. If `read_only` is set to `1`, then the SET PASSWORD statement is limited only to users with the SUPER privilege (<= MariaDB 10.5.1) or READ ONLY ADMIN privilege (>= MariaDB 10.5.2). Attempting to set this variable to `1` will fail if the current session has table locks or transactions pending, while if other sessions hold table locks, the statement will wait until these locks are released before completing. While the attempt to set `read_only` is waiting, other requests for table locks or transactions will also wait until `read_only` has been set. See Read-Only Replicas for more.
- **Commandline:** `--read-only`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

read_rnd_buffer_size

- **Description:** Size in bytes of the buffer used when reading rows from a MyISAM table in sorted order after a key sort. Larger values improve ORDER BY performance, although rather increase the size by SESSION where the need arises to avoid excessive memory use.
- **Commandline:** `--read-rnd-buffer-size=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `262144`
- **Range:** `8200` to `2147483647`

---

redirect_url

- **Description:** URL of another server to redirect clients to. Empty string means no redirection.
- **Commandline:** `--redirect_url=val`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** Empty
- **Introduced:** MariaDB 11.3.0

-

-

require_secure_transport

- **Description:** When this option is enabled, connections attempted using insecure transport will be rejected. Secure transports are SSL/TLS, Unix sockets or named pipes. Note that per-account requirements take precedence.
- **Commandline:** `--require-secure-transport[={0|1}]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`

- **Default Value:** `OFF`
- **Introduced:** MariaDB 10.5.2

---

rowid_merge_buff_size

- **Description:** The maximum size in bytes of the memory available to the Rowid-merge strategy. See Non-semi-join subquery optimizations for more information.
- **Commandline:** `--rowid-merge-buff-size=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `8388608`
- **Range:** `0` to `2147483647`

---

rpl_recovery_rank

- **Description:** Unused.
- **Removed:** MariaDB 10.1.2

---

safe_show_database

- **Description:** This variable was removed in MariaDB 5.5, and has been replaced by the more flexible SHOW DATABASES privilege.
- **Commandline:** `--safe-show-database` (until MySQL 4.1.1)
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Removed:** MariaDB 5.5

---

secure_auth

- **Description:** Connections will be blocked if they use the the `mysql_old_password` authentication plugin. The server will also fail to start if the privilege tables are in the old, pre-MySQL 4.1 format.
- **Commandline:** `--secure-auth`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `ON`

---

secure_file_priv

- **Description:** LOAD DATA, SELECT ... INTO and LOAD FILE() will only work with files in the specified path. If not set, the default, or set to empty string, the statements will work with any files that can be accessed.
- **Commandline:** `--secure-file-priv=path`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `path name`
- **Default Value:** None

---

secure_timestamp

- **Description:** Restricts direct setting of a session timestamp. Possible levels are:
  - YES - timestamp cannot deviate from the system clock
  - REPLICATION - replication thread can adjust timestamp to match the master's
  - SUPER - a user with this privilege and a replication thread can adjust timestamp
  - NO - historical behavior, anyone can modify session timestamp
- **Commandline:** `--secure-timestamp=value`
- **Scope:** Global
- **Dynamic:** No

- **Data Type:** `enum`
- **Default Value:** `NO`
- **Introduced:** MariaDB 10.3.7

---

session_track_schema

- **Description:** Whether to track changes to the default schema within the current session.
- **Commandline:** `--session-track-schema={0|1}`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `ON`

---

session_track_state_change

- **Description:** Whether to track changes to the session state.
- **Commandline:** `--session-track-state-change={0|1}`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

session_track_system_variables

- **Description:** Comma-separated list of session system variables for which to track changes. For compatibility with MySQL defaults, this variable should be set to "autocommit, character_set_client, character_set_connection, character_set_results, time_zone". The `*` character tracks all session variables.
- **Commandline:** `--session-track-system-variables=value`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:**
  - >= MariaDB 11.3: `autocommit,character_set_client,character_set_connection,character_set_results,redirect_ur`
  - <= MariaDB 11.2: `autocommit, character_set_client, character_set_connection, character_set_results, time_zone`

---

session_track_transaction_info

- **Description:** Track changes to the transaction attributes. OFF to disable; STATE to track just transaction state (Is there an active transaction? Does it have any data? etc.); CHARACTERISTICS to track transaction state and report all statements needed to start a transaction with the same characteristics (isolation level, read only/read write,snapshot - but not any work done / data modified within the transaction).
- **Commandline:** `--session-track-transaction-info=value`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `enum`
- **Default Value:** `OFF`
- **Valid Values:** `OFF`, `STATE`, `CHARACTERISTICS`

---

shared_memory

- **Description:** Windows only, determines whether the server permits shared memory connections. See also shared_memory_base_name.
- **Scope:** Global
- **Dynamic:** No

---

shared_memory_base_name

- **Description:** Windows only, specifies the name of the shared memory to use for shared memory connection. Mainly used when running more than one instance on the same physical machine. By default the name is `MYSQL` and is case sensitive. See also shared_memory.
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** `MYSQL`

---

skip_external_locking

- **Description:** If this system variable is set, then some kinds of external table locks will be disabled for some storage engines.
  - If this system variable is set, then the MyISAM storage engine will not use file-based locks. Otherwise, it will use the `fcntl()` function with the `F_SETLK` option to get file-based locks on Unix, and it will use the `LockFileEx()` function to get file-based locks on Windows.
  - If this system variable is set, then the Aria storage engine will not lock a table when it decrements the table's in-file counter that keeps track of how many connections currently have the table open. See MDEV-19393 for more information.
- **Commandline:** `--skip-external-locking`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `1`

---

skip_grant_tables

- **Description:** Start without grant tables. This gives all users FULL ACCESS to all tables. Before MariaDB 10.10, available as an option only. Use mariadb-admin flush-privileges, mariadb-admin reload or FLUSH PRIVILEGES to resume using the grant tables.
- **Commandline:** `--skip-grant-tables`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB 10.10

---

skip_name_resolve

- **Description:** If set to 1 (0 is the default), only IP addresses are used for connections. Host names are not resolved. All host values in the GRANT tables must be IP addresses (or localhost).
- **Commandline:** `--skip-name-resolve`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `0`

---

skip_networking

- **Description:** If set to 1, (0 is the default), the server does not listen for TCP/IP connections. All interaction with the server will be through socket files (Unix) or named pipes or shared memory (Windows). It's recommended to use this option if only local clients are permitted to connect to the server.
- **Commandline:** `--skip-networking`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `0`

---

skip_show_database

- **Description:** If set to 1, (0 is the default), only users with the SHOW DATABASES privilege can use the SHOW

DATABASES statement to see all database names.
- **Commandline:** `--skip-show-database`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `0`

---

slow_launch_time

- **Description:** Time in seconds. If a thread takes longer than this to launch, the `slow_launch_threads` server status variable is incremented.
- **Commandline:** `--slow-launch-time=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `2`

---

slow_query_log

- **Description:** If set to 0, the default unless the --slow-query-log option is used, the slow query log is disabled, while if set to 1 (both global and session variables), the slow query log is enabled. From MariaDB 10.11.0, an alias for log_slow_query.
- **Commandline:** `--slow-query-log`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Data Type:** `boolean`
- **Default Value:** `0`
- **See also:** See log_output to see how log files are written. If that variable is set to `NONE`, no logs will be written even if slow_query_log is set to `1`.

---

slow_query_log_file

- **Description:** Name of the slow query log file. From MariaDB 10.11, an alias for log_slow_query_file.
- **Commandline:** `--slow-query-log-file=file_name`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `file name`
- **Default Value:** *`host_name`*`-slow.log`

---

socket

- **Description:** On Unix-like systems, this is the name of the socket file used for local client connections, by default `/tmp/mysql.sock`, often changed by the distribution, for example `/var/lib/mysql/mysql.sock`. On Windows, this is the name of the named pipe used for local client connections, by default `MySQL`. On Windows, this is not case-sensitive.
- **Commandline:** `--socket=name`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `file name`
- **Default Value:** `/tmp/mysql.sock` (Unix), `MySQL` (Windows)

---

sort_buffer_size

- **Description:** Each session performing a sort allocates a buffer with this amount of memory. Not specific to any storage engine. If the status variable sort_merge_passes is too high, you may need to look at improving your query indexes, or increasing this. Consider reducing where there are many small sorts, such as OLTP, and increasing where needed by session. 16k is a suggested minimum.
- **Commandline:** `--sort-buffer-size=#`
- **Scope:** Global, Session

- **Dynamic:** Yes
- **Data Type:** `number`
- **Default Value:** `2M (2097152)` (some distributions increase the default)

---

sql_auto_is_null

- **Description:** If set to 1, the query `SELECT * FROM table_name WHERE auto_increment_column IS NULL` will return an auto-increment that has just been successfully inserted, the same as the LAST_INSERT_ID() function. Some ODBC programs make use of this IS NULL comparison.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `0`

---

sql_big_selects

- **Description:** If set to 0, MariaDB will not perform large SELECTs. See max_join_size for details. If max_join_size is set to anything but DEFAULT, sql_big_selects is automatically set to 0. If sql_big_selects is again set, max_join_size will be ignored.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `1`

---

sql_big_tables

- **Description:** Old variable, which if set to 1, allows large result sets by saving all temporary sets to disk, avoiding 'table full' errors. No longer needed, as the server now handles this automatically.
  - This is a synonym for `big_tables` .
- **Commandline:** `--sql-big-tables`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `0`
- **Removed:** MariaDB 10.0

---

sql_buffer_result

- **Description:** If set to 1 (0 is default), results from SELECT statements are always placed into temporary tables. This can help the server when it takes a long time to send the results to the client by allowing the table locks to be freed early.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `0`

---

sql_if_exists

- **Description:** If set to 1, adds an implicit IF EXISTS to ALTER, RENAME and DROP of TABLES, VIEWS, FUNCTIONS and PACKAGES. This variable is mainly used in replication to tag DDLs that can be ignored on the slave if the target table doesn't exist.
- **Commandline:** `--sql-if-exists[={0|1}]`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB 10.5.2

---

sql_log_off

- **Description:** If set to 1 (0 is the default), no logging to the general query log is done for the client. Only clients with the SUPER privilege can update this variable.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `0`

---

sql_log_update

- **Description:** Removed. Use sql_log_bin instead.
- **Removed:** MariaDB/MySQL 5.5

---

sql_low_priority_updates

- **Description:** If set to 1 (0 is the default), for storage engines that use only table-level locking (Aria, MyISAM, MEMORY and MERGE), all INSERTs, UPDATEs, DELETEs and LOCK TABLE WRITEs will wait until there are no more SELECTs or LOCK TABLE READs pending on the relevant tables. Set this to 1 if reads are prioritized over writes.
  - This is a synonym for `low_priority_updates`.
- **Commandline:** `--sql-low-priority-updates`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `0`
- **Removed:** MariaDB 10.0

---

sql_max_join_size

- **Description:** Synonym for max_join_size, the preferred name.
- **Deprecated:** MariaDB 5.5
- **Removed:** MariaDB 10.0

---

sql_mode

- **Description:** Sets the SQL Mode. Multiple modes can be set, separated by a comma.
- **Commandline:** `--sql-mode=value[,value[,value...]]`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:**
  - `STRICT_TRANS_TABLES,ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION`
- **Valid Values:** See SQL Mode for the full list.

---

sql_notes

- **Description:** If set to 1, the default, warning_count is incremented each time a Note warning is encountered. If set to 0, Note warnings are not recorded. mariadb-dump has outputs to set this variable to 0 so that no unnecessary increments occur when data is reloaded. See also note_verbosity, which defines which notes should be given. The recommended way, as of MariaDB 10.6.16, to disable notes is to set `note_verbosity` to "".
- **Commandline:** None
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `1`

---

sql_quote_show_create

- **Description:** If set to 1, the default, the server will quote identifiers for SHOW CREATE DATABASE, SHOW CREATE TABLE and SHOW CREATE VIEW statements. Quoting is disabled if set to 0. Enable to ensure

replications works when identifiers require quoting.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `1`

---

sql_safe_updates

- **Description:** If set to 1, UPDATEs and DELETEs must be executed by using an index (simply mentioning an indexed column in a WHERE clause is not enough, optimizer must actually use it) or they must mention an indexed column and specify a LIMIT clause. Otherwise a statement will be aborted. Prevents the common mistake of accidentally deleting or updating every row in a table. Until MariaDB 10.3.11, could not be set as a command-line option or in my.cnf.
- **Commandline:** `--sql-safe-updates[={0|1}]`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

sql_select_limit

- **Description:** Maximum number of rows that can be returned from a SELECT query. Default is the maximum number of rows permitted per table by the server, usually 2^{32}-1 or 2^{64}-1. Can be restored to the default value after being changed by assigning it a value of DEFAULT.
- **Commandline:** None
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `18446744073709551615`

---

sql_warnings

- **Description:** If set to 1, single-row INSERTs will produce a string containing warning information if a warning occurs.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF (0)`

---

storage_engine

- **Description:** See default_storage_engine.
- **Deprecated:** MariaDB 5.5

---

standard_compliant_cte

- **Description:** Allow only standard-compliant common table expressions. Prior to MariaDB 10.2.4, this variable was named `standards_compliant_cte`.
- **Commandline:** `--standard-compliant-cte={0|1}`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `ON`
- **Introduced:** MariaDB 10.2.2

---

stored_program_cache

- **Description:** Limit to the number of stored routines held in the stored procedures and stored functions caches. Each time a stored routine is executed, this limit is first checked, and if the number held in the cache exceeds this, that cache is flushed and memory freed.

- **Commandline:** `--stored-program-cache=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `256`
- **Range:** `256` to `524288`

---

strict_password_validation

- **Description:** When password validation plugins are enabled, reject passwords that cannot be validated (passwords specified as a hash). This excludes direct updates to the privilege tables.
- **Commandline:** `--strict-password-validation`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `ON`

---

sync_frm

- **Description:** If set to 1, the default, each time a non-temporary table is created, its .frm definition file is synced to disk. Fractionally slower, but safer in case of a crash.
- **Commandline:** `--sync-frm`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `TRUE`

---

system_time_zone

- **Description:** The system time zone is determined when the server starts. The system time zone is usually read from the operating system's environment but can be overridden by setting the 'TZ' environment variable before starting the server. See Time Zones: System Time Zone for the various ways to change the system time zone. This variable is not the same as the time_zone system variable, which is the variable that actually controls a session's active time zone. The system time zone is used for a session when `time_zone` is set to the special value `SYSTEM`.
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`

---

table_definition_cache

- **Description:** Number of table definitions that can be cached. Table definitions are taken from the .frm files, and if there are a large number of tables increasing the cache size can speed up table opening. Unlike the table_open_cache, as the table_definition_cache doesn't use file descriptors, and is much smaller.
- **Commandline:** `--table-definition-cache=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `400`
- **Range:**
  - `400` to `2097152` (>= MariaDB 10.4.2, MariaDB 10.3.13, MariaDB 10.2.22, MariaDB 10.1.38)
  - `400` to `524288` (<= MariaDB 10.4.1, MariaDB 10.3.12, MariaDB 10.2.21, MariaDB 10.1.37)

---

table_lock_wait_timeout

- **Description:** Unused, and removed.
- **Commandline:** `--table-lock-wait-timeout=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `50`

- **Range:** `1` to `1073741824`
- **Removed:** MariaDB 5.5

---

table_open_cache

- **Description:** Maximum number of open tables cached in one table cache instance. See Optimizing table_open_cache for suggestions on optimizing. Increasing table_open_cache increases the number of file descriptors required.
- **Commandline:** `--table-open-cache=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `2000`
- **Range:**
  - `1` to `1048576` (1024K)

---

table_open_cache_instances

- **Description:** This system variable specifies the maximum number of table cache instances. MariaDB Server initially creates just a single instance. However, whenever it detects contention on the existing instances, it will automatically create a new instance. When the number of instances has been increased due to contention, it does not decrease again. The default value of this system variable is `8`, which is expected to handle up to 100 CPU cores. If your system is larger than this, then you may benefit from increasing the value of this system variable.
  - Depending on the ratio of actual available file handles, and `table_open_cache` size, the max. instance count may be auto adjusted to a lower value on server startup.
  - The implementation and behavior of this feature is different than the same feature in MySQL 5.6.
  - See Optimizing table_open_cache: Automatic Creation of New Table Open Cache Instances for more information.
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `8` (>= MariaDB 10.2.2)
- **Range:** `1` to `64`
- **Introduced:** MariaDB 10.2.2

---

table_type

- **Description:** Removed and replaced by storage_engine. Use default_storage_engine instead.

---

tcp_keepalive_interval

- **Description:** The interval, in seconds, between when successive keep-alive packets are sent if no acknowledgement is received. If set to 0, the system dependent default is used.
- **Commandline:** `--tcp-keepalive-interval=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `2147483`
- **Introduced:** MariaDB 10.3.3

---

tcp_keepalive_probes

- **Description:** The number of unacknowledged probes to send before considering the connection dead and notifying the application layer. If set to 0, a system dependent default is used.
- **Commandline:** `--tcp-keepalive-probes=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`

- **Range:** `0` to `2147483`
- **Introduced:** MariaDB 10.3.3

---

tcp_keepalive_time

- **Description:** Timeout, in seconds, with no activity until the first TCP keep-alive packet is sent. If set to 0, a system dependent default is used.
- **Commandline:** `--tcp-keepalive-time=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `2147483`
- **Introduced:** MariaDB 10.3.3

---

tcp_nodelay

- **Description:** Set the TCP_NODELAY option (disable Nagle's algorithm) on socket.
- **Commandline:** `--tcp-nodelay={0|1}`
- **Scope:** Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `1`
- **Introduced:** MariaDB 10.4.0

---

thread_cache_size

- **Description:** Number of threads server caches for re-use. If this limit hasn't been reached, when a client disconnects, its threads are put into the cache, and re-used where possible. In MariaDB 10.2.0 and newer the threads are freed after 5 minutes of idle time. Normally this setting has little effect, as the other aspects of the thread implementation are more important, but increasing it can help servers with high volumes of connections per second so that most can use a cached, rather than a new, thread. The cache miss rate can be calculated as the server status variables threads_created/connections. If the thread pool is active, `thread_cache_size` is ignored. If `thread_cache_size` is set to greater than the value of max_connections, `thread_cache_size` will be set to the max_connections value.
- **Commandline:** `--thread-cache-size=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `256` (adjusted if thread pool is active)
- **Range:** `0` to `16384`

---

thread_concurrency

- **Description:** Allows applications to give the system a hint about the desired number of threads. Specific to Solaris only, invokes thr_setconcurrency(). Deprecated and has no effect from MariaDB 5.5.
- **Commandline:** `--thread-concurrency=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `10`
- **Range:** `1` to `512`
- **Deprecated:** MariaDB 5.5
- **Removed:** MariaDB 10.5.1

---

thread_stack

- **Description:** Stack size for each thread. If set too small, limits recursion depth of stored procedures and complexity of SQL statements the server can handle in memory. Also affects limits in the crash-me test.
- **Commandline:** `--thread-stack=#`

- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:**
  - `299008`
- **Range:** `131072` to `18446744073709551615`

---

time_format

- **Description:** Unused.
- **Removed:** MariaDB 11.3.0

---

time_zone

- **Description:** The global value determines the default time zone for sessions that connect. The session value determines the session's active time zone. When it is set to `SYSTEM`, the session's time zone is determined by the `system_time_zone` system variable.
- **Commandline:** `--default-time-zone=string`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `SYSTEM`

---

timed_mutexes

- **Description:** Determines whether InnoDB mutexes are timed. `OFF`, the default, disables mutex timing, while `ON` enables it. See also SHOW ENGINE for more on mutex statistics. Deprecated and has no effect.
- **Commandline:** `--timed-mutexes`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Deprecated:** MariaDB 5.5.39
- **Removed:** MariaDB 10.5.1

---

timestamp

- **Description:** Sets the time for the client. This will affect the result returned by the NOW() function, not the SYSDATE() function, unless the server is started with the --sysdate-is-now option, in which case SYSDATE becomes an alias of NOW, and will also be affected. Also used to get the original timestamp when restoring rows from the binary log.
- **Scope:** Session
- **Dynamic:** Yes
- **Valid Values:** `timestamp_value` (Unix epoch timestamp, not MariaDB timestamp), `DEFAULT`

---

tmp_disk_table_size

- **Description:** Max size for data for an internal temporary on-disk MyISAM or Aria table. These tables are created as part of complex queries when the result doesn't fit into the memory engine. You can set this variable if you want to limit the size of temporary tables created in your temporary directory tmpdir.
- **Commandline:** `--tmp-disk-table-size=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `18446744073709551615` (max unsigned integer, no limit)
- **Range:** `1024` to `18446744073709551615`
- **Introduced:** MariaDB 10.2.7

---

tmp_memory_table_size

- **Description:** An alias for tmp_table_size.
- **Commandline:** `--tmp-memory-table-size=#`
- **Introduced:** MariaDB 10.2.7

---

tmp_table_size

- **Description:** The largest size for temporary tables in memory (not MEMORY tables) although if max_heap_table_size is smaller the lower limit will apply. You can see if it's necessary to increase by comparing the status variables `Created_tmp_disk_tables` and `Created_tmp_tables` to see how many temporary tables out of the total created needed to be converted to disk. Often complex GROUP BY queries are responsible for exceeding the limit. Defaults may be different on some systems, see for example Differences in MariaDB in Debian. From MariaDB 10.2.7, tmp_memory_table_size is an alias.
- **Commandline:** `--tmp-table-size=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `16777216` (16MB)
- **Range:**
  - `1024` to `4294967295` (< MariaDB 10.5)
  - `0` to `4294967295` (>= MariaDB 10.5.0)

---

tmpdir

- **Description:** Directory for storing temporary tables and files. Can specify a list (separated by semicolons in Windows, and colons in Unix) that will then be used in round-robin fashion. This can be used for load balancing across several disks. Note that if the server is a replication replica, and slave_load_tmpdir, which overrides `tmpdir` for replicas, is not set, you should not set `tmpdir` to a directory that is cleared when the machine restarts, or else replication may fail.
- **Commandline:** `--tmpdir=path` or `-t path`
- **Scope:** Global
- **Dynamic:** No
- **Type:** directory name/s
- **Default:**
  - `$TMPDIR` (environment variable) if set
  - otherwise `$TEMP` if set and on Windows
  - otherwise `$TMP` if set and on Windows
  - otherwise `P_tmpdir` (`"/tmp"`) or `C:\TEMP` (unless overridden during buid time)

---

transaction_alloc_block_size

- **Description:** Size in bytes to increase the memory pool available to each transaction when the available pool is not large enough. See transaction_prealloc_size.
- **Commandline:** `--transaction-alloc-block-size=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Type:** numeric
- **Default Value:** `8192`
- **Range:** `1024` to `4294967295`
- **Block Size:** `1024`

---

transaction_isolation

- **Description:** The transaction isolation level. See also SET TRANSACTION ISOLATION LEVEL. Introduced in MariaDB 11.1.1 to replace the tx_isolation system variable and align the option and the system variable name.
- **Commandline:** `--transaction-isolation=name`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Type:** enumeration
- **Default Value:** `REPEATABLE-READ`

- **Valid Values:** `READ-UNCOMMITTED`, `READ-COMMITTED`, `REPEATABLE-READ`, `SERIALIZABLE`
- **Introduced:** MariaDB 11.1.1

---

transaction_prealloc_size

- **Description:** Initial size of a memory pool available to each transaction for various memory allocations. If the memory pool is not large enough for an allocation, it is increased by transaction_alloc_block_size bytes, and truncated back to transaction_prealloc_size bytes when the transaction is completed. If set large enough to contain all statements in a transaction, extra malloc() calls are avoided.
- **Commandline:** `--transaction-prealloc-size=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Type:** numeric
- **Default Value:** `4096`
- **Range:** `1024` to `4294967295`
- **Block Size:** `1024`

---

transaction_read_only

- **Description:** Default transaction access mode. If set to `OFF`, the default, access is read/write. If set to `ON`, access is read-only. The `SET TRANSACTION` statement can also change the value of this variable. See SET TRANSACTION and START TRANSACTION.
- **Commandline:** `--transaction-read-only=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Type:** boolean
- **Default Value:** `OFF`
- **Introduced:** MariaDB 11.1

---

tx_isolation

- **Description:** The transaction isolation level. Setting this session variable via `set @@tx_isolation=` will take effect for only the subsequent transaction in the current session, much like SET TRANSACTION ISOLATION LEVEL. To set for a session, use `SET SESSION tx_isolation` or `SET @@session.tx_isolation`. See MDEV-31751. See also SET TRANSACTION ISOLATION LEVEL. In MariaDB 11.1, this system variable is deprecated and replaced by transaction_isolation.
- **Commandline:** `--transaction-isolation=name`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Type:** enumeration
- **Default Value:** `REPEATABLE-READ`
- **Valid Values:** `READ-UNCOMMITTED`, `READ-COMMITTED`, `REPEATABLE-READ`, `SERIALIZABLE`
- **Deprecated:** MariaDB 11.1

---

tx_read_only

- **Description:** Default transaction access mode. If set to `OFF`, the default, access is read/write. If set to `ON`, access is read-only. The `SET TRANSACTION` statement can also change the value of this variable. See SET TRANSACTION and START TRANSACTION. In MariaDB 11.1, this system variable is deprecated and replaced by transaction_read_only.
- **Commandline:** `--transaction-read-only=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Type:** boolean
- **Default Value:** `OFF`
- **Deprecated:** MariaDB 11.1

---

unique_checks

- **Description:** If set to 0, storage engines can (but are not required to) assume that duplicate keys are not present in

input data. If set to 0, inserting duplicates into a `UNIQUE` index can succeed, causing the table to become corrupted. Set to 0 to speed up imports of large tables to InnoDB.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Type:** boolean
- **Default Value:** `1`

---

updatable_views_with_limit

- **Description:** Determines whether view updates can be made with an UPDATE or DELETE statement with a LIMIT clause if the view does not contain all primary or not null unique key columns from the underlying table. `0` prohibits this, while `1` permits it while issuing a warning (the default).
- **Commandline:** `--updatable-views-with-limit=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Type:** boolean
- **Default Value:** `1`

---

use_stat_tables

- **Description:** Controls the use of engine-independent table statistics.
  - `never` : The optimizer will not use data from statistics tables.
  - `complementary` : The optimizer uses data from statistics tables if the same kind of data is not provided by the storage engine.
  - `preferably` : Prefer the data from statistics tables, if it's not available there, use the data from the storage engine.
  - `complementary_for_queries` : Same as `complementary` , but for queries only (to avoid needlessly collecting for ANALYZE TABLE). From MariaDB 10.4.1.
  - `preferably_for_queries` : Same as `preferably` , but for queries only (to avoid needlessly collecting for ANALYZE TABLE). From MariaDB 10.4.1.
- **Commandline:** `--use-stat-tables=mode`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `enum`
- **Default Value:** `preferably_for_queries` (>= MariaDB 10.4.1), `never` (<= MariaDB 10.4.0)

---

version

- **Description:** Server version number. It may also include a suffix with configuration or build information. `-debug` indicates debugging support was enabled on the server, and `-log` indicates at least one of the binary log, general log or slow query log are enabled, for example `10.0.1-MariaDB-mariadb1precise-log` . Can be set at startup in order to fake the server version.
- **Commandline:** `-V` , `--version[=name]`
- **Scope:** Global
- **Dynamic:** No
- **Type:** string

---

version_comment

- **Description:** Value of the COMPILATION_COMMENT option specified by CMake when building MariaDB, for example `mariadb.org binary distribution` .
- **Scope:** Global
- **Dynamic:** No
- **Type:** string

---

version_compile_machine

- **Description:** The machine type or architecture MariaDB was built on, for example `i686` .
- **Scope:** Global
- **Dynamic:** No

- **Type:** string

`version_compile_os`

- **Description:** Operating system that MariaDB was built on, for example `debian-linux-gnu`.
- **Scope:** Global
- **Dynamic:** No
- **Type:** string

`version_malloc_library`

- **Description:** Version of the used malloc library.
- **Commandline:** No
- **Scope:** Global
- **Dynamic:** No
- **Type:** string

`version_source_revision`

- **Description:** Source control revision id for MariaDB source code, enabling one to see exactly which version of the source was used for a build.
- **Commandline:** None
- **Scope:** Global
- **Dynamic:** No
- **Type:** string
- **Introduced:** MariaDB 10.3.2

`wait_timeout`

- **Description:** Time in seconds that the server waits for a connection to become active before closing it. The session value is initialized when a thread starts up from either the global value, if the connection is non-interactive, or from the interactive_timeout value, if the connection is interactive.
- **Commandline:** `--wait-timeout=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Type:** numeric
- **Default Value:** `28800`
- **Range: (Windows):** `1` **to** `2147483`
- **Range: (Other):** `1` **to** `31536000`

`warning_count`

- **Description:** Read-only variable indicating the number of warnings, errors and notes resulting from the most recent statement that generated messages. See SHOW WARNINGS for more. Note warnings will only be recorded if sql_notes is true (the default).
- **Scope:** Session
- **Dynamic:** No
- **Type:** numeric

2.7.3 OLD_MODE

The old_mode system variable was introduced in MariaDB 5.5.35 to replace the old variable with a new one with better granularity.

Contents

MariaDB supports several different modes which allow you to tune it to suit your needs.

The most important ways for doing this are with SQL_MODE and `OLD_MODE`.

SQL_MODE is used for getting MariaDB to emulate behavior from other SQL servers, while `OLD_MODE` is used for emulating behavior from older MariaDB or MySQL versions.

`OLD_MODE` is a string with different options separated by commas (' , ') without spaces. The options are case insensitive.

Normally `OLD_MODE` should be empty. It's mainly used to get old behavior when switching to MariaDB or to a new major version of MariaDB, until you have time to fix your application.

Between major versions of MariaDB various options supported by `OLD_MODE` may be removed. This is intentional as we assume that the application will be fixed to conform with the new MariaDB behavior between releases.

In other words, `OLD_MODE` options are by design deprecated from the day they were added and will eventually be removed as any other deprecated feature.

You can check the variable's local and global value with:

```
SELECT @@OLD_MODE, @@GLOBAL.OLD_MODE;
```

You can set the `OLD_MODE`
either from the command line (option `--old-mode`) or by setting the old_mode system variable.

Non-default old mode features are deprecated by design, and from MariaDB 11.3, a warning will be issued when set.

Modes

The different values of `OLD_MODE` are:

NO_DUP_KEY_WARNINGS_WITH_IGNORE

Don't print duplicate key warnings when using INSERT IGNORE.

NO_PROGRESS_INFO

Don't show progress information in SHOW PROCESSLIST.

UTF8_IS_UTF8MB3

From MariaDB 10.6.1, the main name of the previous 3 byte `utf` character set has been changed to `utf8mb3`. If set, the default, `utf8` is an alias for `utf8mb3`. If not set, `utf8` would be an alias for `utf8mb4`.

ZERO_DATE_TIME_CAST

When a TIME value is cast to a DATETIME, the date part will be `0000-00-00`, not CURRENT_DATE (as dictated by the SQL standard).

IGNORE_INDEX_ONLY_FOR_JOIN

From MariaDB 10.9, the --old option is deprecated. This option allows behaviour of the --old option for disabling the index only for joins, but allow it for ORDER BY.

COMPAT_5_1_CHECKSUM

From MariaDB 10.9, the --old option is deprecated. This option allows behaviour of the --old option for enabling the old-style checksum for `CHECKSUM TABLE` that MySQL 5.1 supports

LOCK_ALTER_TABLE_COPY

From MariaDB 11.2. The non-locking copy ALTER introduced in MDEV-16329 should be beneficial in the vast majority of cases, but scenarios can exist which significantly impact performance. For example, RBR on tables without a primary key. When non-locking ALTER is performed on such a table, and DML affecting a large number of records is run in parallel, the ALTER can become extremely slow, and further DML can also be affected. If there is a chance of such scenarios (and no possibility of improving the schema by immediately adding primary keys), ALTER should be performed with the explicit LOCK=SHARED clause. If this is also impossible, then LOCK_ALTER_TABLE_COPY flag should be added to the old_mode variable until the schema can be improved.

OLD_MODE and Stored Programs

In contrast to SQL_MODE, stored programs use the current user's `OLD_MODE` value.

Changes to `OLD_MODE` are not sent to replicas.

Examples

This example shows how to get a readable list of enabled OLD_MODE flags:

```
SELECT REPLACE(@@OLD_MODE, ',', '\n');
+-------------------------------------------------+
| REPLACE(@@OLD_MODE, ',', '\n')                  |
+-------------------------------------------------+
| NO_DUP_KEY_WARNINGS_WITH_IGNORE                 |
| NO_PROGRESS_INFO                                |
+-------------------------------------------------+
```

Adding a new flag:

```
SET @@OLD_MODE = CONCAT(@@OLD_MODE, ',NO_PROGRESS_INFO');
```

If the specified flag is already ON, the above example has no effect but does not produce an error.

How to unset a flag:

```
SET @@OLD_MODE = REPLACE(@@OLD_MODE, 'NO_PROGRESS_INFO', '');
```

How to check if a flag is set:

```
SELECT @@OLD_MODE LIKE '%NO_PROGRESS_INFO';
+-----------------------------------+
| @@OLD_MODE LIKE '%NO_PROGESS_INFO' |
+-----------------------------------+
|                                 1 |
+-----------------------------------+
```

From MariaDB 11.3:

```
SET @@OLD_MODE = CONCAT(@@OLD_MODE, ',NO_PROGRESS_INFO');
Query OK, 0 rows affected, 1 warning (0.000 sec)

SHOW WARNINGS;
+---------+------+-----------------------------------------------------------------------------+
| Level   | Code | Message                                                                     |
+---------+------+-----------------------------------------------------------------------------+
| Warning | 1287 | 'NO_PROGRESS_INFO' is deprecated and will be removed in a future release |
+---------+------+-----------------------------------------------------------------------------+
```

2.7.4 SQL_MODE

Contents

MariaDB supports several different modes which allow you to tune it to suit your needs.

The most important ways for doing this are using `SQL_MODE` (controlled by the sql_mode system variable) and OLD_MODE (the old_mode system variable). `SQL_MODE` is used for getting MariaDB to emulate behavior from other SQL servers, while OLD_MODE is used for emulating behavior from older MariaDB or MySQL versions.

`SQL_MODE` is a string with different options separated by commas (' , ') without spaces. The options are case insensitive.

You can check the local and global value of it with:

```
SELECT @@SQL_MODE, @@GLOBAL.SQL_MODE;
```

Setting SQL_MODE

Defaults

From version	Default sql_mode setting
MariaDB 10.2.4	STRICT_TRANS_TABLES, ERROR_FOR_DIVISION_BY_ZERO , NO_AUTO_CREATE_USER, NO_ENGINE_SUBSTITUTION

MariaDB 10.1.7	NO_ENGINE_SUBSTITUTION, NO_AUTO_CREATE_USER
<= MariaDB 10.1.6	No value

You can set the `SQL_MODE` either from the command line (the `--sql-mode` option) or by setting the sql_mode system variable.

```
SET sql_mode = 'modes';
SET GLOBAL sql_mode = 'modes';
```

The session value only affects the current client, and can be changed by the client when required. To set the global value, the SUPER privilege is required, and the change affects any clients that connect from that point on.

SQL_MODE Values

The different `SQL_MODE` values are:

ALLOW_INVALID_DATES

Allow any day between 1-31 in the day part. This is convenient when you want to read in all (including wrong data) into the database and then manipulate it there.

ANSI

Changes the SQL syntax to be closer to ANSI SQL.

Sets: REAL_AS_FLOAT, PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE.

It also adds a restriction: an error will be returned if a subquery uses an aggregating function with a reference to a column from an outer query in a way that cannot be resolved.

If set, SHOW CREATE TABLE output will not display MariaDB-specific table attributes.

ANSI_QUOTES

Changes `"` to be treated as `` ` ``, the identifier quote character. This may break old MariaDB applications which assume that `"` is used as a string quote character.

DB2

Same as: PIPES_AS_CONCAT, ANSI_QUOTES , IGNORE_SPACE, DB2, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS

If set, SHOW CREATE TABLE output will not display MariaDB-specific table attributes.

EMPTY_STRING_IS_NULL

Oracle-compatibility option that translates Item_string created in the parser to Item_null, and translates binding an empty string as prepared statement parameters to binding NULL. For example, `SELECT '' IS NULL` returns TRUE, `INSERT INTO t1 VALUES ('')` inserts NULL. Since MariaDB 10.3.3

ERROR_FOR_DIVISION_BY_ZERO

If not set, division by zero returns NULL. If set returns an error if one tries to update a column with 1/0 and returns a warning as well. Also see MDEV-8319. Default since MariaDB 10.2.4.

HIGH_NOT_PRECEDENCE

Compatibility option for MySQL 5.0.1 and before; This changes `NOT a BETWEEN b AND c` to be parsed as `(NOT a) BETWEEN b AND c`

IGNORE_BAD_TABLE_OPTIONS

If this is set generate a warning (not an error) for wrong table option in CREATE TABLE. Also, since 10.0.13, do not comment out these wrong table options in SHOW CREATE TABLE.

IGNORE_SPACE

Allow one to have spaces (including tab characters and new line characters) between function name and '('. The drawback is that this causes built in functions to become reserved words.

MAXDB

Same as: PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, MAXDB, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS, NO_AUTO_CREATE_USER.

Also has the effect of silently converting TIMESTAMP fields into DATETIME fields when created or modified.

If set, SHOW CREATE TABLE output will not display MariaDB-specific table attributes.

MSSQL

Additionally implies the following: PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS.

Additionally from MariaDB 10.4.5, implements a limited subset of Microsoft SQL Server's language. See SQL_MODE=MSSQL for more.

If set, SHOW CREATE TABLE output will not display MariaDB-specific table attributes.

MYSQL323

Same as: NO_FIELD_OPTIONS, HIGH_NOT_PRECEDENCE.

MYSQL40

Same as: NO_FIELD_OPTIONS, HIGH_NOT_PRECEDENCE.

NO_AUTO_CREATE_USER

Don't automatically create users with `GRANT` unless authentication information is specified. If none is provided, will produce a 1133 error: "Can't find any matching row in the user table". Default since MariaDB 10.1.7.

NO_AUTO_VALUE_ON_ZERO

If set, don't generate an AUTO_INCREMENT on INSERT of zero in an `AUTO_INCREMENT` column, or when adding an AUTO_INCREMENT attribute with the ALTER TABLE statement. Normally both `zero` and `NULL` generate new `AUTO_INCREMENT` values.

NO_BACKSLASH_ESCAPES

Disables using the backslash character `\` as an escape character within strings, making it equivalent to an ordinary character.

NO_DIR_IN_CREATE

Ignore all INDEX DIRECTORY and DATA DIRECTORY directives when creating a table. Can be useful on slave replication servers.

NO_ENGINE_SUBSTITUTION

If not set, if the available storage engine specified by a CREATE TABLE is not available, a warning is given and the default storage engine is used instead. If set, generate a 1286 error when creating a table if the specified storage engine is not available. See also enforce_storage_engine. Default since MariaDB 10.1.7.

NO_FIELD_OPTIONS

Remove MariaDB-specific column options from the output of SHOW CREATE TABLE. This is also used by the portability mode of mariadb-dump.

NO_KEY_OPTIONS

Remove MariaDB-specific index options from the output of SHOW CREATE TABLE. This is also used by the portability mode of mariadb-dump.

NO_TABLE_OPTIONS

Remove MariaDB-specific table options from the output of SHOW CREATE TABLE. This is also used by the portability mode of mariadb-dump.

NO_UNSIGNED_SUBTRACTION

When enabled, subtraction results are signed even if the operands are unsigned.

NO_ZERO_DATE

Don't allow '0000-00-00' as a valid date in strict mode (produce a 1525 error). Zero dates can be inserted with IGNORE. If not in strict mode, a warning is generated.

NO_ZERO_IN_DATE

Don't allow dates where the year is not zero but the month or day parts of the date *are* zero (produce a 1525 error). For example, with this set, '0000-00-00' is allowed, but '1970-00-10' or '1929-01-00' are not. If the ignore option is used, MariaDB will insert '0000-00-00' for those types of dates. If not in strict mode, a warning is generated instead.

ONLY_FULL_GROUP_BY

For SELECT ... GROUP BY queries, disallow SELECTing columns which are not referred to in the GROUP BY clause, unless they are passed to an aggregate function like COUNT() or MAX(). Produce a 1055 error.

ORACLE

In all versions of MariaDB up to MariaDB 10.2, this sets `sql_mode` that is equivalent to: PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS, NO_AUTO_CREATE_USER

From MariaDB 10.3, this mode also sets SIMULTANEOUS_ASSIGNMENT and configures the server to understand a large subset of Oracle's PL/SQL language instead of MariaDB's traditional syntax for stored routines. See SQL_MODE=ORACLE From MariaDB 10.3.

If set, SHOW CREATE TABLE output will not display MariaDB-specific table attributes.

PAD_CHAR_TO_FULL_LENGTH

Trailing spaces in CHAR columns are by default trimmed upon retrieval. With PAD_CHAR_TO_FULL_LENGTH enabled, no trimming occurs. Does not apply to VARCHARs.

PIPES_AS_CONCAT

Allows using the pipe character (ASCII 124) as string concatenation operator. This means that `"A" || "B"` can be used in place of `CONCAT("A", "B")`.

POSTGRESQL

Same as: PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, POSTGRESQL, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS.

If set, SHOW CREATE TABLE output will not display MariaDB-specific table attributes.

REAL_AS_FLOAT

`REAL` is a synonym for FLOAT rather than DOUBLE.

SIMULTANEOUS_ASSIGNMENT

Setting this makes the SET part of the UPDATE statement evaluate all assignments simultaneously, not left-to-right. From MariaDB 10.3.5.

STRICT_ALL_TABLES

Strict mode. Statements with invalid or missing data are aborted and rolled back. For a non-transactional storage engine with a statement affecting multiple rows, this may mean a partial insert or update if the error is found in a row beyond the first.

STRICT_TRANS_TABLES

Strict mode. Statements with invalid or missing data are aborted and rolled back, except that for non-transactional storage engines and statements affecting multiple rows where the invalid or missing data is not the first row, MariaDB will convert the invalid value to the closest valid value, or, if a value is missing, insert the column default value. Default since MariaDB 10.2.4.

TIME_ROUND_FRACTIONAL

With this mode unset, MariaDB truncates fractional seconds when changing precision to smaller. When set, MariaDB will round when converting to TIME, DATETIME and TIMESTAMP, and truncate when converting to DATE. Since MariaDB 10.4.1

TRADITIONAL

Makes MariaDB work like a traditional SQL server. Same as: STRICT_TRANS_TABLES, STRICT_ALL_TABLES, NO_ZERO_IN_DATE, NO_ZERO_DATE, ERROR_FOR_DIVISION_BY_ZERO, TRADITIONAL, NO_AUTO_CREATE_USER.

Strict Mode

A mode where at least one of `STRICT_TRANS_TABLES` or `STRICT_ALL_TABLES` is enabled is called *strict mode*.

With strict mode set (default from MariaDB 10.2.4), statements that modify tables (either transactional for `STRICT_TRANS_TABLES` or all for `STRICT_ALL_TABLES`) will fail, and an error will be returned instead. The IGNORE keyword can be used when strict mode is set to convert the error to a warning.

With strict mode not set (default in version <= MariaDB 10.2.3), MariaDB will automatically adjust invalid values, for example, truncating strings that are too long, or adjusting numeric values that are out of range, and produce a warning.

Statements that don't modify data will return a warning when adjusted regardless of mode.

SQL_MODE and Stored Programs

Stored programs and views always use the SQL_MODE that was active when they were created. This means that users can safely change session or global SQL_MODE; the stored programs they use will still work as usual.

It is possible to change session SQL_MODE within a stored program. In this case, the new SQL_MODE will be in effect only in the body of the current stored program. If it calls some stored procedures, they will not be affected by the change.

Some Information Schema tables (such as ROUTINES) and SHOW CREATE statements such as SHOW CREATE PROCEDURE show the SQL_MODE used by the stored programs.

Examples

This example shows how to get a readable list of enabled SQL_MODE flags:

```
SELECT REPLACE(@@SQL_MODE, ',', '\n');
+--------------------------------------------------------------------------+
| REPLACE(@@SQL_MODE, ',', '\n')                                           |
+--------------------------------------------------------------------------+
| STRICT_TRANS_TABLES
NO_ZERO_IN_DATE
NO_ZERO_DATE
NO_ENGINE_SUBSTITUTION |
+--------------------------------------------------------------------------+
```

Adding a new flag:

```
SET @@SQL_MODE = CONCAT(@@SQL_MODE, ',NO_ENGINE_SUBSTITUTION');
```

If the specified flag is already ON, the above example has no effect but does not produce an error.

How to unset a flag:

```
SET @@SQL_MODE = REPLACE(@@SQL_MODE, 'NO_ENGINE_SUBSTITUTION', '');
```

How to check if a flag is set:

```
SELECT @@SQL_MODE LIKE '%NO_ZERO_DATE%';
+-----------------------------------+
| @@SQL_MODE LIKE '%NO_ZERO_DATE%' |
+-----------------------------------+
|                                 1 |
+-----------------------------------+
```

Without and with strict mode:

```
CREATE TABLE strict (s CHAR(5), n TINYINT);

INSERT INTO strict VALUES ('MariaDB', '128');
Query OK, 1 row affected, 2 warnings (0.14 sec)

SHOW WARNINGS;
+---------+------+--------------------------------------------+
| Level   | Code | Message                                    |
+---------+------+--------------------------------------------+
| Warning | 1265 | Data truncated for column 's' at row 1     |
| Warning | 1264 | Out of range value for column 'n' at row 1 |
+---------+------+--------------------------------------------+
2 rows in set (0.00 sec)

SELECT * FROM strict;
+-------+------+
| s     | n    |
+-------+------+
| Maria |  127 |
+-------+------+

SET sql_mode='STRICT_TRANS_TABLES';

INSERT INTO strict VALUES ('MariaDB', '128');
ERROR 1406 (22001): Data too long for column 's' at row 1
```

Overriding strict mode with the IGNORE keyword:

```
INSERT IGNORE INTO strict VALUES ('MariaDB', '128');
Query OK, 1 row affected, 2 warnings (0.15 sec)
```

2.1.14.2.14 SQL_MODE-MSSQL

2.8 Copying Tables Between Different MariaDB Databases and MariaDB Servers

Contents

With MariaDB it's very easy to copy tables between different MariaDB databases and different MariaDB servers. This works for tables created with the Archive, Aria, CSV, InnoDB, MyISAM, MERGE, and XtraDB engines.

The normal procedures to copy a table is:

```
FLUSH TABLES db_name.table_name FOR EXPORT

# Copy the relevant files associated with the table

UNLOCK TABLES;
```

The table files can be found in datadir/databasename (you can execute `SELECT @@datadir` to find the correct directory). When copying the files, you should copy all files with the same table_name + various extensions. For example, for an Aria table of name foo, you will have files foo.frm, foo.MAI, foo.MAD and possibly foo.TRG if you have triggers.

If one wants to distribute a table to a user that doesn't need write access to the table and one wants to minimize the storage size of the table, the recommended engine to use is Aria or MyISAM as one can pack the table with aria_pack or myisampack respectively to make it notablly smaller. MyISAM is the most portable format as it's not dependent on whether the server settings are different. Aria and InnoDB require the same block size on both servers.

Copying Tables When the MariaDB Server is Down

The following storage engines support export without `FLUSH TABLES ... FOR EXPORT`, assuming the source server is down and the receiving server is not accessing the files during the copy.

Engine	Comment
Archive	
Aria	Requires clean shutdown. Table will automatically be fixed on the receiving server if `aria_chk --zerofill` was not run. If `aria_chk --zerofill` is run, then the table is immediately usable without any delays
CSV	
MyISAM	
MERGE	.MRG files can be copied even while server is running as the file only contains a list of tables that are part of merge.

Copying Tables Live From a Running MariaDB Server

For all of the above storage engines (Archive, Aria, CSV, MyISAM and MERGE), one can copy tables even from a live server under the following circumstances:

- You have done a `FLUSH TABLES` or `FLUSH TABLE table_name` for the specific table.
- The server is not accessing the tables during the copy process.

The advantage of FLUSH TABLES table_name FOR EXPORT is that the table is read locked until UNLOCK TABLES is executed.

Warning: If you do the above live copy, you are doing this on **your own risk** as if you do something wrong, the copied table is very likely to be corrupted. The original table will of course be fine.

An Efficient Way to Give Someone Else Access to a Read Only Table

If you want to give a user access to some data in a table for the user to use in their MariaDB server, you can do the following:

First let's create the table we want to export. To speed up things, we create this without any indexes. We use `TRANSACTIONAL=0` `ROW_FORMAT=DYNAMIC` for Aria to use the smallest possible row format.

```
CREATE TABLE new_table ... ENGINE=ARIA TRANSACTIONAL=0;
ALTER TABLE new_table DISABLE_KEYS;
# Fill the table with data:
INSERT INTO new_table SELECT * ...
FLUSH TABLE new_table WITH READ LOCK;

# Copy table data to some external location, like /tmp with something
# like cp /my/data/test/new_table.* /tmp/

UNLOCK TABLES;
```

Then we pack it and generate the indexes. We use a big sort buffer to speed up generating the index.

```
> ls -l /tmp/new_table.*
-rw-rw---- 1 mysql my 42396148 Sep 21 17:58 /tmp/new_table.MAD
-rw-rw---- 1 mysql my     8192 Sep 21 17:58 /tmp/new_table.MAI
-rw-rw---- 1 mysql my     1039 Sep 21 17:58 /tmp/new_table.frm
> aria_pack /tmp/new_table
Compressing /tmp/new_table.MAD: (922666 records)
- Calculating statistics
- Compressing file
46.07%
> aria_chk -rq --ignore-control-file --sort_buffer_size=1G /tmp/new_table
Recreating table '/tmp/new_table'
- check record delete-chain
- recovering (with sort) Aria-table '/tmp/new_table'
Data records: 922666
- Fixing index 1
State updated
> ls -l /tmp/new_table.*
-rw-rw---- 1 mysql my 26271608 Sep 21 17:58 /tmp/new_table.MAD
-rw-rw---- 1 mysql my 10207232 Sep 21 17:58 /tmp/new_table.MAI
-rw-rw---- 1 mysql my     1039 Sep 21 17:58 /tmp/new_table.frm
```

The procedure for MyISAM tables is identical, except that myisamchk doesn't have the `--ignore-control-file` option.

Copying InnoDB's Transportable Tablespaces

InnoDB's file-per-table tablespaces are transportable, which means that you can copy a file-per-table tablespace from one MariaDB Server to another server. See Copying Transportable Tablespaces for more information.

Importing Tables

Tables that use most storage engines are immediately usable when their files are copied to the new `datadir`.

However, this is not true for tables that use InnoDB. InnoDB tables have to be imported with ALTER TABLE ... IMPORT TABLESPACE. See Copying Transportable Tablespaces for more information.

3 High Availability & Performance Tuning

Information on replication, clustering, and multi-master solutions for MariaDB, as well as performance tuning.

MariaDB Replication

Documentation on standard primary and replica replication.

MariaDB Galera Cluster

MariaDB Galera Cluster is a virtually synchronous multi-master cluster.

Optimization and Tuning

Using indexes, writing better queries and adjusting variables for better performance.

Connection Redirection Mechanism in the MariaDB Client/Server Protocol

Connection Redirection Mechanism in the MariaDB Client/Server Protocol.

There are 2 related questions.

3.1 MariaDB Replication

The terms *master* and *slave* have historically been used in replication, but the terms terms *primary* and *replica* are now preferred. The old terms are used still used in parts of the documentation, and in MariaDB commands, although MariaDB 10.5 has begun the process of renaming. The documentation process is ongoing. See MDEV-18777 to follow progress on this effort.

Replication is a feature allowing the contents of one or more primary servers to be mirrored on one or more replica servers.

Replication Overview
Allow the contents of one or more primary servers to be mirrored on one or more replicas.

Replication Commands
List of replication-related commands.

Setting Up Replication
Getting replication working involves steps on both the primary server/s and the replica server/s.

Setting up a Replica with Mariabackup
Setting up a replica with Mariabackup.

Read-Only Replicas
Making replicas read-only.

Replication as a Backup Solution
Replication can be used to support the backup strategy.

Multi-Source Replication
Using replication with many masters.

Replication Threads
Types of threads that are used to enable replication.

Global Transaction ID
Improved replication using global transaction IDs.

Parallel Replication
Executing queries replicated from the primary in parallel on the replica.

Replication and Binary Log System Variables
Replication and binary log system variables.

Replication and Binary Log Status Variables
Replication and binary log status variables.

Binary Log
Contains a record of all changes to the databases, both data and structure

Unsafe Statements for Statement-based Replication
Statements that are not safe for statement-based replication.

Replication and Foreign Keys
Cascading deletes or updates based on foreign key relations are not written to the binary log

Relay Log
Event log created by the replica from the primary binary log.

Enhancements for START TRANSACTION WITH CONSISTENT SNAPSHOT
Enhancements for START TRANSACTION WITH CONSISTENT SNAPSHOT.

Group Commit for the Binary Log
Optimization when the server is run with innodb_flush_logs_at_trx_commit or sync_binlog.

Selectively Skipping Replication of Binlog Events
@@skip_replication and --replicate-events-marked-for-skip.

Binlog Event Checksums
Including a checksum in binlog events.

Binlog Event Checksum Interoperability
Replicating between servers with differing binlog checksum availability

Annotate_rows_log_event

Annotate_rows events accompany row events and describe the query which caused the row event.

Row-based Replication With No Primary Key

MariaDB improves on row-based replication of tables with no primary key

Replication Filters

Replication filters allow users to configure replicas to intentionally skip certain events.

Running Triggers on the Replica for Row-based Events

Running triggers on the replica for row-based events.

Semisynchronous Replication

Semisynchronous replication.

Using MariaDB Replication with MariaDB Galera Cluster

Information on using MariaDB replication with MariaDB Galera Cluster.

Delayed Replication

Specify that a replica should lag behind the primary by (at least) a specified amount of time.

Replication When the Primary and Replica Have Different Table Definitions

Slave and the primary table definitions can differ while replicating.

Restricting Speed of Reading Binlog from Primary by a Replica

The read_binlog_speed_limit option can be used to reduce load on the primary.

Changing a Replica to Become the Primary

How to change a replica to primary and old primary as a replica for the new primary.

Replication with Secure Connections

Enabling TLS encryption in transit for MariaDB replication.

Obsolete Replication Information

This section is for replication-related items that are obsolete

There are 23 related questions.

3.1.1 Replication Overview

Contents

Replication is a feature allowing the contents of one or more servers (called primaries) to be mirrored on one or more servers (called replicas).

You can exert control over which data to replicate. All databases, one or more databases, or tables within a database can each be selectively replicated.

The main mechanism used in replication is the binary log. If binary logging is enabled, all updates to the database (data manipulation and data definition) are written into the binary log as binlog events. Replicas read the binary log from each primary in order to access the data to replicate. A relay log is created on the replica, using the same format as the binary log, and this is used to perform the replication. Old relay log files are removed when no longer needed.

A replica server keeps track of the position in the primary's binlog of the last event applied on the replica. This allows the replica server to re-connect and resume from where it left off after replication has been temporarily stopped. It also allows a replica to disconnect, be cloned and then have the new replica resume replication from the same primary.

Primaries and replicas do not need to be in constant communication with each other. It's quite possible to take servers offline or disconnect from the network, and when they come back, replication will continue where it left off.

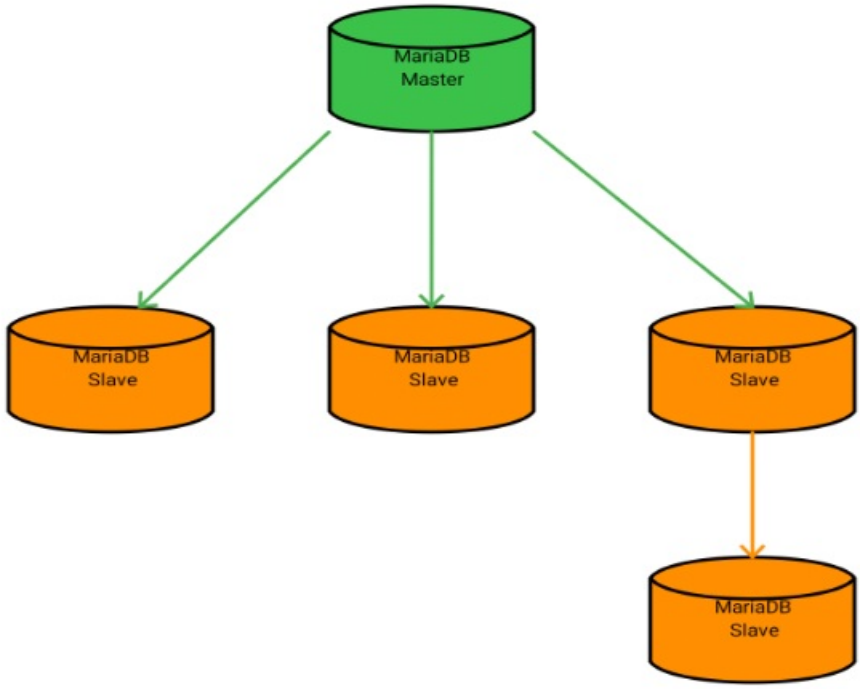
Replication Uses

Replication is used in a number of common scenarios. Uses include:

- Scalability. By having one or more replicas, reads can be spread over multiple servers, reducing the load on the primary. The most common scenario for a high-read, low-write environment is to have one primary, where all the writes occur, replicating to multiple replicas, which handle most of the reads.
- Data analysis. Analyzing data may have too much of an impact on a primary server, and this can similarly be handled on a replica, while the primary continues unaffected by the extra load.
- Backup assistance. Backups can more easily be run if a server is not actively changing the data. A common scenario is to replicate the data to a replica, which is then disconnected from the primary with the data in a stable state. Backup is then performed from this server. See Replication as a Backup Solution.
- Distribution of data. Instead of being connected to a remote primary, it's possible to replicate the data locally and work from this data instead.

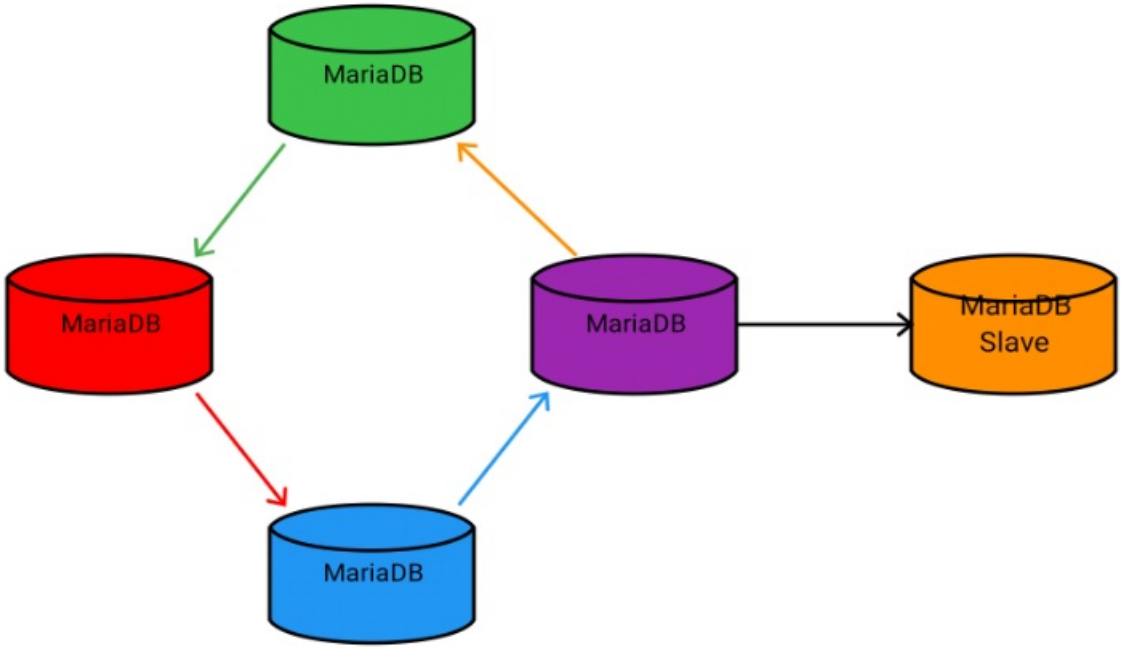
Common Replication Setups

Standard Replication



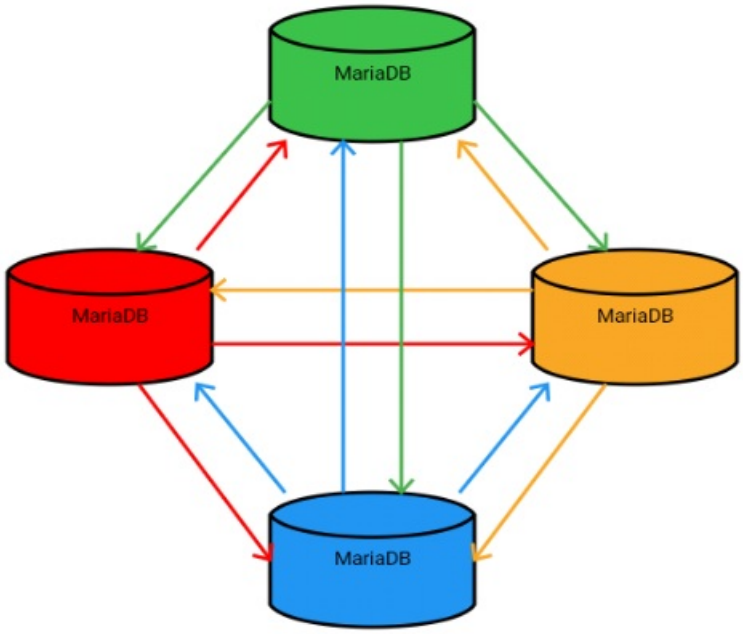
- Provides infinite read scale out.
- Provides high-availability by upgrading replica to primary.

Ring Replication



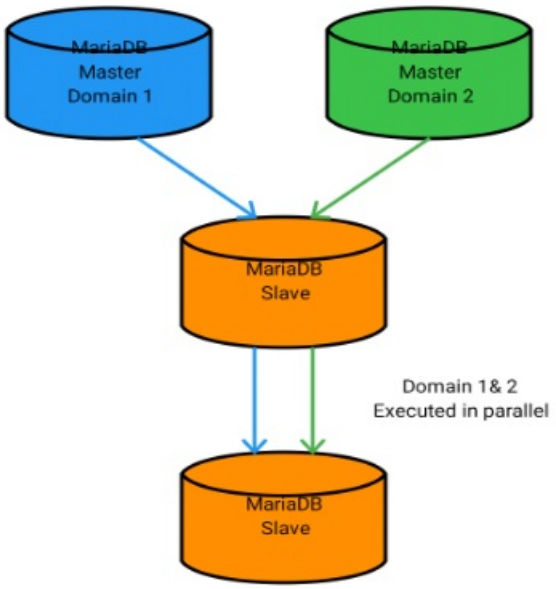
- Provides read and write scaling.
- Doesn’t handle conflicts.
- If one primary fails, replication stops.

Star Replication



- Provides read and write scaling.
- Doesn’t handle conflicts.
- Have to use replication filters to avoid duplication of data.

Multi-Source Replication



- Allows you to combine data from different sources.
- Different domains executed independently in parallel on all replicas.

Cross-Version Replication Compatibility

The following table describes replication compatibility between different MariaDB Server versions. In general, the replica should always be at least equivalent in version to the primary:

Primary→ / Replica ↓	MariaDB 10.3	MariaDB 10.4	MariaDB 10.5	MariaDB 10.6	MariaDB 10.11
MariaDB 10.3	✓	□	□	□	□
MariaDB 10.4	✓	✓	□	□	□
MariaDB 10.5	✓	✓	✓	□	□
MariaDB 10.6	✓	✓	✓	✓	□
MariaDB 10.11	✓	✓	✓	✓	✓

- ✓: This combination is supported.
- □: This combination is **not** supported.

For replication compatibility details between MariaDB and MySQL, see MariaDB versus MySQL - Compatibility: Replication Compatibility.

1.1.1.2.5 Replication Commands

3.1.3 Setting Up Replication

The terms *master* and *slave* have historically been used in replication, but the terms terms *primary* and *replica* are now preferred. The old terms are used still used in parts of the documentation, and in MariaDB commands, although MariaDB 10.5 has begun the process of renaming. The documentation process is ongoing. See MDEV-18777 to follow progress on this effort.

Contents

Getting replication working involves steps on both the master server/s and steps on the slave server/s.

MariaDB 10.0 introduced replication with global transaction IDs. These have a number of benefits, and it is generally recommended to use this feature from MariaDB 10.0.

Setting up a Replication Slave with Mariabackup

If you would like to use Mariabackup to set up a replication slave, then you might find the information at Setting up a Replication Slave with Mariabackup helpful.

Versions

In general, when replicating across different versions of MariaDB, it is best that the master is an older version than the slave. MariaDB versions are usually backward compatible, while of course older versions cannot always be forward compatible. See also Replicating from MySQL Master to MariaDB Slave.

Configuring the Master

- Enable binary logging if it's not already enabled. See Activating the Binary Log and Binary log formats for details.
- Give the master a unique server_id. All slaves must also be given a server_id. This can be a number from 1 to 2^{32}-1, and must be unique for each server in the replicating group.
- Specify a unique name for your replication logs with --log-basename. If this is not specified your host name will be used and there will be problems if the hostname ever changes.
- Slaves will need permission to connect and start replicating from a server. Usually this is done by creating a dedicated slave user, and granting that user permission only to replicate (REPLICATION SLAVE permission).

Example Enabling Replication for MariaDB

Add the following into your my.cnf file and restart the database.

```
[mariadb]
log-bin
server_id=1
log-basename=master1
binlog-format=mixed
```

The server id is a unique number for each MariaDB/MySQL server in your network. binlog-format specifies how your statements are logged. This mainly affects the size of the binary log that is sent between the Master and the Slaves.

Then execute the following SQL with the `mysql` command line client:

```
CREATE USER 'replication_user'@'%' IDENTIFIED BY 'bigs3cret';
GRANT REPLICATION SLAVE ON *.* TO 'replication_user'@'%';
```

Example Enabling Replication for MySQL

If you want to enable replication from MySQL to MariaDB, you can do it in almost the same way as between MariaDB servers. The main difference is that MySQL doesn't support `log-basename`.

```
[mysqld]
log-bin
server_id=1
```

Settings to Check

There are a number of options that may impact or break replication. Check the following settings to avoid problems.

- skip-networking. If `skip-networking=1`, the server will limit connections to localhost only, and prevent all remote slaves from connecting.
- bind-address. Similarly, if the address the server listens for TCP/IP connections is 127.0.0.1 (localhost), remote slaves connections will fail.

Configuring the Slave

- Give the slave a unique server_id. All servers, whether masters or slaves, are given a server_id. This can be a number from 1 to 2^{32}-1, and must be unique for each server in the replicating group. The server will need to be restarted in order for a change in this option to take effect.

Getting the Master's Binary Log Co-ordinates

Now you need prevent any changes to the data while you view the binary log position. You'll use this to tell the slave at exactly which point it should start replicating from.

- On the master, flush and lock all tables by running `FLUSH TABLES WITH READ LOCK`. Keep this session running - exiting it will release the lock.
- Get the current position in the binary log by running `SHOW MASTER STATUS`:

```
SHOW MASTER STATUS;
+--------------------+----------+--------------+------------------+
| File               | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+--------------------+----------+--------------+------------------+
| master1-bin.000096 |      568 |              |                  |
+--------------------+----------+--------------+------------------+
```

- Record the *File* and *Position* details. If binary logging has just been enabled, these will be blank.
- Now, with the lock still in place, copy the data from the master to the slave. See Backup, Restore and Import for details on how to do this.
- Note for live databases: You just need to make a local copy of the data, you don't need to keep the master locked until the slave has imported the data.
- Once the data has been copied, you can release the lock on the master by running UNLOCK TABLES.

```
UNLOCK TABLES;
```

Start the Slave

- Once the data has been imported, you are ready to start replicating. Begin by running a CHANGE MASTER TO, making sure that *MASTER_LOG_FILE* matches the file and *MASTER_LOG_POS* the position returned by the earlier SHOW MASTER STATUS. For example:

```
CHANGE MASTER TO
  MASTER_HOST='master.domain.com',
  MASTER_USER='replication_user',
  MASTER_PASSWORD='bigs3cret',
  MASTER_PORT=3306,
  MASTER_LOG_FILE='master1-bin.000096',
  MASTER_LOG_POS=568,
  MASTER_CONNECT_RETRY=10;
```

If you are starting a slave against a fresh master that was configured for replication from the start, then you don't have to specify `MASTER_LOG_FILE` and `MASTER_LOG_POS`.

Use Global Transaction Id (GTID)

MariaDB starting with 10.0

MariaDB 10.0 introduced global transaction IDs (GTIDs) for replication. It is generally recommended to use (GTIDs) from MariaDB 10.0, as this has a number of benefits. All that is needed is to add the `MASTER_USE_GTID` option to the `CHANGE MASTER` statement, for example:

```
CHANGE MASTER TO MASTER_USE_GTID = slave_pos
```

See Global Transaction ID for a full description.

- Now start the slave with the `START SLAVE` command:

```
START SLAVE;
```

- Check that the replication is working by executing the `SHOW SLAVE STATUS` command:

```
SHOW SLAVE STATUS \G
```

- If replication is working correctly, both the values of `Slave_IO_Running` and `Slave_SQL_Running` should be `Yes`:

```
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
```

Replicating from MySQL Master to MariaDB Slave

- Replicating from MySQL 5.5 to MariaDB 5.5+ should just work. When using a MariaDB 10.2+ as a slave, it may be necessary to set binlog_checksum to NONE.
- Replicating from MySQL 5.6 without GTID to MariaDB 10+ should work.
- Replication from MySQL 5.6 with GTID, binlog_rows_query_log_events and ignorable events works starting from MariaDB 10.0.22 and MariaDB 10.1.8. In this case MariaDB will remove the MySQL GTIDs and other unneeded events and instead adds its own GTIDs.

2.3.4.7 Setting up a Replica with Mariabackup

3.1.5 Read-Only Replicas

Contents

1. Older MariaDB Versions

A common replication setup is to have the replicas read-only to ensure that no one accidentally updates them. If the replica has binary logging enabled and gtid_strict_mode is used, then any update that causes changes to the binary log will stop replication.

When the variable `read_only` is set to 1, no updates are permitted except from users with the SUPER privilege (<= MariaDB 10.5.1) or READ ONLY ADMIN privilege (>= MariaDB 10.5.2) or replica servers updating from a primary. Inserting rows to log tables, updates to temporary tables and OPTIMIZE TABLE or ANALYZE TABLE statements on temporary tables are excluded from this limitation.

If read_only is set to 1, then the SET PASSWORD statement is limited only to users with the SUPER privilege (<= MariaDB 10.5.1) or READ ONLY ADMIN privilege (>= MariaDB 10.5.2).

Attempting to set the `read_only` variable to 1 will fail if the current session has table locks or transactions pending.

The statement will wait for other sessions that hold table locks. While the attempt to set read_only is waiting, other requests for table locks or transactions will also wait until read_only has been set.

From MariaDB 10.3.19, some issues related to read only replicas are fixed:

- CREATE, DROP, ALTER, INSERT and DELETE of temporary tables are not logged to binary log, even in statement or mixed mode. With earlier MariaDB versions, one can avoid the problem with temporary tables by using binlog_format=ROW in which cases temporary tables are never logged.
- Changes to temporary tables created during `read_only` will not be logged even after `read_only` mode is disabled (for example if the replica is promoted to a primary).
- The admin statements ANALYZE, CHECK, OPTIMIZE and REPAIR will not be logged to the binary log under read-only.

Older MariaDB Versions

If you are using an older MariaDB version with read-only replicas and binary logging enabled on the replica, and you need to do some changes but don't want to have them logged to the binary log, the easiest way to avoid the logging is to disable binary logging while running as root during maintenance:

```
set sql_log_bin=0;
alter table test engine=rocksdb;
```

The above changes the test table on the replica to rocksdb without registering the change in the binary log.

3.1.6 Replication as a Backup Solution

Replication can be used to support the backup strategy.

Replication alone is *not* sufficient for backup. It assists in protecting against hardware failure on the primary server, but does not protect against data loss. An accidental or malicious `DROP DATABASE` or `TRUNCATE TABLE` statement will be replicated onto the replica as well. Care needs to be taken to prevent data getting out of sync between the primary and the replica.

The terms *master* and *slave* have historically been used in replication, but the terms terms *primary* and *replica* are now preferred. The old terms are used still used in parts of the documentation, and in MariaDB commands, although MariaDB 10.5 has begun the process of renaming. The documentation process is ongoing. See MDEV-18777 to follow progress on this effort.

Replication is most commonly used to support backups as follows:

- A primary server replicates to a replica
- Backups are then run off the replica without any impact on the primary.

Backups can have a significant effect on a server, and a high-availability primary may not be able to be stopped, locked or simply handle the extra load of a backup. Running the backup from a replica has the advantage of being able to shutdown or lock the replica and perform a backup without any impact on the primary server.

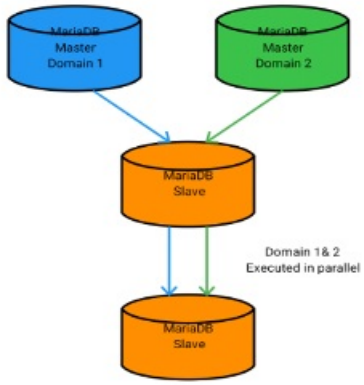
Note that when backing up off a replica server, it is important to ensure that the servers keep the data in sync. See for example Replication and Foreign Keys for a situation when identical statements can result in different data on a replica and a primary.

3.1.7 Multi-Source Replication

The terms *master* and *slave* have historically been used in replication, but the terms terms *primary* and *replica* are now preferred. The old terms are used still used in parts of the documentation, and in MariaDB commands, although MariaDB 10.5 has begun the process of renaming. The documentation process is ongoing. See MDEV-18777 to follow progress on this effort.

Contents

Multi-source replication means that one server has many primaries from which it replicates.



New Syntax

You specify which primary connection you want to work with by either specifying the connection name in the command or setting default_master_connection to the connection you want to work with.

The connection name may include any characters and should be less than 64 characters. Connection names are compared without regard to case (case insensitive). You should preferably keep the connection name short as it will be used as a suffix for relay logs and primary info index files.

The new syntax introduced to handle many connections:

- `CHANGE MASTER ['connection_name'] TO ...` . This creates or modifies a connection to a primary.
- `FLUSH RELAY LOGS ['connection_name']`
- `MASTER_POS_WAIT(....,['connection_name'])`
- `RESET SLAVE ['connection_name'] [ALL]` . This is used to reset a replica's replication position or to remove a replica permanently.
- `SHOW RELAYLOG ['connection_name'] EVENTS`
- `SHOW SLAVE ['connection_name'] STATUS`
- `SHOW ALL SLAVES STATUS`
- `START SLAVE ['connection_name'...]]`
- `START ALL SLAVES ...`
- `STOP SLAVE ['connection_name'] ...`
- `STOP ALL SLAVES ...`

The original old-style connection is an empty string `''` . You don't have to use this connection if you don't want to.

You create new primary connections with CHANGE MASTER.
You delete the connection permanently with RESET SLAVE 'connection_name' ALL.

Replication Variables for Multi-Source

The new replication variable default_master_connection specifies which connection will be used for commands and variables if you don't specify a connection. By default this is `''` (the default connection name).

The following replication variables are local for the connection. (In other words, they show the value for the `@@default_master_connection` connection). We are working on making all the important ones local for the connection.

Type	Name	Description
Variable	max_relay_log_size	Max size of relay log. Is set at startup to `max_binlog_size` if 0
Variable	replicate_do_db	Tell the replica to restrict replication to updates of tables whose names appear in the comma-separated list. For statement-based replication, only the default database (that is, the one selected by USE) is considered, not any explicitly mentioned tables in the query. For row-based replication, the actual names of table(s) being updated are checked.
Variable	replicate_do_table	Tells the replica to restrict replication to tables in the comma-separated list
Variable	replicate_ignore_db	Tell the replica to restrict replication to updates of tables whose names do not appear in the comma-separated list. For statement-based replication, only the default database (that is, the one selected by USE) is considered, not any explicitly mentioned tables in the query. For row-based replication, the actual names of table(s) being updated are checked.

Variable	replicate_ignore_table	Tells the replica thread to not replicate any statement that updates the specified table, even if any other tables might be updated by the same statement.
Variable	replicate_rewrite_db	From MariaDB 10.11. Allows one to configure a replica to rewrite database names. It uses the format primary_database->replica_database. If a replica encounters a binary log event in which the default database (i.e. the one selected by the USE statement) is primary_database, then the replica will apply the event in replica_database instead.
Variable	replicate_wild_do_table	Tells the replica thread to restrict replication to statements where any of the updated tables match the specified database and table name patterns.
Variable	replicate_wild_ignore_table	Tells the replica thread to not replicate to the tables that match the given wildcard pattern.
Status	Slave_heartbeat_period	How often to request a heartbeat packet from the primary (in seconds).
Status	Slave_received_heartbeats	How many heartbeats we have got from the primary.
Status	Slave_running	Shows if the replica is running. YES means that the sql thread and the IO thread are active. No means either one is not running. '' means that `@@default_master_connection` doesn't exist.
Variable	Sql_slave_skip_counter	How many entries in the replication log that should be skipped (mainly used in case of errors in the log).

You can access all of the above variables with either `SESSION` or `GLOBAL`.

Note that in contrast to MySQL, all variables always show the correct active value!

Example:

```
set @@default_master_connection='';
show status like 'Slave_running';
set @@default_master_connection='other_connection';
show status like 'Slave_running';
```

If `@@default_master_connection` contains a non existing name, you will get a warning.

All other primary-related variables are global and affect either only the '' connections or all connections. For example, Slave_retried_transactions now shows the total number of retried transactions over all replicas.

If you need to set gtid_slave_pos you need to set this for all primaries at the same time.

New status variables:

Name	Description
`Com_start_all_slaves`	Number of executed `START ALL SLAVES` commands.
`Com_start_slave`	Number of executed `START SLAVE` commands. This replaces `Com_slave_start`.
`Com_stop_slave`	Number of executed `STOP SLAVE` commands. This replaces `Com_slave_stop`.
`Com_stop_all_slaves`	Number of executed `STOP ALL SLAVES` commands.

`SHOW ALL SLAVES STATUS` has the following new columns:

Name	Description
`Connection_name`	Name of the primary connection. This is the first variable.
`Slave_SQL_State`	State of SQL thread.
`Retried_transactions`	Number of retried transactions for this connection.
`Max_relay_log_size`	Max relay log size for this connection.
`Executed_log_entries`	How many log entries the replica has executed.
`Slave_received_heartbeats`	How many heartbeats we have got from the primary.
`Slave_heartbeat_period`	How often to request a heartbeat packet from the primary (in seconds).

New Files

The basic principle of the new files used by multi source replication is that they have the same name as the original relay log files suffixed with `connection_name` before the extension. The main exception is the file that holds all connection is named as the normal `master-info-file` with a `multi-` prefix.

When you are using multi source, the following new files are created:

Name	Description
`multi-master-info-file`	The `master-info-file` (normally `master.info`) with a `multi-` prefix. This contains all primary connections in use.
`master-info-file`-connection_name`.extension`	Contains the current primary position for what's applied to in the replica. Extension is normally `.info`
`relay-log`-connection_name`.xxxxx`	The relay-log name with a connection_name suffix. The xxxxx is the relay log number. This contains the replication data read from the primary.
`relay-log-index`-connection_name`.extension`	Contains the name of the active `relay-log`-connection_name`.xxxxx` files. Extension is normally `.index`
`relay-log-info-file`-connection_name`.extension`	Contains the current primary position for the relay log. Extension is normally `.info`

When creating the file, the connection name is converted to lower case and all special characters in the connection name are converted, the same way as MySQL table names are converted. This is done to make the file name portable across different systems.

Hint:

Instead of specifying names for `mysqld` with --relay-log, --relay-log-index, --general-log-file, --slow-query-log-file, --log-bin and --log-bin-index, you can just specify --log-basename and all the other variables are set with this as a prefix.

Other Things

- All error messages from a replica with a connection name, that are written to the error log, are prefixed with `Master 'connection_name':`. This makes it easy to see from where an error originated.
- Errors `ER_MASTER_INFO` and `WARN_NO_MASTER_INFO` now includes connection_name.
- There is no conflict resolution. The assumption is that there are no conflicts in data between the different primaries.
- All executed commands are stored in the normal binary log (nothing new here).
- If the server variable `log_warnings` > 1 then you will get some information in the log about how the multi-master-info file is updated (mainly for debugging).
- The output of SHOW ALL SLAVES STATUS has one more column than `SHOW SLAVE STATUS`, since it includes the `connection_name` column.
- `RESET SLAVE` now deletes all relay-log files.

replicate-... Variables

- One can set the values for the `replicate-...` variables from the command line or in `my.cnf` for a given connection by prefixing the variable with the connection name.
- If one doesn't use any connection name prefix for a `replicate..` variable, then the value will be used as the default value for all connections that don't have a value set for this variable.

Example:

```
mysqld --main_connection.replicate_do_db=main_database --replicate_do_db=other_database
```

The have sets the `replicate_do_db` variable to `main_database` for the connection named `main_connection`. All other connections will use the value `other_database`.

One can also use this syntax to set `replicate-rewrite-db` for a given connection.

Typical Use Cases

- You are partitioning your data over many primaries and would like to get it all together on one machine to do analytical queries on all data.
- You have many databases spread over many MariaDB/MySQL servers and would like to have all of them on one machine as an extra backup.
- In a Galera cluster the default replication filter rules like `replicate-do-db` do not apply to replication connections,

but also to Galera write set applier threads. By using a named multi-primary replication connection instead, even when replicating from just one primary into the cluster, the primary-replica replication rules can be kept separate from the Galera intra-node replication traffic.

Limitations

- Each active connection will create 2 threads (as is normal for MariaDB replication).
- You should ensure that all primaries have different `server-id`'s. If you don't do this, you will get into trouble if you try to replicate from the multi-source replica back to your primaries.
- One can change max_relay_log_size for any active connection, but new connections will always use the server startup value for `max_relay_log_size`, which can't be changed at runtime.
- Option innodb-recovery-update-relay-log (xtradb feature to store and restore relay log position for replicas) only works for the default connection ''. As this option is not really safe and can easily cause loss of data if you use storage engines other than InnoDB, we don't recommend this option be used.
- slave_net_timeout affects all connections. We don't check anymore if it's less than Slave_heartbeat_period, as this doesn't make sense in a multi-source setup.

Incompatibilities with MariaDB/MySQL 5.5

- max_relay_log_size is now (almost) a normal variable and not automatically changed if max_binlog_size is changed. To keep things compatible with old config files, we set it to `max_binlog_size` at startup if its value is 0.
- You can now access replication variables that depend on the active connection with either `GLOBAL` or `SESSION`.
- We only write information about relay log positions for recovery if innodb-recovery-update-relay-log is set.
- Slave_retried_transactions now shows the total count of retried transactions over all replicas.
- The status variable `Com_slave_start` is replaced with Com_start_slave.
- The status variable `Com_slave_stop` is replaced with Com_stop_slave.
- `FLUSH RELAY LOGS` are not replicated anymore. This is not safe as connection names may be different on the replica.

3.1.8 Replication Threads

The terms *master* and *slave* have historically been used in replication, but the terms terms *primary* and *replica* are now preferred. The old terms are used still used in parts of the documentation, and in MariaDB commands, although MariaDB 10.5 has begun the process of renaming. The documentation process is ongoing. See MDEV-18777 to follow progress on this effort.

Contents

MariaDB's replication implementation requires several types of threads.

Threads on the Primary

The primary usually only has one type of replication-related thread: the binary log dump thread.

If semisynchronous replication is enabled, then the primary also has an ACK receiver thread.

Binary Log Dump Thread

The binary log dump thread runs on the primary and dumps the binary log to the replica. This thread can be identified by running the SHOW PROCESSLIST statement and finding the thread where the thread command is `"Binlog Dump"`.

The primary creates a separate binary log dump thread for each replica connected to the primary. You can identify which replicas are connected to the primary by executing the SHOW SLAVE HOSTS statement.

Binary Log Dump Threads and the Shutdown Process

When a primary server is shutdown and it goes through the normal shutdown process, the primary kills client threads in random order. By default, the primary also considers its binary log dump threads to be regular client threads. As a consequence, the binary log dump threads can be killed while client threads still exist, and this means that data can be written on the primary during a normal shutdown that won't be replicated. This is true even if semi-synchronous replication is being used. Data is not lost, it is stored in the primary server's binary log. The replicas on reconnection, after the primary server restarts, will resume at the exact position they where killed off during the primary shutdown. No data is lost.

In MariaDB 10.4 and later, this problem can be solved by shutting down the server using either the mariadb-admin utility or the SHUTDOWN command, and providing a special option.

For example, this problem can be solved by shutting down the server with the mariadb-admin utility and by providing the `--wait-for-all-slaves` option to the utility and by executing the `shutdown` command with the utility:

```
mariadb-admin --wait-for-all-slaves shutdown
```

Or this problem can be solved by shutting down the server with the SHUTDOWN command and by providing the `WAIT FOR ALL SLAVES` option to the command:

```
SHUTDOWN WAIT FOR ALL SLAVES;
```

When one of these special options is provided, the server only kills its binary log dump threads after all client threads have been killed, and it only completes the shutdown after the last binary log has been sent to all connected replicas.

In MariaDB 10.4 and later, it is still not possible to enable this behavior by default. This means that this behavior is currently inaccessible when shutting down the server using tools like systemd or sysVinit.

In MariaDB 10.3 and before, it is recommended to manually switchover replicas to a new primary before shutting down the old primary.

ACK Receiver Thread

When semisynchronous replication is enabled, semisynchronous replicas send acknowledgements (ACKs) to their primary to confirm that they have received some transaction. The primary creates an ACK receiver thread to receive these ACKs.

Threads on the Replica

The replica has three types of replication-related threads: the replica I/O thread, the replica SQL thread, and worker threads, which are only applicable when parallel replication is in use.

When multi-source replication is in use, each independent replication connection has its own replica threads of each type.

Replica I/O Thread

The replica's I/O thread receives the binary log events from the primary and writes them to its relay log.

Binary Log Position

The binary log position of the replica's I/O thread can be checked by executing the SHOW SLAVE STATUS statement. It will be shown as the `Master_Log_File` and `Read_Master_Log_Pos` columns.

The binary log position of the replica's I/O thread can be set by setting the MASTER_LOG_FILE and MASTER_LOG_POS options with the CHANGE MASTER statement.

The binary log position of the replica's I/O thread and the values of most other CHANGE MASTER options are written to either the default `master.info` file or the file that is configured by the master_info_file option. The replica's I/O thread keeps this binary log position updated as it downloads events only when the MASTER_USE_GTID option is set to `NO`. Otherwise the file is not updated on a per event basis. See CHANGE MASTER TO: Option Persistence for more information.

Replica SQL Thread

The replica's SQL thread reads events from the relay log. What it does with them depends on whether parallel replication is

in use. If parallel replication is not in use, then the SQL thread applies the events to its local copy of the data. If parallel replication is in use, then the SQL thread hands off the events to its worker threads to apply in parallel.

Relay Log Position

The relay log position of the replica's SQL thread can be checked by executing the SHOW SLAVE STATUS statement. It will be shown as the `Relay_Log_File` and `Relay_Log_Pos` columns.

The relay log position of the replica's SQL thread can be set by setting the RELAY_LOG_FILE and RELAY_LOG_POS options with the CHANGE MASTER statement.

The relay log position of the replica's SQL thread is written to either the default `relay-log.info` file or the file that is configured by the relay_log_info_file system variable. The replica's SQL thread keeps this relay log position updated as it applies events. See CHANGE MASTER TO: Option Persistence for more information.

Binary Log Position

The corresponding binary log position of the current relay log position of the replica's SQL thread can be checked by executing the SHOW SLAVE STATUS statement. It will be shown as the `Relay_Master_Log_File` and `Exec_Master_Log_Pos` columns.

GTID Position

If the replica is replicating binary log events that contain GTIDs, then the replica's's SQL thread will write every GTID that it applies to the mysql.gtid_slave_pos table. This GTID can be inspected and modified through the gtid_slave_pos system variable.

If the replica has the log_slave_updates system variable enabled and if the replica has the binary log enabled, then every write by the replica's SQL thread will also go into the replica's binary log. This means that GTIDs of replicated transactions would be reflected in the value of the gtid_binlog_pos system variable.

See CHANGE MASTER TO: GTID Persistence for more information.

Worker Threads

When parallel replication is in use, then the SQL thread hands off the events to its worker threads to apply in parallel.

3.1.9 Global Transaction ID

> The terms *master* and *slave* have historically been used in replication, but the terms terms *primary* and *replica* are now preferred. The old terms are used still used in parts of the documentation, and in MariaDB commands, although MariaDB 10.5 has begun the process of renaming. The documentation process is ongoing. See MDEV-18777 to follow progress on this effort.

Contents

Note that MariaDB and MySQL have different GTID implementations, and that these are not compatible with each other. MariaDB can be a replica for a MySQL primary but MySQL cannot be a replica for a MariaDB primary.

Overview

MariaDB replication in general works as follows (see Replication overview for more information):

On a master server, all updates to the database (DML and DDL) are written into the binary log as binlog events. A replica server connects to the primary and reads the binlog events, then applies the events locally to replicate the same changes as done on the primary. A server can be both a primary and a replica at the same time, and it is thus possible for binlog events to replicated through multiple levels of servers.

A replica server keeps track of the position in the primary's binlog of the last event applied on the replica. This allows the replica server to re-connect and resume from where it left off after replication has been temporarily stopped. It also allows a replica to disconnect, be cloned and then have the new replica resume replication from the same primary.

Global transaction ID introduces a new event attached to each event group in the binlog. (An event group is a collection of events that are always applied as a unit. They are best thought of as a "transaction", though they also include non-transactional DML statements, as well as DDL). As an event group is replicated from primary server to replica server, the global transaction ID is preserved. Since the ID is globally unique across the entire group of servers, this makes it easy to uniquely identify the same binlog events on different servers that replicate each other (this was not easily possible before MariaDB 10.0.2).

Benefits

Using global transaction ID provides two main benefits:

1. Easy to change a replica server to connect to and replicate from a different primary server.

The replica remembers the global transaction ID of the last event group applied from the old primary. This makes it easy to know where to resume replication on the new primary, since the global transaction IDs are known throughout the entire replication hierarchy. This is not the case when using old-style replication; in this case the replica knows only the specific file name and offset of the old primary server of the last event applied. There is no simple way to guess from this the correct file name and offset on a new primary.

2. The state of the replica is recorded in a crash-safe way.

The replica keeps track of its current position (the global transaction ID of the last transaction applied) in the mysql.gtid_slave_pos system table. If this table is using a transactional storage engine (such as InnoDB, which is the default), then updates to the state are done in the same transaction as the updates to the data. This makes the state crash-safe; if the replica server crashes, crash recovery on restart will make sure that the recorded replication position matches the changes that were actually replicated. This is not the case for old-style replication, where the state is recorded in a file relay-log.info, which is updated independently of the actual data changes and can easily get out of sync if the replica server crashes. (This works for DML to transactional tables; non-transactional tables and DDL in general are not crash-safe in MariaDB.)

Because of these two benefits, it is generally recommended to use global transaction ID for any replication setups based on MariaDB 10.0.2 or later. However, old-style replication continues to work as always, so there is no pressing need to change existing setups. Global transaction ID integrates smoothly with old-style replication, and the two can be used freely together in the same replication hierarchy. There is no special configuration needed of the server to start using global transaction ID. However, it must be explicitly set for a replica server with the appropriate CHANGE MASTER option; by default old-style replication is used by a replication replica, to maintain backwards compatibility.

Implementation

A global transaction ID, or GTID for short, consists of three numbers separated with dashes '-'. For example:

```
0-1-10
```

- The first number 0 is the domain ID, which is specific for global transaction ID (more on this below). It is a 32-bit unsigned integer.
- The second number is the server ID, the same as is also used in old-style replication. It is a 32-bit unsigned integer.
- The third number is the sequence number. This is a 64-bit unsigned integer that is monotonically increasing for each new event group logged into the binlog.

The server ID is set to the server ID of the server where the event group is first logged into the binlog. The sequence number is increased on a server for every event group logged. Since server IDs must be unique for every server, this makes the (server_id, sequence_number) pair, and hence the whole GTID, globally unique.

Using a 64-bit number provides ample range that there should be no risk of it overflowing in the foreseeable future. However, one should not artificially (by setting `gtid_seq_no`) inject a GTID with a very high sequence number close to the limit of 64-bit.

The Domain ID

When events are replicated from a primary server to a replica server, the events are always logged into the replica's binlog in the same order that they were read from the primary's binlog. Thus, if there is only ever a single primary server receiving (non-replication) updates at a time, then the binlog order will be identical on every server in the replication hierarchy.

This consistent binlog order is used by the replica to keep track of its current position in the replication. Basically, the replica remembers the GTID of the last event group replicated from the primary. When reconnecting to a primary, whether the same one or a new one, it sends this GTID position to the primary, and the primary starts sending events from the first event after the corresponding event group.

However, if user updates are done independently on multiple servers at the same time, then in general it is not possible for binlog order to be identical across all servers. This can happen when using multi-source replication, with multi-primary ring topologies, or just if manual updates are done on a replica that is replicating from active primary. If the binlog order is different on the new primary from the order on the old primary, then it is not sufficient for the replica to keep track of a single GTID to completely record the current state.

The domain ID, the first component of the GTID, is used to handle this.

In general, the binlog is not a single ordered stream. Rather, it consists of a number of different streams, each one identified by its own domain ID. Within each stream, GTIDs always have the same order in every server binlog. However, different streams can be interleaved in different ways on different servers.

A replica server then keeps track of its replication position by recording the last GTID applied within each replication stream. When connecting to a new primary, the replica can start replication from a different point in the binlog for each domain ID.

For more details on using multi-primary setups and multiple domain IDs, see Use with multi-source replication and other multi-primary setups.

Simple replication setups only have a single primary being updated by the application at any one time. In such setups, there is only a single replication stream needed. Then domain ID can be ignored, and left as the default of 0 on all servers.

Using Global Transaction IDs

Global transaction ID is enabled automatically. Each event group logged to the binlog receives a GTID event, as can be seen with mariadb-binlog or SHOW BINLOG EVENTS.

The replica automatically keeps track of the GTID of the last applied event group, as can be seen from the gtid_slave_pos variable:

```
SELECT @@GLOBAL.gtid_slave_pos
0-1-1
```

When a replica connects to a primary, it can use either global transaction ID or old-style filename/offset to decide where in the primary binlogs to start replicating from. To use global transaction ID, use the CHANGE MASTER *master_use_gtid* option:

```
CHANGE MASTER TO master_use_gtid = { slave_pos | current_pos | no }
```

A replica is configured to use GTID by `CHANGE MASTER TO master_use_gtid=slave_pos`. When the replica connects to the primary, it will start replication at the position of the last GTID replicated to the replica, which can be seen in the variable gtid_slave_pos. Since GTIDs are the same across all replication servers, the replica can then be pointed to a different primary, and the correct position will be determined automatically.

But suppose that we set up two servers A and B and let A be the primary and B the replica. It runs for a while. Then at some point we take down A, and B becomes the new primary. Then later we want to add A back, this time as a replica.

Since A was never a replica before, it does not have any prior replicated GTIDs, and gtid_slave_pos will be empty. To allow A to be added as a replica automatically, `master_use_gtid=current_pos` can be used. This will connect using the value of the variable gtid_current_pos instead of gtid_slave_pos, which also takes into account GTIDs written into the binlog when the server was a primary.

When using `master_use_gtid=current_pos` there is no need to consider whether a server was a primary or a replica prior to using CHANGE MASTER. But care must be taken not to inject extra transactions into the binlog on the replica server that are not intended to be replicated to other servers. If such an extra transaction is the most recent when the replica starts, it will be used as the starting point of replication. This will probably fail because that transaction is not present on the primary. To avoid local changes on a replica server to go into the binlog, set sql_log_bin to 0.

If it is undesirable that changes to the binlog on the replica affects the GTID replication position, then `master_use_gtid=slave_pos` should be used. Then the replica will always connect to the primary at the position of the last replicated GTID. This may avoid some surprises for users that expect behavior consistent with traditional replication, where the replication position is never changed by local changes done on a server.

When GTID strict mode is enabled (by setting `@@GLOBAL.gtid_strict_mode` to 1), it is normally best to use `current_pos`. In strict mode, extra transactions on the primary are disallowed.

If a replica is configured with the binlog disabled, `current_pos` and `slave_pos` are equivalent.

Even when a replica is configured to connect with the old-style binlog filename and offset (`CHANGE MASTER TO master_log_file=..., master_log_pos=...`), it will still keep track of the current GTID position in `@@GLOBAL.gtid_slave_pos`. This means that an existing replica previously configured and running can be changed to connect with GTID (to the same or a new master) simply with:

```
CHANGE MASTER TO master_use_gtid = slave_pos
```

The replica remembers that `master_use_gtid=slave_pos|master_pos` was specified and will use it also for subsequent connects, until it is explicitly changed by specifying `master_log_file/pos=...` or `master_use_gtid=no`. The current value can be seen as the field Using_Gtid of SHOW SLAVE STATUS:

```
SHOW SLAVE STATUS\G
...
Using_Gtid: Slave_pos
```

The replica server internally uses the mysql.gtid_slave_pos table to store the GTID position (and so preserve the value of `@@GLOBAL.gtid_slave_pos` across server restarts). After upgrading a server to 10.0, it is necessary to run mysql_upgrade (as always) to get the table created.

In order to be crash-safe, this table must use a transactional storage engine such as InnoDB. When MariaDB is first installed (or upgraded to 10.0.2+) the table is created using the default storage engine - which itself defaults to InnoDB. If there is a need to change the storage engine for this table (to make it transactional on a system configured with MyISAM as the default storage engine, for example), use ALTER TABLE:

```
ALTER TABLE mysql.gtid_slave_pos ENGINE = InnoDB
```

The mysql.gtid_slave_pos table should not be modified in any other way. In particular, do not try to update the rows in the table to change the replica's idea of the current GTID position; instead use

```
SET GLOBAL gtid_slave_pos = '0-1-1'
```

Starting from MariaDB 10.3.1, the server variable gtid_pos_auto_engines can preferably be set to make the server handle this automatically. See the description of the mysql.gtid_slave_pos table for details.

Using `current_pos` vs. `slave_pos`

When setting the MASTER_USE_GTID replication parameter, you have the option of enabling Global Transaction IDs to use either the `current_pos` or `slave_pos` values.

Using the value `current_pos` causes the replica to set its position based on the gtid_current_pos system variable, which is a union of gtid_binlog_pos and gtid_slave_pos. Using the value `slave_pos` causes the replica to instead set its position based on the gtid_slave_pos system variable.

You may run into issues when you use the value `current_pos` if you write any local transactions on the replica. For instance, if you issue an INSERT statement or otherwise write to a table while the replica threads are stopped, then new local GTIDs may be generated in gtid_binlog_pos, which will affect the replica's value of gtid_current_pos. This may cause errors when the replica threads are restarted, since the local GTIDs will be absent from the primary.

You can correct this issue by setting the MASTER_USE_GTID replication parameter to `slave_pos` instead of `current_pos`. For example:

```
CHANGE MASTER TO MASTER_USE_GTID = slave_pos;
START SLAVE;
```

Using GTIDs with Parallel Replication

If parallel replication is in use, then events that were logged with GTIDs with different gtid_domain_id values can be applied in parallel in an out-of-order manner.

Using GTIDs with MariaDB Galera Cluster

Starting with MariaDB 10.1.4, MariaDB Galera Cluster has limited support for GTIDs. See Using MariaDB GTIDs with MariaDB Galera Cluster for more information.

Setting up a New Replica Server with Global Transaction ID

Setting up a new replica server with global transaction ID is not much different from setting up an old-style replica. The basic steps are:

1. Setup the new server and load it with the initial data.

2. Start the replica replicating from the appropriate point in the primary's binlog.

Setting up a New Replica with an Empty Server

The simplest way for testing purposes is probably to setup a new, empty replica server and replicate all of the primary's binlogs from the start (this is usually not feasible in a realistic production setup, as the initial binlog files will probably have been purged or take too long to apply).

The replica server is installed in the normal way. By default, the GTID position for a newly installed server is empty, which makes the replica replicate from the start of the primary's binlogs. But if the replica was used for other purposes before, the initial position can be explicitly set to empty first:

```
SET GLOBAL gtid_slave_pos = "";
```

Next, point the replica to the master with CHANGE MASTER. Specify master_host etc. as usual. But instead of specifying master_log_file and master_log_pos manually, use `master_use_gtid=current_pos` (or `slave_pos` to have GTID do it

automatically:

```
CHANGE MASTER TO master_host="127.0.0.1", master_port=3310, master_user="root",
master_use_gtid=current_pos;
START SLAVE;
```

Setting up a New Replica From a Backup

The normal way to set up a new replication replica is to take a backup from an existing server (either a primary or replica in the replication topology), and then restore that backup on the server acting as the new replica, and the configure it to start replicating from the appropriate position in the primary's binary log.

It is important that the position at which replication is started corresponds exactly to the state of the data at the point in time that the backup was taken. Otherwise, the replica can end up with different data than the primary because of missing or duplicated transactions. Of course, if there are no writes to the server being backed up during the backup process, then a simple SHOW MASTER STATUS will give the correct position.

See the description of the specific backup tool to determine how to get the binary log position that corresponds to the backup.

Once the current binary log position for the backup has been obtained, in the form of a binary log file name and position, the corresponding GTID position can be obtained from BINLOG_GTID_POS() on the server that was backed up:

```
SELECT BINLOG_GTID_POS("master-bin.000001", 600);
```

The new replica can then start replicating from the primary by setting the correct value for gtid_slave_pos, and then executing CHANGE MASTER with the relevant values for the primary, and then starting the replica threads by executing START SLAVE. For example:

```
SET GLOBAL gtid_slave_pos = "0-1-2";
CHANGE MASTER TO master_host="127.0.0.1", master_port=3310, master_user="root",
master_use_gtid=slave_pos;
START SLAVE;
```

This method is particularly useful when setting up a new replica from a backup of the primary. Remember to ensure that the value of server_id configured on the new replica is different from that of any other server in the replication topology.

If the backup was taken of an existing replica server, then the new replica should already have the correct GTID position stored in the mysql.gtid_slave_pos table. This is assuming that this table was backed up and that it was backed up in a consistent manner with changes to other tables. In this case, there is no need to explicitly look up the GTID position on the old server and set it on the new replica - it will be already correctly loaded from the mysql.gtid_slave_pos table. This however does not work if the backup was taken from the primary - because then the current GTID position is contained in the binary log, not in the mysql.gtid_slave_pos table or any other table.

Setting up a New Replica with Mariabackup

A new replica can easily be set up with Mariabackup, which is a fork of Percona XtraBackup. See Setting up a Replica with Mariabackup for more information.

Setting up a New Replica with mariadb-dump

A new replica can also be set up with mariadb-dump.

mariadb-dump automatically includes the GTID position as a comment in the backup file if either the --master-data or --dump-slave option is used. It also automatically includes the commands to set gtid_slave_pos and execute CHANGE MASTER in the backup file if the --gtid option is used with either the --master-data or --dump-slave option.

Switching An Existing Old-Style Replica To Use GTID.

If there is already an existing replica running using old-style binlog filename/offset position, then this can be changed to use GTID directly. This can be useful for upgrades for example, or where there are already tools to setup new replica using old-style binlog positions.

When a replica connects to a primary using old-style binlog positions, and the primary supports GTID (i.e. is MariaDB 10.0.2 or later), then the replica automatically downloads the GTID position at connect and updates it during replication. Thus, once a replica has connected to the GTID-aware primary at least once, it can be switched to using GTID without any other actions needed;

```
STOP SLAVE;
CHANGE MASTER TO master_host="127.0.0.1", master_port=3310, master_user="root",
master_use_gtid=current_pos;
START SLAVE;
```

(A later version will probably add a way to setup the replica so that it will connect with old-style binlog file/offset the first time, and automatically switch to using GTID on subsequent connects.)

Changing a Replica to Replicate From a Different Primary

Once replication is running with GTID (master_use_gtid=current_pos|slave_pos), the replica can be pointed to a new primary simply by specifying in CHANGE MASTER the new master_host (and if required master_port, master_user, and master_password):

```
STOP SLAVE;
CHANGE MASTER TO master_host='127.0.0.1', master_port=3312;
START SLAVE;
```

The replica has a record of the GTID of the last applied transaction from the old primary, and since GTIDs are identical across all servers in a replication hierarchy, the replica will just continue from the appropriate point in the new primary's binlog.

It is important to understand how this change of primary work. The binlog is an ordered stream of events (or multiple streams, one per replication domain, (see Use with multi-source replication and other multi-primary setups). Events within the stream are always applied in the same order on every replica that replicates it. The MariaDB GTID relies on this ordering, so that it is sufficient to remember just a single point within the stream. Since event order is the same on every server, switching to the point of the same GTID in the binlog of another server will give the same result.

This translates into some responsibility for the user. The MariaDB GTID replication is fully asynchronous, and fully flexible in how it can be configured. This makes it possible to use it in ways where the assumption that binlog sequence is the same on all servers is violated. In such cases, when changing primary, GTID will still attempt to continue at the point of current GTID in the new binlog.

The most common way that binlog sequence gets different between servers is when the user/DBA does updates directly on a replica server (and these updates are written into the replica's binlog). This results in events in the replica's binlog that are not present on the primary or any other replicas. This can be avoided by setting the session variable sql_log_bin false while doing such updates, so they do not go into the binlog.

It is normally best to avoid any differences in binlogs between servers. That being said, MariaDB replication is designed for maximum flexibility, and there can be valid reasons for introducing such differences from time to time. It this case, it just needs to be understood that the GTID position is a single point in each binlog stream (one per replication domain), and how this affects the users particular setup.

Differences can also occur when two primary are active at the same time in a replication hierarchy. This happens when using a multi-primary ring. But it can also occur in a simple primary-replica setup, during switch to a new primary, if changes on the old primary is not allowed to fully replicate to all replica servers before switching primary. Normally, to switch primary, first writes to the old primary should be stopped, then one should wait for all changes to be replicated to the new primary, and only then should writes begin on the new primary. Deliberately using multiple active primary is also supported, this is described in the next section.

The GTID strict mode can be used to enforce identical binlogs across servers. When it is enabled, most actions that would cause differences are rejected with an error.

Use With Multi-Source Replication and Other Multi-Primary Setups

MariaDB global transaction ID supports having multiple primarys active at the same time. Typically this happens with either multi-source replication or multi-primary ring setups.

In such setups, each active primary must be configured with its own distinct replication domain ID, gtid_domain_id. The binlog will then in effect consists of multiple independent streams, one per active primary. Within one replication domain, binlog order is always the same on every server. But two different streams can be interleaved differently in different server binlogs.

The GTID position of a given replica is then not a single GTID. Rather, it becomes the GTID of the last event group applied for each value of domain ID, in effect the position reached in each binlog stream. When the replica connects to a primary, it can continue from one stream in a different binlog position than another stream. Since order within one stream is consistent across all servers, this is sufficient to always be able to continue replication at the correct point in any new primary

server(s).

Domain IDs are assigned by the DBA, according to the need of the application. The default value of @@GLOBAL.gtid_domain_id is 0. This is appropriate for most replication setups, where only a single primary is active at a time. The MariaDB server will never by itself introduce new domain_id values into the binlog.

When using multi-source replication, where a single replica connects to multiple primaries at the same time, each such primary should be configured with its own distinct domain ID.

Similarly, in a multi-primary ring topology, where all primary in the ring are updated by the application concurrently (with some mechanism to avoid conflicts), a distinct domain ID should be configured for each server (In a multi-primary ring where the application is careful to only do updates on one primary at a time, a single domain ID is sufficient).

Normally, a replica server should not receive direct updates (as this creates binlog differences compared to the primary). Thus it does not matter what value of gtid_domain_id is set on a replica, though it may make sense to make it the same as the primary (if not using multi-primary) to make it easy to promote the replica as a new primary. Of course, if a replica is itself an active primary, as in a multi-primary ring topology, the domain ID should be set according to the server's role as active primary.

Note that domain ID and server ID are distinct concepts. It is possible to use a different domain ID on each server, but this is normally not desirable. It makes the current GTID position (@@global.gtid_slave_pos) more complicated to understand and work with, and loses the concept of a single ordered binlog stream across all servers. It is recommended only to configure as many domain IDs as there are primary servers actively being updated by the application at the same time.

It is not an error in itself to configure domain IDs incorrectly (for example, not configuring them at all). For example, this will be typical in an upgrade scenario where a multi-primary ring using 5.5 is upgraded to 10.0. The ring will continue to work as before even though everything is configured to use the default domain ID 0. It is even possible to use GTID for replication between the servers. However, care must be taken when switching a replica to a different primary. If the binlog order between the old and the new primary differs, then a single GTID position to start replication from in the new primary's binlog may not be sufficient.

Multiple Redundant Replication Paths

Using GTID with multi-source replication, it is possible to set up multiple redundant replication paths. For example:

```
M1 <-> M2
M1 -> S1
M1 -> S2
M2 -> S1
M2 -> S2
```

Here, M1 and M2 are setup in a master-master ring. S1 and S2 both replicate from each of M1 and M2. Each event generated on M1 will now arrive twice at S1, through the paths M1->S1 and M1->M2->S1. This way, if the network connection between M1 and S1 is broken, the replication can continue uninterrupted through the alternate path through M2. Note that this is an advanced setup, and good familiarity with MariaDB replication is recommended to successfully operate it.

The option --gtid-ignore-duplicates must be enabled to use multiple redundant replication paths. This is necessary to avoid each event being applied twice on the replica as it arrives through each path. The GTID of every event will be compared against the sequence number of the current GTID replica position (within each domain), and will be skipped if less than or equal. Thus it is required that sequence numbers are strictly increasing within each domain for --gtid-ignore-duplicates to function correctly, and setting --gtid-strict-mode=1 to help enforce this is recommended.

The --gtid-ignore-duplicates options also relaxes the requirement for connection to the master. In the above example, when S1 connects to M2, it may connect at a GTID position from M1 that has not yet been applied on M2.

When --gtid-ignore-duplicates is enabled, the connection will be allowed, and S1 will start receiving events from M2 once the GTID has been replicated from M1 to M2. This can also be used to use replication filters in parts of a replication topology, to allow a replica to connect to a GTID position which was filtered on a master. When --gtid-ignore-duplicates is enabled, the connecting replica will start receiving events from the master at the first GTID sequence number that is larger than the connect-position.

Deleting Unused Domains

`FLUSH BINARY LOGS DELETE_DOMAIN_ID=(list-of-domains)` can be used to discard obsolete GTID domains from the server's binary log state. In order for this to be successful, no event group from the listed GTID domains can be present in existing binary log files. If some still exist, then they must be purged prior to executing this command.

If the command completes successfully, then it also rotates the binary log.

The old domains will still appear in gtid_io_pos. To get rid of these, you can stop the replica and execute on the replica:

```
SET gtid_slave_pos="<position with domains removed>"
```

New Syntax For Global Transaction ID

CHANGE MASTER

CHANGE MASTER has a new option, `master_use_gtid=[current_pos|slave_pos|no]`. When enabled (set to *current_pos* or *slave_pos*), the replica will connect to the master using the GTID position. When disabled (set to "no"), the old-style binlog filename/offset position is used to decide where to start replicating when connecting. Unlike in the old-style, when GTID is enabled, the values of the MASTER_LOG_FILE and MASTER_LOG_POS options are not updated per received event in master_info_file file.

The value of `master_use_gtid` is saved across server restarts (in master.info). The current value can be seen as the field Using_Gtid in the output of SHOW SLAVE STATUS.

For a detailed look at the difference between the *current_pos* and *slave_pos* options, see Using global transaction IDs

START SLAVE UNTIL master_gtid_pos=xxx

When starting replication with START SLAVE, it is possible to request the replica to run only until a specific GTID position is reached. Once that position is reached, the replica will stop.

The syntax for this is:

```
START SLAVE UNTIL master_gtid_pos = <GTID position>
```

The replica will start replication from the current GTID position, run up to and including the event with the GTID specified, and then stop. Note that this stops both the IO thread and the SQL thread (unlike START SLAVE UNTIL MASTER_LOG_FILE/MASTER_LOG_POS, which stops only the SQL thread).

If multiple GTIDs are specified, then they must be with distinct replication domain ID, for example:

```
START SLAVE UNTIL master_gtid_pos = "1-11-100,2-21-50"
```

With multiple domains in the UNTIL condition, each domain runs only up to and including the specified position, so it is possible for different domains to stop at different places in the binlog (each domain will resume from the stopped position when the replica is started the next time).

Not specifying a replication domain at all in the UNTIL condition means that the domain is stopped immediately, nothing is replicated from that domain. In particular, specifying the empty string will stop the replica immediately.

When using `START SLAVE UNTIL master_gtid_pos = XXX`, if the UNTIL position is present in the primary's binlog then it is permissible for the start position to be missing on the primary. In this case, replication for the associated domains stop immediately.

Both replica threads must be already stopped when using UNTIL master_gtid_pos, otherwise an error occurs. It is also an error if the replica is not configured to use GTID (`CHANGE MASTER TO master_use_gtid=current_pos|slave_pos`). And both threads must be started at the same time, the `IO_THREAD` or `SQL_THREAD` options can not be used to start only one of them.

`START SLAVE UNTIL master_gtid_pos=XXX` is particularly useful for promoting a new primary among a set of replicas when the old master goes away and replicas may have reached different positions in the old primary's binlog. The new primary needs to be ahead of all the other replicas to avoid losing events. This can be achieved by picking one server, say S1, and replicating any missing events from each other server S2, S3, ..., Sn:

```
CHANGE MASTER TO master_host="S2";
START SLAVE UNTIL master_gtid_pos = "<S2 GTID position>";
...
CHANGE MASTER TO master_host="Sn";
START SLAVE UNTIL master_gtid_pos = "<Sn GTID position>";
```

Once this is completed, S1 will have all events present on any of the servers. It can now be selected as the new primary, and all the other servers set to replicate from it.

MariaDB starting with 11.3.0

SQL_BEFORE_GTIDS|SQL_AFTER_GTIDS

MariaDB 11.3 extended the START SLAVE UNTIL command with the options `SQL_BEFORE_GTIDS` and `SQL_AFTER_GTIDS` to allow control of whether the replica stops before or after a provided GTID state. Its syntax is:

```
START SLAVE UNTIL (SQL_BEFORE_GTIDS|SQL_AFTER_GTIDS)="<gtid_list>"
```

When providing SQL_BEFORE_GTIDS="<gtid_list>", the replica will execute all transactions up to the first GTID found in the provided list, and stop immediately. In contrast to the default behavior of UNTIL, this will execute transactions from all domains on the primary until the replica stops due to seeing a GTID on the list.

START SLAVE UNTIL SQL_AFTER_GTIDS="<gtid_list>" is an alias to the default behavior of START SLAVE UNTIL master_gtid_pos="<gtid_list>". That is, the replica will only execute transactions originating from domain ids provided in the list, and will stop once all transactions provided in the UNTIL list have all been executed.

Example

If a primary server has a binary log consisting of the following GTIDs:

- 0-1-1
- 1-1-1
- 0-1-2
- 1-1-2
- 0-1-3
- 1-1-3

If a fresh replica (i.e. one with an empty GTID position, @@gtid_slave_pos='') is started with `SQL_BEFORE_GTIDS`, i.e. `START SLAVE UNTIL SQL_BEFORE_GTIDS="1-1-2"`, the resulting gtid_slave_pos of the replica will be "0-1-2,1-1-1". This is because the replica will execute all events until it sees the transaction with GTID 1-1-2 and immediately stop without executing it.

However, if a replica is started with `SQL_AFTER_GTIDS`, i.e. `START SLAVE UNTIL SQL_AFTER_GTIDS="1-1-2"` then the resulting gtid_slave_pos of the replica will be "1-1-2". This is because it will only execute events from domain 1 until it has executed the provided GTID.

BINLOG_GTID_POS().

The BINLOG_GTID_POS() function takes as input an old-style binary log position in the form of a file name and a file offset. It looks up the position in the current binlog, and returns a string representation of the corresponding GTID position. If the position is not found in the current binlog, NULL is returned.

MASTER_GTID_WAIT

The MASTER_GTID_WAIT function is useful in replication for controlling primary/replica synchronization, and blocks until the replica has read and applied all updates up to the specified position in the primary log. See MASTER_GTID_WAIT for details.

System Variables

`gtid_slave_pos`

This system variable contains the GTID of the last transaction applied to the database by the server's replica threads for each replication domain. This system variable's value is automatically updated whenever a replica thread applies an event group. This system variable's value can also be manually changed by users, so that the user can change the GTID position of the replica threads.

When using multi-source replication, the same GTID position is shared by all replica connections. In this case, different primaries should use different replication domains by configuring different gtid_domain_id values. If one primary was using a gtid_domain_id value of `1`, and if another primary was using a gtid_domain_id value of `2`, then any replicas replicating from both primaries would have GTIDs with both gtid_domain_id values in `gtid_slave_pos`.

This system variable's value can be manually changed by executing SET GLOBAL, but all replica threads to be stopped with STOP SLAVE first. For example:

```
STOP ALL SLAVES;
SET GLOBAL gtid_slave_pos = "1-10-100,2-20-500";
START ALL SLAVES;
```

This system variable's value can be reset by manually changing its value to the empty string. For example:

```
SET GLOBAL gtid_slave_pos = '';
```

The GTID position defined by `gtid_slave_pos` can be used as a replica's starting replication position by setting MASTER_USE_GTID=slave_pos when the replica is configured with the CHANGE MASTER TO statement. As an alternative, the gtid_current_pos system variable can also be used as a replica's starting replication position.

If a user sets the value of the `gtid_slave_pos` system variable, and gtid_binlog_pos contains later GTIDs for certain replication domains, then gtid_current_pos will contain the GTIDs from gtid_binlog_pos for those replication domains. To protect users in this scenario, if a user sets the `gtid_slave_pos` system variable to a GTID position that is behind the GTID position in gtid_binlog_pos, then the server will give the user a warning.

This can help protect the user when the replica is configured to use gtid_current_pos as its replication position. This can also help protect the user when a server has been rolled back to restart replication from an earlier point in time, but the user has forgotten to reset gtid_binlog_pos with RESET MASTER.

The mysql.gtid_slave_pos system table is used to store the contents of global.gtid_slave_pos and preserve it over restarts.

- **Commandline:** None
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default:** Null

`gtid_binlog_pos`

This variable is the GTID of the last event group written to the binary log, for each replication domain.

Note that when the binlog is empty (such as on a fresh install or after RESET MASTER), there are no event groups written in any replication domain, so in this case the value of `gtid_binlog_pos` will be the empty string.

The value is read-only, but it is updated whenever a DML or DDL statement is written to the binary log. The value can be reset by executing RESET MASTER, which will also delete all binary logs. However, note that RESET MASTER does not also reset gtid_slave_pos. Since gtid_current_pos is the union of gtid_slave_pos and `gtid_binlog_pos`, that means that new GTIDs added to `gtid_binlog_pos` can lag behind those in gtid_current_pos if gtid_slave_pos contains GTIDs in the same domain with higher sequence numbers. If you want to reset gtid_current_pos for a specific GTID domain in cases like this, then you will also have to change gtid_slave_pos in addition to executing RESET MASTER. See gtid_slave_pos for notes on how to change its value.

- **Commandline:** None
- **Scope:** Global
- **Dynamic:** Read-only
- **Data Type:** `string`
- **Default:** Null

`gtid_binlog_state`

The variable gtid_binlog_state holds the internal state of the binlog. The state consists of the last GTID ever logged to the binary log for every combination of domain_id and server_id. This information is used by the primary to determine whether a given GTID has been logged to the binlog in the past, even if it has later been deleted due to binlog purge. For each domain_id, the last entry in @@gtid_binlog_state is the last GTID logged into binlog, ie. this is the value that appears in @@gtid_binlog_pos.

Normally this internal state is not needed by users, as @@gtid_binlog_pos is more useful in most cases. The main usage of @@gtid_binlog_state is to restore the state of the binlog after RESET MASTER (or equivalently if the binlog files are lost). If the value of @@gtid_binlog_state is saved before RESET MASTER and restored afterwards, the primary will retain information about past history, same as if PURGE BINARY LOGS had been used (of course the actual events in the binary logs are still deleted).

Note that to set the value of @@gtid_binlog_state, the binary log must be empty, that is it must not contain any GTID events and the previous value of @@gtid_binlog_state must be the empty string. If not, then RESET MASTER must be used first to erase the binary log first.

The value of @@gtid_binlog_state is preserved by the server across restarts by writing a file MASTER-BIN.state, where MASTER-BIN is the base name of the binlog set with the --log-bin option. This file is written at server shutdown, and re-read at next server start. (In case of a server crash, the data in the MASTER-BIN.state is not correct, and the server instead recovers the correct value during binlog crash recovery by scanning the binlog files and recording each GTID found).

For completeness, note that setting @@gtid_binlog_state internally executes a RESET MASTER. This is normally not noticeable as it can only be changed when the binlog is empty of GTID events. However, if executed e.g. immediately after upgrading to MariaDB 10, it is possible that the binlog is non-empty but without any GTID events, in which case all such events will be deleted, just as if RESET MASTER had been run.

- **Commandline:** None
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default:** Null

`gtid_current_pos`

This system variable contains the GTID of the last transaction applied to the database for each replication domain.

The value of this system variable is constructed from the values of the gtid_binlog_pos and gtid_slave_pos system variables. It gets GTIDs of transactions executed locally from the value of the gtid_binlog_pos system variable. It gets GTIDs of replicated transactions from the value of the gtid_slave_pos system variable.

For each replication domain, if the server_id of the corresponding GTID in gtid_binlog_pos is equal to the servers own server_id, *and* the sequence number is higher than the corresponding GTID in gtid_slave_pos, then the GTID from gtid_binlog_pos will be used. Otherwise the GTID from gtid_slave_pos will be used for that domain.

GTIDs from gtid_binlog_pos in which the server_id of the GTID is **not** equal to the server's own server_id are effectively ignored. If gtid_binlog_pos contains a GTID for a given replication domain, but the server_id of the GTID is **not** equal to the server's own server_id, and gtid_slave_pos does **not** contain a GTID for that given replication domain, then `gtid_current_pos` will **not** contain any GTID for that replication domain.

Thus, `gtid_current_pos` contains the most recent GTID executed on the server, whether this was done as a primary or as a replica.

The GTID position defined by `gtid_current_pos` can be used as a replica's starting replication position by setting MASTER_USE_GTID=current_pos when the replica is configured with the CHANGE MASTER TO statement. As an alternative, the gtid_slave_pos system variable can also be used as a replica's starting replication position.

The value of `gtid_current_pos` is read-only, but it is updated whenever a transaction is written to the binary log and/or replicated by a replica thread, and that transaction's GTID is considered *newer* than the current GTID for that domain. See above for the rules on how to determine if a GTID would be considered *newer*.

If you need to reset the value, see the notes on resetting gtid_slave_pos and gtid_binlog_pos, since `gtid_current_pos` is formed from the values of those variables.

- **Commandline:** None
- **Scope:** Global
- **Dynamic:** Read-only
- **Data Type:** `string`
- **Default:** Null

`gtid_strict_mode`

The GTID strict mode is an optional setting that can be used to help the DBA enforce a strict discipline about keeping binlogs identical across multiple servers replicating using global transaction ID.

When GTID strict mode is enabled, some additional errors are enabled for situations that could otherwise cause differences between binlogs on different servers in a replication hierarchy:

1. If a replica server tries to replicate a GTID with a sequence number lower than what is already in the binlog for that replication domain, the SQL thread stops with an error (this indicates an extra transaction in the replica binlog not present on the primary).
2. Similarly, an attempt to manually binlog a GTID with a lower sequence number (by setting `@@SESSION.gtid_seq_no`) is rejected with an error.
3. If the replica tries to connect starting at a GTID that is missing in the primary's binlog, this is an error in GTID strict mode even if a GTID exists with a higher sequence number (this indicates a GTID on the replica missing on the primary). Note that this error is controlled by the setting of GTID strict mode on the connecting replica server.

GTID mode is off by default; this is needed to preserve backwards compatibility with existing replication setups (older versions of the server did not enforce any strict mode for binlog order). Global transaction ID is designed to work correctly even when strict mode is not enabled. However, with strict mode enforced, the semantics is simpler and thus easier to understand, because binlog order is always identical across servers and sequence numbers are always strictly increasing within each replication domain. This can also make automated scripting of large replication setups easier to implement correctly.

When GTID strict mode is enabled, the replica will stop with an error when a problem is encountered. This allows the DBA to become aware of the problem and take corrective actions to avoid similar issues in the future. One way to recover from such an error is to temporarily disable GTID strict mode on the offending replica, to be able to replicate past the problem

point (perhaps using `START SLAVE UNTIL master_gtid_pos=XXX`).

- **Commandline:** `--gtid-strict-mode[={0|1}]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default:** `Off`

---

gtid_domain_id

- **Description:** This variable is used to decide which replication domain new GTIDs are logged in for a primary server. See Use with multi-source replication and other multi-primary setups for details. This variable can also be set on the session level by a user with the SUPER privilege. This is used by mariadb-binlog to preserve the domain ID of GTID events.
- **Commandline:** `--gtid-domain-id=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric (32-bit unsigned integer)`
- **Default Value:** `0`
- **Range:** `0` to `4294967295`

---

last_gtid

- **Description:** Holds the GTID that was assigned to the last transaction, or statement that was logged to the binary log. If the binary log is disabled, or if no transaction or statement was executed in the session yet, then the value is an empty string.
- **Scope:** Session
- **Dynamic:** Read-only
- **Data Type:** `string`

---

server_id

- **Description:** Server_id can be set on the session level to change which server_id value is logged in binlog events (both GTID and other events). This is used by mariadb-binlog to preserve the server ID of GTID events.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** numeric (32-bit unsigned integer)

---

gtid_seq_no

- **Description:** gtid_seq_no can be set on the session level to change which sequence number is logged in the following GTID event. The variable, along with @@gtid_domain_id and @@server_id, is typically used by mariadb-binlog to set up the gtid value of the transaction being decoded into the output.
- **Commandline:** None
- **Scope:** Session
- **Dynamic:** Yes
- **Data Type:** `numeric (64-bit unsigned integer)`
- **Default:** Null

---

gtid_ignore_duplicates

- **Description:** When set, different primary connections in multi-source replication are allowed to receive and process event groups with the same GTID (when using GTID mode). Only one will be applied, any others will be ignored. Within a given replication domain, just the sequence number will be used to decide whether a given GTID has been already applied; this means it is the responsibility of the user to ensure that GTID sequence numbers are strictly increasing. With gtid_ignore_duplicates=OFF, a duplicate event based on domain id and sequence number, will be executed. When --gtid-ignore-duplicate is set, a replica is allowed to connect at a GTID position that does not exist on the primary. The replica will start receiving events once a GTID with a higher sequence number is available on the primary (within that domain). This can be used to allow a replica to connect at a GTID position that was filtered on the primary, eg. using --replicate-ignore-table. See also Multiple Redundant Replication Paths
- **Commandline:** `--gtid-ignore-duplicates=#`

- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default:** `OFF`

`gtid_pos_auto_engines`

This variable is used to enable multiple versions of the mysql.gtid_slave_pos table, one for each transactional storage engine in use. This can improve replication performance if a server is using multiple different storage engines in different transactions.

The value is a list of engine names, separated by commas (','). Replication of transactions using these engines will automatically create new versions of the mysql.gtid_slave_pos table in the same engine and use that for future transactions (table creation takes place in a background thread). This avoids introducing a cross-engine transaction to update the GTID position. Only transactional storage engines are supported for gtid_pos_auto_engines (this currently means InnoDB, TokuDB, or MyRocks).

The variable can be changed dynamically, but replica SQL threads should be stopped when changing it, and it will take effect when the replicas are running again.

When setting the variable on the command line or in a configuration file, it is possible to specify engines that are not enabled in the server. The server will then still start if, for example, that engine is no longer used. Attempting to set a non-enabled engine dynamically in a running server (with SET GLOBAL gtid_pos_auto_engines) will still result in an error.

Removing a storage engine from the variable will have no effect once the new tables have been created - as long as these tables are detected, they will be used.

- **Commandline:** `--gtid-pos-auto-engines=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string` (comma-separated list of engine names)
- **Default:** empty

`gtid_cleanup_batch_size`

- **Description:** Normally does not need tuning. How many old rows must accumulate in the mysql.gtid_slave_pos table before a background job will be run to delete them. Can be increased to reduce number of commits if using many different engines with gtid_pos_auto_engines, or to reduce CPU overhead if using a huge number of different gtid_domain_ids. Can be decreased to reduce number of old rows in the table.
- **Commandline:** `--gtid-cleanup-batch-size=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default:** `64`
- **Range:** `0` to `2147483647`
- **Introduced:** MariaDB 10.4.1

3.1.10 Parallel Replication

> The terms *master* and *slave* have historically been used in replication, but the terms terms *primary* and *replica* are now preferred. The old terms are used still used in parts of the documentation, and in MariaDB commands, although MariaDB 10.5 has begun the process of renaming. The documentation process is ongoing. See MDEV-18777 to follow progress on this effort.

Contents

Some writes, replicated from the primary can be executed in parallel (simultaneously) on the replica. Note that for parallel replication to work, both the primary and replica need to be MariaDB 10.0.5 or later.

Parallel Replication Overview

MariaDB replication in general takes place in three parts:

- Replication events are read from the primary by the IO thread and queued in the relay log.
- Replication events are fetched one at a time by the SQL thread from the relay log
- Each event is applied on the replica to replicate all changes done on the primary.

Before MariaDB 10, the third step was also performed by the SQL thread; this meant that only one event could execute at a time, and replication was essentially single-threaded. Since MariaDB 10, the third step can optionally be performed by a pool of separate replication worker threads, and thereby potentially increase replication performance by applying multiple events in parallel.

How to Enable Parallel Replica

To enable, specify slave-parallel-threads=# in your my.cnf file as an argument to mysql. Parallel replication can in addition be disabled on a per-multi-source connection by setting @@connection_name.slave-parallel-mode to "none".

The value (#) of slave_parallel_threads specifies how many threads will be created in a pool of worker threads used to apply events in parallel for *all* your replicas (this includes multi-source replication). If the value is zero, then no worker threads are created, and old-style replication is used where events are applied inside the SQL thread. Usually the value, if non-zero, should be at least two times the number of multi-source primary connections used. It makes little sense to use only a single worker thread for one connection; this will incur some overhead in inter-thread communication between the SQL thread and the worker thread, but with just a single worker thread events can not be applied in parallel anyway.

`slave-parallel-threads=#` is a dynamic variable that can be changed without restarting mysqld. All replicas connections must however be stopped when changing the value.

Configuring the Replica Parallel Mode

Parallel replication can be in-order or out-of-order:

- In-order executes transactions in parallel, but orders the commit step of the transactions to happen in the exact same order as on the primary. Transactions are only executed in parallel to the extent that this can be automatically verified to be possible without any conflicts. This means that the use of parallelism is completely transparent to the application.
- Out-of-order can execute and commit transactions in different order on the replica than originally on the primary. This means that the application must be tolerant to seeing updates occur in different order. The application is also responsible for ensuring that there are no conflicts between transactions that are replicated out-of-order. Out-of-order is only used in GTID mode and only when explicitly enabled by the application, using the replication domain that is part of the GTID.

In-Order Parallel Replication

Optimistic Mode of In-Order Parallel Replication

Optimistic mode of in-order parallel replication provides a lot of opportunities for parallel apply on the replica while still

preserving exact transaction semantics from the point of view of applications. It is the default mode from MariaDB 10.5.1.

Optimistic mode of in-order parallel replication can be configured by setting the slave_parallel_mode system variable to `optimistic` on the replica.

Any transactional DML (INSERT/UPDATE/DELETE) is allowed to run in parallel, up to the limit of @@slave_domain_parallel_threads. This may cause conflicts on the replica, eg. if two transactions try to modify the same row. Any such conflict is detected, and the latter of the two transactions is rolled back, allowing the former to proceed. The latter transaction is then re-tried once the former has completed.

The term "optimistic" is used for this mode, because the server optimistically assumes that few conflicts will occur, and that the extra work spent rolling back and retrying conflicting transactions is justified from the gain from running most transactions in parallel.

There are a few heuristics to try to avoid needless conflicts. If a transaction executed a row lock wait on the primary, it will not be run in parallel on the replica. Transactions can also be marked explicitly as potentially conflicting on the primary, by setting the variable @@skip_parallel_replication. More such heuristics may be added in later MariaDB versions. There is a further --slave-parallel-mode called "aggressive", where these heuristics are disabled, allowing even more transactions to be applied in parallel.

Non-transactional DML and DDL is not safe to optimistically apply in parallel, as it cannot be rolled back in case of conflicts. Thus, in optimistic mode, non-transactional (such as MyISAM) updates are not applied in parallel with earlier events (it is however possible to apply a MyISAM update in parallel with a later InnoDB update). DDL statements are not applied in parallel with any other transactions, earlier or later.

The different kind of transactions can be identified in the output of mariadb-binlog. For example:

```
#150324 13:06:26 server id 1  end_log_pos 6881  GTID 0-1-42 ddl
...
#150324 13:06:26 server id 1  end_log_pos 7816  GTID 0-1-47
...
#150324 13:06:26 server id 1  end_log_pos 8177  GTID 0-1-49 trans
/*!100101 SET @@session.skip_parallel_replication=1*//*!*/;
...
#150324 13:06:26 server id 1  end_log_pos 9836  GTID 0-1-59 trans waited
```

GTID 0-1-42 is marked as being DDL. GTID 0-1-47 is marked as being non-transactional DML, while GTID 0-1-49 is transactional DML (seen on the "trans" keyword). GTID 0-1-49 was additionally run with @@skip_parallel_replication set on the primary. GTID 0-1-59 is transactional DML that had a row lock wait when run on the primary (the "waited" keyword).

Aggressive Mode of In-Order Parallel Replication

Aggressive mode of in-order parallel replication is very similar to optimistic mode. The main difference is that the replica does not consider whether transactions conflicted on the primary when deciding whether to apply the transactions in parallel.

Aggressive mode of in-order parallel replication can be configured by setting the slave_parallel_mode system variable to `aggressive` on the replica.

Conservative Mode of In-Order Parallel Replication

Conservative mode of in-order parallel replication uses the group commit on the primary to discover potential for parallel apply of events on the replica. If two transactions commit together in a group commit on the primary, they are written into the binlog with the same commit id. Such events are certain to not conflict with each other, and they can be scheduled by the parallel replication to run in different worker threads.

Conservative mode of in-order parallel replication is the default mode until MariaDB 10.5.0, but it can also be configured by setting the slave_parallel_mode system variable to `conservative` on the replica.

Two transactions that were committed separately on the primary can potentially conflict (eg. modify the same row of a table). Thus, the worker that applies the second transaction will not start immediately, but wait until the first transaction begins the commit step; at this point it is safe to start the second transaction, as it can no longer disrupt the execution of the first one.

Here is example output from mariadb-binlog that shows how GTID events are marked with commit id. The GTID 0-1-47 has no commit id, and can not run in parallel. The GTIDs 0-1-48 and 0-1-49 have the same commit id 630, and can thus replicate in parallel with one another on a replica:

```
#150324 12:54:24 server id 1  end_log_pos 20052  GTID 0-1-47 trans
...
#150324 12:54:24 server id 1  end_log_pos 20212  GTID 0-1-48 cid=630 trans
...
#150324 12:54:24 server id 1  end_log_pos 20372  GTID 0-1-49 cid=630 trans
```

In either case, when the two transactions reach the point where the low-level commit happens and commit order is determined, the two commits are sequenced to happen in the same order as on the primary, so that operation is transparent to applications.

The opportunities for parallel replication on replicas can be highly increased if more transactions are committed in a group commit on the primary. This can be tuned using the binlog_commit_wait_count and binlog_commit_wait_usec variables. If for example the application can tolerate up to 50 milliseconds extra delay for transactions on the primary, one can set `binlog_commit_wait_usec=50000` and `binlog_commit_wait_count=20` to get up to 20 transactions at a time available for replication in parallel. Care must however be taken to not set `binlog_commit_wait_usec` too high, as this could cause significant slowdown for applications that run a lot of small transactions serially one after the other.

Note that even if there is no parallelism available from the primary group commit, there is still an opportunity for speedup from in-order parallel replication, since the actual commit steps of different transactions can run in parallel. This can be particularly effective on a replica with binlog enabled (log_slave_updates=1), and more so if replica is configured to be crash-safe (sync_binlog=1 and innodb_flush_log_at_trx_commit=1), as this makes group commit possible on the replica.

Minimal Mode of In-Order Parallel Replication

Minimal mode of in-order parallel replication *only* allows the commit step of transactions to be applied in parallel; all other steps are applied serially.

Minimal mode of in-order parallel replication can be configured by setting the slave_parallel_mode system variable to `minimal` on the replica.

Out-of-Order Parallel Replication

Out-of-order parallel replication happens (only) when using GTID mode, when GTIDs with different replication domains are used. The replication domain is set by the DBA/application using the variable `gtid_domain_id`.

Two transactions having GTIDs with different domain_id are scheduled to different worker threads by parallel replication, and are allowed to execute completely independently from each other. It is the responsibility of the application to only set different domain_ids for transactions that are truly independent, and are guaranteed to not conflict with each other. The application must also be able to work correctly even though the transactions with different domain_id are seen as committing in different order between the replica and the primary, and between different replicas.

Out-of-order parallel replication can potentially give more performance gain than in-order parallel replication, since the application can explicitly give more opportunities for running transactions in parallel than what the server can determine on its own automatically.

One simple but effective usage is to run long-running statements, such as ALTER TABLE, in a separate replication domain. This allows replication of other transactions to proceed uninterrupted:

```
SET SESSION gtid_domain_id=1
ALTER TABLE t ADD INDEX myidx(b)
SET SESSION gtid_domain_id=0
```

Normally, a long-running ALTER TABLE or other query will stall all following transactions, causing the replica to become behind the primary as least as long time as it takes to run the long-running query. By using out-of-order parallel replication by setting the replication domain id, this can be avoided. The DBA/application must ensure that no conflicting transactions will be replicated while the ALTER TABLE runs.

Another common opportunity for out-of-order parallel replication comes in connection with multi-source replication. Suppose we have two different primaries M1 and M2, and we are using multi-source replication to have S1 as a replica of both M1 and M2. S1 will apply events received from M1 in parallel with events received from M2. If we now have a third-level replica S2 that replicates from S1 as primary, we want S2 to also be able to apply events that originated on M1 in parallel with events that originated on M2. This can be achieved with out-of-order parallel replication, by setting `gtid_domain_id` different on M1 and M2.

Note that there are no special restrictions on what operations can be replicated in parallel using out-of-order; such operations can be on the same database/schema or even on the same table. The only restriction is that the operations must not conflict, that is they must be able to be applied in any order and still end up with the same result.

When using out-of-order parallel replication, the current replica position in the primary's binlog becomes multi-dimensional - each replication domain can have reached a different point in the primary binlog at any one time. The current position can be seen from the variable `gtid_slave_pos`. When the replica is stopped, restarted, or switched to replicate from a different primary using CHANGE MASTER, MariaDB automatically handles restarting each replication domain at the appropriate point in the binlog.

Out-of-order parallel replication is disabled when --slave-parallel-mode=minimal (or none).

Checking Worker Thread Status in SHOW PROCESSLIST

The worker threads will be listed as "system user" in SHOW PROCESSLIST. Their state will show the query they are currently working on, or it can show one of these:

- "Waiting for work from main SQL threads". This means that the worker thread is idle, no work is available for it at the moment.
- "Waiting for prior transaction to start commit before starting next transaction". This means that the previous batch of transactions that committed together on the primary primary has to complete first. This worker thread is waiting for that to happen before it can start working on the following batch.
- "Waiting for prior transaction to commit". This means that the transaction has been executed by the worker thread. In order to ensure in-order commit, the worker thread is waiting to commit until the previous transaction is ready to commit before it.

Expected Performance Gain

Here is an article showing up to ten times improvement when using parallel replication: http://kristiannielsen.livejournal.com/18435.html.

Configuring the Maximum Size of the Parallel Replica Queue

The slave_parallel_max_queued system variable can be used to configure the maximum size of the parallel replica queue. This system variable is only meaningful when parallel replication is configured (i.e. when slave_parallel_threads > `0`).

When parallel replication is used, the SQL thread will read ahead in the relay logs, queueing events in memory while looking for opportunities for executing events in parallel. The slave_parallel_max_queued system variable sets a limit for how much memory it will use for this.

The configured value of the slave_parallel_max_queued system variable is actually allocated for each worker thread, so the total allocation is actually equivalent to the following:

slave_parallel_max_queued * slave_parallel_threads

If this value is set too high, and the replica is far (eg. gigabytes of binlog) behind the primary, then the SQL thread can quickly read all of that and fill up memory with huge amounts of binlog events faster than the worker threads can consume them.

On the other hand, if set too low, the SQL thread might not have sufficient space for queuing enough events to keep the worker threads busy, which could reduce performance. In this case, the SQL thread will have the thread state that states `Waiting for room in worker thread event queue`. For example:

```
+----+-------------+-----------+------+---------+--------+----------------------------------
------------+------------------+----------+
| Id | User        | Host      | db   | Command | Time   | State
| Info             | Progress |
+----+-------------+-----------+------+---------+--------+----------------------------------
------------+------------------+----------+
|  3 | system user |           | NULL | Connect |    139 | closing tables
| NULL             |    0.000 |
|  4 | system user |           | NULL | Connect |    139 | Waiting for work from SQL thread
| NULL             |    0.000 |
|  6 | system user |           | NULL | Connect | 264274 | Waiting for master to send event
| NULL             |    0.000 |
| 10 | root        | localhost | NULL | Sleep   |     43 |
| NULL             |    0.000 |
| 21 | system user |           | NULL | Connect |     45 | Waiting for room in worker thread
event queue | NULL             |    0.000 |
| 54 | root        | localhost | NULL | Query   |      0 | init
| SHOW PROCESSLIST |    0.000 |
+----+-------------+-----------+------+---------+--------+----------------------------------
------------+------------------+----------+
```

The slave_parallel_max_queued system variable does not define a hard limit, since the binary log events that are currently executing always need to be held in-memory. This means that at least two events per worker thread can always be queued in-memory, regardless of the value of slave_parallel_threads.

Usually, the slave_parallel_threads system variable should be set large enough that the SQL thread is able to read far enough ahead in the binary log to exploit all possible parallelism. In normal operation, the replica will hopefully not be too far behind, so there will not be a need to queue much data in-memory. The slave_parallel_max_queued system variable could be set fairly high (eg. a few hundred kilobytes) to not limit throughput. It should just be set low enough that total allocation of the parallel replica queue will not cause the server to run out of memory.

Configuration Variable slave_domain_parallel_threads

The pool of replication worker threads is shared among all multi-source primary connections, and among all replication domains that can replicate in parallel using out-of-order.

If one primary connection or replication domain is currently processing a long-running query, it is possible that it will allocate all the worker threads in the pool, only to have them wait for the long-running query to complete, stalling any other primary connection or replication domain, which will have to wait for a worker thread to become free.

This can be avoided by setting slave_domain_parallel_threads to a number that is lower than `slave_parallel_threads`. When set different from zero, each replication domain in one primary connection can reserve at most that many worker threads at any one time, leaving the rest (up to the value of slave_parallel_threads) free for other primary connections or replication domains to use in parallel.

The `slave_domain_parallel_threads` variable is dynamic and can be changed without restarting the server; all replicas must be stopped while changing it, though.

Implementation Details

The implementation is described in MDEV-4506.

3.1.11 Replication and Binary Log System Variables

The terms *master* and *slave* have historically been used in replication, but the terms terms *primary* and *replica* are now preferred. The old terms are used still used in parts of the documentation, and in MariaDB commands, although MariaDB 10.5 has begun the process of renaming. The documentation process is ongoing. See MDEV-18777 to follow progress on this effort.

Contents

28. gtid_slave_pos
29. gtid_strict_mode
30. gtid_pos_auto_engines
31. init_slave
32. last_gtid
33. log_bin
34. log_bin_basename
35. log_bin_compress
36. log_bin_compress_min_len
37. log_bin_index
38. log_bin_trust_function_creators
39. log_slow_slave_statements
40. log_slave_updates
41. master_verify_checksum
42. max_binlog_cache_size
43. max_binlog_size
44. max_binlog_stmt_cache_size
45. max_relay_log_size
46. read_binlog_speed_limit
47. relay_log
48. relay_log_basename
49. relay_log_index
50. relay_log_info_file
51. relay_log_purge
52. relay_log_recovery
53. relay_log_space_limit
54. replicate_annotate_row_events
55. replicate_do_db
56. replicate_do_table
57. replicate_events_marked_for_skip
58. replicate_ignore_db
59. replicate_ignore_table
60. replicate_rewrite_db
61. replicate_wild_do_table
62. replicate_wild_ignore_table
63. report_host
64. report_password
65. report_port
66. report_user
67. server_id
68. skip_parallel_replication
69. skip_replication
70. slave_compressed_protocol
71. slave_ddl_exec_mode
72. slave_domain_parallel_threads
73. slave_exec_mode
74. slave_load_tmpdir
75. slave_max_allowed_packet
76. slave_max_statement_time
77. slave_net_timeout
78. slave_parallel_max_queued
79. slave_parallel_mode
80. slave_parallel_threads
81. slave_parallel_workers
82. slave_run_triggers_for_rbr
83. slave_skip_errors
84. slave_sql_verify_checksum
85. slave_transaction_retries
86. slave_transaction_retry_errors
87. slave_transaction_retry_interval
88. slave_type_conversions
89. sql_log_bin
90. sql_slave_skip_counter
91. sync_binlog
92. sync_master_info
93. sync_relay_log
94. sync_relay_log_info

This page lists system variables that are related to binary logging and replication.

See Server System Variables for a complete list of system variables and instructions on setting them, as well as System variables for global transaction ID.

Also see mariadbd replication options for related options that are not system variables (such as binlog_do_db and binlog_ignore_db).

See also the Full list of MariaDB options, system and status variables.

auto_increment_increment

- **Description:** The increment for all AUTO_INCREMENT values on the server, by default `1`. Intended for use in primary-to-primary replication.
- **Commandline:** `--auto-increment-increment[=#]`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1`
- **Range:** `1` to `65535`

---

auto_increment_offset

- **Description:** The offset for all AUTO_INCREMENT values on the server, by default `1`. Intended for use in primary-to-primary replication. Should be not be larger than auto_increment_increment. See AUTO_INCREMENT#Replication.
- **Commandline:** `--auto-increment-offset[=#]`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1`
- **Range:** `1` to `65535`

---

binlog_alter_two_phase

- **Description:** When set, split ALTER at binary logging into two statements: START ALTER and COMMIT/ROLLBACK ALTER. The ON setting is recommended for long-running ALTER-table so it could start on replica before its actual execution on primary.
- **Commandline:** `--binlog-alter-two-phase[={0|1}]`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB 10.8.1

---

binlog_annotate_row_events

- **Description:** This option tells the primary to write annotate_rows_events to the binary log.
- **Commandline:** `--binlog-annotate-row-events[={0|1}]`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** boolean
- **Default Value:**
  - `ON` (>= MariaDB 10.2.4)
  - `OFF` (<= MariaDB 10.2.3)

binlog_do_db

- **Description:** This option allows you to configure a replication primary to write statements and transactions affecting databases that match a specified name into its binary log. Since the filtered statements or transactions will not be present in the binary log, its replicas will not be able to replicate them.
  - This option will **not** work with cross-database updates with statement-based logging. See the Statement-Based Logging section for more information.
  - Until MariaDB 11.2.0, only available as an option, not a system variable. This option can **not** be set

dynamically.
    - When setting it on the command-line or in a server option group in an option file, the option does not accept a comma-separated list. If you would like to specify multiple filters, then you need to specify the option multiple times.
    - See Replication Filters for more information.
- **Commandline:** `--binlog-do-db=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** NULL
- **Introduced:** MariaDB 11.2.0 (as a system variable)

---

binlog_cache_size

- **Description:** If the binary log is active, this variable determines the size in bytes, per-connection, of the cache holding a record of binary log changes during a transaction. A separate variable, binlog_stmt_cache_size, sets the upper limit for the statement cache. The binlog_cache_disk_use and binlog_cache_use server status variables will indicate whether this variable needs to be increased (you want a low ratio of binlog_cache_disk_use to binlog_cache_use).
- **Commandline:** `--binlog-cache-size=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `32768`
- **Range - 32 bit:** `4096` to `4294967295`
- **Range - 64 bit:** `4096` to `18446744073709547520`

---

binlog_checksum

- **Description:** Specifies the type of BINLOG_CHECKSUM_ALG for log events in the binary log.
- **Commandline:**
    - `--binlog-checksum=name`
    - `--binlog-checksum=[0|1]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:**
    - `CRC32` (>= MariaDB 10.2.1)
    - `NONE` (<= MariaDB 10.2.0)
- **Valid Values:** `NONE` (`0`), `CRC32` (`1`)

---

binlog_commit_wait_count

- **Description:** Configures the behavior of group commit for the binary log, which can help increase transaction throughput and is used to enable conservative mode of in-order parallel replication . With group commit for the binary log, the server can delay flushing a committed transaction into binary log until the given number of transactions are ready to be flushed as a group. The delay will however not be longer than the value set by binlog_commit_wait_usec. The default value of 0 means that no delay is introduced. Setting this value can reduce I/O on the binary log and give an increased opportunity for parallel apply on the replica when conservative mode of in-order parallel replication is enabled, but too high a value will decrease the transaction throughput. By monitoring the status variable binlog_group_commit_trigger_count (>=MariaDB 10.1.5) it is possible to see how often this is occurring.
- Starting with MariaDB 10.0.18 and MariaDB 10.1.4: If the server detects that one of the committing transactions T1 holds an InnoDB row lock that another transaction T2 is waiting for, then the commit will complete immediately without further delay. This helps avoid losing throughput when many transactions need conflicting locks. This often makes it safe to use this option without losing throughput on a replica with conservative mode of in-order parallel replication, provided the value of slave_parallel_threads is sufficiently high.
- **Commandline:** `--binlog-commit-wait-count=#]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `18446744073709551615`

binlog_commit_wait_usec

- **Description:** Configures the behavior of group commit for the binary log, which can help increase transaction throughput and is used to enable conservative mode of in-order parallel replication. With group commit for the binary log, the server can delay flushing a committed transaction into binary log until the transaction has waited the configured number of microseconds. By monitoring the status variable binlog_group_commit_trigger_timeout (>=MariaDB 10.1.5) it is possible to see how often group commits are made due to `binlog_commit_wait_usec`. As soon as the number of pending commits reaches binlog_commit_wait_count, the wait will be terminated, though. Thus, this setting only takes effect if `binlog_commit_wait_count` is non-zero.
- **Commandline:** `--binlog-commit-wait-usec#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `100000`
- **Range:** `0` to `18446744073709551615`

binlog_direct_non_transactional_updates

- **Description:** Replication inconsistencies can occur due when a transaction updates both transactional and non-transactional tables and the updates to the non-transactional tables are visible before being written to the binary log. This is because, to preserve causality, the non-transactional statements are written to the transaction cache, which is only flushed on commit. Setting binlog_direct_non_transactional_updates to 1 (0 is default) will cause non-transactional tables to be written straight to the binary log, rather than the transaction cache. This setting has no effect when row-based binary logging is used, as it requires statement-based logging. See binlog_format. Use with care, and only in situations where no dependencies exist between the non-transactional and transactional tables, for example INSERTing into a non-transactional table based upon the results of a SELECT from a transactional table.
- **Commandline:** `--binlog-direct-non-transactional-updates[=value]`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF (0)`

binlog_expire_logs_seconds

- **Description:** If non-zero, binary logs will be purged after `binlog_expire_logs_seconds` seconds. Possible purges happen at startup and at binary log rotation. From MariaDB 10.6.1, `binlog_expire_logs_seconds` and expire_logs_days are forms of aliases, such that changes to one automatically reflect in the other.
- **Commandline:** `--binlog-expire-logs-seconds=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `4294967295`
- **Introduced:** MariaDB 10.6.1

binlog_file_cache_size

- **Description:** Size of in-memory cache that is allocated when reading binary log and relay log files.
- **Commandline:** `--binlog-file-cache-size=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `16384`
- **Range:** `8192` to `18446744073709551615`
- **Introduced:** MariaDB 10.3.3

binlog_format

- **Description:** Determines whether replication is row-based, statement-based or mixed. Statement-based was the default until MariaDB 10.2.3. Be careful of changing the binary log format when a replication environment is

already running. See Binary Log Formats. Starting from MariaDB 10.0.22 a replica will apply any events it gets from the primary, regardless of the binary log format. `binlog_format` only applies to normal (not replicated) updates.

- **Commandline:** `--binlog-format=format`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `enumeration`
- **Default Value:**
  - `MIXED` (>= MariaDB 10.2.4)
  - `STATEMENT` (<= MariaDB 10.2.3)
- **Valid Values:** `ROW`, `STATEMENT` or `MIXED`

---

binlog_ignore_db

- **Description:** This option allows you to configure a replication primary to **not** write statements and transactions affecting databases that match a specified name into its binary log. Since the filtered statements or transactions will not be present in the binary log, its replicas will not be able to replicate them.
  - This option will **not** work with cross-database updates with statement-based logging. See the Statement-Based Logging section for more information.
  - Until MariaDB 11.2.0, only available as an option, not a system variable. This option can **not** be set dynamically.
  - When setting it on the command-line or in a server option group in an option file, the option does not accept a comma-separated list. If you would like to specify multiple filters, then you need to specify the option multiple times.
  - See Replication Filters for more information.
- **Commandline:** `--binlog-ignore-db=name`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** NULL
- **Introduced:** MariaDB 11.2.0

---

binlog_optimize_thread_scheduling

- **Description:** Run fast part of group commit in a single thread, to optimize kernel thread scheduling. On by default. Disable to run each transaction in group commit in its own thread, which can be slower at very high concurrency. This option is mostly for testing one algorithm versus another, and it should not normally be necessary to change it.
- **Commandline:** `--binlog-optimize-thread-scheduling` or `--skip-binlog-optimize-thread-scheduling`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `ON`

---

binlog_row_event_max_size

- **Description:** The maximum size of a row-based binary log event in bytes. Rows will be grouped into events smaller than this size if possible. The value has to be a multiple of 256. Until MariaDB 11.2.0, only available as an option, not a system variable.
- **Commandline:** `--binlog-row-event-max-size=val`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `8192`
- **Range:** `256` to `4294967040` (in multiples of 256)
- **Introduced:** MariaDB 11.2.0

---

binlog_row_image

- **Description:** Controls whether, in row-based replication, rows should be logged in 'FULL', 'NOBLOB' or 'MINIMAL' formats. In row-based replication (the variable has no effect with statement-based replication), each row change event contains an image for matching against when choosing the row to be updated, and another image containing the changes. Before the introduction of this variable, all columns were logged for both of these images. In certain circumstances, this is not necessary, and memory, disk and network resources can be saved by partial logging. Note

that to safely change this setting from the default, the table being replicated to must contain identical primary key definitions, and columns must be present, in the same order, and use the same data types as the original table. If these conditions are not met, matches may not be correctly determined and updates and deletes may diverge on the replica, with no warnings or errors returned.
    - `FULL` : All columns in the before and after image are logged. This is the default, and the only behavior in earlier versions.
    - `NOBLOB` : mariadbd avoids logging blob and text columns whenever possible (eg, blob column was not changed or is not part of primary key).
    - `MINIMAL` : A PK equivalent (PK columns or full row if there is no PK in the table) is logged in the before image, and only changed columns are logged in the after image.
- **Commandline:** `--binlog-row-image=value`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `enum`
- **Default Value:** `FULL`
- **Valid Values:** `FULL`, `NOBLOB` or `MINIMAL`

---

binlog_row_metadata

- **Description:** Controls the format used for binlog metadata logging.
    - `NO_LOG` : No metadata is logged (default).
    - `MINIMAL` : Only metadata required by a replica is logged.
    - `FULL` : All metadata is logged.
- **Commandline:** `--binlog-row-metadata=value`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `enum`
- **Default Value:** `NO_LOG`
- **Valid Values:** `NO_LOG`, `MINIMAL`, `FULL`
- **Introduced:** MariaDB 10.5.0

---

binlog_stmt_cache_size

- **Description:** If the binary log is active, this variable determines the size in bytes of the cache holding a record of binary log changes outside of a transaction. The variable binlog_cache_size, determines the cache size for binary log statements inside a transaction. The binlog_stmt_cache_disk_use and binlog_stmt_cache_use server status variables will indicate whether this variable needs to be increased (you want a low ratio of binlog_stmt_cache_disk_use to binlog_stmt_cache_use).
- **Commandline:** `--binlog-stmt-cache-size=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `32768`
- **Range - 32 bit:** `4096` to `4294967295`
- **Range - 64 bit:** `4096` to `18446744073709547520`

---

default_master_connection

- **Description:** In multi-source replication, specifies which connection will be used for commands and variables if you don't specify a connection.
- **Commandline:** None
- **Scope:** Session
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `''` (empty string)

---

encrypt_binlog

- **Description:** Encrypt binary logs (including relay logs). See Data at Rest Encryption and Encrypting Binary Logs.
- **Commandline:** `--encrypt-binlog[={0|1}]`
- **Scope:** Global

- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

expire_logs_days

- **Description:** Number of days after which the binary log can be automatically removed. By default 0, or no automatic removal. When using replication, should always be set higher than the maximum lag by any replica. Removals take place when the server starts up, when the binary log is flushed, when the next binary log is created after the previous one reaches the maximum size, or when running PURGE BINARY LOGS. Units are whole days (integer) until MariaDB 10.6.0, or 1/1000000 precision (double) from MariaDB 10.6.1.
  Starting from MariaDB 10.6.1, `expire_logs_days` and binlog_expire_logs_seconds are forms of aliases, such that changes to one automatically reflect in the other.
- **Commandline:** `--expire-logs-days=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0.000000` (>= MariaDB 10.6.1), `0` (<= MariaDB 10.6.0)
- **Range:** `0` to `99`

---

init_slave

- **Description:** Similar to init_connect, but the string contains one or more SQL statements, separated by semicolons, that will be executed by a replica server each time the SQL thread starts. These statements are only executed after the acknowledgement is sent to the replica and START SLAVE completes.
- **Commandline:** `--init-slave=name`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`
- **Related variables:** `init_connect`

---

log_bin

- **Description:** Whether binary logging is enabled or not. If the --log-bin option is used, log_bin will be set to ON, otherwise it will be OFF. If no `name` option is given for `--log-bin`, `datadir/'log-basename'-bin` or `'datadir'/mysql-bin` will be used (the latter if --log-basename is not specified). We strongly recommend you use either `--log-basename` or specify a filename to ensure that replication doesn't stop if the real hostname of the computer changes. The name option can optionally include an absolute path. If no path is specified, the log will be written to the data directory. The name can optionally include the file extension; it will be stripped and only the file basename will be used.
- **Commandline:** `--log-bin[=name]`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Related variables:** `sql_log_bin`

---

log_bin_basename

- **Description:** The full path of the binary log file names, excluding the extension. Its value is derived from the rules specified in `log_bin` system variable. This is a read-only variable only, there is no corresponding configuration file setting or command line option by the same name, use `log_bin` to set the basename path instead.
- **Commandline:** `No commandline option`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** None
- **Dynamic:** `No`

---

log_bin_compress

- **Description:** Whether or not the binary log can be compressed. `0` (the default) means no compression. See Compressing Events to Reduce Size of the Binary Log.
- **Commandline:** `--log-bin-compress`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

log_bin_compress_min_len

- **Description:** Minimum length of sql statement (in statement mode) or record (in row mode) that can be compressed. See Compressing Events to Reduce Size of the Binary Log.
- **Commandline:** `--log-bin-compress-min-len`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `256`
- **Range:** `10` to `1024`

---

log_bin_index

- **Description:** File that holds the names for last binlog files.
- **Commandline:** `--log-bin-index=name`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** None

---

log_bin_trust_function_creators

- **Description:** Functions and triggers can be dangerous when used with replication. Certain types of functions and triggers may have unintended consequences when the statements are applied on a replica. For that reason, there are some restrictions on the creation of functions and triggers when the binary log is enabled by default, such as:
  - When `log_bin_trust_function_creators` is `OFF` and `log_bin` is `ON`, `CREATE FUNCTION` and `ALTER FUNCTION` statements will trigger an error if the function is defined with any of the `NOT DETERMINISTIC`, `CONTAINS SQL` or `MODIFIES SQL DATA` characteristics.
  - This means that when `log_bin_trust_function_creators` is `OFF` and `log_bin` is `ON`, `CREATE FUNCTION` and `ALTER FUNCTION` statements will only succeed if the function is defined with any of the `DETERMINISTIC`, `NO SQL`, or `READS SQL DATA` characteristics.
  - When `log_bin_trust_function_creators` is `OFF` and `log_bin` is `ON`, the `SUPER` privilege is also required to execute the following statements:
    - `CREATE FUNCTION`
    - `CREATE TRIGGER`
    - `DROP TRIGGER`
  - Setting `log_bin_trust_function_creators` to `ON` removes these requirements around functions characteristics and the `SUPER` privileges.
  - See Binary Logging of Stored Routines for more information.
- **Commandline:** `--log-bin-trust-function-creators[={0|1}]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

log_slow_slave_statements

- **Description:** Log slow statements executed by replica thread to the slow log if it is open. Before MariaDB 10.1.13, this was only available as a mariadbd option, not a server variable.
- **Commandline:** `--log-slow-slave-statements`
- **Scope:** Global

- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:**
  - `ON` (>= MariaDB 10.2.4)
  - `OFF` (<= MariaDB 10.2.3)

---

log_slave_updates

- **Description:** If set to `0`, the default, updates on a replica received from a primary during replication are not logged in the replica's binary log. If set to `1`, they are. The replica's binary log needs to be enabled for this to have an effect. Set to `1` if you want to daisy-chain the replicas.
- **Commandline:** `--log-slave-updates`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

master_verify_checksum

- **Description:** Verify binlog checksums when reading events from the binlog on the primary.
- **Commandline:** `--master-verify-checksum=[0|1]`
- **Scope:** Global
- **Access Type:** Can be changed dynamically
- **Data Type:** `bool`
- **Default Value:** `OFF (0)`

---

max_binlog_cache_size

- **Description:** Restricts the size in bytes used to cache a multi-transactional query. If more bytes are required, a `Multi-statement transaction required more than 'max_binlog_cache_size' bytes of storage` error is generated. If the value is changed, current sessions are unaffected, only sessions started subsequently. See max_binlog_stmt_cache_size and binlog_cache_size.
- **Commandline:** `--max-binlog-cache-size=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `18446744073709547520`
- **Range:** `4096` to `18446744073709547520`

---

max_binlog_size

- **Description:** If the binary log exceeds this size after a write, the server rotates it by closing it and opening a new binary log. Single transactions will always be stored in the same binary log, so the server will wait for open transactions to complete before rotating. This figure also applies to the size of relay logs if max_relay_log_size is set to zero.
- **Commandline:** `--max-binlog-size=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1073741824` (1GB)
- **Range:** `4096 to 1073741824` (4KB to 1GB)

---

max_binlog_stmt_cache_size

- **Description:** Restricts the size used to cache non-transactional statements. See max_binlog_cache_size and binlog_stmt_cache_size.
- **Commandline:** `--max-binlog-stmt-cache-size=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`

- **Default Value:** `18446744073709547520` (64 bit), `4294963200` (32 bit)
- **Range:** `4096` to `18446744073709547520`

---

max_relay_log_size

- **Description:** Replica will rotate its relay log if it exceeds this size after a write. If set to 0, the max_binlog_size setting is used instead. Previously global only, since the implementation of multi-source replication, it can be set per session as well.
- **Commandline:** `--max-relay-log-size=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` , or `4096 to 1073741824` (4KB to 1GB)

---

read_binlog_speed_limit

- **Description:** Used to restrict the speed at which a replica can read the binlog from the primary. This can be used to reduce the load on a primary if many replicas need to download large amounts of old binlog files at the same time. The network traffic will be restricted to the specified number of kilobytes per second.
- **Commandline:** `--read-binlog-speed-limit=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0` (no limit)
- **Range:** `0` to `18446744073709551615`

---

relay_log

- **Description:** Relay log basename. If not set, the basename of the files will be `hostname-relay-bin` .
- **Commandline:** `--relay-log=file_name`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `filename`
- **Default Value:** `''` (none)

---

relay_log_basename

- **Description:** The full path of the relay log file names, excluding the extension. Its value is derived from the relay-log variable value.
- **Commandline:** `No commandline option`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** None
- **Dynamic:** `No`

---

relay_log_index

- **Description:** Name and location of the relay log index file, the file that keeps a list of the last relay logs. Defaults to *hostname*-relay-bin.index.
- **Commandline:** `--relay-log-index=name`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** None

---

relay_log_info_file

- **Description:** Name and location of the file where the `RELAY_LOG_FILE` and `RELAY_LOG_POS` options (i.e. the relay log position) for the CHANGE MASTER statement are written. The replica's SQL thread keeps this relay log position updated as it applies events.
  - See CHANGE MASTER TO: Option Persistence for more information.
- **Commandline:** `--relay-log-info-file=file_name`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** `relay-log.info`

---

relay_log_purge

- **Description:** If set to `1` (the default), relay logs will be purged as soon as they are no longer necessary.
- **Commandline:** `--relay-log-purge={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `ON`
- **Note:** In MySQL and in MariaDB before version 10.0.8 this variable was silently changed if you did CHANGE MASTER.

---

relay_log_recovery

- **Description:** If set to `1` (`0` is default), on startup the replica will drop all relay logs that haven't yet been processed, and retrieve relay logs from the primary. Can be useful after the replica has crashed to prevent the processing of corrupt relay logs. relay_log_recovery should always be set together with relay_log_purge. Setting `relay-log-recovery=1` with `relay-log-purge=0` can cause the relay log to be read from files that were not purged, leading to data inconsistencies.
- **Commandline:** `--relay-log-recovery`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

relay_log_space_limit

- **Description:** Specifies the maximum space to be used for the relay logs. The IO thread will stop until the SQL thread has cleared the backlog. By default `0`, or no limit.
- **Commandline:** `--relay-log-space-limit=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range - 32 bit:** `0` to `4294967295`
- **Range - 64 bit:** `0` to `18446744073709547520`

---

replicate_annotate_row_events

- **Description:** Tells the replica to reproduce annotate_rows_events received from the primary in its own binary log. This option is sensible only when used in tandem with the log_slave_updates option.
- **Commandline:** `--replicate-annotate-row-events`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:**
  - `ON` (>= MariaDB 10.2.4)
  - `OFF` (<= MariaDB 10.2.3)

---

replicate_do_db

- **Description:** This system variable allows you to configure a replica to apply statements and transactions affecting databases that match a specified name.
  - This system variable will **not** work with cross-database updates with statement-based logging. See the Statement-Based Logging section for more information.
  - When setting it dynamically with `SET GLOBAL`, the system variable accepts a comma-separated list of filters.
  - When setting it on the command-line or in a server option group in an option file, the system variable does not accept a comma-separated list. If you would like to specify multiple filters, then you need to specify the system variable multiple times.
  - See Replication Filters for more information.
- **Commandline:** `--replicate-do-db=name`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `''` (empty)

---

replicate_do_table

- **Description:** This system variable allows you to configure a replica to apply statements and transactions that affect tables that match a specified name. The table name is specified in the format: `dbname.tablename`.
  - This system variable will **not** work with cross-database updates with statement-based logging. See the Statement-Based Logging section for more information.
  - When setting it dynamically with `SET GLOBAL`, the system variable accepts a comma-separated list of filters.
  - When setting it on the command-line or in a server option group in an option file, the system variable does not accept a comma-separated list. If you would like to specify multiple filters, then you need to specify the system variable multiple times.
  - See Replication Filters for more information.
- **Commandline:** `--replicate-do-table=name`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `''` (empty)

---

replicate_events_marked_for_skip

- **Description:** Tells the replica whether to replicate events that are marked with the `@@skip_replication` flag. See Selectively skipping replication of binlog events for more information.
- **Commandline:** `--replicate-events-marked-for-skip`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `enumeration`
- **Default Value:** `replicate`
- **Valid Values:** `REPLICATE`, `FILTER_ON_SLAVE`, `FILTER_ON_MASTER`

---

replicate_ignore_db

- **Description:** This system variable allows you to configure a replica to ignore statements and transactions affecting databases that match a specified name.
  - This system variable will **not** work with cross-database updates with statement-based logging. See the Statement-Based Logging section for more information.
  - When setting it dynamically with `SET GLOBAL`, the system variable accepts a comma-separated list of filters.
  - When setting it on the command-line or in a server option group in an option file, the system variable does not accept a comma-separated list. If you would like to specify multiple filters, then you need to specify the system variable multiple times.
  - See Replication Filters for more information.
- **Commandline:** `--replicate-ignore-db=name`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `''` (empty)

---

replicate_ignore_table

- **Description:** This system variable allows you to configure a replica to ignore statements and transactions that affect tables that match a specified name. The table name is specified in the format: `dbname.tablename`.
  - This system variable will **not** work with cross-database updates with statement-based logging. See the Statement-Based Logging section for more information.
  - When setting it dynamically with `SET GLOBAL`, the system variable accepts a comma-separated list of filters.
  - When setting it on the command-line or in a server option group in an option file, the system variable does not accept a comma-separated list. If you would like to specify multiple filters, then you need to specify the system variable multiple times.
  - See Replication Filters for more information.
- **Commandline:** `--replicate-ignore-table=name`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `''` (empty)

---

replicate_rewrite_db

- **Description:** This option allows you to configure a replica to rewrite database names. It uses the format `primary_database->replica_database`. If a replica encounters a binary log event in which the default database (i.e. the one selected by the `USE` statement) is `primary_database`, then the replica will apply the event in `replica_database` instead.
  - This option will **not** work with cross-database updates with statement-based logging. See the Statement-Based Logging section for more information.
  - This option only affects statements that involve tables. This option does not affect statements involving the database itself, such as CREATE DATABASE, ALTER DATABASE, and DROP DATABASE.
  - When setting it on the command-line or in a server option group in an option file, the option does not accept a comma-separated list. If you would like to specify multiple filters, then you need to specify the option multiple times.
  - See Replication Filters for more information.
  - Before MariaDB 10.11, `replicate_rewrite_db` was not available as a system variable, only as a mariadbd option, and could not be set dynamically. From MariaDB 10.11 it is available as a dynamic system variable
- **Commandline:** `--replicate-rewrite-db=primary_database->replica_database`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `''` (empty)
- **Introduced:** MariaDB 10.11.0

---

replicate_wild_do_table

- **Description:** This system variable allows you to configure a replica to apply statements and transactions that affect tables that match a specified wildcard pattern. The wildcard pattern uses the same semantics as the `LIKE` operator.
  - This system variable will work with cross-database updates with statement-based logging. See the Statement-Based Logging section for more information.
  - When setting it dynamically with `SET GLOBAL`, the system variable accepts a comma-separated list of filters.
  - When setting it on the command-line or in a server option group in an option file, the system variable does not accept a comma-separated list. If you would like to specify multiple filters, then you need to specify the system variable multiple times.
  - See Replication Filters for more information.
- **Commandline:** `--replicate-wild-do-table=name`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `''` (empty)

---

replicate_wild_ignore_table

- **Description:** This system variable allows you to configure a replica to ignore statements and transactions that affect tables that match a specified wildcard pattern. The wildcard pattern uses the same semantics as the LIKE operator.
  - This system variable will work with cross-database updates with statement-based logging. See the Statement-Based Logging section for more information.
  - When setting it dynamically with SET GLOBAL, the system variable accepts a comma-separated list of filters.
  - When setting it on the command-line or in a server option group in an option file, the system variable does not

accept a comma-separated list. If you would like to specify multiple filters, then you need to specify the system variable multiple times.
    - See Replication Filters for more information.
- **Commandline:** `--replicate-wild-ignore-table=name`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `''` (empty)

---

report_host

- **Description:** The host name or IP address the replica reports to the primary when it registers. If left unset, the replica will not register itself. Reported by SHOW SLAVE HOSTS. Note that it is not sufficient for the primary to simply read the IP of the replica from the socket once the replica connects. Due to NAT and other routing issues, that IP may not be valid for connecting to the replica from the primary or other hosts.
- **Commandline:** `--report-host=host_name`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`

---

report_password

- **Description:** Replica password reported to the primary when it registers. Reported by SHOW SLAVE HOSTS if `--show-slave-auth-info` is set. This password has no connection with user privileges or with the replication user account password.
- **Commandline:** `--report-password=password`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`

---

report_port

- **Description:** The commandline option sets the TCP/IP port for connecting to the replica that will be reported to the replicating primary during the replica's registration. Viewing the variable will show this value.
- **Commandline:** `--report-port=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `65535`

---

report_user

- **Description:** Replica's account user name reported to the primary when it registers. Reported by SHOW SLAVE HOSTS if `--show-slave-auth-info` is set. This username has no connection with user privileges or with the replication user account.
- **Commandline:** `--report-user=name`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`

---

server_id

- **Description:** This system variable is used with MariaDB replication to identify unique primary and replica servers in a topology. This system variable is also used with the binary log to determine which server a specific transaction originated on.
    - When MariaDB replication is used with standalone MariaDB Server, each server in the replication topology must have a unique `server_id` value.
    - When MariaDB replication is used with MariaDB Galera Cluster, see Using MariaDB Replication with MariaDB Galera Cluster: Setting server_id on Cluster Nodes for more information on how to set the `server_id` values.

    - In MariaDB 10.2.1 and below, the default `server_id` value is `0`. If a replica's `server_id` value is `0`, then all primary's will refuse its connection attempts. If a primary's `server_id` value is `0`, then it will refuse all replica connection attempts.
- **Commandline:** `--server-id =#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1`
- **Range:** `1` to `4294967295`

---

skip_parallel_replication

- **Description:** If set when a transaction is written to the binlog, parallel apply of that transaction will be avoided on a replica where slave_parallel_mode is not `aggressive`. Can be used to avoid unnecessary rollback and retry for transactions that are likely to cause a conflict if replicated in parallel. See parallel replication.
- **Commandline:** None
- **Scope:** Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

skip_replication

- **Description:** Changes are logged into the binary log with the @@skip_replication flag set. Such events will not be replicated by replica that run with `--replicate-events-marked-for-skip` set different from its default of `REPLICATE`. See Selectively skipping replication of binlog events for more information.
- **Commandline:** None
- **Scope:** Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

slave_compressed_protocol

- **Description:** If set to 1 (0 is the default), will use compression for the replica/primary protocol if both primary and replica support this.
- **Commandline:** `--slave-compressed-protocol`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `0`

---

slave_ddl_exec_mode

- **Description:** Modes for how replication of DDL events should be executed. Legal values are `STRICT` and `IDEMPOTENT` (default). In `IDEMPOTENT` mode, the replica will not stop for failed DDL operations that would not cause a difference between the primary and the replica. In particular CREATE TABLE is treated as CREATE OR REPLACE TABLE and DROP TABLE is treated as `DROP TABLE IF EXISTS`.
- **Commandline:** `--slave-ddl-exec-mode=name`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `enumeration`
- **Default Value:** `IDEMPOTENT`
- **Valid Values:** `IDEMPOTENT`, `STRICT`

---

slave_domain_parallel_threads

- **Description:** When set to a non-zero value, each replication domain in one primary connection can reserve at most that many worker threads at any one time, leaving the rest (up to the value of slave_parallel_threads) free for other primary connections or replication domains to use in parallel. See Parallel Replication for details.

- **Commandline:** `--slave-domain-parallel-threads=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Valid Values:** `0` to `16383`

---

slave_exec_mode

- **Description:** Determines the mode used for replication error checking and conflict resolution. STRICT mode is the default, and catches all errors and conflicts. IDEMPOTENT mode suppresses duplicate key or no key errors, which can be useful in certain replication scenarios, such as when there are multiple primaries, or circular replication.
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `enumeration`
- **Default Value:** `IDEMPOTENT` (NDB), `STRICT` (All)
- **Valid Values:** `IDEMPOTENT`, `STRICT`

---

slave_load_tmpdir

- **Description:** Directory where the replica stores temporary files for replicating LOAD DATA INFILE statements. If not set, the replica will use tmpdir. Should be set to a disk-based directory that will survive restarts, or else replication may fail.
- **Commandline:** `--slave-load-tmpdir=path`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `file name`
- **Default Value:** `/tmp`

---

slave_max_allowed_packet

- **Description:** Maximum packet size in bytes for replica SQL and I/O threads. This value overrides max_allowed_packet for replication purposes. Set in multiples of 1024 (the minimum) up to 1GB
- **Commandline:** `--slave-max-allowed-packet=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1073741824`
- **Range:** `1024` to `1073741824`

---

slave_max_statement_time

- **Description:** A query that has taken more than this in seconds to run on the replica will be aborted. The argument will be treated as a decimal value with microsecond precision. A value of 0 (default) means no timeout.
- **Commandline:** `--slave-max-statement-time=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0.000000`
- **Range:** `0` to `31536000`
- **Introduced:** MariaDB 10.10

---

slave_net_timeout

- **Description:** Time in seconds for the replica to wait for more data from the primary before considering the connection broken, after which it will abort the read and attempt to reconnect. The retry interval is determined by the MASTER_CONNECT_RETRY open for the CHANGE MASTER statement, while the maximum number of reconnection attempts is set by the master-retry-count option. The first reconnect attempt takes place immediately.
- **Commandline:** `--slave-net-timeout=#`
- **Scope:** Global

- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:**
  - `60 (1 minute)`
- **Range:** `1` to `31536000`

---

slave_parallel_max_queued

- **Description:** When parallel_replication is used, the SQL thread will read ahead in the relay logs, queueing events in memory while looking for opportunities for executing events in parallel. This system variable sets a limit for how much memory it will use for this.
  - The configured value of this system variable is actually allocated for each worker thread, so the total allocation is actually equivalent to the following:
    - `slave_parallel_max_queued * slave_parallel_threads`
  - This system variable is only meaningful when parallel replication is configured (i.e. when `slave_parallel_threads > 0`).
  - See Parallel Replication: Configuring the Maximum Size of the Parallel Slave Queue for more information.
- **Commandline:** `--slave-parallel-max-queued=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `131072`
- **Range:** `0` to `2147483647`

---

slave_parallel_mode

- **Description:** Controls what transactions are applied in parallel when using parallel replication.
  - `optimistic` : tries to apply most transactional DML in parallel, and handles any conflicts with rollback and retry. See optimistic mode.
  - `conservative` : limits parallelism in an effort to avoid any conflicts. See conservative mode.
  - `aggressive` : tries to maximize the parallelism, possibly at the cost of increased conflict rate.
  - `minimal` : only parallelizes the commit steps of transactions.
  - `none` disables parallel apply completely.
- **Commandline:** None
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `enum`
- **Default Value:** `optimistic` (>= MariaDB 10.5.1), `conservative` (<= MariaDB 10.5.0)
- **Valid Values:** `conservative`, `optimistic`, `none`, `aggressive` and `minimal`

---

slave_parallel_threads

- **Description:** This system variable is used to configure parallel replication.
  - If this system variable is set to a value greater than `0`, then its value will determine how many replica worker threads will be created to apply binary log events in parallel.
  - If this system variable is set to `0` (which is the default value), then no replica worker threads will be created. Instead, when replication is enabled, binary log events are applied by the replica's SQL thread.
  - The replica threads must be stopped in order to change this option's value dynamically.
  - Events that were logged with GTIDs with different `gtid_domain_id` values can be applied in parallel in an out-of-order manner. Each `gtid_domain_id` can use the number of threads configured by `slave_domain_parallel_threads`.
  - Events that were group-committed on the primary can be applied in parallel in an in-order manner, and the specific behavior can be configured by setting `slave_parallel_mode`.
- **Commandline:** `--slave-parallel-threads=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `16383`

---

slave_parallel_workers

- **Description:** Alias for slave_parallel_threads.
- **Commandline:** `--slave-parallel-workers=#`

---

slave_run_triggers_for_rbr

- **Description:** See Running triggers on the slave for Row-based events for a description and use-case for this setting.
- **Commandline:** `--slave-run-triggers-for-rbr=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `enum`
- **Default Value:** `NO`
- **Valid Values:** `NO`, `YES`, `LOGGING`, or `ENFORCE` (>= MariaDB 10.5.2)

---

slave_skip_errors

- **Description:** When an error occurs on the replica, replication usually halts. This option permits a list of error codes to ignore, and for which replication will continue. This option should never be needed in normal use, and careless use could lead to replica that are out of sync with primary's. Error codes are in the format of the number from the replica error log. Using `all` as an option permits the replica the keep replicating no matter what error it encounters, an option you would never normally need in production and which could rapidly lead to data inconsistencies. A count of these is kept in slave_skipped_errors.
- **Commandline:** `--slave-skip-errors=[error_code1,error_code2,...|all|ddl_exist_errors]`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** `OFF`
- **Valid Values:** `[list of error codes]`, `ALL`, `OFF`

---

slave_sql_verify_checksum

- **Description:** Verify binlog checksums when the replica SQL thread reads events from the relay log.
- **Commandline:** `--slave-sql-verify-checksum=[0|1]`
- **Scope:** Global
- **Access Type:** Can be changed dynamically
- **Data Type:** `bool`
- **Default Value:** `ON (1)`

---

slave_transaction_retries

- **Description:** Number of times a replication replica retries to execute an SQL thread after it fails due to InnDB deadlock or by exceeding the transaction execution time limit. If after this number of tries the SQL thread has still failed to execute, the replica will stop with an error. See also the innodb_lock_wait_timeout system variable.
- **Commandline:** `--slave-transaction-retries=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `10`
- **Range - 32 bit:** `0` to `4294967295`
- **Range - 64 bit:** `0` to `18446744073709547520`

---

slave_transaction_retry_errors

- **Description:** When an error occurs during a transaction on the replica, replication usually halts. By default, transactions that caused a deadlock or elapsed lock wait timeout will be retried. One can add other errors to the the list of errors that should be retried by adding a comma-separated list of error numbers to this variable. This is particularly useful in some Spider setups. Some recommended errors to retry for Spider are 1158,1159,1160,1161,1429,2013,12701.(From MariaDB 10.4.5, these are in the default value)
- **Commandline:** `--slave-transaction_retry-errors=[error_code1,error_code2,...]`

- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:**
  - `1158,1159,1160,1161,1205,1213,1429,2013,12701` (>= MariaDB 10.4.5)
  - `1213,1205` (>= MariaDB 10.3.3)
- **Valid Values:** `comma-separated list of error codes`
- **Introduced:** MariaDB 10.3.3

---

slave_transaction_retry_interval

- **Description:** Interval in seconds for the replica SQL thread to retry a failed transaction due to a deadlock, elapsed lock wait timeout or an error listed in slave_transaction_retry_errors. The interval is calculated as `max(slave_transaction_retry_interval, min(retry_count, 5))`.
- **Commandline:** `--slave-transaction-retry-interval=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `3600`
- **Introduced:** MariaDB 10.3.3

---

slave_type_conversions

- **Description:** Determines the type conversion mode on the replica when using row-based replication, including replications in MariaDB Galera cluster. Multiple options can be set, delimited by commas. If left empty, the default, type conversions are disallowed. The variable is dynamic and a change in its value takes effect immediately. This variable tells the server what to do if the table definition is different between the primary and replica (for example a column is 'int' on the primary and 'bigint' on the replica).
  - `ALL_NON_LOSSY` means that all safe conversions (no data loss) are allowed.
  - `ALL_LOSSY` means that all lossy conversions are allowed (for example 'bigint' to 'int'). This, however, does not imply that safe conversions (non-lossy) are allowed as well. In order to allow all conversions, one needs to allow both lossy as well as non-lossy conversions by setting this variable to ALL_NON_LOSSY,ALL_LOSSY.
  - Empty (default) means that the server should give an error and replication should stop if the table definition is different between the primary and replica.
- **Commandline:** `--slave-type-conversions=set`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `set`
- **Default Value:** `Empty variable`
- **Valid Values:** `ALL_LOSSY`, `ALL_NON_LOSSY`, empty

---

sql_log_bin

- **Description:** If set to 0 (1 is the default), no logging to the binary log is done for the client. Only clients with the SUPER privilege can update this variable. Does not affect the replication of events in a Galera cluster.
- **Scope:** Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `1`

---

sql_slave_skip_counter

- **Description:** Number of events that a replica skips from the primary. If this would cause the replica to begin in the middle of an event group, the replica will instead begin from the beginning of the next event group. See SET GLOBAL sql_slave_skip_counter.
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`

---

sync_binlog

- **Description:** MariaDB will synchronize its binary log file to disk after this many events. The default is 0, in which case the operating system handles flushing the file to disk. 1 is the safest, but slowest, choice, since the file is flushed after each write. If autocommit is enabled, there is one write per statement, otherwise there's one write per transaction. If the disk has cache backed by battery, synchronization will be fast and a more conservative number can be chosen.
- **Commandline:** `--sync-binlog=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `4294967295`

---

sync_master_info

- **Description:** A replication replica will synchronize its master.info file to disk after this many events. If set to 0, the operating system handles flushing the file to disk.
- **Commandline:** `--sync-master-info=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `10000`

---

sync_relay_log

- **Description:** The MariaDB server will synchronize its relay log to disk after this many writes to the log. The default until MariaDB 10.1.7 was 0, in which case the operating system handles flushing the file to disk. 1 is the safest, but slowest, choice, since the file is flushed after each write. If autocommit is enabled, there is one write per statement, otherwise there's one write per transaction. If the disk has cache backed by battery, synchronization will be fast and a more conservative number can be chosen.
- **Commandline:** `--sync-relay-log=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `10000`

---

sync_relay_log_info

- **Description:** A replication replica will synchronize its relay-log.info file to disk after this many transactions. The default until MariaDB 10.1.7 was 0, in which case the operating system handles flushing the file to disk. 1 is the most secure choice, because at most one event could be lost in the event of a crash, but it's also the slowest.
- **Commandline:** `--sync-relay-log-info=#`
- **Scope:** Global,
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `10000`
- **Range:** `0` to `4294967295`

3.1.12 Replication and Binary Log Status Variables

The terms *master* and *slave* have historically been used in replication, but the terms terms *primary* and *replica* are now preferred. The old terms are used still used in parts of the documentation, and in MariaDB commands, although MariaDB 10.5 has begun the process of renaming. The documentation process is ongoing. See MDEV-18777 to follow progress on this effort.

Contents

The following status variables are useful in binary logging and replication. See Server Status Variables for a complete list of status variables that can be viewed with SHOW STATUS.

See also the Full list of MariaDB options, system and status variables.

`Binlog_bytes_written`

- **Description:** The number of bytes written to the binary log.
- **Scope:** Global
- **Data Type:** `numeric`

`Binlog_cache_disk_use`

- **Description:** Number of transactions which used a temporary disk cache because they could not fit in the regular binary log cache, being larger than binlog_cache_size. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

`Binlog_cache_use`

- **Description:** Number of transaction which used the regular binary log cache, being smaller than binlog_cache_size. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global

- **Data Type:** `numeric`

`Binlog_commits`

- **Description:** Total number of transactions committed to the binary log.
- **Scope:** Global
- **Data Type:** `numeric`

`Binlog_group_commit_trigger_count`

- **Description:** Total number of group commits triggered because of the number of binary log commits in the group reached the limit set by the variable binlog_commit_wait_count. See Group commit for the binary log.
- **Scope:** Global
- **Data Type:** `numeric`

`Binlog_group_commit_trigger_lock_wait`

- **Description:** Total number of group commits triggered because a binary log commit was being delayed because of a lock wait where the lock was held by a prior binary log commit. When this happens the later binary log commit is placed in the next group commit. See Group commit for the binary log.
- **Scope:** Global
- **Data Type:** `numeric`

`Binlog_group_commit_trigger_timeout`

- **Description:** Total number of group commits triggered because of the time since the first binary log commit reached the limit set by the variable binlog_commit_wait_usec. See Group commit for the binary log.
- **Scope:** Global
- **Data Type:** `numeric`

`Binlog_group_commits`

- **Description:** Total number of group commits done to the binary log. See Group commit for the binary log.
- **Scope:** Global
- **Data Type:** `numeric`

`Binlog_snapshot_file`

- **Description:** The binary log file. Unlike SHOW MASTER STATUS, can be queried in a transactionally consistent way, irrespective of which other transactions have been committed since the snapshot was taken. See Enhancements for START TRANSACTION WITH CONSISTENT SNAPSHOT.
- **Scope:** Global
- **Data Type:** `string`

`Binlog_snapshot_position`

- **Description:** The binary log position. Unlike SHOW MASTER STATUS, can be queried in a transactionally consistent way, irrespective of which other transactions have been committed since the snapshot was taken. See Enhancements for START TRANSACTION WITH CONSISTENT SNAPSHOT.
- **Scope:** Global
- **Data Type:** `numeric`

`Binlog_stmt_cache_disk_use`

- **Description:** Number of non-transaction statements which used a temporary disk cache because they could not fit in the regular binary log cache, being larger than binlog_stmt_cache_size. The global value can be flushed by `FLUSH STATUS` .

- **Scope:** Global
- **Data Type:** `numeric`

`Binlog_stmt_cache_use`

- **Description:** Number of non-transaction statement which used the regular binary log cache, being smaller than binlog_stmt_cache_size. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

`Com_change_master`

- **Description:** Number of CHANGE MASTER TO statements executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Com_show_binlog_status`

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.5.2

`Com_show_master_status`

- **Description:** Number of SHOW MASTER STATUS commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Removed:** MariaDB 10.5.2

`Com_show_new_master`

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Removed:** `MariaDB 5.5`

`Com_show_slave_hosts`

- **Description:** Number of SHOW SLAVE HOSTS commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Com_show_slave_status`

- **Description:** Number of SHOW SLAVE STATUS commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Com_slave_start`

- **Description:** Number of START SLAVE commands executed. Removed in MariaDB 10.0, see Com_start_slave.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Removed:** `MariaDB 10.0`

Com_slave_stop

- **Description:** Number of STOP SLAVE commands executed. Removed in MariaDB 10.0, see Com_stop_slave.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Removed:** `MariaDB 10.0`

Com_start_all_slaves

- **Description:** Number of START ALL SLAVES commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_start_slave

- **Description:** Number of START SLAVE commands executed. Replaces the old Com_slave_start.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_stop_all_slaves

- **Description:** Number of STOP ALL SLAVES commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_stop_slave

- **Description:** Number of STOP SLAVE commands executed. Replaces the old Com_slave_stop.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Master_gtid_wait_count

- **Description:** Number of times MASTER_GTID_WAIT called.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Master_gtid_wait_time

- **Description:** Total number of time spent in MASTER_GTID_WAIT.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Master_gtid_wait_timeouts

- **Description:** Number of timeouts occurring in MASTER_GTID_WAIT.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rpl_status

- **Description:** For showing the status of fail-safe replication. Removed in MySQL 5.6, still present in MariaDB 10.0.

Rpl_transactions_multi_engine

- **Description:** Number of replicated transactions that involved changes in multiple (transactional) storage engines,

before considering the update of `mysql.gtid_slave_pos`. These are transactions that were already cross-engine, independent of the GTID position update introduced by replication. The global value can be flushed by `FLUSH STATUS`.

- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.3.1

Slave_connections

- **Description:** Number of REGISTER_SLAVE attempts. In practice the number of times slaves has tried to connect to the master.
- **Scope:** Global
- **Data Type:** `numeric`

Slave_heartbeat_period

- **Description:** Time in seconds that a heartbeat packet is requested from the master by a slave.
- **Scope:** Global
- **Data Type:** `numeric`

Slave_open_temp_tables

- **Description:** Number of temporary tables the slave has open.
- **Scope:** Global
- **Data Type:** `numeric`

Slave_received_heartbeats

- **Description:** Number of heartbeats the slave has received from the master.
- **Scope:** Global
- **Data Type:** `numeric`

Slave_retried_transactions

- **Description:** Number of times the slave has retried transactions since the server started. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

Slave_running

- **Description:** Whether the default connection slave is running (both I/O and SQL threads are running) or not.
- **Scope:** Global
- **Data Type:** `numeric`

Slave_skipped_errors

- **Description:** The number of times a slave has skipped errors defined by slave-skip-errors.
- **Scope:** Global
- **Data Type:** `numeric`

Slaves_connected

- **Description:** Number of slaves connected.
- **Scope:** Global
- **Data Type:** `numeric`

`Slaves_running`

- **Description:** Number of slave SQL threads running.
- **Scope:** Global
- **Data Type:** `numeric`

`Transactions_gtid_foreign_engine`

- **Description:** Number of replicated transactions where the update of the `gtid_slave_pos` table had to choose a storage engine that did not otherwise participate in the transaction. This can indicate that setting gtid_pos_auto_engines might be useful. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.3.1

`Transactions_multi_engine`

- **Description:** Number of transactions that changed data in multiple (transactional) storage engines. If this is significantly larger than Rpl_transactions_multi_engine, it indicates that setting gtid_pos_auto_engines could reduce the need for cross-engine transactions. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.3.1

3.1.13 Binary Log

The binary log contains a record of all changes to the databases, both data and structure. It consists of a set of binary log files and an index.

It is necessary for replication, and can also be used to restore data after a backup.

Overview of the Binary Log

The binary log contains a record of all changes to the databases

Activating the Binary Log

Activating the Binary Log.

Using and Maintaining the Binary Log

Using and maintaining the binary log.

Binary Log Formats

The three binary logging formats.

Binary Logging of Stored Routines

Stored routines require extra consideration when binary logging.

SHOW BINARY LOGS

SHOW BINARY LOGS lists all binary logs on the server.

PURGE BINARY LOGS

PURGE BINARY LOGS removes all binary logs from the server, prior to the provided date or log file.

SHOW BINLOG EVENTS

Show events in the binary log.

SHOW MASTER STATUS

Status information about the binary log.

Binlog Event Checksums

Including a checksum in binlog events.

Binlog Event Checksum Interoperability

Replicating between servers with differing binlog checksum availability

Group Commit for the Binary Log

Optimization when the server is run with innodb_flush_logs_at_trx_commit or sync_binlog.

mariadb-binlog

mariadb-binlog utility for processing binary log files.

Transaction Coordinator Log

The transaction coordinator log (tc_log) is used to coordinate transactions...

Compressing Events to Reduce Size of the Binary Log

Binlog events can be compressed to save space on disk and in network transfers.

Encrypting Binary Logs

Data-at-rest encryption for binary logs and relay logs.

Flashback

Rollback instances/databases/tables to an old snapshot.

Relay Log

Event log created by the replica from the primary binary log.

Replication and Binary Log System Variables

Replication and binary log system variables.

There are 5 related questions.

3.1.13.1 Overview of the Binary Log

The binary log contains a record of all changes to the databases, both data and structure, as well as how long each statement took to execute. It consists of a set of binary log files and an index.

This means that statements such as CREATE, ALTER, INSERT, UPDATE and DELETE will be logged, but statements that have no effect on the data, such as SELECT and SHOW, will not be logged. If you want to log these (at a cost in performance), use the general query log.

If a statement may potentially have an effect, but doesn't, such as an UPDATE or DELETE that returns no rows, it will still be logged (this applies to the default statement-based logging, not to row-based logging - see Binary Log Formats).

The purpose of the binary log is to allow replication, where data is sent from one or more masters to one or more slave servers based on the contents of the binary log, as well as assisting in backup operations.

A MariaDB server with the binary log enabled will run slightly more slowly.

It is important to protect the binary log, as it may contain sensitive information, including passwords.

Binary logs are stored in a binary, not plain text, format, and so are not viewable with a regular editor. However, MariaDB includes mariadb-binlog, a commandline tool for plain text processing of binary logs.

3.1.13.2 Activating the Binary Log

Contents

To enable binary logging, start the server with the `--log-bin[=name]` option.

If you specify a filename with an extension (for example `.log`), the extension will be silently ignored.

If you don't provide a name (which can, optionally, include an absolute path), the default will be `datadir/log-basename-bin`, `datadir/mysql-bin` or `datadir/mariadb-bin` (the latter two if --log-basename is not specified, and dependent on server version). Datadir is determined by the value of the datadir system variable.

We strongly recommend you use either --log-basename or specify a filename to ensure that replication doesn't stop if the

hostname of the computer changes.

The directory storing the binary logs will contain a binary log index, as well as the individual binary log files.

The binary log files will have a series of numbers as filename extensions. Each additional binary log will increment the extension number, so the oldest binary logs will have lower numbers, the most recent, higher numbers.

A new binary log, with a new extension, is created every time the server starts, the logs are flushed, or the maximum size is reached (determined by max_binlog_size).

The binary log index file contains a master list of all the binary logs, in order.

A sample listing from a directory containing the binary logs:

```
shell> ls -l
total 100
...
-rw-rw---- 1 mysql adm 2098 Apr 19 00:46 mariadb-bin.000079
-rw-rw---- 1 mysql adm  332 Apr 19 00:56 mariadb-bin.000080
-rw-rw---- 1 mysql adm  347 Apr 19 07:36 mariadb-bin.000081
-rw-rw---- 1 mysql adm  306 Apr 20 07:15 mariadb-bin.000082
-rw-rw---- 1 mysql adm  332 Apr 20 07:41 mariadb-bin.000083
-rw-rw---- 1 mysql adm  373 Apr 21 07:56 mariadb-bin.000084
-rw-rw---- 1 mysql adm  347 Apr 21 09:09 mariadb-bin.000085
-rw-rw---- 1 mysql adm  398 Apr 21 21:24 mariadb-bin.000086
-rw-rw---- 1 mysql adm  816 Apr 21 17:05 mariadb-bin.index
```

The binary log index file will by default have the same name as the individual binary logs, with the extension .index. You can specify an alternative name with the `--log-bin-index[=filename]` option.

Clients with the SUPER privilege can disable and re-enable the binary log for the current session by setting the sql_log_bin variable.

```
SET sql_log_bin = 0;

SET sql_log_bin = 1;
```

Binary Log Format

There are three formats for the binary log. The default is mixed logging, which is a mix of statement-based and row-based logging. See Binary Log Formats for a full discussion.

3.1.13.3 Using and Maintaining the Binary Log

Contents

See Overview of the Binary Log for a general overview of what the binary log is, and Activating the Binary Log for how to make sure it's running on your system.

For details on using the binary log for replication, see the Replication section.

Purging Log Files

To delete all binary log files on the server, run the RESET MASTER command. To delete all binary logs before a certain datetime, or up to a certain number, use PURGE BINARY LOGS.

> If a replica is active but has yet to read from a binary log file you attempt to delete, the statement will fail with an error. However, if the replica is not connected and has yet to read from a log file you delete, the file will be deleted, but the replica will be unable to continue replicating once it connects again.

Log files can also be removed automatically with the expire_logs_days system variable. This is set to 0 by default (no removal), but can be set to a time, in days, after which a binary log file will be automatically removed. Log files will only be checked for being older than expire_logs_days upon log rotation, so if your binary log only fills up slowly and does not reach max_binlog_size on a daily basis, you may see older log files still being kept. You can also force log rotation, and so expiry deletes, by running FLUSH BINARY LOGS on a regular basis. Always set expire_logs_days higher than any possible replica lag.

From MariaDB 10.6, the binlog_expire_logs_seconds variable allows more precise control over binlog deletion, and takes precedence if both are non-zero.

> If the binary log index file has been removed, or incorrectly manually edited, all of the above forms of purging log files will fail. The .index file is a plain text file, and can be manually recreated or edited so that it lists only the binary log files that are present, in numeric/age order.

Examples

```
PURGE BINARY LOGS TO 'mariadb-bin.000063';
```

```
PURGE BINARY LOGS BEFORE '2013-04-22 09:55:22';
```

Safely Purging Binary Log Files While Replicating

To be sure replication is not broken while deleting log files, perform the following steps:

- Get a listing of binary log files on the primary by running SHOW BINARY LOGS.
- Go to each replica server and run SHOW SLAVE STATUS to check which binary log file each replica is currently reading.
- Find the earliest log file still being read by a replica. No log files before this one will be needed.
- If you wish, make a backup of the log files to be deleted
- Purge all log files before (not including) the file identified above.

Binary Log Format

There are three formats for the binary log. The default is statement-based logging, while row-based logging and a mix of the two formats are also possible. See Binary Log Formats for a full discussion.

Selectively Logging to the Binary Log

By default, all changes to data or data structure are logged. This behavior can be changed by starting the server with the `--binlog-ignore-db=database_name` or `--binlog-do-db=database_name` options.

`--binlog-ignore-db=database_name` specified a database to ignore for logging purposes, while `--binlog-do-db=database_name` will not log any statements unless they apply to the specified database.

> Neither option accepts comma-delimited lists of multiple databases as an option, since a database name can contain a comma. To apply to multiple databases, use the option multiple times.

`--binlog-ignore-db=database_name` behaves differently depending on whether statement-based or row-based logging is used. For statement-based logging, the server will not log any statement where the *default database* is database_name. The default database is set with the USE statement.

Similarly, `--binlog-do-db=database_name` also behaves differently depending on whether statement-based or row-based logging is used.

For statement-based logging, the server will only log statement where the *default database* is database_name. The default database is set with the USE statement.

For row-based logging, the server will log any updates to any tables in the named database/s, irrespective of the current database.

Examples

Assume the server has started with the option `--binlog-ignore-db=employees`. The following example *is* logged if

statement-based logging is used, and *is not* logged with row-based logging.

```
USE customers;
UPDATE employees.details SET bonus=bonus*1.2;
```

This is because statement-based logging examines the default database, in this case, `customers`. Since `customers` is not specified in the ignore list, the statement will be logged. If row-based logging is used, the example will not be logged as updates are written to the tables in the `employees` database.

Assume instead the server started with the option `--binlog-do-db=employees`. The following example *is not* logged if statement-based logging is used, and *is* logged with row-based logging.

```
USE customers;
UPDATE employees.details SET bonus=bonus*1.2;
```

This is again because statement-based logging examines the default database, in this case, `customers`. Since `customers` is not specified in the do list, the statement will not be logged. If row-based logging is used, the example will be logged as updates are written to the tables in the `employees` database.

Effects of Full Disk Errors on Binary Logging

If MariaDB encounters a full disk error while trying to write to a binary log file, then it will keep retrying the write every 60 seconds. Log messages will get written to the error log every 600 seconds. For example:

```
2018-11-27  2:46:46 140278181563136 [Warning] mysqld: Disk is full writing '/var/lib/mariadb-
bin.00001' (Errcode: 28 "No space left on device"). Waiting for someone to free space... (Expect
up to 60 secs delay for server to continue after freeing disk space)
2018-11-27  2:46:46 140278181563136 [Warning] mysqld: Retry in 60 secs. Message reprinted in 600
secs
```

However, if MariaDB encounters a full disk error while trying to open a new binary log file, then it will disable binary logging entirely. A log message like the following will be written to the error log:

```
2018-11-27  3:30:49 140278181563136 [ERROR] Could not open '/var/lib/mariadb-bin.00002 for
logging (error 28). Turning logging off for the whole duration of the MySQL server process. To
turn it on again: fix the cause, shutdown the MySQL server and restart it.
2018-11-27  3:30:49 140278181563136 [ERROR] mysqld: Error writing file '(null)' (errno: 9 "Bad
file descriptor")
2018-11-27  3:30:49 140278181563136 [ERROR] mysqld: Error writing file '(null)' (errno: 28 "No
space left on device")
```

3.1.13.4 Binary Log Formats

Contents

Supported Binary Log Formats

There are three supported formats for binary log events:

- Statement-Based Logging
- Row-Based Logging
- Mixed Logging

Regardless of the format, binary log events are always stored in a binary format, rather than in plain text. MariaDB includes the mariadb-binlog utility that can be used to output binary log events in a human-readable format.

You may want to set the binary log format in the following cases:

- If you execute single statements that update many rows, then statement-based logging will be more efficient than row-based logging for the replica to download.
- If you execute many statements that don't affect any rows, then row-based logging will be more efficient than statement-based logging for the replica to download.
- If you execute statements that take a long time to complete, but they ultimately only insert, update, or delete a few rows in the table, then row-based logging will be more efficient than statement-based logging for the replica to apply.

The default is mixed logging which is replication-safe and requires less storage space than row logging.

The storage engine API also allows storage engines to set or limit the logging format, which helps reduce errors with replicating between primaries and replicas with different storage engines.

Statement-Based Logging

In MariaDB 10.2.3 and before, statement-based logging was the default. Mixed logging is now the default.

When statement-based logging is enabled, statements are logged to the binary log exactly as they were executed. Temporary tables created on the primary will also be created on the replica. This mode is only recommended where one needs to keep the binary log as small as possible, the primary and replica have identical data (including using the same storage engines for all tables), and all functions being used are deterministic (repeatable with the same arguments). Statements and tables using timestamps or auto_increment are safe to use with statement-based logging.

This mode can be enabled by setting the binlog_format system variable to `STATEMENT`.

In certain cases when it would be impossible to execute the statement on the replica, the server will switch to row-based logging for the statement. Some cases of this are:

- When replication has been changed from row-based to statement-based and a statement uses data from a temporary table created during row-based mode. In this case, the temporary tables are not stored on the replica, so row logging is the only alternative.
- ALTER TABLE of a table using a storage engine that stores data remotely, such as the S3 storage engine, to another storage engine.
- One is using SEQUENCE's in the statement or the CREATE TABLE definition.

In certain cases, a statement may not be deterministic, and therefore not safe for replication. If MariaDB determines that an unsafe statement has been executed, then it will issue a warning. For example:

```
[Warning] Unsafe statement written to the binary log using statement format since
  BINLOG_FORMAT = STATEMENT. The statement is unsafe because it uses a LIMIT clause. This
  is unsafe because the set of rows included cannot be predicted.
```

See Unsafe Statements for Statement-based Replication for more information.

If you need to execute non-deterministic statements, then it is safer to use mixed logging or row-based.

Mixed Logging

Mixed logging is the default binary log format.

When mixed logging is enabled, the server uses a combination of statement-based logging and row-based logging. Statement-based logging is used where possible, but when the server determines a statement may not be safe for statement-based logging, it will use row-based logging instead. See Unsafe Statements for Statement-based Replication: Unsafe Statements for a list of unsafe statements.

During one transaction, some statements may be logged with row logging while others are logged with statement-based logging.

This mode can be enabled by setting the binlog_format system variable to `MIXED`.

Row-Based Logging

When row-based logging is enabled, DML statements are **not** logged to the binary log. Instead, each insert, update, or delete performed by the statement for each row is logged to the binary log separately. DDL statements are still logged to the binary log.

Row-based logging uses more storage than the other log formats but is the safest to use. In practice mixed logging should be as safe.

If one wants to be able to see the original query that was logged, one can enable annotated rows events, that is shown with mariadb-binlog, with --binlog-annotate-row-events. This option is on by default.

This mode can be enabled by setting the binlog_format system variable to `ROW`.

Compression of the Binary Log

Compression of the binary log can be used with any of the binary log formats, but the best results come from using mixed or row-based logging. You can enable compression by using the --log_bin_compress startup option.

Configuring the Binary Log Format

The format for binary log events can be configured by setting the binlog_format system variable. If you have the SUPER privilege, then you can change it dynamically with SET GLOBAL. For example:

```
SET GLOBAL binlog_format='ROW';
```

You can also change it dynamically for just a specific session with SET SESSION. For example:

```
SET SESSION binlog_format='ROW';
```

It can also be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
binlog_format=ROW
```

Be careful when changing the binary log format when using replication. When you change the binary log format on a server, it only changes the format for that server. Changing the binary log format on a primary has no effect on the replica's binary log format. This can cause replication to give inconsistent results or to fail.

Be careful changing the binary log format dynamically when the server is a replica and parallel replication is enabled. If you change the global value dynamically, then that does not also affect the session values of any currently running threads. This can cause problems with parallel replication, because the worker threads will remain running even after STOP SLAVE is executed. This can be worked around by resetting the slave_parallel_threads system variable. For example:

```
STOP SLAVE;
SET GLOBAL slave_parallel_threads=0;
SET GLOBAL binlog_format='ROW';
SET GLOBAL slave_parallel_threads=4;
START SLAVE
```

Effect of the Binary Log Format on Replicas

In MariaDB 10.0.22 and later, a replica will apply any events it gets from the primary, regardless of the binary log format. The binlog_format system variable only applies to normal (not replicated) updates.

If you are running MySQL or an older MariaDB than 10.0.22, you should be aware of that if you are running the replica in `binlog_format=STATEMENT` mode, the replica will stop if the primary is used with `binlog_format` set to anything else than `STATEMENT`.

The binary log format is upwards-compatible. This means replication should always work if the replica is the same or a newer version of MariaDB than the primary.

The mysql Database

Statements that affect the `mysql` database can be logged in a different way to that expected.

If the mysql database is edited directly, logging is performed as expected according to the binlog_format. Statements that directly edit the mysql database include INSERT, UPDATE, DELETE, REPLACE, DO, LOAD DATA INFILE, SELECT, and

TRUNCATE TABLE.

If the `mysql` database is edited indirectly, statement logging is used regardless of binlog_format setting. Statements editing the `mysql` database indirectly include GRANT, REVOKE, SET PASSWORD, RENAME USER, ALTER, DROP and CREATE (except for the situation described below).

CREATE TABLE ... SELECT can use a combination of logging formats. The CREATE TABLE portion of the statement is logged using statement-based logging, while the SELECT portion is logged according to the value of `binlog_format`.

3.1.13.5 Binary Logging of Stored Routines

Contents

Binary logging can be row-based, statement-based, or a mix of the two. See Binary Log Formats for more details on the formats. If logging is statement-based, it is possible that a statement will have different effects on the master and on the slave.

Stored routines are particularly prone to this, for two main reasons:

- stored routines can be non-deterministic, in other words non-repeatable, and therefore have different results each time they are run.
- the slave thread executing the stored routine on the slave holds full privileges, while this may not be the case when the routine was run on the master.

The problems with replication will only occur with statement-based logging. If row-based logging is used, since changes are made to rows based on the master's rows, there is no possibility of the slave and master getting out of sync.

By default, with row-based replication, triggers run on the master, and the effects of their executions are replicated to the slaves. However, starting from MariaDB 10.1.1, it is possible to run triggers on the slaves. See Running triggers on the slave for Row-based events.

How MariaDB Handles Statement-Based Binary Logging of Routines

If the following criteria are met, then there are some limitations on whether stored routines can be created:

- The binary log is enabled, and the `binlog_format` system variable is set to `STATEMENT`. See Binary Log Formats for more information.
- The `log_bin_trust_function_creators` is set to `OFF`, which is the default value.

If the above criteria are met, then the following limitations apply:

- When a stored function is created, it must be declared as either `DETERMINISTIC`, `NO SQL` or `READS SQL DATA`, or else an error will occur. MariaDB cannot check whether a function is deterministic, and relies on the correct definition being used.
- To create or modify a stored function, a user requires the `SUPER` privilege as well as the regular privileges. See Stored Routine Privileges for these details.
- Triggers work in the same way, except that they are always assumed to be deterministic for logging purposes, even if this is obviously not the case, such as when they use the UUID function.
- Triggers can also update data. The slave uses the DEFINER attribute to determine which user is taken to have created the trigger.
- Note that the above limitations do no apply to stored procedures or to events.

Examples

A deterministic function:

```
DELIMITER //

CREATE FUNCTION trust_me(x INT)
RETURNS INT
DETERMINISTIC
READS SQL DATA
BEGIN
   RETURN (x);
END //

DELIMITER ;
```

A non-deterministic function, since it uses the UUID_SHORT function:

```
DELIMITER //

CREATE FUNCTION dont_trust_me()
RETURNS INT
BEGIN
   RETURN UUID_SHORT();
END //

DELIMITER ;
```

1.1.1.2.8.4 SHOW BINARY LOGS

1.1.1.2.11 PURGE BINARY LOGS

1.1.1.2.8.5 SHOW BINLOG EVENTS

1.1.1.2.5.8 SHOW MASTER STATUS

3.1.19 Binlog Event Checksums

3.1.13.11 Binlog Event Checksum Interoperability

> The terms *master* and *slave* have historically been used in replication, but the terms terms *primary* and *replica* are now preferred. The old terms are used still used in parts of the documentation, and in MariaDB commands, although MariaDB 10.5 has begun the process of renaming. The documentation process is ongoing. See MDEV-18777 to follow progress on this effort.

The introduction of checksums on binlog events changes the format that events are stored in binary log files and sent over the network to replicas. This raises the question on what happens when replicating between different versions of the server, where one server is a newer version that has the binlog checksum feature implemented, while the other server is an older version that does not know about binlog checksums.

When checksums are disabled on the primary (or the primary has the old version with no checksums implemented), there is no problem. In this case the binlog format is backwards compatible, and replication works fine.

When the primary is a newer version with checksums enabled in the binlog, but the replica is an old version that does not understand checksums, replication will fail. The primary will disconnect the replica with an error, and also log a warning in its own error log. This prevents sending events to the replica that it will be unable to interpret correctly, but means that binlog checksums can not be used with older replicas. (With the recommended upgrade path, where replicas are upgraded before primaries, this is not a problem of course).

Replicating from a new MySQL primary with checksums enabled to a new MariaDB which also understands checksums works, and the MariaDB replica will verify checksums on replicated events.

There is however a problem when a newer MySQL replica replicates against a newer MariaDB primary with checksums enabled. The replica server looks at the primary server version to know whether events include checksums or not, and MySQL has not yet been updated to learn that MariaDB does this already from version 5.3.0 (as of the time of writing,

MySQL 5.6.2). Thus, if MariaDB at least version 5.3.0 but less that 5.6.1 is used as a primary with binlog checksums enabled, a MySQL replica will interpret the received events incorrectly as it does not realise the last part of the events is the checksum. So replication will fail with an error about corrupt events or even silent corruption of replicated data in unlucky cases. This requires changes to the MySQL server to fix.

Here is a summary table of the status of replication between different combination of primary and replica servers and checksum enabled/disabled:

- **OLD:** MySQL <5.6.1 or MariaDB < 5.3.0 with no checksum capabilities
- **NEW-MARIA:** MariaDB >= 5.3.0 with checksum capabilities
- **NEW-MYSQL:** MySQL >= 5.6.1 with checksum capabilities

Primary mariadb-lbinlog	Replica / enabled?	Checksums	Status
OLD	OLD	-	Ok
OLD	NEW-MARIA	-	Ok
OLD	MYSQL	-	Ok
NEW-MARIA	OLD	No	Ok
NEW-MARIA	OLD	Yes	Primary will refuse with error
NEW-MARIA	NEW-MARIA	Yes/No	Ok
NEW-MARIA	NEW-MYSQL	No	Ok
NEW-MARIA	NEW-MYSQL	Yes	Fail. Requires changes in MySQL, otherwise it will not realise MariaDB < 5.6.1 does checksums and will be confused.
NEW-MYSQL	OLD	No	Ok
NEW-MYSQL	OLD	Yes	Primary will refuse with error
NEW-MYSQL	NEW-MARIA	Yes/No	Ok
NEW-MYSQL	NEW-MYSQL	Yes/No	Ok

Checksums and mariadb-binlog

When using the mariadb-binlog client program, there are similar issues.

A version of `mariadb-binlog` which understands checksums can read binlog files from either old or new servers, with or without checksums enabled.

An old version of `mariadb-binlog` can read binlog files produced by a new server version **if** checksums were disabled when the log was produced. Old versions of `mariadb-binlog` reading a new binlog file containing checksums will be confused, and output will be garbled, with the added checksums being interpreted as extra garbage at the end of query strings and similar entries. No error will be reported in this case, just wrong output.

A version of `mysqlbinlog` (the MySQL equivalent to mariadb-binlog and the old MariaDB name for the binary) from MySQL >= 5.6.1 will have similar problems as a replica until this is fixed in MySQL. When reading a binlog file with checksums produced by MariaDB >= 5.3.0 but < 5.6.1, it will not realise that checksums are included, and will produce garbled output just like an old version of `mysqlbinlog` . The MariaDB version of `mariadb-binlog` can read binlog files produced by either MySQL or MariaDB just fine.

3.1.13.12 Group Commit for the Binary Log

Contents

Overview

The server supports group commit. This is an important optimization that helps MariaDB reduce the number of expensive

disk operations that are performed.

Durability

In ACID terminology, the "D" stands for durability. In order to ensure durability with group commit, innodb_flush_log_at_trx_commit=1 and/or sync_binlog=1 should be set. These settings are needed to ensure that if the server crashes, then any transaction that was committed prior to the time of crash will still be present in the database after crash recovery.

Durable InnoDB Data and Binary Logs

Setting both innodb_flush_log_at_trx_commit=1 and sync_binlog=1 provides the most durability and the best guarantee of replication consistency after a crash.

Non-Durable InnoDB Data

If sync_binlog=1 is set, but innodb_flush_log_at_trx_commit is not set to `1` or `3`, then it is possible after a crash to end up in a state where a transaction present in a server's binary log is missing from the server's InnoDB redo log. If the server is a replication master, then that means that the server can become inconsistent with its slaves, since the slaves may have replicated transactions from the master's binary log that are no longer present in the master's local InnoDB data.

Non-Durable Binary Logs

If innodb_flush_log_at_trx_commit is set to `1` or `3`, but sync_binlog=1 is not set, then it is possible after a crash to end up in a state where a transaction present in a server's InnoDB redo log is missing from the server's binary log. If the server is a replication master, then that also means that the server can become inconsistent with its slaves, since the server's slaves would not be able to replicate the missing transactions from the server's binary log.

Non-Durable InnoDB Data and Binary Logs

Setting innodb_flush_log_at_trx_commit=1 when sync_binlog=1 is not set can also cause the transaction to be missing from the server's InnoDB redo log due to some optimizations added in those versions. In that case, it is recommended to always set sync_binlog=1. If you can't do that, then it is recommended to set innodb_flush_log_at_trx_commit to `3`, rather than `1`. See Non-durable Binary Log Settings for more information.

Amortizing Disk Flush Costs

After every transaction COMMIT, the server normally has to flush any changes the transaction made to the InnoDB redo log and the binary log to disk (i.e. by calling system calls such as `fsync()` or `fdatasync()` or similar). This helps ensure that the data changes made by the transaction are stored durably on the disk. Disk flushing is a time-consuming operation, and can easily impose limits on throughput in terms of the number of transactions-per-second (TPS) which can be committed.

The idea with group commit is to amortize the costs of each flush to disk over multiple commits from multiple parallel transactions. For example, if there are 10 transactions trying to commit in parallel, then we can force all of them to be flushed disk at once with a single system call, rather than do one system call for each commit. This can greatly reduce the need for flush operations, and can consequently greatly improve the throughput of transactions-per-second (TPS).

However, to see the positive effects of group commit, the workload must have sufficient parallelism. A good rule of thumb is that at least three parallel transactions are needed for group commit to be effective. For example, while the first transaction is waiting for its flush operation to complete, the other two transactions will queue up waiting for their turn to flush their changes to disk. When the first transaction is done, a single system call can be used to flush the two queued-up transactions, saving in this case one of the three system calls.

In addition to sufficient parallelism, it is also necessary to have enough transactions per second wanting to commit that the flush operations are a bottleneck. If no such bottleneck exists (i.e. transactions never or rarely need to wait for the flush of another to complete), then group commit will provide little to no improvement.

Changing Group Commit Frequency

The frequency of group commits can be changed by configuring the binlog_commit_wait_usec and binlog_commit_wait_count system variables.

Measuring Group Commit Ratio

Two status variables are available for checking how effective group commit is at reducing flush overhead. These are the Binlog_commits and Binlog_group_commits status variables. We can obtain those values with the following query:

```
SHOW GLOBAL STATUS WHERE Variable_name IN('Binlog_commits', 'Binlog_group_commits');
```

Binlog_commits is the total number of transactions committed to the binary log.

Binlog_group_commits is the total number of groups committed to the binary log. As explained in the previous sections of this page, a group commit is when a group of transactions is flushed to the binary log together by sharing a single flush system call. When sync_binlog=1 is set, then this is also the total number of flush system calls executed in the process of flushing commits to the binary log.

Thus the extent to which group commit is effective at reducing the number of flush system calls on the binary log can be determined by the ratio between these two status variables. Binlog_commits will always be as equal to or greater than Binlog_group_commits. The greater the difference is between these status variables, the more effective group commit was at reducing flush overhead.

To calculate the group commit ratio, we actually need the values of these status variables from two snapshots. Then we can calculate the ratio with the following formula:

`transactions/group commit` = (`Binlog_commits` (*snapshot2*) - `Binlog_commits` (*snapshot1*))/(`Binlog_group_commits` (*snapshot2*) - `Binlog_group_commits` (*snapshot1*))

For example, if we had the following first snapshot:

```
SHOW GLOBAL STATUS WHERE Variable_name IN('Binlog_commits', 'Binlog_group_commits');
+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| Binlog_commits        | 120   |
| Binlog_group_commits  | 120   |
+-----------------------+-------+
2 rows in set (0.00 sec)
```

And the following second snapshot:

```
SHOW GLOBAL STATUS WHERE Variable_name IN('Binlog_commits', 'Binlog_group_commits');
+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| Binlog_commits        | 220   |
| Binlog_group_commits  | 145   |
+-----------------------+-------+
2 rows in set (0.00 sec)
```

Then we would have:

`transactions/group commit` = (220 - 120) / (145 - 120) = 100 / 25 = 4 `transactions/group commit`

If your group commit ratio is too close to 1, then it may help to change your group commit frequency.

Use of Group Commit with Parallel Replication

Group commit is also used to enable conservative mode of in-order parallel replication .

Effects of Group Commit on InnoDB Performance

When both innodb_flush_log_at_trx_commit=1 (the default) is set and the binary log is enabled, there is now one less sync to disk inside InnoDB during commit (2 syncs shared between a group of transactions instead of 3). See Binary Log Group Commit and InnoDB Flushing Performance for more information.

Status Variables

Binlog_commits is the total number of transactions committed to the binary log.

Binlog_group_commits is the total number of groups committed to the binary log.

Binlog_group_commit_trigger_count is the total number of group commits triggered because of the number of binary log commits in the group reached the limit set by the system variable binlog_commit_wait_count.

Binlog_group_commit_trigger_lock_wait is the total number of group commits triggered because a binary log commit was being delayed because of a lock wait where the lock was held by a prior binary log commit. When this happens the later binary log commit is placed in the next group commit.

Binlog_group_commit_trigger_timeout is the total number of group commits triggered because of the time since the first binary log commit reached the limit set by the system variable binlog_commit_wait_usec.

To query these variables, use a statement such as:

```
SHOW GLOBAL STATUS LIKE 'Binlog_%commit%';
```

1.3.16 mariadb-binlog

2.4.11 Transaction Coordinator Log

3.1.13.15 Compressing Events to Reduce Size of the Binary Log

Selected events in the binary log can be optionally compressed, to save space in the binary log on disk and in network transfers.

The events that can be compressed are the events that normally can be of a significant size: Query events (for DDL and DML in statement-based replication), and row events (for DML in row-based replication).

Compression is fully transparent. Events are compressed on the primary before being written into the binary log, and are uncompressed by the I/O thread on the replica before being written into the relay log. The mariadb-binlog command will likewise uncompress events for its output.

Currently, the zlib compression algorithm is used to compress events.

Compression will have the most impact when events are of a non-negligible size, as each event is compressed individually. For example, batch INSERT statements that insert many rows or large values, or row-based events that touch a number of rows in one query.

The log_bin_compress option is used to enable compression of events. Only events with data (query text or row data) above a certain size are compressed; the limit is set with the log_bin_compress_min_len option.

2.2.1.1.2.4 Encrypting Binary Logs

3.1.13.17 Flashback

Contents

Flashback is a feature that allows instances, databases or tables to be rolled back to an old snapshot.

Flashback is currently supported only over DML statements (INSERT, DELETE, UPDATE). An upcoming version of MariaDB will add support for flashback over DDL statements (DROP, TRUNCATE, ALTER, etc.) by copying or moving the current table to a reserved and hidden database, and then copying or moving back when using flashback. See MDEV-10571.

Flashback is achieved in MariaDB Server using existing support for full image format binary logs (binlog_row_image=FULL), so it supports all engines.

The real work of Flashback is done by mariadb-binlog with `--flashback`. This causes events to be translated: INSERT to DELETE, DELETE to INSERT, and for UPDATEs, the before and after images are swapped.

When executing `mariadb-binlog` with `--flashback`, the Flashback events will be stored in memory. You should make sure your server has enough memory for this feature.

Arguments

- mariadb-binlog has the option `--flashback` or `-B` that will let it work in flashback mode.
- mariadbd / mysqld has the option --flashback that enables the binary log and sets `binlog_format=ROW`. It is not mandatory to use this option if you have already enabled those options directly.

Do not use `-v -vv` options, as this adds verbose information to the binary log which can cause problems when importing. See MDEV-12066 and MDEV-12067.

Example

With a table "mytable" in database "test", you can compare the output with `--flashback` and without.

```
mariadb-binlog /var/lib/mysql/mysql-bin.000001 -vv -d test -T mytable \
   --start-datetime="2013-03-27 14:54:00" > review.sql
```

```
mariadb-binlog /var/lib/mysql/mysql-bin.000001 -vv -d test -T mytable \
   --start-datetime="2013-03-27 14:54:00" --flashback > flashback.sql
```

If you know the exact position, `--start-position` can be used instead of `--start-datetime`.

Then, by importing the output file (`mysql < flashback.sql`), you can flash your database/table back to the specified time or position.

Common Use Case

A common use case for Flashback is the following scenario:

- You have one primary and two replicas, one started with `--flashback` (i.e. with binary logging enabled, using binlog_format=ROW, and binlog_row_image=FULL).
- Something goes wrong on the primary (like a wrong update or delete) and you would like to revert to a state of the database (or just a table) at a certain point in time.
- Remove the flashback-enabled replica from replication.
- Invoke mariadb-binlog to find the exact log position of the first offending operation after the state you want to revert to.
- Run `mariadb-binlog --flashback --start-position=xyz | mysql` to pipe the output of `mariadb-binlog` directly to the `mariadb` client, or save the output to a file and then direct the file to the command-line client.

3.1.13.18 Relay Log

Contents

The relay log is a set of log files created by a replica during replication.

It's the same format as the binary log, containing a record of events that affect the data or structure; thus, mariadb-binlog can be used to display its contents. It consists of a set of relay log files and an index file containing a list of all relay log files.

Events are read from the primary's binary log and written to the replica's relay log. They are then performed on the replica. Old relay log files are automatically removed once they are no longer needed.

Creating Relay Log Files

New relay log files are created by the replica at the following times:

- when the IO thread starts
- when the logs are flushed, with FLUSH LOGS or mariadb-admin flush-logs.
- when the maximum size, determined by the max_relay_log_size system variable, has been reached

Relay Log Names

By default, the relay log will be given a name `host_name-relay-bin.nnnnnn`, with `host_name` referring to the server's host name, and #nnnnnn `the sequence number.`

This will cause problems if the replica's host name changes, returning the error `Failed to open the relay log` and `Could not find target log during relay log initialization`. To prevent this, you can specify the relay log file name by setting the relay_log and relay_log_index system variables.

If you need to overcome this issue while replication is already underway,you can stop the replica, prepend the old relay log index file to the new relay log index file, and restart the replica.

For example:

```
shell> cat NEW_relay_log_name.index >> OLD_relay_log_name.index
shell> mv OLD_relay_log_name.index NEW_relay_log_name.index
```

Viewing Relay Logs

The SHOW RELAYLOG EVENTS shows events in the relay log, and, since relay log files are the same format as binary log files, they can be read with the mariadb-binlog utility.

Removing Old Relay Logs

Old relay logs are automatically removed once all events have been implemented on the replica, and the relay log file is no longer needed. This behavior can be changed by adjusting the relay_log_purge system variable from its default of `1` to `0`, in which case the relay logs will be left on the server.

Relay logs are also removed by the CHANGE MASTER statement unless a relay log option is used.

One can also flush the logs with the FLUSH RELAY LOGS commands.

If the relay logs are taking up too much space on the replica, the relay_log_space_limit system variable can be set to limit the size. The IO thread will stop until the SQL thread has cleared the backlog. By default there is no limit.

3.1.11 Replication and Binary Log System Variables

3.1.14 Unsafe Statements for Statement-based Replication

Contents

A safe statement is generally deterministic; in other words the statement will always produce the same result. For example, an INSERT statement producing a random number will most likely produce a different result on the primary than on the replica, and so cannot be replicated safely.

When an unsafe statement is run, the current binary logging format determines how the server responds.

- If the binary logging format is statement-based (the default until MariaDB 10.2.3), unsafe statements generate a warning and are logged normally.
- If the binary logging format is mixed (the default from MariaDB 10.2.4), unsafe statements are logged using the row-based format, while safe statements use the statement-based format.
- If the binary logging format is row-based, all statements are logged normally, and the distinction between safe and unsafe is not made.

MariaDB tries to detect unsafe statements. When an unsafe statement is issued, a warning similar to the following is produced:

```
Note (Code 1592): Unsafe statement written to the binary log using statement format since
  BINLOG_FORMAT = STATEMENT. The statement is unsafe because it uses a LIMIT clause. This
  is unsafe because the set of rows included cannot be predicted.
```

MariaDB also issues this warning for some classes of statements that are safe.

Unsafe Statements

The following statements are regarded as unsafe:

- INSERT ... ON DUPLICATE KEY UPDATE statements upon tables with multiple primary or unique keys, as the order that the keys are checked in, and which affect the rows chosen to update, is not deterministic. Before MariaDB 5.5.24, these statements were not regarded as unsafe. In MariaDB 10.0 this warning has been removed as we always check keys in the same order on the primary and replica if the primary and replica are using the same storage engine.
- INSERT-DELAYED. These statements are inserted in an indeterminate order.
- INSERTs on tables with a composite primary key that has an AUTO_INCREMENT column that isn't the first column of the composite key.
- When a table has an AUTO_INCREMENT column and a trigger or stored procedure executes an UPDATE statement against the table. Before MariaDB 5.5, all updates on tables with an AUTO_INCREMENT column were considered unsafe, as the order that the rows were updated could differ across servers.
- UPDATE statements that use LIMIT, since the order of the returned rows is unspecified. This applies even to statements using an ORDER BY clause, which are deterministic (a known bug). However, since MariaDB 10.0.11, `LIMIT 0` is an exception to this rule (see MDEV-6170), and these statements are safe for replication.
- When using a user-defined function.
- Statements using using any of the following functions, which can return different results on the replica:
  - CURRENT_ROLE()
  - CURRENT_USER()
  - FOUND_ROWS()
  - GET_LOCK()
  - IS_FREE_LOCK()
  - IS_USED_LOCK()
  - JSON_TABLE()
  - LOAD_FILE()
  - MASTER_POS_WAIT()
  - RAND()
  - RANDOM_BYTES()
  - RELEASE_ALL_LOCKS()
  - RELEASE_LOCK()
  - ROW_COUNT()
  - SESSION_USER()
  - SLEEP()
  - SYSDATE()
  - SYSTEM_USER()
  - USER()
  - UUID()
  - UUID_SHORT().
- Statements which refer to log tables, since these may differ across servers.
- Statements which refer to self-logging tables. Statements following a read or write to a self-logging table within a transaction are also considered unsafe.
- Statements which refer to system variables (there are a few exceptions).
- LOAD DATA INFILE statements (since MariaDB 5.5).
- Non-transactional reads or writes that execute after transactional reads within a transaction.
- If row-based logging is used for a statement, and the session executing the statement has any temporary tables, row-based logging is used for the remaining statements until the temporary table is dropped. This is because temporary tables can't use row-based logging, so if it is used due to one of the above conditions, all subsequent statements using that table are unsafe. The server deals with this situation by treating all statements in the session as unsafe for statement-based logging until the temporary table is dropped.

Safe Statements

The following statements are not deterministic, but are considered safe for binary logging and replication:

- CONNECTION_ID()
- CURDATE()
- CURRENT_DATE()
- CURRENT_TIME()
- CURRENT_TIMESTAMP()
- CURTIME()
- LAST_INSERT_ID()
- LOCALTIME()
- LOCALTIMESTAMP()
- NOW()
- UNIX_TIMESTAMP()

- UTC_DATE()
- UTC_TIME()
- UTC_TIMESTAMP()

Isolation Levels

Even when using safe statements, not all transaction isolation levels are safe with statement-based or mixed binary logging. The REPEATABLE READ and SERIALIZABLE isolation levels can only be used with the row-based format.

This restriction does not apply if only non-transactional storage engines are used.

3.1.15 Replication and Foreign Keys

> The terms *master* and *slave* have historically been used in replication, but the terms terms *primary* and *replica* are now preferred. The old terms are used still used in parts of the documentation, and in MariaDB commands, although MariaDB 10.5 has begun the process of renaming. The documentation process is ongoing. See MDEV-18777 to follow progress on this effort.

Replication is based upon the binary log. However, cascading deletes or updates based on foreign key relations are an internal mechanism, and are not written to the binary log.

Because of this, an identical statement run on the master and the slave may result in different outcomes if the foreign key relations are not identical on both master and slave This could be the case if the storage engine on one supports cascading deletes (e.g. InnoDB) and the storage engine on the other does not (e.g. MyISAM), or the one has specified a foreign key relation, and the other hasn't.

Take the following example:

```
CREATE TABLE employees (
    x INT PRIMARY KEY,
    name VARCHAR(10)
) ENGINE = InnoDB;

CREATE TABLE children (
    y INT PRIMARY KEY,
    f INT,
    name VARCHAR(10),
    FOREIGN KEY fk (f) REFERENCES employees (x)
        ON DELETE CASCADE
) ENGINE = InnoDB;
```

The slave, however, has been set up without InnoDB support, and defaults to MyISAM, so the foreign key restrictions are not in place.

```
INSERT INTO employees VALUES (1, 'Yaser'), (2, 'Prune');

INSERT INTO children VALUES (1, 1, 'Haruna'), (2, 1, 'Hera'), (3, 2, 'Eva');
```

At this point, the slave and the master are in sync:

```
SELECT * FROM employees;
+---+--------+
| x | name   |
+---+--------+
| 1 | Yaser |
| 2 | Prune |
+---+--------+
2 rows in set (0.00 sec)

SELECT * FROM children;
+---+------+---------+
| y | f    | name    |
+---+------+---------+
| 1 |    1 | Haruna |
| 2 |    1 | Hera    |
| 3 |    2 | Eva     |
+---+------+---------+
```

However, after:

```
DELETE FROM employees WHERE x=1;
```

there are different outcomes on the slave and the master.

On the master, the cascading deletes have taken effect:

```
SELECT * FROM children;
+---+------+------+
| y | f    | name |
+---+------+------+
| 3 |    2 | Eva  |
+---+------+------+
```

On the slave, the cascading deletes did not take effect:

```
SELECT * FROM children;
+---+------+--------+
| y | f    | name   |
+---+------+--------+
| 1 |    1 | Haruna |
| 2 |    1 | Hera   |
| 3 |    2 | Eva    |
+---+------+--------+
```

3.1.13.18 Relay Log

3.1.13.12 Group Commit for the Binary Log

3.1.18 Selectively Skipping Replication of Binlog Events

The terms *master* and *slave* have historically been used in replication, but the terms terms *primary* and *replica* are now preferred. The old terms are used still used in parts of the documentation, and in MariaDB commands, although MariaDB 10.5 has begun the process of renaming. The documentation process is ongoing. See MDEV-18777 to follow progress on this effort.

Contents

Normally, all changes that are logged as events in the binary log are also replicated to all replicas (though still subject to filtering by replicate-do-db, replicate-ignore-db, and similar options). However, sometimes it may be desirable to have certain events be logged into the binlog, but not be replicated to all or a subset of replicas, where the distinction between events that should be replicated or not is under the control of the application making the changes.

This could be useful if an application does some replication external to the server outside of the built-in replication, or if it has some data that should not be replicated for whatever reason.

This is possible with the following system variables.

Primary Session Variable: skip_replication

When the skip_replication variable is set to true, changes are logged into the binary log with the flag `@@skip_replication` set. Such events will not be replicated by replicas that run with `--replicate-events-marked-for-skip` set different from its default of `REPLICATE`.

Variable Name	skip_replication

Scope	Session only
Access Type	Dynamic
Data Type	`bool`
Default Value	`OFF`

The `skip_replication` option only has effect if binary logging is enabled and sql_log_bin is true.

Attempting to change `@@skip_replication` in the middle of a transaction will fail; this is to avoid getting half of a transaction replicated while the other half is not replicated. Be sure to end any current transaction with `COMMIT` / `ROLLBACK` before changing the variable.

Replica Option: --replicate-events-marked-for-skip

The replicate_events_marked_for_skip option tells the replica whether to replicate events that are marked with the `@@skip_replication` flag. Default is `REPLICATE`, to ensure that all changes are replicated to the replica. If set to `FILTER_ON_SLAVE`, events so marked will be skipped on the replica and not replicated. If set to `FILTER_ON_MASTER`, the filtering will be done on the primary, saving on network bandwidth as the events will not be received by the replica at all.

Variable Name	`replicate_events_marked_for_skip`
Scope	Global
Access Type	Dynamic
Data Type	enum: `REPLICATE` | `FILTER_ON_SLAVE` | `FILTER_ON_MASTER`
Default Value	`REPLICATE`

Note: `replicate_events_marked_for_skip` is a dynamic variable (it can be changed without restarting the server), however the replica threads must be stopped when it is changed, otherwise an error will be thrown.

When events are filtered due to `@@skip_replication`, the filtering happens on the primary side; in other words, the event is never sent to the replica. If many events are filtered like this, a replica can sit a long time without receiving any events from the primary. This is not a problem in itself, but must be kept in mind when inquiring on the replica about events that are filtered. For example `START SLAVE UNTIL <some position>` will stop when the first event that is **not** filtered is encountered at the given position or beyond. If the event at the given position is filtered, then the replica thread will only stop when the next non-filtered event is encountered. In effect, if an event is filtered, to the replica it appears that it was never written to the binlog on the primary.

Note that when events are filtered for a replica, the data in the database will be different on the replica and on the primary. It is the responsibility of the application to replicate the data outside of the built-in replication or otherwise ensure consistency of operation. If this is not done, it is possible for replication to encounter, for example, `UNIQUE` contraint violations or other problems which will cause replication to stop and require manual intervention to fix.

The session variable `@@skip_replication` can be changed without requiring special privileges. This makes it possible for normal applications to control it without requiring `SUPER` privileges. But it must be kept in mind when using replicas with `--replicate-events-marked-for-skip` set different from `REPLICATE`, as it allows any connection to do changes that are not replicated.

skip_replication and sql_log_bin

`@@sql_log_bin` and `@@skip_replication` are somewhat related, as they can both be used to prevent a change on the primary from being replicated to the replica. The difference is that with `@@skip_replication`, changes are still written into the binlog, and replication of the events is only skipped on replicas that explicitly are configured to do so, with `--replicate-events-marked-for-skip` different from `REPLICATE`. With `@@sql_log_bin`, events are not logged into the binlog, and so are not replicated by any replica.

skip_replication and the Binlog

When events in the binlog are marked with the `@@skip_replication` flag, the flag will be preserved if the events are dumped by the mariadb-binlog program and re-applied against a server with the mariadb client program. Similarly, the BINLOG statement will preserve the flag from the event being replayed. And a replica which runs with `--log-slave-updates` and does not filter events (`--replicate-events-marked-for-skip=REPLICATE`) will also preserve the flag in

the events logged into the binlog on the replica.

3.1.19 Binlog Event Checksums

> The terms *master* and *slave* have historically been used in replication, but the terms terms *primary* and *replica* are now preferred. The old terms are used still used in parts of the documentation, and in MariaDB commands, although MariaDB 10.5 has begun the process of renaming. The documentation process is ongoing. See MDEV-18777 to follow progress on this effort.

MariaDB includes a feature to include a checksum in binary log events.

Checksums are enabled with the binlog_checksum option. Until MariaDB 10.2.1, this was disabled by default. From MariaDB 10.2.1, the option is set to `CRC32`.

The variable can be changed dynamically without restarting the server. Setting the variable in any way (even to the existing value) forces a rotation of the binary log (the intention is to avoid having a single binlog where some events are checksummed and others are not).

When checksums are enabled, replication slaves will check events received over the network for checksum errors, and will stop with an error if a corrupt event is detected.

In addition, the server can be configured to verify checksums in two other places.

One is when reading events from the binlog on the master, for example when sending events to a slave or for something like SHOW BINLOG EVENTS. This is controlled by option master_verify_checksum, and is thus used to detect file system corruption of the binlog files.

The other is when the slave SQL thread reads events from the relay log. This is controlled by the slave_sql_verify_checksum option, and is used to detect file system corruption of slave relay log files.

`master_verify_checksum`

- **Description:** Verify binlog checksums when reading events from the binlog on the master.
- **Commandline:** `--master_verify_checksum=[0|1]`
- **Scope:** Global
- **Access Type:** Can be changed dynamically
- **Data Type:** `bool`
- **Default Value:** `OFF (0)`

`slave_sql_verify_checksum`

- **Description:** Verify binlog checksums when the slave SQL thread reads events from the relay log.
- **Commandline:** `--slave_sql_verify_checksum=[0|1]`
- **Scope:** Global
- **Access Type:** Can be changed dynamically
- **Data Type:** `bool`
- **Default Value:** `ON (1)`

The mysqlbinlog client program by default does not verify checksums when reading a binlog file, however it can be instructed to do so with the option verify-binlog-checksum:

- **Variable Name:** `verify-binlog-checksum`
- **Data Type:** `bool`
- **Default Value:** `OFF`

1.3.16.3 Annotate_rows_log_event

3.1.21 Row-based Replication With No Primary Key

> The terms *master* and *slave* have historically been used in replication, but the terms terms *primary* and *replica* are now preferred. The old terms are used still used in parts of the documentation, and in MariaDB commands, although MariaDB 10.5 has begun the process of renaming. The documentation process is ongoing. See MDEV-18777 to follow progress on this effort.

MariaDB improves on row-based replication (see binary log formats) of tables which have no primary key but do have some other index. This is based in part on the original Percona patch "row_based_replication_without_primary_key.patch", with some additional fixes and enhancements.

When row-based replication is used with UPDATE or DELETE, the slave needs to locate each replicated row based on the value in columns. If the table contains at least one index, an index lookup will be used (otherwise a table scan is needed for each row, which is extremely inefficient for all but the smallest table and generally to be avoided).

In MariaDB, the slave will try to choose a good index among any available:

- The primary key is used, if there is one.
- Else, the first unique index without NULL-able columns is used, if there is one.
- Else, a choice is made among any normal indexes on the table (e.g. a FULLTEXT index is not considered).

The choice of which of several non-unique indexes to use is based on the cardinality of indexes; the one that is most selective (has the smallest average number of rows per distinct tuple of column values) is preferred. Note that for this choice to be effective, for most storage engines (like MyISAM, InnoDB) it is necessary to make sure ANALYZE TABLE has been run on the slave, otherwise statistics about index cardinality will not be available. In the absence of index cardinality, the first unique index will be chosen, if any, else the first non-unique index.

Prior to MariaDB 5.3, the slave would always choose the first index without considering cardinality. The slave could even choose an unusable index (like FULLTEXT) if no other index was available (MySQL Bug #58997), causing row-based replication to break in this case; this was also fixed in MariaDB 5.3.

3.1.22 Replication Filters

The terms *master* and *slave* have historically been used in replication, but the terms terms *primary* and *replica* are now preferred. The old terms are used still used in parts of the documentation, and in MariaDB commands, although MariaDB 10.5 has begun the process of renaming. The documentation process is ongoing. See MDEV-18777 to follow progress on this effort.

Contents

Replication filters allow users to configure replicas to intentionally skip certain events.

Binary Log Filters for Replication Primaries

MariaDB provides options that can be used on a replication primary to restrict local changes to specific databases from getting written to the binary log, which also determines whether any replicas replicate those changes.

Binary Log Filter Options

The following options are available, and they are evaluated in the order that they are listed below. If there are conflicting settings, *binlog_do_db* prevails. Before MariaDB 11.2.0, they are only available as options; from MariaDB 11.2.0 they are also available as system variables.

binlog_do_db

The binlog_do_db option allows you to configure a replication primary to write statements and transactions affecting databases that match a specified name into its binary log. Since the filtered statements or transactions will not be present in the binary log, its replicas will not be able to replicate them.

This option will **not** work with cross-database updates with statement-based logging. See the Statement-Based Logging section for more information.

This option can **not** be set dynamically.

When setting it on the command-line or in a server option group in an option file, the option does not accept a comma-separated list. If you would like to specify multiple filters, then you need to specify the option multiple times. For example:

```
[mariadb]
...
binlog_do_db=db1
binlog_do_db=db2
```

This will tell the primary to do the following:

- Write statements and transactions affecting the database named *db1* into the binary log.
- Write statements and transactions affecting the database named *db2* into the binary log.
- Don't write statements and transactions affecting any other databases into the binary log.

binlog_ignore_db

The binlog_ignore_db option allows you to configure a replication primary to **not** write statements and transactions affecting databases that match a specified name into its binary log. Since the filtered statements or transactions will not be present in the binary log, its replicas will not be able to replicate them.

This option will **not** work with cross-database updates with statement-based logging. See the Statement-Based Logging section for more information.

This option can **not** be set dynamically.

When setting it on the command-line or in a server option group in an option file, the option does not accept a comma-separated list. If you would like to specify multiple filters, then you need to specify the option multiple times. For example:

```
[mariadb]
...
binlog_ignore_db=db1
binlog_ignore_db=db2
```

This will tell the primary to do the following:

- Don't write statements and transactions affecting the database named *db1* into the binary log.
- Don't write statements and transactions affecting the database named *db2* into the binary log.
- Write statements and transactions affecting any other databases into the binary log.

> The binlog_ignore_db option is effectively ignored if the binlog_do_db option is set, so those two options should not be set together.

Replication Filters for Replicas

MariaDB provides options and system variables that can be used on used on a replicas to filter events replicated in the binary log.

Replication Filter Options

The following options and system variables are available, and they are evaluated in the order that they are listed below. If there are conflicting settings, the respective *replicate_do_* prevails.

replicate_rewrite_db

The replicate_rewrite_db option (and, from MariaDB 10.11, system variable), allows you to configure a replica to rewrite database names. It uses the format `primary_database->replica_database`. If a replica encounters a binary log event in which the default database (i.e. the one selected by the USE statement) is `primary_database`, then the replica will apply the event in `replica_database` instead.

This option will **not** work with cross-database updates with statement-based logging. See the Statement-Based Logging section for more information.

This option only affects statements that involve tables. This option does not affect statements involving the database itself, such as CREATE DATABASE, ALTER DATABASE, and DROP DATABASE.

This option's rewrites are evaluated *before* any other replication filters configured by the `replicate_*` system variables.

Statements that use table names qualified with database names do not work with other replication filters such as replicate_do_table.

Until MariaDB 10.11, this option could not be set dynamically.

When setting it on the command-line or in a server option group in an option file, the option does not accept a comma-separated list. If you would like to specify multiple filters, then you need to specify the option multiple times. For example:

```
[mariadb]
...
replicate_rewrite_db=db1->db3
replicate_rewrite_db=db2->db4
```

This will tell the replica to do the following:

- If a binary log event is encountered in which the default database was *db1*, then apply the event in *db3* instead.
- If a binary log event is encountered in which the default database was *db2*, then apply the event in *db4* instead.

See Configuring Replication Filter Options with Multi-Source Replication for how to configure this system variable with multi-source replication.

`replicate_do_db`

The replicate_do_db system variable allows you to configure a replica to apply statements and transactions affecting databases that match a specified name.

This system variable will **not** work with cross-database updates with statement-based logging or when using mixed-based logging and the statement is logged statement based. For statement-based replication, only the default database (that is, the one selected by USE) is considered, not any explicitly mentioned tables in the query. See the Statement-Based Logging section for more information.

When setting it dynamically with SET GLOBAL, the system variable accepts a comma-separated list of filters.

When setting it dynamically, it is not possible to specify database names that contain commas. If you need to specify database names that contain commas, then you will need to specify them by either providing the command-line options or configuring them in a server option group in an option file when the server is started.

When setting it dynamically, the replica threads must be stopped. For example:

```
STOP SLAVE;
SET GLOBAL replicate_do_db='db1,db2';
START SLAVE;
```

When setting it on the command-line or in a server option group in an option file, the system variable does not accept a comma-separated list. If you would like to specify multiple filters, then you need to specify the system variable multiple times. For example:

```
[mariadb]
...
replicate_do_db=db1
replicate_do_db=db2
```

This will tell the replica to do the following:

- Replicate statements and transactions affecting the database named *db1*.
- Replicate statements and transactions affecting the database named *db2*.
- Ignore statements and transactions affecting any other databases.

See Configuring Replication Filter Options with Multi-Source Replication for how to configure this system variable with multi-source replication.

replicate_ignore_db

The replicate_ignore_db system variable allows you to configure a replica to ignore statements and transactions affecting databases that match a specified name.

This system variable will **not** work with cross-database updates with statement-based logging or when using mixed-based logging and the statement is logged statement based. For statement-based replication, only the default database (that is, the one selected by USE) is considered, not any explicitly mentioned tables in the query. See the Statement-Based Logging section for more information.

When setting it dynamically with SET GLOBAL, the system variable accepts a comma-separated list of filters.

When setting it dynamically, it is not possible to specify database names that contain commas. If you need to specify names or patterns that contain commas, then you will need to specify them by either providing the command-line options or configuring them in a server option group in an option file when the server is started.

When setting it dynamically, the replica threads must be stopped. For example:

```
STOP SLAVE;
SET GLOBAL replicate_ignore_db='db1,db2';
START SLAVE;
```

When setting it on the command-line or in a server option group in an option file, the system variable does not accept a comma-separated list. If you would like to specify multiple filters, then you need to specify the system variable multiple times. For example:

```
[mariadb]
...
replicate_ignore_db=db1
replicate_ignore_db=db2
```

This will tell the replica to do the following:

- Ignore statements and transactions affecting databases named *db1*.
- Ignore statements and transactions affecting databases named *db2*.
- Replicate statements and transactions affecting any other databases.

> The replicate_ignore_db system variable is effectively ignored if the replicate_do_db system variable is set, so those two system variables should not be set together.

See Configuring Replication Filter Options with Multi-Source Replication for how to configure this system variable with multi-source replication.

replicate_do_table

The replicate_do_table system variable allows you to configure a replica to apply statements and transactions that affect tables that match a specified name. The table name is specified in the format: `dbname.tablename`.

This system variable will **not** work with cross-database updates with statement-based logging. See the Statement-Based Logging section for more information.

This option only affects statements that involve tables. This option does not affect statements involving the database itself, such as CREATE DATABASE, ALTER DATABASE, and DROP DATABASE.

When setting it dynamically with SET GLOBAL, the system variable accepts a comma-separated list of filters.

When setting it dynamically, it is not possible to specify database or table names or patterns that contain commas. If you need to specify database or table names that contain commas, then you will need to specify them by either providing the command-line options or configuring them in a server option group in an option file when the server is started.

When setting it dynamically, the replica threads must be stopped. For example:

```
STOP SLAVE;
SET GLOBAL replicate_do_table='db1.tab,db2.tab';
START SLAVE;
```

When setting it on the command-line or in a server option group in an option file, the system variable does not accept a comma-separated list. If you would like to specify multiple filters, then you need to specify the system variable multiple times. For example:

```
[mariadb]
...
replicate_do_table=db1.tab
replicate_do_table=db2.tab
```

This will tell the replica to do the following:

- Replicate statements and transactions affecting tables in databases named *db1* and which are named *tab*.
- Replicate statements and transactions affecting tables in databases named *db2* and which are named *tab*.
- Ignore statements and transactions affecting any other tables.

See Configuring Replication Filter Options with Multi-Source Replication for how to configure this system variable with multi-source replication.

replicate_ignore_table

The replicate_ignore_table system variable allows you to configure a replica to ignore statements and transactions that affect tables that match a specified name. The table name is specified in the format: `dbname.tablename`.

This system variable will **not** work with cross-database updates with statement-based logging. See the Statement-Based Logging section for more information.

When setting it dynamically with SET GLOBAL, the system variable accepts a comma-separated list of filters.

When setting it dynamically, it is not possible to specify database or table names that contain commas. If you need to specify database or table names that contain commas, then you will need to specify them by either providing the command-line options or configuring them in a server option group in an option file when the server is started.

When setting it dynamically, the replica threads must be stopped. For example:

```
STOP SLAVE;
SET GLOBAL replicate_ignore_table='db1.tab,db2.tab';
START SLAVE;
```

When setting it on the command-line or in a server option group in an option file, the system variable does not accept a comma-separated list. If you would like to specify multiple filters, then you need to specify the system variable multiple times. For example:

```
[mariadb]
...
replicate_ignore_table=db1.tab
replicate_ignore_table=db2.tab
```

This will tell the replica to do the following:

- Ignore statements and transactions affecting tables in databases named *db1* and which are named *tab*.
- Ignore statements and transactions affecting tables in databases named *db2* and which are named *tab*.
- Replicate statements and transactions affecting any other tables.

The replicate_ignore_table system variable is effectively ignored if either the replicate_do_table system variable or the replicate_wild_do_table system variable is set, so the replicate_ignore_table system variable should not be used with those two system variables.

See Configuring Replication Filter Options with Multi-Source Replication for how to configure this system variable with multi-source replication.

replicate_wild_do_table

The replicate_wild_do_table system variable allows you to configure a replica to apply statements and transactions that affect tables that match a specified wildcard pattern.

The wildcard pattern uses the same semantics as the LIKE operator. This means that the the following characters have a special meaning:

- `_` - The `_` character matches any single character.
- `%` - The `%` character matches zero or more characters.
- `\` - The `\` character is used to escape the other special characters in cases where you need the literal character.

This system variable will work with cross-database updates with statement-based logging. See the Statement-Based Logging section for more information.

The system variable does filter databases, tables, views and triggers.

The system variable does not filter stored procedures, stored functions, and events. The replicate_do_db system variable will need to be used to filter those.

If the table name pattern for a filter is just specified as `%`, then all tables in the database will be matched. In this case, the filter will also affect certain database-level statements, such as CREATE DATABASE, ALTER DATABASE and DROP DATABASE.

When setting it dynamically with SET GLOBAL, the system variable accepts a comma-separated list of filters.

When setting it dynamically, it is not possible to specify database or table names or patterns that contain commas. If you need to specify database or table names or patterns that contain commas, then you will need to specify them by either providing the command-line options or configuring them in a server option group in an option file when the server is started.

When setting it dynamically, the replica threads must be stopped. For example:

```
STOP SLAVE;
SET GLOBAL replicate_wild_do_table='db%.tab%,app1.%';
START SLAVE;
```

When setting it on the command-line or in a server option group in an option file, the system variable does not accept a comma-separated list. If you would like to specify multiple filters, then you need to specify the system variable multiple times. For example:

```
[mariadb]
...
replicate_wild_do_table=db%.tab%
replicate_wild_do_table=app1.%
```

This will tell the replica to do the following:

- Replicate statements and transactions affecting tables in databases that start with *db* and whose table names start with *tab*.
- Replicate statements and transactions affecting the database named *app1*.
- Ignore statements and transactions affecting any other tables and databases.

See Configuring Replication Filter Options with Multi-Source Replication for how to configure this system variable with multi-source replication.

replicate_wild_ignore_table

The replicate_wild_ignore_table system variable allows you to configure a replica to ignore statements and transactions that affect tables that match a specified wildcard pattern.

The wildcard pattern uses the same semantics as the LIKE operator. This means that the the following characters have a special meaning:

- `_` - The `_` character matches any single character.
- `%` - The `%` character matches zero or more characters.
- `\` - The `\` character is used to escape the other special characters in cases where you need the literal character.

This system variable will work with cross-database updates with statement-based logging. See the Statement-Based Logging section for more information.

The system variable does filter databases, tables, views and triggers.

The system variable does not filter stored procedures, stored functions, and events. The replicate_ignore_db system variable will need to be used to filter those.

If the table name pattern for a filter is just specified as `%`, then all tables in the database will be matched. In this case, the filter will also affect certain database-level statements, such as CREATE DATABASE, ALTER DATABASE and DROP DATABASE.

When setting it dynamically with SET GLOBAL, the system variable accepts a comma-separated list of filters.

When setting it dynamically, it is not possible to specify database or table names or patterns that contain commas. If you need to specify database or table names or patterns that contain commas, then you will need to specify them by either providing the command-line options or configuring them in a server option group in an option file when the server is started.

When setting it dynamically, the replica threads must be stopped. For example:

```
STOP SLAVE;
SET GLOBAL replicate_wild_ignore_table='db%.tab%,app1.%';
START SLAVE;
```

When setting it on the command-line or in a server option group in an option file, the system variable does not accept a comma-separated list. If you would like to specify multiple filters, then you need to specify the system variable multiple times. For example:

```
[mariadb]
...
replicate_wild_ignore_table=db%.tab%
replicate_wild_ignore_table=app1.%
```

This will tell the replica to do the following:

- Ignore statements and transactions affecting tables in databases that start with *db* and whose table names start with *tab*.
- Ignore statements and transactions affecting all the tables in the database named *app1*.
- Replicate statements and transactions affecting any other tables and databases.

> The replicate_ignore_table system variable is effectively ignored if either the replicate_do_table system variable or the replicate_wild_do_table system variable is set, so the replicate_ignore_table system variable should not be used with those two system variables.

See Configuring Replication Filter Options with Multi-Source Replication for how to configure this system variable with multi-source replication.

Configuring Replication Filter Options with Multi-Source Replication

How you configure replication filters with multi-source replication depends on whether you are configuring them dynamically or whether you are configuring them in a server option group in an option file.

Setting Replication Filter Options Dynamically with Multi-Source Replication

The usage of dynamic replication filters changes somewhat when multi-source replication is in use. By default, the variables are addressed to the default connection, so in a multi-source environment, the required connection needs to be specified. There are two ways to do this.

Prefixing the Replication Filter Option with the Connection Name

One way to change a replication filter for a multi-source connection is to explicitly specify the name when changing the filter. For example:

```
STOP SLAVE 'gandalf';
SET GLOBAL gandalf.replicate_do_table='database1.table1,database1.table2,database1.table3';
START SLAVE 'gandalf';
```

Changing the Default Connection

Alternatively, the default connection can be changed by setting the default_master_connection system variable, and then the replication filter can be changed in the usual fashion. For example:

```
SET default_master_connection = 'gandalf';
STOP SLAVE;
SET GLOBAL replicate_do_table='database1.table1,database1.table2,database1.table3';
START SLAVE;
```

Setting Replication Filter Options in Option Files with Multi-Source Replication

If you are using multi-source replication and if you would like to make this filter persist server restarts by adding it to a server option group in an option file, then the option file can also include the connection name that each filter would apply to. For example:

```
[mariadb]
...
gandalf.replicate_do_db=database1
saruman.replicate_do_db=database2
```

CHANGE MASTER Options

The CHANGE MASTER statement has a few options that can be used to filter certain types of binary log events.

`IGNORE_SERVER_IDS`

The IGNORE_SERVER_IDS option for `CHANGE MASTER` can be used to configure a replica to ignore binary log events that originated from certain servers. Filtered binary log events will not get logged to the replica's relay log, and they will not be applied by the replica.

`DO_DOMAIN_IDS`

The DO_DOMAIN_IDS option for `CHANGE MASTER` can be used to configure a replica to only apply binary log events if the transaction's GTID is in a specific gtid_domain_id value. Filtered binary log events will not get logged to the replica's relay log, and they will not be applied by the replica.

`IGNORE_DOMAIN_IDS`

The IGNORE_DOMAIN_IDS option for `CHANGE MASTER` can be used to configure a replica to ignore binary log events if the transaction's GTID is in a specific gtid_domain_id value. Filtered binary log events will not get logged to the replica's relay log, and they will not be applied by the replica.

Replication Filters and Binary Log Formats

The way that a replication filter is interpreted can depend on the binary log format.

Statement-Based Logging

When an event is logged in its statement-based format, many replication filters that affect a database will test the filter against the default database (i.e. the one selected by the USE statement). This applies to the following replication filters:

- binlog_do_db
- binlog_ignore_db
- replicate_rewrite_db
- replicate_do_db
- replicate_ignore_db

When an event is logged in its statement-based format, many replication filters that affect a table will test the filter against the table in the default database (i.e. the one selected by the USE statement). This applies to the following replication filters:

- replicate_do_table
- replicate_ignore_table

This means that cross-database updates **not** work with replication filters and statement-based binary logging. For example, if replicate_do_table=db2.tab were set, then the following would not replicate with statement-based binary logging:

```
USE db1;
INSERT INTO db2.tab VALUES (1);
```

If you need to be able to support cross-database updates with replication filters and statement-based binary logging, then you should use the following replication filters:

- replicate_wild_do_table
- replicate_wild_ignore_table

Row-Based Logging

When an event is logged in its row-based format, many replication filters that affect a database will test the filter against the database that is actually affected by the event.

Similarly, when an event is logged in its row-based format, many replication filters that affect a table will test the filter against the table in the the database that is actually affected by the event.

This means that cross-database updates work with replication filters and statement-based binary logging.

> Keep in mind that DDL statements are always logged to the binary log in statement-based format, even when the binlog_format system variable is set to `ROW`. This means that the notes mentioned in Statement-Based Logging

always apply to DDL.

Replication Filters and Galera Cluster

When using Galera cluster, replication filters should be used with caution. See Configuring MariaDB Galera Cluster: Replication Filters for more details.

3.1.23 Running Triggers on the Replica for Row-based Events

The terms *master* and *slave* have historically been used in replication, but the terms terms *primary* and *replica* are now preferred. The old terms are used still used in parts of the documentation, and in MariaDB commands, although MariaDB 10.5 has begun the process of renaming. The documentation process is ongoing. See MDEV-18777 to follow progress on this effort.

Contents

MariaDB can force the replica thread to run triggers for row-based binlog events.

The setting is controlled by the slave_run_triggers_for_rbr global variable. It can be also specified as a command-line option or in my.cnf.

Possible values are:

Value	Meaning
NO (Default)	Don't invoke triggers for row-based events
YES	Invoke triggers for row-based events, don't log their effect into the binary log
LOGGING	Invoke triggers for row-based events, and log their effect into the binary log
ENFORCE	From MariaDB 10.5.2 only. Triggers will always be run on the replica, even if there are triggers on the master. ENFORCE implies LOGGING.

Note that if you just want to use triggers together with replication, you most likely don't need this option. Read below for details.

When to Use slave_run_triggers_for_rbr

Background

Normally, MariaDB's replication system can replicate trigger actions automatically.

- When one uses statement-based replication, the binary log contains SQL statements. Replica server(s) execute the SQL statements. Triggers are run on the master and on each replica, independently.
- When one uses row-based replication, the binary log contains row changes. It will have both the changes made by the statement itself, and the changes made by the triggers that were invoked by the statement. Replica server(s) do not need to run triggers for row changes they are applying.

Target Usecase

One may want to have a setup where a replica has triggers that are not present on the master (Suppose the replica needs to update summary tables or perform some other ETL-like process).

If one uses statement-based replication, they can just create the required triggers on the replica. The replica will run the statements from the binary log, which will cause the triggers to be invoked.

However, there are cases where you have to use row-based replication. It could be because the master runs non-deterministic statements, or the master could be a node in a Galera cluster. In that case, you would want row-based events

to invoke triggers on the replica. This is what the `slave_run_triggers_for_rbr` option is for. Setting the option to `YES` will cause the SQL replica thread to invoke triggers for row-based events; setting it to `LOGGING` will also cause the changes made by the triggers to be written into the binary log.

The following triggers are invoked:

- Update_row_event runs an UPDATE trigger
- Delete_row_event runs a DELETE trigger
- Write_row_event runs an INSERT trigger. Additionally it runs a DELETE trigger if there was a conflicting row that had to be deleted.

Preventing Multiple Trigger Invocations

There is a basic protection against triggers being invoked both on the master and replica. If the master modifies a table that has triggers, it will produce row-based binlog events with the "triggers were invoked for this event" flag. The replica will not invoke any triggers for flagged events.

3.1.24 Semisynchronous Replication

Contents

Description

Standard MariaDB replication is asynchronous, but MariaDB also provides a semisynchronous replication option.

With regular asynchronous replication, replicas request events from the primary's binary log whenever the replicas are ready. The primary does not wait for a replica to confirm that an event has been received.

With fully synchronous replication, all replicas are required to respond that they have received the events. See Galera Cluster.

Semisynchronous replication waits for just one replica to acknowledge that it has received and logged the events.

Semisynchronous replication therefore comes with some negative performance impact, but increased data integrity. Since the delay is based on the roundtrip time to the replica and back, this delay is minimized for servers in close proximity over fast networks.

In MariaDB 10.3 and later, semisynchronous replication is built into the server, and is no longer a plugin so it can be enabled immediately in those versions. This removes some overhead and improves performance. See MDEV-13073 for more information.

In MariaDB 10.2 and before, semisynchronous replication requires the user to install a plugin on both the primary and the replica before it can be enabled.

Installing the Plugin

> **MariaDB starting with 10.3.3**
> In MariaDB 10.3.3 and later, the Semisynchronous Replication feature is built into MariaDB server and is no longer provided by a plugin. **This means that installing the plugin is not supported on those versions.** In MariaDB 10.3.3 and later, you can skip right to Enabling Semisynchronous Replication.

The semisynchronous replication plugin is actually two different plugins--one for the primary, and one for the replica. Shared libraries for both plugins are included with MariaDB. Although the plugins' shared libraries distributed with MariaDB by default, the plugin is not actually installed by MariaDB by default prior to MariaDB 10.3.3. There are two methods that can be used to install the plugin with MariaDB.

The first method can be used to install the plugin without restarting the server. You can install the plugin dynamically by executing INSTALL SONAME or INSTALL PLUGIN.

For example, if it's a primary:

```
INSTALL SONAME 'semisync_master';
```

Or if it's a replica:

```
INSTALL SONAME 'semisync_slave';
```

The second method can be used to tell the server to load the plugin when it starts up. The plugin can be installed this way by providing the --plugin-load or the --plugin-load-add options. This can be specified as a command-line argument to mysqld or it can be specified in a relevant server option group in an option file.

For example, if it's a primary:

```
[mariadb]
...
plugin_load_add = semisync_master
```

Or if it's a replica:

```
[mariadb]
...
plugin_load_add = semisync_slave
```

Uninstalling the Plugin

> **MariaDB starting with 10.3.3**
> In MariaDB 10.3.3 and later, the Semisynchronous Replication feature is built into MariaDB server and is no longer provided by a plugin. **This means that uninstalling the plugin is not supported on those versions.**

You can uninstall the plugin dynamically by executing UNINSTALL SONAME or UNINSTALL PLUGIN.

For example, if it's a primary:

```
UNINSTALL SONAME 'semisync_master';
```

Or if it's a replica:

```
UNINSTALL SONAME 'semisync_slave';
```

If you installed the plugin by providing the --plugin-load or the --plugin-load-add options in a relevant server option group in an option file, then those options should be removed to prevent the plugin from being loaded the next time the server is restarted.

Enabling Semisynchronous Replication

Semisynchronous replication can be enabled by setting the relevant system variables on the primary and the replica.

If a server needs to be able to switch between acting as a primary and a replica, then you can enable both the primary and

replica system variables on the server. For example, you might need to do this if MariaDB MaxScale is being used to enable auto-failover or switchover with MariaDB Monitor.

Enabling Semisynchronous Replication on the Primary

Semisynchronous replication can be enabled on the primary by setting the rpl_semi_sync_master_enabled system variable to `ON`. It can be set dynamically with SET GLOBAL. For example:

```
SET GLOBAL rpl_semi_sync_master_enabled=ON;
```

It can also be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
rpl_semi_sync_master_enabled=ON
```

Enabling Semisynchronous Replication on the Replica

Semisynchronous replication can be enabled on the replica by setting the rpl_semi_sync_slave_enabled system variable to `ON`. It can be set dynamically with SET GLOBAL. For example:

```
SET GLOBAL rpl_semi_sync_slave_enabled=ON;
```

It can also be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
rpl_semi_sync_slave_enabled=ON
```

When switching between semisynchronous replication and asynchronous replication on a replica with replica IO threads already running, the replica I/O thread will need to be restarted. For example:

```
STOP SLAVE IO_THREAD;
START SLAVE IO_THREAD;
```

If this is not done, and the replica IO thread will continue to use the previous setting.

Configuring the Primary Timeout

In semisynchronous replication, only after the events have been written to the relay log and flushed does the replica acknowledge receipt of a transaction's events. If the replica does not acknowledge the transaction before a certain amount of time has passed, then a timeout occurs and the primary switches to asynchronous replication. This will be reflected in the primary's error log with messages like the following:

```
[Warning] Timeout waiting for reply of binlog (file: mariadb-1-bin.000002, pos: 538), semi-sync
up to file , position 0.
[Note] Semi-sync replication switched OFF.
```

When this occurs, the Rpl_semi_sync_master_status status variable will be switched to `OFF`.

When at least one semisynchronous replica catches up, semisynchronous replication is resumed. This will be reflected in the primary's error log with messages like the following:

```
[Note] Semi-sync replication switched ON with replica (server_id: 184137206) at (mariadb-1-
bin.000002, 215076)
```

When this occurs, the Rpl_semi_sync_master_status status variable will be switched to `ON`.

The number of times that semisynchronous replication has been switched off can be checked by looking at the value of the Rpl_semi_sync_master_no_times status variable.

If you see a lot of timeouts like this in your environment, then you may want to change the timeout period. The timeout period can be changed by setting the rpl_semi_sync_master_timeout system variable. It can be set dynamically with SET GLOBAL. For example:

```
SET GLOBAL rpl_semi_sync_master_timeout=20000;
```

It can also be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
rpl_semi_sync_master_timeout=20000
```

To determine a good value for the rpl_semi_sync_master_timeout system variable, you may want to look at the values of the Rpl_semi_sync_master_net_avg_wait_time and Rpl_semi_sync_master_tx_avg_wait_time status variables.

Configuring the Primary Wait Point

In semisynchronous replication, there are two potential points at which the primary can wait for the replica acknowledge the receipt of a transaction's events. These two wait points have different advantages and disadvantages.

The wait point is configured by the rpl_semi_sync_master_wait_point system variable. The supported values are:

- `AFTER_SYNC`
- `AFTER_COMMIT`

It can be set dynamically with SET GLOBAL. For example:

```
SET GLOBAL rpl_semi_sync_master_wait_point='AFTER_SYNC';
```

It can also be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
rpl_semi_sync_master_wait_point=AFTER_SYNC
```

When this variable is set to `AFTER_SYNC`, the primary performs the following steps:

1. Prepares the transaction in the storage engine.
2. Syncs the transaction to the binary log.
3. Waits for acknowledgement from the replica.
4. Commits the transaction to the storage engine.
5. Returns an acknowledgement to the client.

The effects of the `AFTER_SYNC` wait point are:

- All clients see the same data on the primary at the same time; after acknowledgement by the replica and after being committed to the storage engine on the primary.
- If the primary crashes, then failover should be lossless, because all transactions committed on the primary would have been replicated to the replica.
- However, if the primary crashes, then its binary log may also contain events for transactions that were prepared by the storage engine and written to the binary log, but that were never actually committed by the storage engine. As part of the server's automatic crash recovery process, the server may recover these prepared transactions when the server is restarted. This could cause the "old" crashed primary to become inconsistent with its former replicas when they have been reconfigured to replace the old primary with a new one. The old primary in such a scenario can be re-introduced only as a semisync slave. The server post-crash recovery of the server configured with `rpl_semi_sync_slave_enabled = ON` ensures through MDEV-21117 that the server will not have extra transactions. The reconfigured as semisync replica server's binlog gets truncated to discard transactions proven not to be committed, in any of their branches if they are multi-engine. Truncation does not occur though when there exists a non-transactional group of events beyond the truncation position in which case recovery reports an error. When the semisync replica recovery can't be carried out, the crashed primary may need to be rebuilt.

When this variable is set to `AFTER_COMMIT`, the primary performs the following steps:

1. Prepares the transaction in the storage engine.
2. Syncs the transaction to the binary log.
3. Commits the transaction to the storage engine.
4. Waits for acknowledgement from the replica.
5. Returns an acknowledgement to the client.

The effects of the `AFTER_COMMIT` wait point are:

- Other clients may see the committed transaction before the committing client.

- If the primary crashes, then failover may involve some data loss, because the primary may have committed transactions that had not yet been acknowledged by the replicas.

Versions

Version	Status	Introduced
N/A	N/A	MariaDB 10.3.3
1.0	Stable	MariaDB 10.1.13
1.0	Gamma	MariaDB 10.0.13
1.0	Unknown	MariaDB 10.0.11
1.0	N/A	MariaDB 5.5

System Variables

`rpl_semi_sync_master_enabled`

- **Description:** Set to `ON` to enable semi-synchronous replication primary. Disabled by default.
- **Commandline:** `--rpl-semi-sync-master-enabled[={0|1}]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

`rpl_semi_sync_master_timeout`

- **Description:** The timeout value, in milliseconds, for semi-synchronous replication in the primary. If this timeout is exceeded in waiting on a commit for acknowledgement from a replica, the primary will revert to asynchronous replication.
  - When a timeout occurs, the Rpl_semi_sync_master_status status variable will also be switched to `OFF`.
  - See Configuring the Primary Timeout for more information.
- **Commandline:** `--rpl-semi-sync-master-timeout[=#]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `10000` (10 seconds)
- **Range:** `0` to `18446744073709551615`

`rpl_semi_sync_master_trace_level`

- **Description:** The tracing level for semi-sync replication. Four levels are defined:
  - `1`: General level, including for example time function failures.
  - `16`: More detailed level, with more verbose information.
  - `32`: Net wait level, including more information about network waits.
  - `64`: Function level, including information about function entries and exits.
- **Commandline:** `--rpl-semi-sync-master-trace-level[=#]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `32`
- **Range:** `0` to `18446744073709551615`

`rpl_semi_sync_master_wait_no_slave`

- **Description:** If set to `ON`, the default, the replica count (recorded by Rpl_semi_sync_master_clients) may drop to zero, and the primary will still wait for the timeout period. If set to `OFF`, the primary will revert to asynchronous replication as soon as the replica count drops to zero.
- **Commandline:** `--rpl-semi-sync-master-wait-no-slave[={0|1}]`

- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `ON`

`rpl_semi_sync_master_wait_point`

- **Description:** Whether the transaction should wait for semi-sync acknowledgement after having synced the binlog (`AFTER_SYNC`), or after having committed in storage engine (`AFTER_COMMIT` , the default).
  - When this variable is set to `AFTER_SYNC` , the primary performs the following steps:
    1. Prepares the transaction in the storage engine.
    2. Syncs the transaction to the binary log.
    3. Waits for acknowledgement from the replica.
    4. Commits the transaction to the storage engine.
    5. Returns an acknowledgement to the client.
  - When this variable is set to `AFTER_COMMIT` , the primary performs the following steps:
    1. Prepares the transaction in the storage engine.
    2. Syncs the transaction to the binary log.
    3. Commits the transaction to the storage engine.
    4. Waits for acknowledgement from the replica.
    5. Returns an acknowledgement to the client.
  - In MariaDB 10.1.2 and before, this system variable does not exist. However, in those versions, the primary waits for the acknowledgement from replicas at a point that is equivalent to `AFTER_COMMIT` .
  - See Configuring the Primary Wait Point for more information.
- **Commandline:** `--rpl-semi-sync-master-wait-point=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `enum`
- **Default Value:** `AFTER_COMMIT`
- **Valid Values:** `AFTER_SYNC` , `AFTER_COMMIT`
- **Introduced:** MariaDB 10.1.3

`rpl_semi_sync_slave_delay_master`

- **Description:** Only write primary info file when ack is needed.
- **Commandline:** `--rpl-semi-sync-slave-delay-master[={0|1}]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB 10.3.3

`rpl_semi_sync_slave_enabled`

- **Description:** Set to `ON` to enable semi-synchronous replication replica. Disabled by default.
- **Commandline:** `--rpl-semi-sync-slave-enabled[={0|1}]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

`rpl_semi_sync_slave_kill_conn_timeout`

- **Description:** Timeout for the mysql connection used to kill the replica io_thread's connection on primary. This timeout comes into play when stop slave is executed.
- **Commandline:** `--rpl-semi-sync-slave-kill-conn-timeout[={0|1}]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `5`
- **Range:** `0` to `4294967295`

- **Introduced:** MariaDB 10.3.3

rpl_semi_sync_slave_trace_level

- **Description:** The tracing level for semi-sync replication. The levels are the same as for rpl_semi_sync_master_trace_level.
- **Commandline:** `--rpl-semi-sync-slave-trace_level[=#]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `32`
- **Range:** `0` to `18446744073709551615`

Options

rpl_semi_sync_master

- **Description:** Controls how the server should treat the plugin when the server starts up.
  - Valid values are:
    - `OFF` - Disables the plugin without removing it from the mysql.plugins table.
    - `ON` - Enables the plugin. If the plugin cannot be initialized, then the server will still continue starting up, but the plugin will be disabled.
    - `FORCE` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error.
    - `FORCE_PLUS_PERMANENT` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error. In addition, the plugin cannot be uninstalled with UNINSTALL SONAME or UNINSTALL PLUGIN while the server is running.
  - See Plugin Overview: Configuring Plugin Activation at Server Startup for more information.
- **Commandline:** `--rpl-semi-sync-master=value`
- **Data Type:** `enumerated`
- **Default Value:** `ON`
- **Valid Values:** `OFF`, `ON`, `FORCE`, `FORCE_PLUS_PERMANENT`
- **Removed:** MariaDB 10.3.3

rpl_semi_sync_slave

- **Description:** Controls how the server should treat the plugin when the server starts up.
  - Valid values are:
    - `OFF` - Disables the plugin without removing it from the mysql.plugins table.
    - `ON` - Enables the plugin. If the plugin cannot be initialized, then the server will still continue starting up, but the plugin will be disabled.
    - `FORCE` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error.
    - `FORCE_PLUS_PERMANENT` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error. In addition, the plugin cannot be uninstalled with UNINSTALL SONAME or UNINSTALL PLUGIN while the server is running.
  - See Plugin Overview: Configuring Plugin Activation at Server Startup for more information.
- **Commandline:** `--rpl-semi-sync-slave=value`
- **Data Type:** `enumerated`
- **Default Value:** `ON`
- **Valid Values:** `OFF`, `ON`, `FORCE`, `FORCE_PLUS_PERMANENT`
- **Removed:** MariaDB 10.3.3

Status Variables

For a list of status variables added when the plugin is installed, see Semisynchronous Replication Plugin Status Variables.

3.1.25 Using MariaDB Replication with MariaDB Galera Cluster

Using MariaDB Replication with MariaDB Galera Cluster

Information on using MariaDB replication with MariaDB Galera Cluster.

Using MariaDB GTIDs with MariaDB Galera Cluster

Information on using MariaDB's GTIDs with MariaDB Galera Cluster.

Configuring MariaDB Replication between MariaDB Galera Cluster and MariaDB Server

Information on configuring replication between MariaDB Galera Cluster and MariaDB Server.

Configuring MariaDB Replication between Two MariaDB Galera Clusters

Information on configuring replication between two MariaDB Galera Clusters.

There are 1 related questions.

3.1.25.1 Using MariaDB Replication with MariaDB Galera Cluster

Contents

MariaDB replication and MariaDB Galera Cluster can be used together. However, there are some things that have to be taken into account.

Tutorials

If you want to use MariaDB replication and MariaDB Galera Cluster together, then the following tutorials may be useful:

- Configuring MariaDB Replication between MariaDB Galera Cluster and MariaDB Server
- Configuring MariaDB Replication between Two MariaDB Galera Clusters

Configuring a Cluster Node as a Replication Master

If a Galera Cluster node is also a replication master, then some additional configuration may be needed.

Like with MariaDB replication, write sets that are received by a node with Galera Cluster's certification-based replication are not written to the binary log by default.

If the node is a replication master, then its replication slaves only replicate transactions which are in the binary log, so this means that the transactions that correspond to Galera Cluster write-sets would not be replicated by any replication slaves by default. If you would like a node to write its replicated write sets to the binary log, then you will have to set `log_slave_updates=ON`. If the node has any replication slaves, then this would also allow those slaves to replicate the transactions that corresponded to those write sets.

See Configuring MariaDB Galera Cluster: Writing Replicated Write Sets to the Binary Log for more information.

Configuring a Cluster Node as a Replication Slave

If a Galera Cluster node is also a replication slave, then some additional configuration may be needed.

If the node is a replication slave, then the node's slave SQL thread will be applying transactions that it replicates from its replication master. Transactions applied by the slave SQL thread will only generate Galera Cluster write-sets if the node has

`log_slave_updates=ON` set. Therefore, in order to replicate these transactions to the rest of the nodes in the cluster, `log_slave_updates=ON` must be set.

If the node is a replication slave, then it is probably also a good idea to enable `wsrep_restart_slave`. When this is enabled, the node will restart its slave threads whenever it rejoins the cluster.

Replication Filters

Both MariaDB replication and MariaDB Galera Cluster support replication filters, so extra caution must be taken when using all of these features together. See Configuring MariaDB Galera Cluster: Replication Filters for more details on how MariaDB Galera Cluster interprets replication filters.

Setting server_id on Cluster Nodes

Setting the Same server_id on Each Cluster Node

It is most common to set `server_id` to the same value on each node in a given cluster. Since MariaDB Galera Cluster uses a virtually synchronous certification-based replication, all nodes should have the same data, so in a logical sense, a cluster can be considered in many cases a single logical server for purposes related to MariaDB replication. The binary logs of each cluster node might even contain roughly the same transactions and GTIDs if `log_slave_updates=ON` is set and if wsrep GTID mode is enabled and if non-Galera transactions are not being executed on any nodes.

Setting a Different server_id on Each Cluster Node

There are cases when it might make sense to set a different `server_id` value on each node in a given cluster. For example, if `log_slave_updates=OFF` is set and if another cluster or a standard MariaDB Server is using multi-source replication to replicate transactions from each cluster node individually, then it would be required to set a different `server_id` value on each node for this to work.

Keep in mind that if replication is set up in a scenario where each cluster node has a different `server_id` value, and if the replication topology is set up in such a way that a cluster node can replicate the same transactions through Galera and through MariaDB replication, then you may need to configure the cluster node to ignore these transactions when setting up MariaDB replication. You can do so by setting `IGNORE_SERVER_IDS` to the server IDs of all nodes in the same cluster when executing `CHANGE MASTER TO`. For example, this might be required when circular replication is set up between two separate clusters, and each cluster node as a different `server_id` value, and each cluster has `log_slave_updates=ON` set.

3.1.25.2 Using MariaDB GTIDs with MariaDB Galera Cluster

Contents

MariaDB's global transaction IDs (GTIDs) are very useful when used with MariaDB replication, which is primarily what that feature was developed for. Galera Cluster, on the other hand, was developed by Codership for all MySQL and MariaDB variants, and the initial development of the technology pre-dated MariaDB's GTID implementation. As a side effect, MariaDB Galera Cluster (at least until MariaDB 10.5.1) only partially supports MariaDB's GTID implementation.

GTID Support for Write Sets Replicated by Galera Cluster

Galera Cluster has its own certification-based replication method that is substantially different from MariaDB replication. However, it would still be beneficial if MariaDB Galera Cluster was able to associate a Galera Cluster write set with a GTID that is globally unique, but that is also consistent for that write set on each cluster node.

Before MariaDB 10.5.1, MariaDB Galera Cluster did not replicate the original GTID with the write set except in cases where the transaction was originally applied by a slave SQL thread. Each node independently generated its own GTID for each write set in most cases. See MDEV-20720.

Wsrep GTID Mode

MariaDB supports wsrep_gtid_mode.

MariaDB has a feature called wsrep GTID mode. When this mode is enabled, MariaDB uses some tricks to try to associate each Galera Cluster write set with a GTID that is globally unique, but that is also consistent for that write set on each cluster node. These tricks work in some cases, but GTIDs can still become inconsistent among cluster nodes.

Enabling Wsrep GTID Mode

Several things need to be configured for wsrep GTID mode to work, such as:

- `wsrep_gtid_mode=ON` needs to be set on all nodes in the cluster.
- `wsrep_gtid_domain_id` needs to be set to the same value on all nodes in a given cluster, so that each cluster node uses the same domain when assigning GTIDs for Galera Cluster's write sets. When replicating between two clusters, each cluster should have this set to a different value, so that each cluster uses different domains when assigning GTIDs for their write sets.
- `log_slave_updates` needs to be enabled on all nodes in the cluster. See MDEV-9855.
- `log_bin` needs to be set to the same path on all nodes in the cluster. See MDEV-9856.

And as an extra safety measure:

- `gtid_domain_id` should be set to a different value on all nodes in a given cluster, and each of these values should be different than the configured `wsrep_gtid_domain_id` value. This is to prevent a node from using the same domain used for Galera Cluster's write sets when assigning GTIDs for non-Galera transactions, such as DDL executed with `wsrep_sst_method=RSU` set or DML executed with `wsrep_on=OFF` set.

For information on setting `server_id`, see Using MariaDB Replication with MariaDB Galera Cluster: Setting server_id on Cluster Nodes.

Known Problems with Wsrep GTID Mode

Until MariaDB 10.5.1, there were known cases where GTIDs could become inconsistent across the cluster nodes.

A known issue (fixed in MariaDB 10.5.1) is:

- Implicitly dropped temporary tables can make GTIDs inconsistent. See MDEV-14153 and MDEV-20720.

This does not necessarily imply that wsrep GTID mode works perfectly in all other situations. If you discover any other issues with it, please report a bug.

GTIDs for Transactions Applied by Slave Thread

If a Galera Cluster node is also a replication slave, then that node's slave SQL thread will be applying transactions that it replicates from its replication master. If the node has `log_slave_updates=ON` set, then each transaction that the slave SQL thread applies will also generate a Galera Cluster write set that is replicated to the rest of the nodes in the cluster.

In MariaDB 10.1.30 and earlier, the node acting as slave would apply the transaction with the original GTID that it received from the master, and the other Galera Cluster nodes would generate their own GTIDs for the transaction when they replicated the write set.

In MariaDB 10.1.31 and later, the node acting as slave will include the transaction's original `Gtid_Log_Event` in the replicated write set, so all nodes should associate the write set with its original GTID. See MDEV-13431 about that.

3.1.25.3 Configuring MariaDB Replication between MariaDB Galera Cluster and MariaDB Server

Contents

MariaDB replication can be used to replicate between MariaDB Galera Cluster and MariaDB Server. This article will discuss how to do that.

Configuring the Cluster

Before we set up replication, we need to ensure that the cluster is configured properly. This involves the following steps:

- Set `log_slave_updates=ON` on all nodes in the cluster. See Configuring MariaDB Galera Cluster: Writing Replicated Write Sets to the Binary Log and Using MariaDB Replication with MariaDB Galera Cluster: Configuring a Cluster Node as a Replication Master for more information on why this is important. This is also needed to enable wsrep GTID mode.
- Set `server_id` to the same value on all nodes in the cluster. See Using MariaDB Replication with MariaDB Galera Cluster: Setting server_id on Cluster Nodes for more information on what this means.

Configuring Wsrep GTID Mode

If you want to use GTID replication, then you also need to configure some things to enable wsrep GTID mode. For example:

- `wsrep_gtid_mode=ON` needs to be set on all nodes in the cluster.
- `wsrep_gtid_domain_id` needs to be set to the same value on all nodes in the cluster, so that each cluster node uses the same domain when assigning GTIDs for Galera Cluster's write sets.
- `log_slave_updates` needs to be enabled on all nodes in the cluster. See MDEV-9855 about that.
- `log_bin` needs to be set to the same path on all nodes in the cluster. See MDEV-9856 about that.

And as an extra safety measure:

- gtid_domain_id should be set to a different value on all nodes in a given cluster, and each of these values should be different than the configured wsrep_gtid_domain_id value. This is to prevent a node from using the same domain used for Galera Cluster's write sets when assigning GTIDs for non-Galera transactions, such as DDL executed with wsrep_sst_method=RSU set or DML executed with wsrep_on=OFF set.

Configuring the Replica

Before we set up replication, we also need to ensure that the MariaDB Server replica is configured properly. This involves the following steps:

- Set server_id to a different value than the one that the cluster nodes are using.
- Set gtid_domain_id to a value that is different than the wsrep_gtid_domain_id and gtid_domain_id values that the cluster nodes are using.
- Set log_bin and log_slave_updates=ON if you want the replica to log the transactions that it replicates.

Setting up Replication

Our process to set up replication is going to be similar to the process described at Setting up a Replication Slave with Mariabackup, but it will be modified a bit to work in this context.

Start the Cluster

The very first step is to start the nodes in the first cluster. The first node will have to be bootstrapped. The other nodes can be started normally.

Once the nodes are started, you need to pick a specific node that will act as the replication primary for the MariaDB Server.

Backup the Database on the Cluster's Primary Node and Prepare It

The first step is to simply take and prepare a fresh full backup of the node that you have chosen to be the replication primary. For example:

```
$ mariabackup --backup \
   --target-dir=/var/mariadb/backup/ \
   --user=mariabackup --password=mypassword
```

And then you would prepare the backup as you normally would. For example:

```
$ mariabackup --prepare \
   --target-dir=/var/mariadb/backup/
```

Copy the Backup to the Replica

Once the backup is done and prepared, you can copy it to the MariaDB Server that will be acting as replica. For example:

```
$ rsync -avrP /var/mariadb/backup dc2-dbserver1:/var/mariadb/backup
```

Restore the Backup on the Second Cluster's Replica

At this point, you can restore the backup to the datadir, as you normally would. For example:

```
$ mariabackup --copy-back \
   --target-dir=/var/mariadb/backup/
```

And adjusting file permissions, if necessary:

```
$ chown -R mysql:mysql /var/lib/mysql/
```

Start the New Replica

Now that the backup has been restored to the MariaDB Server replica, you can start the MariaDB Server process.

Create a Replication User on the Cluster's Primary

Before the MariaDB Server replica can begin replicating from the cluster's primary, you need to create a user account on the primary that the replica can use to connect, and you need to grant the user account the REPLICATION SLAVE privilege. For example:

```
CREATE USER 'repl'@'dc2-dbserver1' IDENTIFIED BY 'password';
GRANT REPLICATION SLAVE ON *.*  TO 'repl'@'dc2-dbserver1';
```

Start Replication on the New Replica

At this point, you need to get the replication coordinates of the primary from the original backup.

The coordinates will be in the xtrabackup_binlog_info file.

Mariabackup dumps replication coordinates in two forms: GTID strings and binary log file and position coordinates, like the

ones you would normally see from SHOW MASTER STATUS output. In this case, it is probably better to use the GTID coordinates.

For example:

```
mariadb-bin.000096 568 0-1-2
```

Regardless of the coordinates you use, you will have to set up the primary connection using CHANGE MASTER TO and then start the replication threads with START SLAVE.

GTIDs

If you want to use GTIDs, then you will have to first set gtid_slave_pos to the GTID coordinates that we pulled from the xtrabackup_binlog_info file, and we would set MASTER_USE_GTID=slave_pos in the CHANGE MASTER TO command. For example:

```
SET GLOBAL gtid_slave_pos = "0-1-2";
CHANGE MASTER TO
   MASTER_HOST="c1dbserver1",
   MASTER_PORT=3310,
   MASTER_USER="repl",
   MASTER_PASSWORD="password",
   MASTER_USE_GTID=slave_pos;
START SLAVE;
```

File and Position

If you want to use the binary log file and position coordinates, then you would set MASTER_LOG_FILE and MASTER_LOG_POS in the CHANGE MASTER TO command to the file and position coordinates that we pulled the xtrabackup_binlog_info file. For example:

```
CHANGE MASTER TO
   MASTER_HOST="c1dbserver1",
   MASTER_PORT=3310,
   MASTER_USER="repl",
   MASTER_PASSWORD="password",
   MASTER_LOG_FILE='mariadb-bin.000096',
   MASTER_LOG_POS=568,
START SLAVE;
```

Check the Status of the New Replica

You should be done setting up the replica now, so you should check its status with SHOW SLAVE STATUS. For example:

```
SHOW SLAVE STATUS\G
```

Now that the MariaDB Server is up, ensure that it does not start accepting writes yet if you want to set up circular replication between the cluster and the MariaDB Server.

Setting up Circular Replication

You can also set up circular replication between the cluster and MariaDB Server, which means that the MariaDB Server replicates from the cluster, and the cluster also replicates from the MariaDB Server.

Create a Replication User on the MariaDB Server Primary

Before circular replication can begin, you also need to create a user account on the MariaDB Server, since it will be acting as replication primary to the cluster's replica, and you need to grant the user account the REPLICATION SLAVE privilege. For example:

```
CREATE USER 'repl'@'c1dbserver1' IDENTIFIED BY 'password';
GRANT REPLICATION SLAVE ON *.*  TO 'repl'@'c1dbserver1';
```

Start Circular Replication on the Cluster

How this is done would depend on whether you want to use the GTID coordinates or the binary log file and position coordinates.

Regardless, you need to ensure that the second cluster is not accepting any writes other than those that it replicates from the cluster at this stage.

GTIDs

To get the GTID coordinates on the MariaDB Server you can check `gtid_current_pos` by executing:

```
SHOW GLOBAL VARIABLES LIKE 'gtid_current_pos';
```

Then on the node acting as replica in the cluster, you can set up replication by setting gtid_slave_pos to the GTID that was returned and then executing CHANGE MASTER TO:

```
SET GLOBAL gtid_slave_pos = "0-1-2";
CHANGE MASTER TO
   MASTER_HOST="c2dbserver1",
   MASTER_PORT=3310,
   MASTER_USER="repl",
   MASTER_PASSWORD="password",
   MASTER_USE_GTID=slave_pos;
START SLAVE;
```

File and Position

To get the binary log file and position coordinates on the MariaDB Server, you can execute SHOW MASTER STATUS:

```
SHOW MASTER STATUS
```

Then on the node acting as replica in the cluster, you would set `master_log_file` and `master_log_pos` in the CHANGE MASTER TO command. For example:

```
CHANGE MASTER TO
   MASTER_HOST="c2dbserver1",
   MASTER_PORT=3310,
   MASTER_USER="repl",
   MASTER_PASSWORD="password",
   MASTER_LOG_FILE='mariadb-bin.000096',
   MASTER_LOG_POS=568;
START SLAVE;
```

Check the Status of the Circular Replication

You should be done setting up the circular replication on the node in the first cluster now, so you should check its status with SHOW SLAVE STATUS. For example:

```
SHOW SLAVE STATUS\G
```

3.1.25.4 Configuring MariaDB Replication between Two MariaDB Galera Clusters

Contents

MariaDB replication can be used to replication between two MariaDB Galera Clusters. This article will discuss how to do that.

Configuring the Clusters

Before we set up replication, we need to ensure that the clusters are configured properly. This involves the following steps:

- Set `log_slave_updates=ON` on all nodes in both clusters. See Configuring MariaDB Galera Cluster: Writing Replicated Write Sets to the Binary Log and Using MariaDB Replication with MariaDB Galera Cluster: Configuring a Cluster Node as a Replication Master for more information on why this is important. This is also needed to enable wsrep GTID mode.
- Set `server_id` to the same value on all nodes in a given cluster, but be sure to use a different value in each cluster. See Using MariaDB Replication with MariaDB Galera Cluster: Setting server_id on Cluster Nodes for more information on what this means.

Configuring Wsrep GTID Mode

If you want to use GTID replication, then you also need to configure some things to enable wsrep GTID mode. For example:

- `wsrep_gtid_mode=ON` needs to be set on all nodes in each cluster.
- `wsrep_gtid_domain_id` needs to be set to the same value on all nodes in a given cluster, so that each cluster node uses the same domain when assigning GTIDs for Galera Cluster's write sets. Each cluster should have this set to a different value, so that each cluster uses different domains when assigning GTIDs for their write sets.
- `log_slave_updates` needs to be enabled on all nodes in the cluster. See MDEV-9855 about that.
- `log_bin` needs to be set to the same path on all nodes in the cluster. See MDEV-9856 about that.

And as an extra safety measure:

- `gtid_domain_id` should be set to a different value on all nodes in a given cluster, and each of these values should be different than the configured `wsrep_gtid_domain_id` value. This is to prevent a node from using the same domain used for Galera Cluster's write sets when assigning GTIDs for non-Galera transactions, such as DDL executed with `wsrep_sst_method=RSU` set or DML executed with `wsrep_on=OFF` set.

Setting up Replication

Our process to set up replication is going to be similar to the process described at Setting up a Replication Slave with Mariabackup, but it will be modified a bit to work in this context.

Start the First Cluster

The very first step is to start the nodes in the first cluster. The first node will have to be bootstrapped. The other nodes can be started normally.

Once the nodes are started, you need to pick a specific node that will act as the replication primary for the second cluster.

Backup the Database on the First Cluster's Primary Node and Prepare It

The first step is to simply take and prepare a fresh full backup of the node that you have chosen to be the replication primary. For example:

```
$ mariabackup --backup \
   --target-dir=/var/mariadb/backup/ \
   --user=mariabackup --password=mypassword
```

And then you would prepare the backup as you normally would. For example:

```
$ mariabackup --prepare \
   --target-dir=/var/mariadb/backup/
```

Copy the Backup to the Second Cluster's Replica

Once the backup is done and prepared, you can copy it to the node in the second cluster that will be acting as replica. For example:

```
$ rsync -avrP /var/mariadb/backup c2dbserver:/var/mariadb/backup
```

Restore the Backup on the Second Cluster's Replica

At this point, you can restore the backup to the datadir, as you normally would. For example:

```
$ mariabackup --copy-back \
   --target-dir=/var/mariadb/backup/
```

And adjusting file permissions, if necessary:

```
$ chown -R mysql:mysql /var/lib/mysql/
```

Bootstrap the Second Cluster's Replica

Now that the backup has been restored to the second cluster's replica, you can start the server by bootstrapping the node.

Create a Replication User on the First Cluster's Primary

Before the second cluster's replica can begin replicating from the first cluster's primary, you need to create a user account on the primary that the replica can use to connect, and you need to grant the user account the REPLICATION SLAVE privilege. For example:

```
CREATE USER 'repl'@'c2dbserver1' IDENTIFIED BY 'password';
GRANT REPLICATION SLAVE ON *.*  TO 'repl'@'c2dbserver1';
```

Start Replication on the Second Cluster's Replica

At this point, you need to get the replication coordinates of the primary from the original backup.

The coordinates will be in the xtrabackup_binlog_info file.

Mariabackup dumps replication coordinates in two forms: GTID strings and binary log file and position coordinates, like the ones you would normally see from SHOW MASTER STATUS output. In this case, it is probably better to use the GTID coordinates.

For example:

```
mariadb-bin.000096 568 0-1-2
```

Regardless of the coordinates you use, you will have to set up the primary connection using CHANGE MASTER TO and then start the replication threads with START SLAVE.

GTIDs

If you want to use GTIDs, then you will have to first set `gtid_slave_pos` to the GTID coordinates that we pulled from the `xtrabackup_binlog_info` file, and we would set `MASTER_USE_GTID=slave_pos` in the `CHANGE MASTER TO` command. For example:

```
SET GLOBAL gtid_slave_pos = "0-1-2";
CHANGE MASTER TO
   MASTER_HOST="c1dbserver1",
   MASTER_PORT=3310,
   MASTER_USER="repl",
   MASTER_PASSWORD="password",
   MASTER_USE_GTID=slave_pos;
START SLAVE;
```

File and Position

If you want to use the binary log file and position coordinates, then you would set `MASTER_LOG_FILE` and `MASTER_LOG_POS` in the `CHANGE MASTER TO` command to the file and position coordinates that we pulled the `xtrabackup_binlog_info` file. For example:

```
CHANGE MASTER TO
   MASTER_HOST="c1dbserver1",
   MASTER_PORT=3310,
   MASTER_USER="repl",
   MASTER_PASSWORD="password",
   MASTER_LOG_FILE='mariadb-bin.000096',
   MASTER_LOG_POS=568,
START SLAVE;
```

Check the Status of the Second Cluster's Replica

You should be done setting up the replica now, so you should check its status with `SHOW SLAVE STATUS`. For example:

```
SHOW SLAVE STATUS\G
```

Start the Second Cluster

If the replica is replicating normally, then the next step would be to start the MariaDB Server process on the other nodes in the second cluster.

Now that the second cluster is up, ensure that it does not start accepting writes yet if you want to set up circular replication between the two clusters.

Setting up Circular Replication

You can also set up circular replication between the two clusters, which means that the second cluster replicates from the first cluster, and the first cluster also replicates from the second cluster.

Create a Replication User on the Second Cluster's Primary

Before circular replication can begin, you also need to create a user account on the second cluster's primary that the first cluster's replica can use to connect, and you need to grant the user account the `REPLICATION SLAVE` privilege. For example:

```
CREATE USER 'repl'@'c1dbserver1' IDENTIFIED BY 'password';
GRANT REPLICATION SLAVE ON *.*  TO 'repl'@'c1dbserver1';
```

Start Circular Replication on the First Cluster

How this is done would depend on whether you want to use the GTID coordinates or the binary log file and position coordinates.

Regardless, you need to ensure that the second cluster is not accepting any writes other than those that it replicates from the first cluster at this stage.

GTIDs

To get the GTID coordinates on the second cluster, you can check `gtid_current_pos` by executing:

```
SHOW GLOBAL VARIABLES LIKE 'gtid_current_pos';
```

Then on the first cluster, you can set up replication by setting `gtid_slave_pos` to the GTID that was returned and then executing CHANGE MASTER TO:

```
SET GLOBAL gtid_slave_pos = "0-1-2";
CHANGE MASTER TO
   MASTER_HOST="c2dbserver1",
   MASTER_PORT=3310,
   MASTER_USER="repl",
   MASTER_PASSWORD="password",
   MASTER_USE_GTID=slave_pos;
START SLAVE;
```

File and Position

To get the binary log file and position coordinates on the second cluster, you can execute SHOW MASTER STATUS:

```
SHOW MASTER STATUS
```

Then on the first cluster, you would set `master_log_file` and `master_log_pos` in the CHANGE MASTER TO command. For example:

```
CHANGE MASTER TO
   MASTER_HOST="c2dbserver1",
   MASTER_PORT=3310,
   MASTER_USER="repl",
   MASTER_PASSWORD="password",
   MASTER_LOG_FILE='mariadb-bin.000096',
   MASTER_LOG_POS=568;
START SLAVE;
```

Check the Status of the Circular Replication

You should be done setting up the circular replication on the node in the first cluster now, so you should check its status with SHOW SLAVE STATUS. For example:

```
SHOW SLAVE STATUS\G
```

3.1.26 Delayed Replication

> The terms *master* and *slave* have historically been used in replication, but the terms terms *primary* and *replica* are now preferred. The old terms are used still used in parts of the documentation, and in MariaDB commands, although MariaDB 10.5 has begun the process of renaming. The documentation process is ongoing. See MDEV-18777 to follow progress on this effort.

Delayed replication allows specifying that a replica should lag behind the primary by (at least) a specified amount of time (specified in seconds). Before executing an event, the replica will first wait, if necessary, until the given time has passed since the event was created on the primary. The result is that the replica will reflect the state of the primary some time back in the past.

The default is zero, or no delay, and the maximum value is 2147483647, or about 68 years.

Delayed replication is enabled using the MASTER_DELAY option to CHANGE MASTER:

```
  CHANGE MASTER TO master_delay=3600;
```

A zero delay disables delayed replication. The replica must be stopped when changing the delay value.

Three fields in SHOW SLAVE STATUS are associated with delayed replication:

1. SQL_Delay: This is the value specified by MASTER_DELAY in CHANGE MASTER (or 0 if none).
2. SQL_Remaining_Delay. When the replica is delaying the execution of an event due to MASTER_DELAY, this is the number of seconds of delay remaining before the event will be applied. Otherwise, the value is NULL.
3. Slave_SQL_Running_State. This shows the state of the SQL driver threads, same as in SHOW PROCESSLIST. When the replica is delaying the execution of an event due to MASTER_DELAY, this fields displays: "Waiting until MASTER_DELAY seconds after master executed event".

When using older versions prior to MariaDB 10.2.3, a 3rd party tool called pt-slave-delay can be used. It is part of the Percona Toolkit. Note that pt-slave-delay does not support MariaDB multi-channel replication syntax.

3.1.27 Replication When the Primary and Replica Have Different Table Definitions

The terms *master* and *slave* have historically been used in replication, but the terms terms *primary* and *replica* are now preferred. The old terms are used still used in parts of the documentation, and in MariaDB commands, although MariaDB 10.5 has begun the process of renaming. The documentation process is ongoing. See MDEV-18777 to follow progress on this effort.

Contents

While replication is usually meant to take place between primaries and replicas with the same table definitions and this is recommended, in certain cases replication can still take place even if the definitions are identical.

Tables on the replica and the primary do not need to have the same definition in order for replication to take place. There can be differing numbers of columns, or differing data definitions and, in certain cases, replication can still proceed.

Different Column Definitions - Attribute Promotion and Demotion

It is possible in some cases to replicate to a replica that has a column of a different type on the replica and the primary. This process is called attribute promotion (to a larger type) or attribute demotion (to a smaller type).

The conditions differ depending on whether statement-based or row-based replication is used.

Statement-Based Replication

When using statement-based replication, generally, if a statement can run successfully on the replica, it will be replicated. If a column definition is the same or a larger type on the replica than on the primary, it can replicate successfully. For example a column defined as `VARCHAR(10)` will successfully be replicated on a replica with a definition of `VARCHAR(12)`.

Replicating to a replica where the column is defined as smaller than on the primary can also work. For example, given the following table definitions:

Master:

```
DESC r;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | tinyint(4)  | YES  |     | NULL    |       |
| v     | varchar(10) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
```

Slave

```
DESC r;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | tinyint(4)  | YES  |     | NULL    |       |
| v     | varchar(8)  | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
```

the statement

```
INSERT INTO r VALUES (6,'hi');
```

would successfully replicate because the value inserted into the `v` field can successfully be inserted on both the primary and the smaller replica equivalent.

However, the following statement would fail:

```
INSERT INTO r VALUES (7,'abcdefghi')
```

In this case, the value fits in the primary definition, but is too long for the replica field, and so replication will fail.

```
SHOW SLAVE STATUS\G
*************************** 1. row ***************************
...
Slave_IO_Running: Yes
Slave_SQL_Running: No
...
Last_Errno: 1406
Last_Error: Error 'Data too long for column 'v' at row 1' on query.
   Default database: 'test'. Query: 'INSERT INTO r VALUES (7,'abcdefghi')'
...
```

Row-Based Replication

When using row-based replication, the value of the slave_type_conversions variable is important. The default value of this variable is empty, in which case MariaDB will not perform attribute promotion or demotion. If the column definitions do not match, replication will stop. If set to `ALL_NON_LOSSY`, safe replication is permitted. If set to `ALL_LOSSY` as well, replication will be permitted even if data loss takes place.

For example:

Master:

```
DESC r;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | smallint(6) | YES  |     | NULL    |       |
| v     | varchar(10) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
```

Slave:

```
SHOW VARIABLES LIKE 'slave_ty%';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| slave_type_conversions |       |
+-------------------------+-------+

 DESC r;
+-------+------------+------+-----+---------+-------+
| Field | Type       | Null | Key | Default | Extra |
+-------+------------+------+-----+---------+-------+
| id    | tinyint(4) | YES  |     | NULL    |       |
| v     | varchar(1) | YES  |     | NULL    |       |
+-------+------------+------+-----+---------+-------+
```

The following query will fail:

```
INSERT INTO r VALUES (3,'c');
```

```
SHOW SLAVE STATUS\G;
...
Slave_IO_Running: Yes
Slave_SQL_Running: No
...
Last_Errno: 1677
Last_Error: Column 0 of table 'test.r' cannot be converted from
  type 'smallint' to type 'tinyint(4)'
...
```

By changing the value of the slave_type_conversions, replication can proceed:

```
SET GLOBAL slave_type_conversions='ALL_NON_LOSSY,ALL_LOSSY';

START SLAVE;
```

```
SHOW SLAVE STATUS\G;
*************************** 1. row ***************************
...
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
...
```

Supported Conversions

- Between TINYINT, SMALLINT, MEDIUMINT, INT and BIGINT. If lossy conversion is supported, the value from the primary will be converted to the maximum or minimum permitted on the replica, which non-lossy conversions require the replica column to be large enough. For example, SMALLINT UNSIGNED can be converted to MEDIUMINT, but not SMALLINT SIGNED.

Different Number or Order of Columns

Replication can also take place when the primary and replica have a different number of columns if the following criteria are met:

- columns must be in the same order on the primary and replica
- common columns must be defined with the same data type
- extra columns must be defined after the common columns

Row-Based

The following example replicates incorrectly (replication proceeds, but the data is corrupted), as the columns are not in the same order.

Master:

```
CREATE OR REPLACE TABLE r (i1 INT, i2 INT);
```

Slave:

```
ALTER TABLE r ADD i3 INT AFTER i1;
```

Master:

```
INSERT INTO r (i1,i2) VALUES (1,1);

SELECT * FROM r;
+------+------+
| i1   | i2   |
+------+------+
|    1 |    1 |
+------+------+
```

Slave:

```
SELECT * FROM r;
+------+------+------+
| i1   | i3   | i2   |
+------+------+------+
|    1 |    1 | NULL |
+------+------+------+
```

Statement-Based

Using statement-based replication, the same example may work, even though the columns are not in the same order.

Master:

```
CREATE OR REPLACE TABLE r (i1 INT, i2 INT);
```

Slave:

```
ALTER TABLE r ADD i3 INT AFTER i1;
```

Master:

```
INSERT INTO r (i1,i2) VALUES (1,1);

SELECT * FROM r;
+------+------+
| i1   | i2   |
+------+------+
|    1 |    1 |
+------+------+
```

Slave:

```
 SELECT * FROM r;
+------+------+------+
| i1   | i3   | i2   |
+------+------+------+
|    1 | NULL |    1 |
+------+------+------+
```

3.1.28 Restricting Speed of Reading Binlog from Primary by a Replica

When a replica starts after being stopped for some time, or a new replica starts that was created from a backup from some time back, a lot of old binlog events may need to be downloaded from the primary. If this happens from many replicas simultaneously, it can put a lot of load on the primary.

The **read_binlog_speed_limit** option can be used to reduce such load, by limiting the speed at which events are downloaded. The limit is given as maximum kilobytes per second to download on one replica connection.

With this option set, the replication I/O thread will limit the rate of download. Since the I/O thread is often much faster to download events than the SQL thread is at applying them, an appropriate value for **read_binlog_speed_limit** may reduce load spikes on the primary without much limit in the speed of the replica.

The option **read_binlog_speed_limit** is available starting from MariaDB 10.2.3.

`read_binlog_speed_limit`

- **Description:** Maximum speed(KB/s) to read binlog from primary.
- **Commandline:** `--read-binlog-speed-limit[=#]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `4294967295`
- **Introduced:** MariaDB 10.2.3

3.1.29 Changing a Replica to Become the Primary

The terms *master* and *slave* have historically been used in replication, but the terms terms *primary* and *replica* are now preferred. The old terms are used still used in parts of the documentation, and in MariaDB commands, although MariaDB 10.5 has begun the process of renaming. The documentation process is ongoing. See MDEV-18777 to follow progress on this effort.

Contents

This article describes how to change a replica to become a primary and optionally to set the old primary as a replica for the new primary.

A typical scenario of when this is useful is if you have set up a new version of MariaDB as a replica, for example for testing, and want to upgrade your primary to the new version.

In MariaDB replication, a replica should be of a version same or newer than the primary. Because of this, one should first upgrades all replicas to the latest version before changing a replica to be a primary. In some cases one can have a replica to be of an older version than the primary, as long as one doesn't execute on the primary any SQL commands that the replica doesn't understand. This is however not guaranteed between all major MariaDB versions.

Note that in the examples below, `[connection_name]` is used as the name of the connection. If you are not using named connections you can ignore this.

Stopping the Original Master.

First one needs to take down the original primary in such a way that the replica has all information on the primary.

If you are using Semisynchronous Replication you can just stop the server with the SHUTDOWN command as the replicas should be automatically up to date.

If you are using MariaDB MaxScale proxy, then you can use MaxScale to handle the whole process of taking down the primary and replacing it with one of the replicas.

If neither of the above is true, you have to do this step manually:

Manually Take Down the Primary

First we have to set the primary to read only to ensure that there are no new updates on the primary:

```
FLUSH TABLES WITH READ LOCK;
```

Note that you should not disconnect this session as otherwise the read lock will disappear and you have to start from the beginning.

Then you should check the current position of the primary:

```
SHOW MASTER STATUS;
+--------------------+----------+--------------+------------------+
| File               | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+--------------------+----------+--------------+------------------+
| mariadb-bin.000003 |      343 |              |                  |
+--------------------+----------+--------------+------------------+
SELECT @@global.gtid_binlog_pos;
+---------------------------+
| @@global.gtid_binlog_pos |
+---------------------------+
| 0-1-2                     |
+---------------------------+
```

And wait until you have the same position on the replica: (The following should be excepted on the replica)

```
SHOW SLAVE [connection_name] STATUS;
+------------------+-------------------+
Master_Log_File     | narttu-bin.000003 +
Read_Master_Log_Pos | 343               +
Exec_Master_Log_Pos | 343               +
...
Gtid_IO_Pos          0-1-2              +
+------------------+-------------------+
```

The most important information to watch are `Master_Log_File` and `Exec_Master_Log_Pos` as when this matches the primary, it signals that all transactions has been committed on the replica.

Note that `Gtid_IO_Pos` on replica can contain many different positions separated with ',' if the replica has been connected to many different primaries. What is important is that all the sequences that are on the primary is also on the replica.

When replica is up to date, you can then take the **PRIMARY** down. This should be on the same connection where you executed FLUSH TABLES WITH READ LOCK.

```
SHUTDOWN;
```

Preparing the Replica to be a Primary

Stop all old connections to the old primary(s) and reset **read only mode**, if you had it enabled. You also want to save the values of SHOW MASTER STATUS and `gtid_binlog_pos`, as you may need these to setup new replicas.

```
STOP ALL SLAVES;
RESET SLAVE ALL;
SHOW MASTER STATUS;
SELECT @@global.gtid_binlog_pos;
SET @@global.read_only=0;
```

Reconnect Other Replicas to the New Primary

On the other replicas you have point them to the new primary (the replica you promoted to a primary).

```
STOP SLAVE [connection_name];
CHANGE MASTER [connection_name] TO MASTER_HOST="new_master_name",
MASTER_PORT=3306, MASTER_USER='root', MASTER_USE_GTID=current_pos,
MASTER_LOG_FILE="XXX", MASTER_LOG_POS=XXX;
START SLAVE;
```

The `XXX` values for `MASTER_LOG_FILE` and `MASTER_LOG_POS` should be the values you got from the `SHOW MASTER STATUS` command you did when you finished setting up the replica.

Changing the Old Primary to be a Replica

Now you can upgrade the new primary to a newer version of MariaDB and then follow the same procedure to connect it as a replica.

When starting the original primary, it's good to start the `mysqld` executable with the `--with-skip-slave-start` and `--read-only` options to ensure that no old replica configurations could cause any conflicts.

For the same reason it's also good to execute the following commands on the old primary (same as for other replicas, but with some extra security). The `read_only` option below is there to ensure that old applications doesn't by accident try to update the old primary by mistake. It only affects normal connections to the replica, not changes from the new primary.

```
set @@global.read_only=1;
STOP ALL SLAVES;
RESET MASTER;
RESET SLAVE ALL;
CHANGE MASTER [connection_name] TO MASTER_HOST="new_master_name",
MASTER_PORT=3306, MASTER_USER='root', MASTER_USE_GTID=current_pos,
MASTER_LOG_FILE="XXX", MASTER_LOG_POS=XXX;
START SLAVE;
```

Moving Applications to Use New Primary

You should now point your applications to use the new primary. If you are using the MariaDB MaxScale proxy, then you don't have to do this step as MaxScale will take care of sending write request to the new primary.

2.2.1.1.1.4 Replication with Secure Connections

3.2 MariaDB Galera Cluster

MariaDB Galera Cluster is a virtually synchronous multi-master cluster that runs on Linux only. It has been a standard part of the server since MariaDB 10.1.

What is MariaDB Galera Cluster?
Basic information on MariaDB Galera Cluster.

About Galera Replication
About Galera replication.

Galera Use Cases
Common use cases for Galera replication.

MariaDB Galera Cluster - Known Limitations
Describing the known limitations of MariaDB Galera Cluster.

Tips on Converting to Galera
Best/required practices when using Galera for High-availability

Getting Started with MariaDB Galera Cluster
Synchronous multi-master cluster for Linux supporting InnoDB storage engines.

Configuring MariaDB Galera Cluster
Details on how to configure MariaDB Galera Cluster.

State Snapshot Transfers (SSTs) in Galera Cluster
In an SST, the cluster provisions nodes by transferring a full data copy from one node to another.

Galera Cluster Status Variables
Galera Cluster status variables.

Galera Cluster System Variables
Listing and description of Galera Cluster system variables.

Building the Galera wsrep Package on Ubuntu and Debian
Short how-to on building the galera package on Debian

Building the Galera wsrep Package on Fedora
Short how-to on building the galera package on Fedora

Installing Galera from Source
Building Galera from source.

Galera Test Repositories
To facilitate development and QA, we have created some test repos for the Galera wsrep provider.

wsrep_provider_options
Galera options set with the wsrep_provider_options variable.

Galera Cluster Address
Galera URLs

Galera Load Balancer
A load balancer specifically designed for Galera Cluster

MariaDB Galera Cluster Releases
Galera Cluster release notes and changelogs.

Upgrading Galera Cluster
Upgrading MariaDB Galera Cluster.

Using MariaDB Replication with MariaDB Galera Cluster
Information on using MariaDB replication with MariaDB Galera Cluster.

Securing Communications in Galera Cluster
Enabling TLS encryption in transit for Galera Cluster.

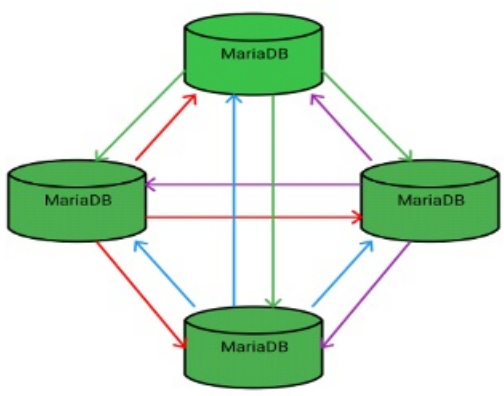
Installing MariaDB Galera on IBM Cloud
Get MariaDB Galera on IBM Cloud You should have an IBM Cloud account, oth...

There are 41 related questions.

3.2.1 What is MariaDB Galera Cluster?

Contents

About



MariaDB Galera Cluster is a virtually synchronous multi-primary cluster for MariaDB. It is available on Linux only, and only supports the InnoDB storage engine (although there is experimental support for MyISAM and, from MariaDB 10.6, Aria. See

the wsrep_replicate_myisam system variable, or, from MariaDB 10.6, the wsrep_mode system variable).

Features

- Virtually synchronous replication
- Active-active multi-primary topology
- Read and write to any cluster node
- Automatic membership control, failed nodes drop from the cluster
- Automatic node joining
- True parallel replication, on row level
- Direct client connections, native MariaDB look & feel

Benefits

The above features yield several benefits for a DBMS clustering solution, including:

- No replica lag
- No lost transactions
- Read scalability
- Smaller client latencies

The Getting Started with MariaDB Galera Cluster page has instructions on how to get up and running with MariaDB Galera Cluster.

> A great resource for Galera users is Codership on Google Groups (`codership-team 'at' googlegroups (dot) com`) - If you use Galera it is recommended you subscribe.

Galera Versions

MariaDB Galera Cluster is powered by:

- MariaDB Server.
- The Galera wsrep provider library.

The functionality of MariaDB Galera Cluster can be obtained by installing the standard MariaDB Server packages and the Galera wsrep provider library package. The following Galera version corresponds to each MariaDB Server version:

- In MariaDB 10.4 and later, MariaDB Galera Cluster uses Galera 4. This means that the wsrep API version is 26 and the Galera wsrep provider library is version 4.X.
- In MariaDB 10.3 and before, MariaDB Galera Cluster uses Galera 3. This means that the wsrep API is version 25 and the Galera wsrep provider library is version 3.X.

See Deciphering Galera Version Numbers for more information about how to interpret these version numbers.

Galera 4 Versions

The following table lists each version of the Galera 4 wsrep provider, and it lists which version of MariaDB each one was first released in. If you would like to install Galera 4 using yum, apt, or zypper, then the package is called `galera-4` .

Galera Version	Released in MariaDB Version
26.4.14	10.10.3, 10.9.5, 10.8.7, 10.7.8, 10.6.12, 10.5.19, 10.4.28
26.4.13	10.10.2, 10.9.4, 10.8.6, 10.7.7, 10.6.11, 10.5.18, 10.4.27
26.4.12	10.10.1, 10.9.2, 10.8.4, 10.7.5, 10.6.9, 10.5.17, 10.4.26
26.4.11	10.8.1, 10.7.2, 10.6.6, 10.5.14, 10.4.22
26.4.9	10.6.4, 10.5.12, 10.4.21
26.4.8	10.6.1, 10.5.10, 10.4.19
26.4.7	10.5.9, 10.4.18
26.4.6	10.5.7, 10.4.16
26.4.5	10.5.4, 10.4.14
26.4.4	10.5.1, 10.4.13

26.4.3	10.5.0, 10.4.9
26.4.2	10.4.4
26.4.1	10.4.3
26.4.0	10.4.2

Galera 3 Versions

The following table lists each version of the Galera 3 wsrep provider, and it lists which version of MariaDB each one was first released in. If you would like to install Galera 3 using yum, apt, or zypper, then the package is called `galera`.

Galera Version	Released in MariaDB Version
25.3.37	MariaDB 10.3.36
25.3.35	MariaDB 10.3.33, MariaDB 10.2.42
25.3.34	MariaDB 10.3.31, MariaDB 10.2.40
25.3.33	MariaDB 10.3.29, MariaDB 10.2.38
25.3.32	MariaDB 10.3.28, MariaDB 10.2.37
25.3.31	MariaDB 10.3.26, MariaDB 10.2.35, MariaDB 10.1.48
25.3.30	MariaDB 10.3.25, MariaDB 10.2.34, MariaDB 10.1.47
25.3.29	MariaDB 10.3.23, MariaDB 10.2.32, MariaDB 10.1.45
25.3.28	MariaDB 10.3.19, MariaDB 10.2.28, MariaDB 10.1.42
25.3.27	MariaDB 10.3.18, MariaDB 10.2.27
25.3.26	MariaDB 10.3.14, MariaDB 10.2.23, MariaDB 10.1.39
25.3.25	MariaDB 10.3.12, MariaDB 10.2.20, MariaDB 10.1.38, MariaDB Galera Cluster 10.0.38, MariaDB Galera Cluster 5.5.63
25.3.24	MariaDB 10.4.0, MariaDB 10.3.10, MariaDB 10.2.18, MariaDB 10.1.37, MariaDB Galera Cluster 10.0.37, MariaDB Galera Cluster 5.5.62
25.3.23	MariaDB 10.3.5, MariaDB 10.2.13, MariaDB 10.1.32, MariaDB Galera Cluster 10.0.35, MariaDB Galera Cluster 5.5.60
25.3.22	MariaDB 10.3.3, MariaDB 10.2.11, MariaDB 10.1.29, MariaDB Galera Cluster 10.0.33, MariaDB Galera Cluster 5.5.59
25.3.21	N/A
25.3.20	MariaDB 10.3.1, MariaDB 10.2.6, MariaDB 10.1.23, MariaDB Galera Cluster 10.0.31, MariaDB Galera Cluster 5.5.56
25.3.19	MariaDB 10.3.0, MariaDB 10.2.3, MariaDB 10.1.20, MariaDB Galera Cluster 10.0.29, MariaDB Galera Cluster 5.5.54
25.3.18	MariaDB 10.2.2, MariaDB 10.1.18, MariaDB Galera Cluster 10.0.28, MariaDB Galera Cluster 5.5.53
25.3.17	MariaDB 10.1.17, MariaDB Galera Cluster 10.0.27, MariaDB Galera Cluster 5.5.51
25.3.16	N/A
25.3.15	MariaDB 10.2.0, MariaDB 10.1.13, MariaDB Galera Cluster 10.0.25, MariaDB Galera Cluster 5.5.49
25.3.14	MariaDB 10.1.12, MariaDB Galera Cluster 10.0.24, MariaDB Galera Cluster 5.5.48
25.3.12	MariaDB 10.1.11
25.3.11	N/A
25.3.10	N/A
25.3.9	MariaDB 10.1.3, MariaDB Galera Cluster 10.0.17, MariaDB Galera Cluster 5.5.42
25.3.8	N/A
25.3.7	N/A
25.3.6	N/A

25.3.5	MariaDB 10.1.1, MariaDB Galera Cluster 10.0.10, MariaDB Galera Cluster 5.5.37
25.3.4	N/A
25.3.3	N/A
25.3.2	MariaDB Galera Cluster 10.0.7, MariaDB Galera Cluster 5.5.35

3.2.2 About Galera Replication

Contents

In MariaDB Cluster, the Server replicates a transaction at commit time by broadcasting the write set associated with the transaction to every node in the cluster. The client connects directly to the DBMS and experiences behavior that is similar to native MariaDB in most cases. The wsrep API (write set replication API) defines the interface between Galera replication and MariaDB.

Synchronous vs. Asynchronous Replication

The basic difference between synchronous and asynchronous replication is that "synchronous" replication guarantees that if a change happened on one node in the cluster, then the change will happen on other nodes in the cluster "synchronously", or at the same time. "Asynchronous" replication gives no guarantees about the delay between applying changes on "master" node and the propagation of changes to "slave" nodes. The delay with "asynchronous" replication can be short or long. This also implies that if master node crashes in an "asynchronous" replication topology, then some of the latest changes may be lost.

Theoretically, synchronous replication has a number of advantages over asynchronous replication:

- Clusters utilizing synchronous replication are always highly available. If one of the nodes crashed, then there would be no data loss. Additionally, all cluster nodes are always consistent.
- Clusters utilizing synchronous replication allow transactions to be executed on all nodes in parallel.
- Clusters utilizing synchronous replication can guarantee causality across the whole cluster. This means that if a `SELECT` is executed on one cluster node after a transaction is executed on a cluster node, it should see the effects of that transaction.

However, in practice, synchronous database replication has traditionally been implemented via the so-called "2-phase commit" or distributed locking which proved to be very slow. Low performance and complexity of implementation of synchronous replication led to a situation where asynchronous replication remains the dominant means for database performance scalability and availability. Widely adopted open-source databases such as MySQL or PostgreSQL offer only asynchronous or semi-synchronous replication solutions.

Galera's replication is not completely synchronous. It is sometimes called **virtually synchronous** replication.

Certification-Based Replication Method

An alternative approach to synchronous replication that uses Group Communication and transaction ordering techniques was suggested by a number of researchers. For example:

- Database State Machine Approach
- Don't Be Lazy, Be Consistent

Prototype implementations have shown a lot of promise. We combined our experience in synchronous database replication and the latest research in the field to create the Galera Replication library and the wsrep API.

Galera replication is a **highly transparent**, **scalable**, and **virtually synchronous** replication solution for database clustering to achieve high availability and improved performance. Galera-based clusters are:

- Highly available
- Highly transparent
- Highly scalable (near linear scalability may be reached depending on the application)

Generic Replication Library

Galera replication functionality is implemented as a shared library and can be linked with any transaction processing system which implements the wsrep API hooks.

The Galera replication library is a protocol stack providing functionality for preparing, replicating and applying of transaction write sets. It consists of:

- **wsrep API** specifies the interface — responsibilities for DBMS and replication provider
- **wsrep hooks** is the wsrep integration in the DBMS engine.
- **Galera provider** implements the wsrep API for Galera library
- **certification** layer takes care of preparing write sets and performing certification
- **replication** manages replication protocol and provides total ordering capabilities
- **GCS framework** provides plugin architecture for group communication systems
- many gcs implementations can be adapted, we have experimented with spread and our in-house implementations: vsbes and gemini

Many components in the Galera replication library were redesigned and improved with the introduction of MariaDB 10.4, which includes Galera 4.

Galera Slave Threads

Although the **Galera provider** certifies the write set associated with a transaction at commit time on each node in the cluster, this write set is not necessarily applied on that cluster node immediately. Instead, the write set is placed in the cluster node's receive queue on the node, and it is eventually applied by one of the cluster node's Galera slave thread.

The number of Galera slave threads can be configured with the wsrep_slave_threads system variable.

The Galera slave threads are able to determine which write sets are safe to apply in parallel. However, if your cluster nodes seem to have frequent consistency problems, then setting the value to `1` will probably fix the problem.

When a cluster node's state, as seen by wsrep_local_state_comment, is `JOINED`, then increasing the number of slave threads may help the cluster node catch up with the cluster more quickly. In this case, it may be useful to set the number of threads to twice the number of CPUs on the system.

Streaming Replication

MariaDB starting with 10.4

Streaming replication was introduced in Galera 4, and so is only available from MariaDB 10.4.

In older versions of MariaDB Cluster there was a 2GB limit on the size of the transaction you could run. The node waits on the transaction commit before performing replication and certification. With large transactions, long running writes, and changes to huge data-sets, there was a greater possibility of a conflict forcing rollback on an expensive operation.

Using Streaming replication, the node breaks huge transactions up into smaller and more manageable fragments, it then replicates these fragments to the cluster as it works instead of waiting for the commit. Once certified, the fragment can no longer be aborted by conflicting transactions. As this can have performance consequences both during execution and in the event of rollback, it is recommended that you only use it with large transactions that are unlikely to experience conflict.

For more information on Streaming Replication, see the Galera documentation.

Group Commits

MariaDB starting with 10.4

Group Commit support for MariaDB Cluster was introduced in Galera 4, and so is only available from MariaDB 10.4.

In MariaDB Group Commit, groups of transactions are flushed together to disk to improve performance. Prior to MariaDB 10.4, this feature was not available in MariaDB Cluster as it interfered with the global-ordering of transactions for replication. Beginning in 10.4, MariaDB Cluster can take advantage of Group Commit.

For more information on Group Commit, see the Galera documentation.

3.2.3 Galera Use Cases

Here are some common use cases for Galera replication:

Read Master	Traditional MariaDB master-slave topology, but with Galera all "slave" nodes are capable masters at all times - it is just the application that treats them as slaves. Galera replication can guarantee zero slave lag for such installations and, due to parallel slave applying, much better throughput for the cluster.

WAN Clustering	Synchronous replication works fine over the WAN network. There will be a delay, which is proportional to the network round trip time (RTT), but it only affects the commit operation.
Disaster Recover	Disaster recovery is a sub-class of WAN replication. Here one data center is passive and only receives replication events, but does not process any client transactions. Such a remote data center will be up to date at all times and no data loss can happen. During recovery, the spare site is just nominated as primary and application can continue as normal with a minimal fail over delay.
Latency Eraser	With WAN replication topology, cluster nodes can be located close to clients. Therefore all read & write operations will be super fast with the local node connection. The RTT related delay will be experienced only at commit time, and even then it can be generally accepted by end user, usually the kill-joy for end user experiences is the slow browsing response time, and read operations are as fast as they possibly can be.

3.2.4 MariaDB Galera Cluster - Known Limitations

This article contains information on known problems and limitations of MariaDB Galera Cluster.

Limitations from codership.com:

- Currently replication works only with the InnoDB storage engine. Any writes to tables of other types, including system (mysql.*) tables are not replicated (this limitation excludes DDL statements such as CREATE USER, which implicitly modify the mysql.* tables — those are replicated). There is however experimental support for MyISAM - see the wsrep_replicate_myisam system variable)
- Unsupported explicit locking include LOCK TABLES, FLUSH TABLES {explicit table list} WITH READ LOCK, (GET_LOCK(), RELEASE_LOCK(),…). Using transactions properly should be able to overcome these limitations. Global locking operators like FLUSH TABLES WITH READ LOCK are supported.
- All tables should have a primary key (multi-column primary keys are supported). DELETE operations are unsupported on tables without a primary key. Also, rows in tables without a primary key may appear in a different order on different nodes.
- The general query log and the slow query log cannot be directed to a table. If you enable these logs, then you must forward the log to a file by setting `log_output=FILE`.
- XA transactions are not supported.
- Transaction size. While Galera does not explicitly limit the transaction size, a writeset is processed as a single memory-resident buffer and as a result, extremely large transactions (e.g. LOAD DATA) may adversely affect node performance. To avoid that, the wsrep_max_ws_rows and wsrep_max_ws_size system variables limit transaction rows to 128K and the transaction size to 2Gb by default. If necessary, users may want to increase those limits. Future versions will add support for transaction fragmentation.

Other observations, in no particular order:

- If you are using mysqldump for state transfer, and it failed for whatever reason (e.g. you do not have the database account it attempts to connect with, or it does not have necessary permissions), you will see an SQL SYNTAX error in the server error log. Don't let it fool you, this is just a fancy way to deliver a message (the pseudo-statement inside of the bogus SQL will actually contain the error message).
- Do not use transactions of any essential size. Just to insert 100K rows, the server might require additional 200-300 Mb. In a less fortunate scenario it can be 1.5 Gb for 500K rows, or 3.5 Gb for 1M rows. See MDEV-466 for some numbers (you'll see that it's closed, but it's not closed because it was fixed).
- Locking is lax when DDL is involved. For example, if your DML transaction uses a table, and a parallel DDL statement is started, in the normal MySQL setup it would have waited for the metadata lock, but in Galera context it will be executed right away. It happens even if you are running a single node, as long as you configured it as a cluster node. See also MDEV-468. This behavior might cause various side-effects, the consequences have not been investigated yet. Try to avoid such parallelism.
- Do not rely on auto-increment values to be sequential. Galera uses a mechanism based on autoincrement increment to produce unique non-conflicting sequences, so on every single node the sequence will have gaps. See http://codership.blogspot.com/2009/02/managing-auto-increments-with-multi.html
- A command may fail with `ER_UNKNOWN_COM_ERROR` producing 'WSREP has not yet prepared node for application use' (or 'Unknown command' in older versions) error message. It happens when a cluster is suspected to be split and the node is in a smaller part — for example, during a network glitch, when nodes temporarily lose each other. It can

also occur during state transfer. The node takes this measure to prevent data inconsistency. Its usually a temporary state which can be detected by checking wsrep_ready value. The node, however, allows SHOW and SET command during this period.

- After a temporary split, if the 'good' part of the cluster was still reachable and its state was modified, resynchronization occurs. As a part of it, nodes of the 'bad' part of the cluster drop all client connections. It might be quite unexpected, especially if the client was idle and did not even know anything wrong was happening. Please also note that after the connection to the isolated node is restored, if there is a flow on the node, it takes a long time for it to synchronize, during which the "good" node says that the cluster is already of the normal size and synced, while the rejoining node says it's only joined (but not synced). The connections keep getting 'unknown command'. It should pass eventually.
- While binlog_format is checked on startup and can only be ROW (see Binary Log Formats), it can be changed at runtime. Do NOT change binlog_format at runtime, it is likely not only cause replication failure, but make all other nodes crash.
- If you are using rsync for state transfer, and a node crashes before the state transfer is over, rsync process might hang forever, occupying the port and not allowing to restart the node. The problem will show up as 'port in use' in the server error log. Find the orphan rsync process and kill it manually.
- Performance: by design performance of the cluster cannot be higher than performance of the slowest node; however, even if you have only one node, its performance can be considerably lower comparing to running the same server in a standalone mode (without wsrep provider). It is particularly true for big enough transactions (even those which are well within current limitations on transaction size quoted above).
- Windows is not supported.
- Replication filters: When using Galera cluster, replication filters should be used with caution. See Configuring MariaDB Galera Cluster: Replication Filters for more details. See also MDEV-421 and MDEV-6229.
- Flashback isn't supported in Galera due to incompatible binary log format.
- `FLUSH PRIVILEGES` is not replicated.
- The query cache needed to be disabled by setting `query_cache_size=0` prior to MariaDB Galera Cluster 5.5.40, MariaDB Galera Cluster 10.0.14, and MariaDB 10.1.2..
- In an asynchronous replication setup where a master replicates to a galera node acting as slave, parallel replication (slave-parallel-threads > 1) on slave is currently not supported (see MDEV-6860).
- The disk-based Galera gcache is not encrypted (MDEV-8072).
- Nodes may have different table definitions, especially temporarily during rolling schema upgrade operations, but the same schema compatibility restrictions apply as they do for ROW based replication

3.2.5 Tips on Converting to Galera

Contents

These topics will be discussed in more detail below.

Dear Schema Designer:

- InnoDB only, always have PK.

Dear Developer:

- Check for errors, even after COMMIT.
- Moderate sized transactions.
- Don't make assumptions about AUTO_INCREMENT values.
- Handling of "critical reads" is quite different (arguably better).
- Read/Write split is not necessary, but is still advised in case the underlying structure changes in the future.

Dear DBA:

- Building the machines is quite different. (Not covered here)
- ALTERs are handled differently.
- TRIGGERs and EVENTs may need checking.
- Tricks in replication (eg, BLACKHOLE) may not work.
- Several variables need to be set differently.

Galera is available in many places

Galera's High Availability replication is available via:

- MariaDB 10.1 and later
- Percona XtraDB Cluster
- Codership's Galera Cluster for MySQL

Overview of cross-colo writing

(This overview is valid even for same-datacenter nodes, but the issues of latency vanish.)

Cross-colo latency is an 'different' than with traditional replication, but not necessarily better or worse with Galera. The latency happens at a very different time for Galera.

In 'traditional' replication, these steps occur:

- Client talks to Master. If Client and Master are in different colos, this has a latency hit.
- Each SQL to Master is another latency hit, including(?) the COMMIT (unless using autocommit).
- Replication to Slave(s) is asynchronous, so this does not impact the client writing to the Master.
- Since replication is asynchronous, a Client (same or subsequent) cannot be guaranteed to see that data on the Slave. This is a "critical read". The async Replication delay forces apps to take some evasive action.

In Galera-based replication:

- Client talks to any Master -- possibly with cross-colo latency. Or you could arrange to have Galera nodes co-located with clients to avoid this latency.
- At COMMIT time (or end of statement, in case of autocommit=1), galera makes one roundtrip to other nodes.
- The COMMIT usually succeeds, but could fail if some other node is messing with the same rows. (Galera retries on autocommit failures.)
- Failure of the COMMIT is reported to the Client, who should simply replay the SQL statements from the BEGIN.
- Later, the whole transaction will be applied (with possibility of conflict) on the other nodes.
- Critical Read -- details below

For an N-statement transaction: In a typical 'traditional' replication setup:

- 0 or N (N+2?) latency hits, depending on whether the Client is co-located with the Master.
- Replication latencies and delays lead to issues with "Critical Reads".

In Galera:

- 0 latency hits (assuming Client is 'near' some node)
- 1 latency hit for the COMMIT.
- 0 (usually) for Critical Read (details below)

Bottom line: Depending on where your Clients are, and whether you clump statements into BEGIN...COMMIT transacitons, Galera may be faster or slower than traditional replication in a WAN topology.

AUTO_INCREMENT

By using wsrep_auto_increment_control = ON, the values of auto_increment_increment and auto_increment_offset will be automatically adjusted as nodes come/go.

If you are building a Galera cluster by starting with one node as a Slave to an existing non-Galera system, and if you have multi-row INSERTs that depend on AUTO_INCREMENTs, the read this Percona blog

Bottom line: There may be gaps in AUTO_INCREMENT values. Consecutive rows, even on one connection, will not have

consecutive ids.

Beware of Proxies that try to implement a "read/write split". In some situations, a reference to LAST_INSERT_ID() will be sent to a "Slave".

InnoDB only

For effective replication of data, you must use only InnoDB. This eliminates

- FULLTEXT index (until 5.6)
- SPATIAL index
- MyISAM's PK as second column

You can use MyISAM and MEMORY for data that does not need to be replicated.

Also, you should use "START TRANSACTION READONLY" wherever appropriate.

Check after COMMIT

Check for errors after issuing COMMIT. A "deadlock" can occur due to writes on other node(s).

Possible exception (could be useful for legacy code without such checks): Treat the system as single-Master, plus Slaves. By writing only to one node, COMMIT should always succeed(?)

What about autocommit = 1? wsrep_retry_autocommit tells Galera to retry if a single statement that is autocommited N times. So, there is still a chance (very slim) of getting a deadlock on such a statement. The default setting of "1" retry is probably good.

Always have PRIMARY KEY

"Row Based Replication" will be used; this requires a PK on every table.

A non-replicated table (eg, MyISAM) does not have to have a PK.

Transaction "size"

(This section assumes you have Galera nodes in multiple colos.) Because of some of the issues discussed, it is wise to group your write statements into moderate sized BEGIN...COMMIT transactions. There is one latency hit per COMMIT or autocommit. So, combining statements will decrease those hits. On the other hand, it is unwise (for other reasons) to make huge transactions, such as inserting/modifying millions of rows in a single transaction.

To deal with failure on COMMIT, design your code so you can redo the SQL statements in the transaction without messing up other data. For example, move "normalization" statements out of the main transaction; there is arguably no compelling reason to roll them back if the main code rolls back.

In any case, doing what is "right" for the business logic overrides other considerations.

Galera's tx_isolation is between Serializable and Repeatable Read. tx_isolation variable is ignored.

Set wsrep_log_conflicts to get errors put in the regular MySQL mysqld.err.

XA transactions cannot be supported. (Galera is already doing a form of XA in order to do its thing.)

Critical reads

Here is a 'simple' (but not 'free') way to assure that a read-after-write, even from a different connection, will see the updated data.

SET SESSION wsrep_sync_wait = 1; SELECT ... SET SESSION wsrep_sync_wait = 0;

For non-SELECTs, use a different bit set for the first select. (TBD: Would 0xffff always work?) (Before Galera 3.6, it was wsrep_causal_reads = ON.) Doc for wsrep_sync_wait

This setting stalls the SELECT until all current updates have been applied to the node. That is sufficient to guarantee that a previous write will be visible. The time cost is usually zero. However, a large UPDATE could lead to a delay. Because of RBR and parallel application, delays are likely to be less than on traditional replication. Zaitsev's blog

It may be more practical (for a web app) to simply set wsrep_sync_wait right after connecting.

MyISAM and MEMORY

As said above, use InnoDB only. However, here is more info on the MyISAM (and hence FULLTEXT, SPATIAL, etc) issues.

MyISAM and MEMORY tables are not replicated.

Having MyISAM not replicated can be a big benefit -- You can "CREATE TEMPORARY TABLE ... ENGINE=MyISAM" and have it exist on only one node. RBR assures that any data transferred from that temp table into a 'real' table can still be replicated.

Replicating GRANTs

GRANTs and related operations act on the MyISAM tables in the database `mysql`. The GRANT statements will(?) be replicated, but the underlying tables will not.

ALTERs

Many DDL changes on Galera can be achieved without downtime, even if they take a long time.

RSU vs TOI:

- Rolling Schema Upgrade (RSU): manually execute the DDL on each node in the cluster. The node will desync while executing the DDL.
- Total Order Isolation (TOI): Galera automatically replicates the DDL to each node in the cluster, and it synchronizes each node so that the statement is executed at same time (in the replication sequence) on all nodes.

Caution: Since there is no way to synchronize the clients with the DDL, you must make sure that the clients are happy with either the old or the new schema. Otherwise, you will probably need to take down the entire cluster while simultaneously switching over both the schema and the client code.

Fast DDL operations can usually be executed in TOI mode:

- DDL operations that support the `NOCOPY` and `INSTANT` algorithms are usually very fast.
- DDL operations that support the `INPLACE` algorithm may be fast or slow, depending on whether the table needs to be rebuilt.
- DDL operations that only support the `COPY` algorithm are usually very slow.

For a list of which operations support which algorithms, see InnoDB Online DDL.

If you need to use RSU mode, then do the following separately for each node:

```
SET SESSION wsrep_OSU_method='RSU';
ALTER TABLE tab <alter options here>;
SET SESSION wsrep_OSU_method='TOI';
```

More discussion of RSU procedures

Single "Master" Configuration

You can 'simulate' Master + Slaves by having clients write only to one node.

- No need to check for errors after COMMIT.
- Lose the latency benefits.

DBA tricks

- Remove node from cluster; back it up; put it back in. Syncup is automatic.
- Remove node from cluster; use it for testing, etc; put it back in. Syncup is automatic.
- Rolling hardware/software upgrade: Remove; upgrade; put back in. Repeat.

Variables that may need to be different

- auto_increment_increment - If you are writing to multiple nodes, and you use AUTO_INCREMENT, then auto_increment_increment will automatically be equal the current number of nodes.
- binlog-do/ignore-db - Do not use.
- binlog_format - ROW is required for Galera.
- innodb_autoinc_lock_mode - 2
- innodb_doublewrite - ON: When an IST occurs, want there to be no torn pages? (With FusionIO or other drives that guarantee atomicity, OFF is better.)
- innodb_flush_log_at_trx_commit - 2 or 0. IST or SST will recover from loss if you have 1.

- query_cache_size - 0
- query_cache_type - 0: The Query cache cannot be used in a Galera context.
- wsrep_auto_increment_control - Normally want ON
- wsrep_on - ON
- wsrep_provider_options - Various settings may need tuning if you are using a WAN.
- wsrep_slave_threads - use for parallel replication
- wsrep_sync_wait (previously wsrep_causal_reads) - used transiently to dealing with "critical reads".

Miscellany

Until recently, FOREIGN KEYs were buggy.

LOAD DATA is auto-chunked. That is, it is passed to other nodes piecemeal, not all at once.

MariaDB's known issues with Galera

DROP USER may not replicate?

A slight difference in ROLLBACK for conflict: InnoDB rolls back smaller transaction; Galera rolls back last.

Slide Deck for Galera

SET GLOBAL wsrep_debug = 1; leads to a lot of debug info in the error log.

Large UPDATEs / DELETEs should be broken up. This admonition is valid for all databases, but there are additional issues in Galera.

WAN: May need to increase (from the defaults) wsrep_provider_options = evs...

MySQL/Perona 5.6 or MariaDB 10 is recommended when going to Galera.

Cluster limitations Slide show

GTIDs

See Using MariaDB GTIDs with MariaDB Galera Cluster.

How many nodes to have in a cluster

If all the servers are in the same 'vulnerability zone' -- eg, rack or data center -- Have an odd number (at least 3) of nodes.

When spanning colos, you need 3 (or more) data centers in order to be 'always' up, even during a colo failure. With only 2 data centers, Galera can automatically recover from one colo outage, but not the other. (You pick which.)

If you use 3 or 4 colos, these number of nodes per colo are safe:

- 3 nodes: 1+1+1 (1 node in each of 3 colos)
- 4 nodes: 1+1+1+1 (4 nodes won't work in 3 colos)
- 5 nodes: 2+2+1, 2+1+1+1 (5 nodes spread 'evenly' across the colos)
- 6 nodes: 2+2+2, 2+2+1+1
- 7 nodes: 3+2+2, 3+3+1, 2+2+2+1, 3+2+1+1 There may be a way to "weight" the nodes differently; that would allow a few more configurations. With "weighting", give each colo the same weight; then subdivide the weight within each colo evenly. Four nodes in 3 colos: (1/6+1/6) + 1/3 + 1/3 That way, any single colo failure cannot lead to "split brain".

Postlog

Posted 2013; VARIABLES: 2015; Refreshed Feb. 2016

3.2.6 Getting Started with MariaDB Galera Cluster

Galera 3.

For convenience, packages containing these libraries are included in the MariaDB YUM and APT repositories.

Currently, MariaDB Galera Cluster only supports the InnoDB storage engine (although there is experimental support for MyISAM and, from MariaDB 10.6, Aria).

A great resource for Galera users is the mailing list run by the developers at Codership. It can be found at Codership on Google Groups. If you use Galera, then it is recommended you subscribe.

Contents

Galera Cluster Support in MariaDB Server

MariaDB Galera Cluster is powered by:

- MariaDB Server.
- The MySQL-wsrep patch for MySQL Server and MariaDB Server developed by Codership. The patch currently supports only Unix-like operating systems.
- The Galera wsrep provider library.

In MariaDB 10.1 and later, the MySQL-wsrep patch has been merged into MariaDB Server. This means that the functionality of MariaDB Galera Cluster can be obtained by installing the standard MariaDB Server packages and the Galera wsrep provider library package. The following Galera version corresponds to each MariaDB Server version:

- In MariaDB 10.4 and later, MariaDB Galera Cluster uses Galera 4. This means that the MySQL-wsrep patch is version 26 and the Galera wsrep provider library is version 4.
- In MariaDB 10.3 and before, MariaDB Galera Cluster uses Galera 3. This means that the MySQL-wsrep patch is version 25 and the Galera wsrep provider library is version 3.

See Deciphering Galera Version Numbers for more information about how to interpret these version numbers.

See What is MariaDB Galera Cluster?: Galera Versions for more information about which specific Galera version is included in each release of MariaDB Server.

In supported builds, Galera Cluster functionality can be enabled by setting some configuration options that are mentioned below. Galera Cluster functionality is not enabled in a standard MariaDB Server installation unless explicitly enabled with these configuration options.

Prerequisites

Swap Size Requirements

During normal operation a MariaDB Galera node does not consume much more memory than a regular MariaDB server. Additional memory is consumed for the certification index and uncommitted writesets, but normally this should not be noticeable in a typical application. There is one exception though:

1. **Writeset caching during state transfer.** When a node is receiving a state transfer it cannot process and apply incoming writesets because it has no state to apply them to yet. Depending on a state transfer mechanism (e.g. mysqldump) the node that sends the state transfer may not be able to apply writesets as well. Thus they need to cache those writesets for a catch-up phase. Currently the writesets are cached in memory and, if the system runs out of memory either the state transfer will fail or the cluster will block waiting for the state transfer to end.

To control memory usage for writeset caching, check the Galera parameters: `gcs.recv_q_hard_limit`, `gcs.recv_q_soft_limit`, and `gcs.max_throttle`.

Limitations

Before using MariaDB Galera Cluster, we would recommend reading through the known limitations, so you can be sure that it is appropriate for your application.

Installing MariaDB Galera Cluster

To use MariaDB Galera Cluster, there are two primary packages that you need to install:

1. A MariaDB Server version that supports Galera Cluster.
2. The Galera wsrep provider library.

As mentioned in the previous section, in MariaDB 10.1 and above, Galera Cluster support is actually included in the standard MariaDB Server packages. That means that installing MariaDB Galera Cluster package is the same as installing standard MariaDB Server package in those versions. However, you will also have to install an additional package to obtain the Galera wsrep provider library.

Some SST methods may also require additional packages to be installed. The `mariabackup` SST method is generally the best option for large clusters that expect a lot of load.

Installing MariaDB Galera Cluster with a Package Manager

MariaDB Galera Cluster can be installed via a package manager on Linux. In order to do so, your system needs to be configured to install from one of the MariaDB repositories.

You can configure your package manager to install it from MariaDB Corporation's MariaDB Package Repository by using the MariaDB Package Repository setup script.

You can also configure your package manager to install it from MariaDB Foundation's MariaDB Repository by using the MariaDB Repository Configuration Tool.

Installing MariaDB Galera Cluster with yum/dnf

On RHEL, CentOS, Fedora, and other similar Linux distributions, it is highly recommended to install the relevant RPM packages from MariaDB's repository using `yum` or `dnf`. Starting with RHEL 8 and Fedora 22, `yum` has been replaced by `dnf`, which is the next major version of `yum`. However, `yum` commands still work on many systems that use `dnf`.

To install MariaDB Galera Cluster with `yum` or `dnf`, follow the instructions at Installing MariaDB Galera Cluster with yum.

Installing MariaDB Galera Cluster with apt-get

On Debian, Ubuntu, and other similar Linux distributions, it is highly recommended to install the relevant DEB packages from MariaDB's repository using `apt-get`.

To install MariaDB Galera Cluster with `apt-get`, follow the instructions at Installing MariaDB Galera Cluster with apt-get.

Installing MariaDB Galera Cluster with zypper

On SLES, OpenSUSE, and other similar Linux distributions, it is highly recommended to install the relevant RPM packages

from MariaDB's repository using `zypper`.

To install MariaDB Galera Cluster with `zypper`, follow the instructions at Installing MariaDB Galera Cluster with ZYpp.

Installing MariaDB Galera Cluster with a Binary Tarball

To install MariaDB Galera Cluster with a binary tarball, follow the instructions at Installing MariaDB Binary Tarballs.

MariaDB Galera Cluster starting with 10.0.24

To make the location of the `libgalera_smm.so` library in binary tarballs more similar to its location in other packages, the library is now found at `lib/galera/libgalera_smm.so` in the binary tarballs, and there is a symbolic link in the `lib` directory that points to it.

Installing MariaDB Galera Cluster from Source

To install MariaDB Galera Cluster by compiling it from source, you will have to compile both MariaDB Server and the Galera wsrep provider library. For some information on how to do this, see the pages at Installing Galera From Source. The pages at Compiling MariaDB From Source and Galera Cluster Documentation: Building Galera Cluster for MySQL may also be helpful.

Configuring MariaDB Galera Cluster

A number of options need to be set in order for Galera Cluster to work when using MariaDB. See Configuring MariaDB Galera Cluster for more information.

Bootstrapping a New Cluster

To first node of a new cluster needs to be bootstrapped by starting mariadbd on that node with the option `--wsrep-new-cluster` option. This option tells the node that there is no existing cluster to connect to. The node will create a new UUID to identify the new cluster.

Do not use the --wsrep-new-cluster option when connecting to an existing cluster. Restarting the node with this option set will cause the node to create new UUID to identify the cluster again, and the node won't reconnect to the old cluster. See the next section about how to reconnect to an existing cluster.

For example, if you were manually starting mariadbd on a node, then you could bootstrap it by executing the following:

```
$ mariadbd --wsrep-new-cluster
```

However, keep in mind that most users are not going to be starting mariadbd manually. Instead, most users will use a service manager to start mariadbd. See the following sections on how to bootstrap a node with the most common service managers.

Systemd and Bootstrapping

On operating systems that use systemd, a node can be bootstrapped in the following way:

```
$ galera_new_cluster
```

This wrapper uses systemd to run mariadbd with the --wsrep-new-cluster option.

If you are using the systemd service that supports the systemd service's method for interacting with multiple MariaDB Server processes, then you can bootstrap a specific instance by specifying the instance name as a suffix. For example:

```
$ galera_new_cluster mariadb@node1
```

Systemd support and the galera_new_cluster script were added in MariaDB 10.1.

SysVinit and Bootstrapping

On operating systems that use sysVinit, a node can be bootstrapped in the following way:

```
$ service mysql bootstrap
```

This runs mariadbd with the `--wsrep-new-cluster` option.

Adding Another Node to a Cluster

Once you have a cluster running and you want to add/reconnect another node to it, you must supply an address of one or more of the existing cluster members in the `wsrep_cluster_address` option. For example, if the first node of the cluster has the address 192.168.0.1, then you could add a second node to the cluster by setting the following option in a server option group in an option file:

```
[mariadb]
...
wsrep_cluster_address=gcomm://192.168.0.1  # DNS names work as well, IP is preferred for performa
```

The new node only needs to connect to one of the existing cluster nodes. Once it connects to one of the existing cluster nodes, it will be able to see all of the nodes in the cluster. However, it is generally better to list all nodes of the cluster in `wsrep_cluster_address`, so that any node can join a cluster by connecting to any of the other cluster nodes, even if one or more of the cluster nodes are down. It is even OK to list a node's own IP address in `wsrep_cluster_address`, since Galera Cluster is smart enough to ignore it.

Once all members agree on the membership, the cluster's state will be exchanged. If the new node's state is different from that of the cluster, then it will request an IST or SST to make itself consistent with the other nodes.

Restarting the Cluster

If you shut down all nodes at the same time, then you have effectively terminated the cluster. Of course, the cluster's data still exists, but the running cluster no longer exists. When this happens, you'll need to bootstrap the cluster again.

If the cluster is not bootstrapped and mariadbd on the first node is just started normally, then the node willl try to connect to at least one of the nodes listed in the `wsrep_cluster_address` option. If no nodes are currently running, then this will fail. Bootstrapping the first node solves this problem.

Determining the Most Advanced Node

In some cases Galera will refuse to bootstrap a node if it detects that it might not be the most advanced node in the cluster. Galera makes this determination if the node was not the last one in the cluster to be shut down or if the node crashed. In those cases, manual intervention is needed.

If you know for sure which node is the most advanced you can edit the `grastate.dat` file in the `datadir`. You can set `safe_to_bootstrap=1` on the most advanced node.

You can determine which node is the most advanced by checking `grastate.dat` on each node and looking for the node with the highest `seqno`. If the node crashed and `seqno=-1`, then you can find the most advanced node by recovering the `seqno` on each node with the `wsrep_recover` option. For example:

```
$ mariadbd --wsrep_recover
```

Systemd and Galera Recovery

On operating systems that use `systemd`, the position of a node can be recovered by running the `galera_recovery` script. For example:

```
$ galera_recovery
```

If you are using the `systemd` service that supports the systemd service's method for interacting with multiple MariaDB Server processes, then you can recover the position of a specific instance by specifying the instance name as a suffix. For example:

```
$ galera_recovery mariadb@node1
```

The `galera_recovery` script recovers the position of a node by running mariadbd with the `wsrep_recover` option.

When the `galera_recovery` script runs mariadbd, it does not write to the error log. Instead, it redirects mariadbd log output to a file named with the format `/tmp/wsrep_recovery.XXXXXX`, where `XXXXXX` is replaced with random characters.

When Galera is enabled, MariaDB's systemd service automatically runs the `galera_recovery` script prior to starting MariaDB, so that MariaDB starts with the proper Galera position.

Support for systemd and the `galera_recovery` script were added in MariaDB 10.1.

State Snapshot Transfers (SSTs)

In a State Snapshot Transfer (SST), the cluster provisions nodes by transferring a full data copy from one node to another. When a new node joins the cluster, the new node initiates a State Snapshot Transfer to synchronize its data with a node that is already part of the cluster.

See Introduction to State Snapshot Transfers (SSTs) for more information.

Incremental State Transfers (ISTs)

In an Incremental State Transfer (SST), the cluster provisions nodes by transferring a node's missing writesets from one node to another. When a new node joins the cluster, the new node initiates a Incremental State Transfer to synchronize its data with a node that is already part of the cluster.

If a node has only been out of a cluster for a little while, then an IST is generally faster than an SST.

Data at Rest Encryption

In MariaDB 10.1 and above, MariaDB Galera Cluster supports Data at Rest Encryption. See SSTs and Data at Rest Encryption for some disclaimers on how SSTs are affected when encryption is configured.

Some data still cannot be encrypted:

- The disk-based Galera gcache is not encrypted (MDEV-8072).

Monitoring

Status Variables

Galera Cluster's status variables can be queried with the standard `SHOW STATUS` command. For example:

```
SHOW GLOBAL STATUS LIKE 'wsrep_%';
```

Cluster Change Notifications

The cluster nodes can be configured to invoke a command when cluster membership or node status changes. This mechanism can also be used to communicate the event to some external monitoring agent. This is configured by setting `wsrep_notify_cmd`. See Galera Cluster documentation: Notification Command for more information.

3.2.7 Configuring MariaDB Galera Cluster

Contents

A number of options need to be set in order for Galera Cluster to work when using MariaDB. These should be set in the MariaDB option file.

Mandatory Options

Several options are mandatory, which means that they *must* be set in order for Galera Cluster to be enabled or to work properly with MariaDB. The mandatory options are:

- `wsrep_provider` — Path to the Galera library

- `wsrep_cluster_address` — See Galera Cluster address format and usage
- `binlog_format=ROW` — See Binary Log Formats
- `wsrep_on=ON` — Enable wsrep replication
- `default_storage_engine=InnoDB` — This is the default value, or alternately `wsrep_replicate_myisam=1` (before MariaDB 10.6) or `galera-cluster-system-variables/#wsrep_mode=REPLICATE_ARIA,REPLICATE_MYISAM` (MariaDB 10.6 and later)
- `innodb_doublewrite=1` — This is the default value, and should not be changed.

Performance-related Options

These are optional optimizations that can be made to improve performance.

- `innodb_flush_log_at_trx_commit=0` — This is not usually recommended in the case of standard MariaDB. However, it is a bit safer with Galera Cluster, since inconsistencies can always be fixed by recovering from another node.

Writing Replicated Write Sets to the Binary Log

Like with MariaDB replication, write sets that are received by a node with Galera Cluster's certification-based replication are not written to the binary log by default. If you would like a node to write its replicated write sets to the binary log, then you will have to set `log_slave_updates=ON`. This is especially helpful if the node is a replication master. See Using MariaDB Replication with MariaDB Galera Cluster: Configuring a Cluster Node as a Replication Master.

Replication Filters

Like with MariaDB replication, replication filters can be used to filter write sets from being replicated by Galera Cluster's certification-based replication. However, they should be used with caution because they may not work as you'd expect.

The following replication filters are honored for InnoDB DML, but not DDL:

- `binlog_do_db`
- `binlog_ignore_db`
- `replicate_wild_do_table`
- `replicate_wild_ignore_table`

The following replication filters are honored for DML and DDL for tables that use both the InnoDB and MyISAM storage engines:

- `replicate_do_table`
- `replicate_ignore_table`

However, it should be kept in mind that if replication filters cause inconsistencies that lead to replication errors, then nodes may abort.

See also MDEV-421 and MDEV-6229.

Network Ports

Galera Cluster needs access to the following ports:

- **Standard MariaDB Port** (default: 3306) - For MySQL client connections and State Snapshot Transfers that use the `mysqldump` method. This can be changed by setting `port`.
- **Galera Replication Port** (default: 4567) - For Galera Cluster replication traffic, multicast replication uses both UDP transport and TCP on this port. Can be changed by setting `wsrep_node_address`.
- **Galera Replication Listening Interface** (default: `0.0.0.0:4567`) needs to be set using gmcast.listen_addr, either
  - in wsrep_provider_options: `wsrep_provider_options='gmcast.listen_addr=tcp://<IP_ADDR>:<PORT>;'`
  - or in wsrep_cluster_address
- **IST Port** (default: 4568) - For Incremental State Transfers. Can be changed by setting `ist.recv_addr` in `wsrep_provider_options`.
- **SST Port** (default: 4444) - For all State Snapshot Transfer methods other than `mysqldump`. Can be changed by setting `wsrep_sst_receive_address`.

Mutiple Galera Cluster Instances on One Server

If you want to run multiple Galera Cluster instances on one server, then you can do so by starting each instance with

`mysqld_multi`, or if you are using systemd, then you can use the relevant systemd method for interacting with multiple MariaDB instances.

You need to ensure that each instance is configured with a different `datadir`.

You also need to ensure that each instance is configured with different network ports.

3.2.8 State Snapshot Transfers (SSTs) in Galera Cluster

Introduction to State Snapshot Transfers (SSTs)
In an SST, the cluster provisions nodes by transferring a full data copy from one node to another.

mariabackup SST Method
The mariabackup SST method uses the Mariabackup utility for performing SSTs.

Manual SST of Galera Cluster Node With Mariabackup
It can be helpful to perform a "manual SST" with Mariabackup when Galera's normal SSTs fail.

xtrabackup-v2 SST Method
The xtrabackup-v2 SST method uses the Percona XtraBackup utility for performing SSTs.

Manual SST of Galera Cluster Node With Percona XtraBackup
It can be helpful to perform a "manual SST" with Xtrabackup when Galera's normal SSTs fail.

3.2.8.1 Introduction to State Snapshot Transfers (SSTs)

Contents

In a State Snapshot Transfer (SST), the cluster provisions nodes by transferring a full data copy from one node to another. When a new node joins the cluster, the new node initiates a State Snapshot Transfer to synchronize its data with a node that is already part of the cluster.

Types of SSTs

There are two conceptually different ways to transfer a state from one MariaDB server to another:

1. **Logical**

The only SST method of this type is the `mysqldump` SST method, which actually uses the `mysqldump` utility to get a logical dump of the donor. This SST method requires the joiner node to be fully initialized and ready to accept connections before the transfer. This method is, by definition, blocking, in that it blocks the donor node from modifying its own state for the duration of the transfer. It is also the slowest of all, and that might be an issue in a cluster with a lot of load.

2. **Physical**

SST methods of this type physically copy the data files from the donor node to the joiner node. This requires that the joiner node is initialized after the transfer. The `mariabackup` SST method and a few other SST methods fall into this category.

These SST methods are much faster than the `mysqldump` SST method, but they have certain limitations. For example, they can be used only on server startup and the joiner node must be configured very similarly to the donor node (e.g. innodb_file_per_table should be the same and so on). Some of the SST methods in this category are non-blocking on the donor node, meaning that the donor node is still able to process queries while donating the SST (e.g. the `mariabackup` SST method is non-blocking).

SST Methods

SST methods are supported via a scriptable interface. New SST methods could potentially be developed by creating new SST scripts. The scripts usually have names of the form `wsrep_sst_<method>` where `<method>` is one of the SST methods listed below.

You can choose your SST method by setting the `wsrep_sst_method` system variable. It can be changed dynamically with `SET GLOBAL` on the node that you intend to be a SST donor. For example:

```
SET GLOBAL wsrep_sst_method='mariabackup';
```

It can also be set in a server option group in an option file prior to starting up a node:

```
[mariadb]
...
wsrep_sst_method = mariabackup
```

For an SST to work properly, the donor and joiner node must use the same SST method. Therefore, it is recommended to set `wsrep_sst_method` to the same value on all nodes, since any node will usually be a donor or joiner node at some point.

MariaDB Galera Cluster comes with the following built-in SST methods:

mariabackup

This SST method uses the Mariabackup utility for performing SSTs. It is one of the two non-locking methods. This is the recommended SST method if you require the ability to run queries on the donor node during the SST. Note that if you use the `mariabackup` SST method, then you also need to have `socat` installed on the server. This is needed to stream the backup from the donor to the joiner. This is a limitation inherited from the `xtrabackup-v2` SST method.

This SST method supports GTID.

This SST method supports Data at Rest Encryption.

This SST method is available from MariaDB 10.1.26 and MariaDB 10.2.10.

With this SST method, it is impossible to upgrade the cluster between some major versions; see MDEV-27437.

See mariabackup SST method for more information.

rsync / rsync_wan

`rsync` is the default method. This method uses the `rsync` utility to create a snapshot of the donor node. `rsync` should be available by default on all modern Linux distributions. The donor node is blocked with a read lock during the SST. This is the fastest SST method, especially for large datasets since it copies binary data. Because of that, this is the recommended SST method if you do not need to allow the donor node to execute queries during the SST.

The `rsync` method runs `rsync` in `--whole-file` mode, assuming that nodes are connected by fast local network links so that the default delta transfer mode would consume more processing time than it may save on data transfer bandwidth. When having a distributed cluster with slow links between nodes, the `rsync_wan` method runs `rsync` in the default delta transfer mode, which may reduce data transfer time substantially when an older datadir state is already present on the joiner node. Both methods are actually implemented by the same script, `wsrep_sst_rsync_wan` is just a symlink to the `wsrep_sst_rsync` script and the actual `rsync` mode to use is determined by the name the script was called by.

This SST method supports GTID.

This SST method supports Data at Rest Encryption.

The rsync SST method does not support tables created with the DATA DIRECTORY or INDEX DIRECTORY clause. Use the mariabackup SST method as an alternative to support this feature.

Use of this SST method **could result in data corruption** when using innodb_use_native_aio (the default) if the donor is older than MariaDB 10.3.35, MariaDB 10.4.25, MariaDB 10.5.16, MariaDB 10.6.8, or MariaDB 10.7.4; see MDEV-

25975. Starting with those donor versions, `wsrep_sst_method=rsync` is a reliable way to upgrade the cluster to a newer major version.

As of MariaDB 10.1.36, MariaDB 10.2.18, and MariaDB 10.3.10, `stunnel` can be used to encrypt data over the wire. Be sure to have `stunnel` installed. You will also need to generate certificates and keys. See the stunnel documentation for information on how to do that. Once you have the keys, you will need to add the `tkey` and `tcert` options to the `[sst]` option group in your MariaDB configuration file, such as:

```
[sst]
tkey = /etc/my.cnf.d/certificates/client-key.pem
tcert = /etc/my.cnf.d/certificates/client-cert.pem
```

You also need to run the certificate directory through `openssl rehash`.

mysqldump

This SST method runs mysqldump on the donor node and pipes the output to the mariadb client connected to the joiner node. The `mysqldump` SST method needs a username/password pair set in the wsrep_sst_auth variable in order to get the dump. The donor node is blocked with a read lock during the SST. This is the slowest SST method.

This SST method supports GTID.

This SST method supports Data at Rest Encryption.

xtrabackup-v2

In MariaDB 10.1 and later, Mariabackup is the recommended backup method to use instead of Percona XtraBackup.

In MariaDB 10.3, Percona XtraBackup is **not supported**. See Percona XtraBackup Overview: Compatibility with MariaDB for more information.

In MariaDB 10.2 and MariaDB 10.1, Percona XtraBackup is only **partially supported**. See Percona XtraBackup Overview: Compatibility with MariaDB for more information.

This SST method uses the Percona XtraBackup utility for performing SSTs. It is one of the two non-blocking methods. Note that if you use the `xtrabackup-v2` SST method, you also need to have `socat` installed on the server. Since Percona XtraBackup is a third party product, this SST method requires an additional installation some additional configuration. Please refer to Percona's xtrabackup SST documentation for information from the vendor.

This SST method does **not** support GTID.

This SST method does **not** support Data at Rest Encryption.

This SST method is available from MariaDB Galera Cluster 5.5.37 and MariaDB Galera Cluster 10.0.10.

See xtrabackup-v2 SST method for more information.

xtrabackup

In MariaDB 10.1 and later, Mariabackup is the recommended backup method to use instead of Percona XtraBackup.

In MariaDB 10.3, Percona XtraBackup is **not supported**. See Percona XtraBackup Overview: Compatibility with MariaDB for more information.

In MariaDB 10.2 and MariaDB 10.1, Percona XtraBackup is only **partially supported**. See Percona XtraBackup Overview: Compatibility with MariaDB for more information.

This SST method is an older SST method that uses the Percona XtraBackup utility for performing SSTs. The `xtrabackup-v2` SST method should be used instead of the `xtrabackup` SST method starting from MariaDB 5.5.33.

This SST method does **not** support GTID.

This SST method does **not** support Data at Rest Encryption.

Authentication

All SST methods except `rsync` require authentication via username and password. You can tell the client what username and password to use by setting the `wsrep_sst_auth` system variable. It can be changed dynamically with `SET GLOBAL` on the node that you intend to be a SST donor. For example:

```
SET GLOBAL wsrep_sst_auth = 'mariabackup:password';
```

It can also be set in a server option group in an option file prior to starting up a node:

```
[mariadb]
...
wsrep_sst_auth = mariabackup:password
```

Some authentication plugins do not require a password. For example, the `unix_socket` and `gssapi` authentication plugins do not require a password. If you are using a user account that does not require a password in order to log in, then you can just leave the password component of `wsrep_sst_auth` empty. For example:

```
[mariadb]
...
wsrep_sst_auth = mariabackup:
```

See the relevant description or page for each SST method to find out what privileges need to be granted to the user and whether the privileges are needed on the donor node or joiner node for that method.

SSTs and Systemd

MariaDB's `systemd` unit file has a default startup timeout of about 90 seconds on most systems. If an SST takes longer than this default startup timeout on a joiner node, then `systemd` will assume that `mysqld` has failed to startup, which causes `systemd` to kill the `mysqld` process on the joiner node. To work around this, you can reconfigure the MariaDB `systemd` unit to have an infinite timeout, such as by executing one of the following commands:

If you are using `systemd` 228 or older, then you can execute the following to set an infinite timeout:

```
sudo tee /etc/systemd/system/mariadb.service.d/timeoutstartsec.conf <<EOF
[Service]

TimeoutStartSec=0
EOF
sudo systemctl daemon-reload
```

Systemd 229 added the infinity option, so if you are using `systemd` 229 or later, then you can execute the following to set an infinite timeout:

```
sudo tee /etc/systemd/system/mariadb.service.d/timeoutstartsec.conf <<EOF
[Service]

TimeoutStartSec=infinity
EOF
sudo systemctl daemon-reload
```

See Configuring the Systemd Service Timeout for more details.

Note that systemd 236 added the `EXTEND_TIMEOUT_USEC` environment variable that allows services to extend the startup timeout during long-running processes. Starting with MariaDB 10.1.35, MariaDB 10.2.17, and MariaDB 10.3.8, on systems with systemd versions that support it, MariaDB uses this feature to extend the startup timeout during long SSTs. Therefore, if you are using `systemd` 236 or later, then you should not need to manually override `TimeoutStartSec`, even if your SSTs run for longer than the configured value. See MDEV-15607 for more information.

SST Failure

An SST failure generally renders the joiner node unusable. Therefore, when an SST failure is detected, the joiner node will abort.

Restarting a node after a `mysqldump` SST failure may require manual restoration of the administrative tables.

SSTs and Data at Rest Encryption

Look at the description of each SST method to determine which methods support Data at Rest Encryption.

For logical SST methods like `mysqldump`, each node should be able to have different encryption keys. For physical SST methods, all nodes need to have the same encryption keys, since the donor node will copy encrypted data files to the joiner node, and the joiner node will need to be able to decrypt them.

Minimal Cluster Size

In order to avoid a split-brain condition, the minimum recommended number of nodes in a cluster is 3.

When using an SST method that blocks the donor, there is yet another reason to require a minimum of 3 nodes. In a 3-node cluster, if one node is acting as an SST joiner and one other node is acting as an SST donor, then there is still one more node to continue executing queries.

Manual SSTs

In some cases, if Galera Cluster's automatic SSTs repeatedly fail, then it can be helpful to perform a "manual SST". See the following pages on how to do that:

- Manual SST of Galera Cluster node with Mariabackup
- Manual SST of Galera Cluster node with Percona XtraBackup

Known Issues

mysqld_multi

SST scripts can't currently read the `[mysqldN]` option groups in option files that are read by instances managed by `mysqld_multi`.

See MDEV-18863 for more information.

2.3.4.13 mariabackup SST Method

3.2.8.3 Manual SST of Galera Cluster Node With Mariabackup

Contents

Sometimes it can be helpful to perform a "manual SST" when Galera's normal SSTs fail. This can be especially useful when the cluster's `datadir` is very large, since a normal SST can take a long time to fail in that case.

A manual SST essentially consists of taking a backup of the donor, loading the backup on the joiner, and then manually editing the cluster state on the joiner node. This page will show how to perform this process with Mariabackup.

Process

- Check that the donor and joiner nodes have the same Mariabackup version.

```
mariabackup --version
```

- Create backup directory on donor.

```
MYSQL_BACKUP_DIR=/mysql_backup
mkdir $MYSQL_BACKUP_DIR
```

- Take a full backup the of the donor node with `mariabackup`. The `--galera-info` option should also be provided, so that the node's cluster state is also backed up.

```
DB_USER=sstuser
DB_USER_PASS=password
mariabackup --backup  --galera-info \
   --target-dir=$MYSQL_BACKUP_DIR \
   --user=$DB_USER \
   --password=$DB_USER_PASS
```

- Verify that the MariaDB Server process is stopped on the joiner node. This will depend on your service manager.

For example, on systemd systems, you can execute::

```
systemctl status mariadb
```

- Create the backup directory on the joiner node.

```
MYSQL_BACKUP_DIR=/mysql_backup
mkdir $MYSQL_BACKUP_DIR
```

- Copy the backup from the donor node to the joiner node.

```
OS_USER=dba
JOINER_HOST=dbserver2.mariadb.com
rsync -av $MYSQL_BACKUP_DIR/* ${OS_USER}@${JOINER_HOST}:${MYSQL_BACKUP_DIR}
```

- Prepare the backup on the joiner node.

```
mariabackup --prepare \
   --target-dir=$MYSQL_BACKUP_DIR
```

- Get the Galera Cluster version ID from the donor node's `grastate.dat` file.

```
MYSQL_DATADIR=/var/lib/mysql
cat $MYSQL_DATADIR/grastate.dat | grep version
```

For example, a very common version number is "2.1".

- Get the node's cluster state from the `xtrabackup_galera_info` file in the backup that was copied to the joiner node.

```
cat $MYSQL_BACKUP_DIR/xtrabackup_galera_info
```

The file contains the values of the `wsrep_local_state_uuid` and `wsrep_last_committed` status variables.

The values are written in the following format:

```
wsrep_local_state_uuid:wsrep_last_committed
```

For example:

```
d38587ce-246c-11e5-bcce-6bbd0831cc0f:1352215
```

- Create a `grastate.dat` file in the backup directory of the joiner node. The Galera Cluster version ID, the cluster uuid, and the seqno from previous steps will be used to fill in the relevant fields.

For example, with the example values from the last two steps, we could do:

```
sudo tee $MYSQL_BACKUP_DIR/grastate.dat <<EOF
# GALERA saved state
version: 2.1
uuid:    d38587ce-246c-11e5-bcce-6bbd0831cc0f
seqno:   1352215
safe_to_bootstrap: 0
EOF
```

- Remove the existing contents of the `datadir` on the joiner node.

```
MYSQL_DATADIR=/var/lib/mysql
rm -Rf $MYSQL_DATADIR/*
```

- Copy the contents of the backup directory to the `datadir` the on joiner node.

```
mariabackup --copy-back \
   --target-dir=$MYSQL_BACKUP_DIR
```

- Make sure the permissions of the `datadir` are correct on the joiner node.

```
chown -R mysql:mysql $MYSQL_DATADIR/
```

- Start the MariaDB Server process on the joiner node. This will depend on your service manager.

For example, on systemd systems, you can execute::

```
systemctl start mariadb
```

- Watch the MariaDB error log on the joiner node and verify that the node does not need to perform a normal SSTs due to the manual SST.

```
tail -f /var/log/mysql/mysqld.log
```

3.2.8.4 xtrabackup-v2 SST Method

Contents

In MariaDB 10.1 and later, Mariabackup is the recommended backup method to use instead of Percona XtraBackup.

In MariaDB 10.3, Percona XtraBackup is **not supported**. See Percona XtraBackup Overview: Compatibility with MariaDB for more information.

In MariaDB 10.2 and MariaDB 10.1, Percona XtraBackup is only **partially supported**. See Percona XtraBackup Overview: Compatibility with MariaDB for more information.

The `xtrabackup-v2` SST method uses the Percona XtraBackup utility for performing SSTs. It is one of the methods that does not block the donor node.

Note that if you use the `xtrabackup-v2` SST method, then you also need to have socat installed on the server. This is needed to stream the backup from the donor node to the joiner node.

Since Percona XtraBackup is a third party product, it may require additional installation and additional configuration. Please refer to Percona's xtrabackup SST documentation for information from the vendor.

Choosing Percona XtraBackup for SSTs

To use the `xtrabackup-v2` SST method, you must set the `wsrep_sst_method=xtrabackup-v2` on both the donor and joiner node. It can be changed dynamically with `SET GLOBAL` on the node that you intend to be a SST donor. For example:

```
SET GLOBAL wsrep_sst_method='xtrabackup-v2';
```

It can be set in a server option group in an option file prior to starting up a node:

```
[mariadb]
...
wsrep_sst_method = xtrabackup-v2
```

For an SST to work properly, the donor and joiner node must use the same SST method. Therefore, it is recommended to set `wsrep_sst_method` to the same value on all nodes, since any node will usually be a donor or joiner node at some point.

Authentication and Privileges

To use the `xtrabackup-v2` SST method, Percona XtraBackup needs to be able to authenticate locally on the donor node, so that it can create a backup to stream to the joiner. You can tell the donor node what username and password to use by setting the `wsrep_sst_auth` system variable. It can be changed dynamically with `SET GLOBAL` on the node that you intend to be a SST donor. For example:

```
SET GLOBAL wsrep_sst_auth = 'xtrabackup:mypassword';
```

It can also be set in a server option group in an option file prior to starting up a node:

```
[mariadb]
...
wsrep_sst_auth = xtrabackup:mypassword
```

Some authentication plugins do not require a password. For example, the `unix_socket` and `gssapi` authentication plugins do not require a password. If you are using a user account that does not require a password in order to log in, then you can just leave the password component of `wsrep_sst_auth` empty. For example:

```
[mariadb]
...
wsrep_sst_auth = xtrabackup:
```

The user account that performs the backup for the SST needs to have the same privileges as Percona XtraBackup, which are the `RELOAD` , `PROCESS` , `LOCK TABLES` and `REPLICATION CLIENT` global privileges. To be safe, you should ensure that these privileges are set on each node in your cluster. Percona XtraBackup connects locally on the donor node to perform the backup, so the following user should be sufficient:

```
CREATE USER 'xtrabackup'@'localhost' IDENTIFIED BY 'mypassword';
GRANT RELOAD, PROCESS, LOCK TABLES, REPLICATION CLIENT ON *.* TO 'xtrabackup'@'localhost';
```

Passwordless Authentication - Unix Socket

It is possible to use the `unix_socket` authentication plugin for the user account that performs SSTs. This would provide the benefit of not needing to configure a plain-text password in `wsrep_sst_auth` .

The user account would have to have the same name as the operating system user account that is running the `mysqld` process. On many systems, this is the user account configured as the `user` option, and it tends to default to `mysql`.

For example, if the `unix_socket` authentication plugin is already installed, then you could execute the following to create the user account:

```
CREATE USER 'mysql'@'localhost' IDENTIFIED VIA unix_socket;
GRANT RELOAD, PROCESS, LOCK TABLES, REPLICATION CLIENT ON *.* TO 'mysql'@'localhost';
```

And then to configure `wsrep_sst_auth`, you could set the following in a server option group in an option file prior to starting up a node:

```
[mariadb]
...
wsrep_sst_auth = mysql:
```

Passwordless Authentication - GSSAPI

It is possible to use the `gssapi` authentication plugin for the user account that performs SSTs. This would provide the benefit of not needing to configure a plain-text password in `wsrep_sst_auth`.

The following steps would need to be done beforehand:

- You need a KDC running MIT Kerberos or Microsoft Active Directory.
- You will need to create a keytab file for the MariaDB server.
- You will need to install the package containing the `gssapi` authentication plugin.
- You will need to install the plugin in MariaDB, so that the `gssapi` authentication plugin is available to use.
- You will need to configure the plugin.
- You will need to create a user account that authenticates with the `gssapi` authentication plugin, so that the user account can be used for SSTs. This user account will need to correspond with a user account that exists on the backend KDC.

For example, you could execute the following to create the user account in MariaDB:

```
CREATE USER 'xtrabackup'@'localhost' IDENTIFIED VIA gssapi;
GRANT RELOAD, PROCESS, LOCK TABLES, REPLICATION CLIENT ON *.* TO 'xtrabackup'@'localhost';
```

And then to configure `wsrep_sst_auth`, you could set the following in a server option group in an option file prior to starting up a node:

```
[mariadb]
...
wsrep_sst_auth = xtrabackup:
```

Choosing a Donor Node

When Percona XtraBackup is used to create the backup for the SST on the donor node, XtraBackup briefly requires a system-wide lock at the end of the backup. This is done with `FLUSH TABLES WITH READ LOCK`.

If a specific node in your cluster is acting as the *primary* node by receiving all of the application's write traffic, then this node should not usually be used as the donor node, because the system-wide lock could interfere with the application. In this case, you can define one or more preferred donor nodes by setting the `wsrep_sst_donor` system variable.

For example, let's say that we have a 5-node cluster with the nodes `node1`, `node2`, `node3`, `node4`, and `node5`, and let's say that `node1` is acting as the *primary* node.The preferred donor nodes for `node2` could be configured by setting the following in a server option group in an option file prior to starting up a node:

```
[mariadb]
...
wsrep_sst_donor=node3,node4,node5,
```

The trailing comma tells the server to allow any other node as donor when the preferred donors are not available. Therefore, if `node1` is the only node left in the cluster, the trailing comma allows it to be used as the donor node.

Socat Dependency

During the SST process, the donor node uses socat to stream the backup to the joiner node. Then the joiner node prepares the backup before restoring it. The socat utility must be installed on both the donor node and the joiner node in order for this to work. Otherwise, the MariaDB error log will contain an error like:

```
WSREP_SST: [ERROR] socat not found in path:
/usr/sbin:/sbin:/usr//bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin (20180122
14:55:32.993)
```

Installing Socat on RHEL/CentOS

On RHEL/CentOS, `socat` can be installed from the Extra Packages for Enterprise Linux (EPEL) repository.

TLS

This SST method supports three different TLS methods. The specific method can be selected by setting the `encrypt` option in the `[sst]` section of the MariaDB configuration file. The options are:

- TLS using OpenSSL encryption built into `socat` (`encrypt=2`)
- TLS using OpenSSL encryption with Galera-compatible certificates and keys (`encrypt=3`)
- TLS using OpenSSL encryption with MariaDB-compatible certificates and keys (`encrypt=4`)

Note that `encrypt=1` refers to a TLS encryption method that has been deprecated and removed.

TLS Using OpenSSL Encryption Built into Socat

To generate keys compatible with this encryption method, you can follow these directions.

For example:

- First, generate the keys and certificates:

```
FILENAME=sst
openssl genrsa -out $FILENAME.key 1024
openssl req -new -key $FILENAME.key -x509 -days 3653 -out $FILENAME.crt
cat $FILENAME.key $FILENAME.crt >$FILENAME.pem
chmod 600 $FILENAME.key $FILENAME.pem
```

- On some systems, you may also have to add dhparams to the certificate:

```
openssl dhparam -out dhparams.pem 2048
cat dhparams.pem >> sst.pem
```

- Then, copy the certificate and keys to all nodes in the cluster.
- Then, configure the following on all nodes in the cluster:

```
[sst]
encrypt=2
tca=/etc/my.cnf.d/certificates/sst.crt
tcert=/etc/my.cnf.d/certificates/sst.pem
```

But replace the paths with whatever is relevant on your system.

This should allow your SSTs to be encrypted.

TLS Using OpenSSL Encryption with Galera-compatible Certificates and Keys

To generate keys compatible with this encryption method, you can follow these directions.

For example:

- First, generate the keys and certificates:

```
# CA
openssl genrsa 2048 > ca-key.pem
openssl req -new -x509 -nodes -days 365000 \
-key ca-key.pem -out ca-cert.pem

# server1
openssl req -newkey rsa:2048 -days 365000 \
-nodes -keyout server1-key.pem -out server1-req.pem
openssl rsa -in server1-key.pem -out server1-key.pem
openssl x509 -req -in server1-req.pem -days 365000 \
-CA ca-cert.pem -CAkey ca-key.pem -set_serial 01 \
-out server1-cert.pem
```

- Then, copy the certificate and keys to all nodes in the cluster.
- Then, configure the following on all nodes in the cluster:

```
[sst]
encrypt=3
tkey=/etc/my.cnf.d/certificates/server1-key.pem
tcert=/etc/my.cnf.d/certificates/server1-cert.pem
```

But replace the paths with whatever is relevant on your system.

This should allow your SSTs to be encrypted.

TLS Using OpenSSL Encryption with MariaDB-compatible Certificates and Keys

To generate keys compatible with this encryption method, you can follow these directions.

For example:

- First, generate the keys and certificates:

```
# CA
openssl genrsa 2048 > ca-key.pem
openssl req -new -x509 -nodes -days 365000 \
-key ca-key.pem -out ca-cert.pem

# server1
openssl req -newkey rsa:2048 -days 365000 \
-nodes -keyout server1-key.pem -out server1-req.pem
openssl rsa -in server1-key.pem -out server1-key.pem
openssl x509 -req -in server1-req.pem -days 365000 \
-CA ca-cert.pem -CAkey ca-key.pem -set_serial 01 \
-out server1-cert.pem
```

- Then, copy the certificate and keys to all nodes in the cluster.
- Then, configure the following on all nodes in the cluster:

```
[sst]
encrypt=4
ssl-ca=/etc/my.cnf.d/certificates/ca-cert.pem
ssl-cert=/etc/my.cnf.d/certificates/server1-cert.pem
ssl-key=/etc/my.cnf.d/certificates/server1-key.pem
```

But replace the paths with whatever is relevant on your system.

This should allow your SSTs to be encrypted.

Logs

The `xtrabackup-v2` SST method has its own logging outside of the MariaDB Server logging.

Logging to SST Logs

By default, on the donor node, it logs to `innobackup.backup.log`. This log file is located in the `datadir`.

By default, on the joiner node, it logs to `innobackup.prepare.log` and `innobackup.move.log`. These log files are located in the `.sst` directory, which is a hidden directory inside the `datadir`.

These log files are overwritten by each subsequent SST, so if an SST fails, it is best to copy them somewhere safe before starting another SST, so that the log files can be analyzed. See MDEV-17973 about that.

Logging to Syslog

You can redirect the SST logs to the syslog instead by setting the following in the `[sst]` option group in an option file:

```
[sst]
sst-syslog=1
```

You can also redirect the SST logs to the syslog by setting the following in the `[mysqld_safe]` option group in an option file:

```
[mysqld_safe]
syslog
```

Performing SSTs with IPv6 Addresses

If you are performing Percona XtraBackup SSTs with IPv6 addresses, then the `socat` utility needs to be passed the `pf=ip6` option. This can be done by setting the `sockopt` option in the `[sst]` option group in an option file. For example:

```
[sst]
sockopt=",pf=ip6"
```

See MDEV-18797 for more information.

Manual SST with Percona XtraBackup

In some cases, if Galera Cluster's automatic SSTs repeatedly fail, then it can be helpful to perform a "manual SST". See the following page on how to do that:

- Manual SST of Galera Cluster node with Percona XtraBackup

3.2.8.5 Manual SST of Galera Cluster Node With Percona XtraBackup

Contents

Mariabackup should be used instead of XtraBackup on all supported releases. See manual SST with Mariabackup.

In MariaDB 10.1 and later, Mariabackup is the recommended backup method to use instead of Percona XtraBackup.

In MariaDB 10.3, Percona XtraBackup is **not supported**. See Percona XtraBackup Overview: Compatibility with MariaDB for more information.

In MariaDB 10.2 and MariaDB 10.1, Percona XtraBackup is only **partially supported**. See Percona XtraBackup Overview: Compatibility with MariaDB for more information.

Sometimes it can be helpful to perform a "manual SST" when Galera's normal SSTs fail. This can be especially useful when the cluster's `datadir` is very large, since a normal SST can take a long time to fail in that case.

A manual SST essentially consists of taking a backup of the donor, loading the backup on the joiner, and then manually editing the cluster state on the joiner node. This page will show how to perform this process with Percona XtraBackup.

Process

- Check that the donor and joiner nodes have the same XtraBackup version.

```
innobackupex --version
```

- Create backup directory on donor.

```
MYSQL_BACKUP_DIR=/mysql_backup
mkdir $MYSQL_BACKUP_DIR
```

- Take a full backup the of the donor node with `innobackupex` . The `--galera-info` option should also be provided, so that the node's cluster state is also backed up.

```
DB_USER=sstuser
DB_USER_PASS=password
innobackupex --user=$DB_USER --password=$DB_USER_PASS --galera-info --no-timestamp $MYSQL_BACKUP_
```

- Verify that the MariaDB Server process is stopped on the joiner node. This will depend on your service manager.

For example, on systemd systems, you can execute::

```
systemctl status mariadb
```

- Create the backup directory on the joiner node.

```
MYSQL_BACKUP_DIR=/mysql_backup
mkdir $MYSQL_BACKUP_DIR
```

- Copy the backup from the donor node to the joiner node.

```
OS_USER=dba
JOINER_HOST=dbserver2.mariadb.com
rsync -av $MYSQL_BACKUP_DIR/* ${OS_USER}@${JOINER_HOST}:${MYSQL_BACKUP_DIR}
```

- Prepare the backup on the joiner node.

```
innobackupex --apply-log $MYSQL_BACKUP_DIR
```

- Get the Galera Cluster version ID from the donor node's `grastate.dat` file.

```
MYSQL_DATADIR=/var/lib/mysql
cat $MYSQL_DATADIR/grastate.dat | grep version
```

For example, a very common version number is "2.1".

- Get the node's cluster state from the `--xtrabackup_galera_info` file in the backup that was copied to the joiner node.

```
cat $MYSQL_BACKUP_DIR/xtrabackup_galera_info
```

Example output:

```
d38587ce-246c-11e5-bcce-6bbd0831cc0f:1352215
```

This output is in the format:

```
uuid:seqno
```

- Create a `grastate.dat` file in the backup directory of the joiner node. The Galera Cluster version ID, the cluster uuid, and the seqno from previous steps will be used to fill in the relevant fields.

For example, with the example values from the last two steps, we could do:

```
sudo tee $MYSQL_BACKUP_DIR/grastate.dat <<EOF
# GALERA saved state
version: 2.1
uuid:    d38587ce-246c-11e5-bcce-6bbd0831cc0f
seqno:   1352215
safe_to_bootstrap: 0
EOF
```

- Remove the existing contents of the `datadir` on the joiner node.

```
MYSQL_DATADIR=/var/lib/mysql
rm -Rf $MYSQL_DATADIR/*
```

- Copy the contents of the backup directory to the `datadir` the on joiner node.

```
cp -R $MYSQL_BACKUP_DIR/* $MYSQL_DATADIR/
```

- Make sure the permissions of the `datadir` are correct on the joiner node.

```
chown -R mysql:mysql $MYSQL_DATADIR/
```

- Start the MariaDB Server process on the joiner node. This will depend on your service manager.

For example, on systemd systems, you can execute::

```
systemctl start mariadb
```

- Watch the MariaDB error log on the joiner node and verify that the node does not need to perform a normal SSTs due to the manual SST.

```
tail -f /var/log/mysql/mysqld.log
```

3.2.9 Galera Cluster Status Variables

Contents

Viewing Galera Cluster Status Variables

Galera status variables can be viewed with the SHOW STATUS statement.

```
SHOW STATUS LIKE 'wsrep%';
```

See also the Full list of MariaDB options, system and status variables.

List of Galera Cluster status variables

MariaDB Galera Cluster has the following status variables:

wsrep_applier_thread_count

- **Description:** Stores current number of applier threads to make clear how many slave threads of this type there are.
- **Introduced:** MariaDB 10.2.26, MariaDB 10.3.17, MariaDB 10.4.7

---

wsrep_apply_oooe

- **Description:** How often writesets have been applied out of order, an indicators of parallelization efficiency.

---

wsrep_apply_oool

- **Description:** How often writesets with a higher sequence number were applied before ones with a lower sequence number, implying slow writesets.

---

wsrep_apply_window

- **Description:** Average distance between highest and lowest concurrently applied seqno.

---

wsrep_cert_deps_distance

- **Description:** Average distance between the highest and the lowest sequence numbers that can possibly be applied in parallel, or the potential degree of parallelization.

---

wsrep_cert_index_size

- **Description:** The number of entries in the certification index.

---

wsrep_cert_interval

- **Description:** Average number of transactions received while a transaction replicates.

---

wsrep_cluster_capabilities

- **Description:**

---

wsrep_cluster_conf_id

- **Description:** Total number of cluster membership changes that have taken place.

---

wsrep_cluster_size

- **Description:** Number of nodes currently in the cluster.

wsrep_cluster_state_uuid

- **Description:** UUID state of the cluster. If it matches the value in wsrep_local_state_uuid, the local and cluster nodes are in sync.

wsrep_cluster_status

- **Description:** Cluster component status. Possible values are `PRIMARY` (primary group configuration, quorum present), `NON_PRIMARY` (non-primary group configuration, quorum lost) or `DISCONNECTED` (not connected to group, retrying).

wsrep_cluster_weight

- **Description:** The total weight of the current members in the cluster. The value is counted as a sum of of pc.weight of the nodes in the current Primary Component.

wsrep_commit_oooe

- **Description:** How often a transaction was committed out of order.

wsrep_commit_oool

- **Description:** No meaning.

wsrep_commit_window

- **Description:** Average distance between highest and lowest concurrently committed seqno.

wsrep_connected

- **Description:** Whether or not MariaDB is connected to the wsrep provider. Possible values are `ON` or `OFF`.

wsrep_desync_count

- **Description:** Returns the number of operations in progress that require the node to temporarily desync from the cluster.

wsrep_evs_delayed

- **Description:** Provides a comma separated list of all the nodes this node has registered on its delayed list.

wsrep_evs_evict_list

- **Description:** Lists the UUID's of all nodes evicted from the cluster. Evicted nodes cannot rejoin the cluster until you restart their mysqld processes.

wsrep_evs_repl_latency

- **Description:** This status variable provides figures for the replication latency on group communication. It measures latency (in seconds) from the time point when a message is sent out to the time point when a message is received. As replication is a group operation, this essentially gives you the slowest ACK and longest RTT in the cluster. Format is min/avg/max/stddev

`wsrep_evs_state`

- **Description:** Shows the internal state of the EVS Protocol.

`wsrep_flow_control_paused`

- **Description:** The fraction of time since the last FLUSH STATUS command that replication was paused due to flow control.

`wsrep_flow_control_paused_ns`

- **Description:** The total time spent in a paused state measured in nanoseconds.

`wsrep_flow_control_recv`

- **Description:** Number of FC_PAUSE events received as well as sent since the most recent status query.

`wsrep_flow_control_sent`

- **Description:** Number of FC_PAUSE events sent since the most recent status query

---

`wsrep_gcomm_uuid`

- **Description:** The UUID assigned to the node.

`wsrep_incoming_addresses`

- **Description:** Comma-separated list of incoming server addresses in the cluster component.

`wsrep_last_committed`

- **Description:** Sequence number of the most recently committed transaction.

`wsrep_local_bf_aborts`

- **Description:** Total number of local transactions aborted by slave transactions while being executed

---

`wsrep_local_cached_downto`

- **Description:** The lowest sequence number, or seqno, in the write-set cache (GCache).

`wsrep_local_cert_failures`

- **Description:** Total number of local transactions that failed the certification test.

`wsrep_local_commits`

- **Description:** Total number of local transactions committed on the node.

`wsrep_local_index`

- **Description:** The node's index in the cluster. The index is zero-based.

`wsrep_local_recv_queue`

- **Description:** Current length of the receive queue, which is the number of writesets waiting to be applied.

`wsrep_local_recv_queue_avg`

- **Description:** Average length of the receive queue since the most recent status query. If this value is noticeably larger than zero, the node is likely to be overloaded, and cannot apply the writesets as quickly as they arrive, resulting in replication throttling.

`wsrep_local_recv_queue_max`

- **Description:** The maximum length of the recv queue since the last FLUSH STATUS command.

`wsrep_local_recv_queue_min`

- **Description:** The minimum length of the recv queue since the last FLUSH STATUS command.

`wsrep_local_replays`

- **Description:** Total number of transaction replays due to asymmetric lock granularity.

`wsrep_local_send_queue`

- **Description:** Current length of the send queue, which is the number of writesets waiting to be sent.

`wsrep_local_send_queue_avg`

- **Description:** Average length of the send queue since the most recent status query. If this value is noticeably larger than zero, there is most likely network throughput or replication throttling issues.

`wsrep_local_send_queue_max`

- **Description:** The maximum length of the send queue since the last FLUSH STATUS command.

`wsrep_local_send_queue_min`

- **Description:** The minimum length of the send queue since the last FLUSH STATUS command.

`wsrep_local_state`

- **Description:** Internal Galera Cluster FSM state number.

`wsrep_local_state_comment`

- **Description:** Human-readable explanation of the state.

`wsrep_local_state_uuid`

- **Description:** The node's UUID state. If it matches the value in wsrep_cluster_state_uuid, the local and cluster nodes are in sync.

`wsrep_open_connections`

- **Description:** The number of open connection objects inside the wsrep provider.

`wsrep_open_transactions`

- **Description:** The number of locally running transactions which have been registered inside the wsrep provider. This means transactions which have made operations which have caused write set population to happen. Transactions which are read only are not counted.

`wsrep_protocol_version`

- **Description:** The wsrep protocol version being used.

`wsrep_provider_name`

- **Description:** The name of the provider. The default is "Galera".

`wsrep_provider_vendor`

- **Description:** The vendor string.

`wsrep_provider_version`

- **Description:** The version number of the Galera wsrep provider

---

`wsrep_ready`

- **Description:** Whether or not the Galera wsrep provider is ready. Possible values are `ON` or `OFF`

`wsrep_received`

- **Description:** Total number of writesets received from other nodes.

`wsrep_received_bytes`

- **Description:** Total size in bytes of all writesets received from other nodes.

`wsrep_repl_data_bytes`

- **Description:** Total size of data replicated.

`wsrep_repl_keys`

- **Description:** Total number of keys replicated.

`wsrep_repl_keys_bytes`

- **Description:** Total size of keys replicated.

`wsrep_repl_other_bytes`

- **Description:** Total size of other bits replicated.

`wsrep_replicated`

- **Description:** Total number of writesets replicated to other nodes.

`wsrep_replicated_bytes`

- **Description:** Total size in bytes of all writesets replicated to other nodes.

`wsrep_rollbacker_thread_count`

- **Description:** Stores current number of rollbacker threads to make clear how many slave threads of this type there are.
- **Introduced:** MariaDB 10.2.26, MariaDB 10.3.17, MariaDB 10.4.7

`wsrep_thread_count`

- **Description:** Total number of wsrep (applier/rollbacker) threads.

3.2.10 Galera Cluster System Variables

Contents

This page documents system variables related to Galera Cluster. For options that are not system variables, see Galera Options.

See Server System Variables for a complete list of system variables and instructions on setting them.

Also see the Full list of MariaDB options, system and status variables.

`wsrep_allowlist`

- **Description:** Allowed IP addresses, comma delimited.
- **Commandline:** `--wsrep-allowlist=value1[,value2...]`
- **Scope:** Global

- **Dynamic:** No
- **Data Type:** String
- **Default Value:** None
- **Introduced:** MariaDB 10.10

---

wsrep_auto_increment_control

- **Description:** If set to `1` (the default), will automatically adjust the auto_increment_increment and auto_increment_offset variables according to the size of the cluster, and when the cluster size changes. This avoids replication conflicts due to auto_increment. In a primary-replica environment, can be set to `OFF`.
- **Commandline:** `--wsrep-auto-increment-control[={0|1}]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** Boolean
- **Default Value:** `ON`

---

wsrep_causal_reads

- **Description:** If set to `ON` (`OFF` is default), enforces read-committed characteristics across the cluster. In the case that a primary applies an event more quickly than a replica, the two could briefly be out-of-sync. With this variable set to `ON`, the replica will wait for the event to be applied before processing further queries. Setting to `ON` also results in larger read latencies. Deprecated by wsrep_sync_wait=1.
- **Commandline:** `--wsrep-causal-reads[={0|1}]`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** Boolean
- **Default Value:** `OFF`
- **Deprecated:** MariaDB 10.1.3
- **Removed:** MariaDB 11.3.0

---

wsrep_certification_rules

- **Description:** Certification rules to use in the cluster. Possible values are:
  - `strict`: Stricter rules that could result in more certification failures. For example with foreign keys, certification failure could result if different nodes receive non-conflicting insertions at about the same time that point to the same row in a parent table
  - `optimized`: relaxed rules that allow more concurrency and cause less certification failures.
- **Commandline:** `--wsrep-certifcation-rules`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** Enumeration
- **Default Value:** `strict`
- **Valid Values:** `strict`, `optimized`
- **Introduced:** MariaDB 10.4.3, MariaDB 10.3.13, MariaDB 10.2.22, MariaDB 10.1.38

---

wsrep_certify_nonPK

- **Description:** When set to `ON` (the default), Galera will still certify transactions for tables with no primary key. However, this can still cause undefined behavior in some circumstances. It is recommended to define primary keys for every InnoDB table when using Galera.
- **Commandline:** `--wsrep-certify-nonPK[={0|1}]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** Boolean
- **Default Value:** `ON`

---

wsrep_cluster_address

- **Description:** The addresses of cluster nodes to connect to when starting up.
  - Good practice is to specify all possible cluster nodes, in the form `gcomm://<node1 or ip:port>,<node2 or`

ip2:port>,<node3 or ip3:port>.
    - Specifying an empty ip (`gcomm://`) will cause the node to start a new cluster (which should not be done in the my.cnf file, as after each restart the server will not rejoin the current cluster).
    - The variable can be changed at runtime in some configurations, and will result in the node closing the connection to any current cluster, and connecting to the new address.
    - If specifying a port, note that this is the Galera port, not the MariaDB port.
    - For example:
        - `gcomm://192.168.0.1,192.168.0.2,192.168.0.3`
        - `gcomm://192.168.0.1:1234,192.168.0.2:1234,192.168.0.3:1234?gmcast.listen_addr=tcp://0.0.0.0:1234`
    - See also gmcast.listen_addr
- **Commandline:** `--wsrep-cluster-address=value`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** String

---

wsrep_cluster_name

- **Description:** The name of the cluster. Nodes cannot connect to clusters with a different name, so needs to be identical on all nodes in the same cluster. The variable can be set dynamically, but note that doing so may be unsafe and cause an outage, and that the wsrep provider is unloaded and loaded.
- **Commandline:** `--wsrep-cluster-name=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** String
- **Default Value:** `my_wsrep_cluster`

---

wsrep_convert_LOCK_to_trx

- **Description:** Converts LOCK/UNLOCK TABLES statements to BEGIN and COMMIT. Used mainly for getting older applications to work with a multi-primary setup, use carefully, as can result in extremely large writesets.
- **Commandline:** `--wsrep-convert-LOCK-to-trx[={0|1}]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** Boolean
- **Default Value:** `OFF`

---

wsrep_data_home_dir

- **Description:** Directory where wsrep provider will store its internal files.
- **Commandline:** `--wsrep-data-home-dir=value`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** String
- **Default Value:** The datadir variable value.

---

wsrep_dbug_option

- **Description:** Unused. The mechanism to pass the DBUG options to the wsrep provider hasn't been implemented.
- **Commandline:** `--wsrep-dbug-option=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** String

---

wsrep_debug

- **Description:** WSREP debug level logging. Before MariaDB 10.4.3 was a boolean, which when set to `ON` (`OFF` was default), ensured debug messages would be logged to the error log as well. Before MariaDB 10.6.1, DDL logging was only logged on the originating node. From MariaDB 10.6.1, it is logged on other nodes as well.
- **Commandline:** `--wsrep-debug[={0|1}]`

- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** Enumeration (>= MariaDB 10.4.3), Boolean (<= MariaDB 10.4.2)
- **Default Value:** `NONE` (>= MariaDB 10.4.3), `OFF` (<= MariaDB 10.4.2)
- **Valid Values**: (>= MariaDB 10.4.3) `NONE`, `SERVER`, `TRANSACTION`, `STREAMING`, `CLIENT`

---

wsrep_desync

- **Description:** When a node receives more write-sets than it can apply, the transactions are placed in a received queue. If the node's received queue has too many write-sets waiting to be applied (as defined by the `gcs.fc_limit` WSREP provider option), then the node would usually engage Flow Control. However, when this option is set to `ON`, Flow Control will be disabled for the desynced node. The desynced node works through the received queue until it reaches a more manageable size. The desynced node continues to receive write-sets from the other nodes in the cluster. The other nodes in the cluster do not wait for the desynced node to catch up, so the desynced node can fall even further behind the other nodes in the cluster. You can check if a node is desynced by checking if the `wsrep_local_state_comment` status variable is equal to `Donor/Desynced`.
- **Commandline:** `--wsrep-desync[={0|1}]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** Boolean
- **Default Value:** `OFF`

---

wsrep_dirty_reads

- **Description:** By default, when not synchronized with the group (wsrep_ready=OFF) a node will reject all queries other than SET and SHOW. If `wsrep_dirty_reads` is set to `1`, queries which do not change data, like SELECT queries (dirty reads), creating of prepare statement, etc. will be accepted by the node.
- **Commandline:** `--wsrep-dirty-reads[={0|1}]`
- **Scope:** Global,Session
- **Dynamic:** Yes
- **Data Type:** Boolean
- **Default Value:** `OFF`
- **Valid Values:** `ON`, `OFF`

---

wsrep_drupal_282555_workaround

- **Description:** If set to `ON`, a workaround for Drupal/MySQL/InnoDB bug #282555 is enabled. This is a bug where, in some cases, when inserting a `DEFAULT` value into an AUTO_INCREMENT column, a duplicate key error may be returned.
- **Commandline:** `--wsrep-drupal-282555-workaround[={0|1}]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** Boolean
- **Default Value:** `OFF`

---

wsrep_forced_binlog_format

- **Description:** A binary log format that will override any session binlog format settings.
- **Commandline:** `--wsrep-forced-binlog-format=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Default Value:** `NONE`
- **Data Type:** Enum
- **Valid Values:** `STATEMENT`, `ROW`, `MIXED` or `NONE` (which resets the forced binlog format state).

---

wsrep_gtid_domain_id

- **Description:** This system variable defines the GTID domain ID that is used for wsrep GTID mode.
  - When `wsrep_gtid_mode` is set to `ON`, `wsrep_gtid_domain_id` is used in place of `gtid_domain_id` for all Galera Cluster write sets.

    - When `wsrep_gtid_mode` is set to `OFF`, `wsrep_gtid_domain_id` is simply ignored to allow for backward compatibility.
    - There are some additional requirements that need to be met in order for this mode to generate consistent GTIDs. For more information, see Using MariaDB GTIDs with MariaDB Galera Cluster.
- **Commandline:** `--wsrep-gtid-domain-id=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `4294967295`

---

wsrep_gtid_mode

- **Description:** Wsrep GTID mode attempts to keep GTIDs consistent for Galera Cluster write sets on all cluster nodes. GTID state is initially copied to a joiner node during an SST. If you are planning to use Galera Cluster with MariaDB replication, then wsrep GTID mode can be helpful.
    - When `wsrep_gtid_mode` is set to `ON`, `wsrep_gtid_domain_id` is used in place of `gtid_domain_id` for all Galera Cluster write sets.
    - When `wsrep_gtid_mode` is set to `OFF`, `wsrep_gtid_domain_id` is simply ignored to allow for backward compatibility.
    - There are some additional requirements that need to be met in order for this mode to generate consistent GTIDs. For more information, see Using MariaDB GTIDs with MariaDB Galera Cluster.
- **Commandline:** `--wsrep-gtid-mode[={0|1}]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

wsrep_gtid_seq_no

- **Description:** Internal server usage, manually set WSREP GTID seqno.
- **Commandline:** None
- **Scope:** Session only
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Range:** `0` to `18446744073709551615`
- **Introduced:** MariaDB 10.5.1

---

wsrep_ignore_apply_errors

- **Description:** Bitmask determining whether errors are ignored, or reported back to the provider.
    - 0: No errors are skipped.
    - 1: Ignore some DDL errors (DROP DATABASE, DROP TABLE, DROP INDEX, ALTER TABLE).
    - 2: Skip DML errors (Only ignores DELETE errors).
    - 4: Ignore all DDL errors.
- **Commandline:** `--wsrep-ignore-apply-errors`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** Numeric
- **Default Value:** `7`
- **Range:** `0` to `7`
- **Introduced:** MariaDB 10.4.2

---

wsrep_load_data_splitting

- **Description:** If set to `ON` (the default for MariaDB 10.4.2 and before), LOAD DATA INFILE supports big data files by introducing transaction splitting. The setting has been deprecated in Galera 4, and defaults to `OFF` from MariaDB 10.4.3.
- **Commandline:** `--wsrep-load-data-splitting[={0|1}]`
- **Scope:** Global
- **Dynamic:** Yes

- **Data Type:** Boolean
- **Default Value:** `OFF` (>= MariaDB 10.4.3), `ON` (<= MariaDB 10.4.2)

---

wsrep_log_conflicts

- **Description:** If set to `ON` (`OFF` is default), details of conflicting MDL as well as InnoDB locks in the cluster will be logged.
- **Commandline:** `--wsrep-log-conflicts[={0|1}]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** Boolean
- **Default Value:** `OFF`

---

wsrep_max_ws_rows

- **Description:** Maximum permitted number of rows per writeset. Before MariaDB Galera 10.0.27 and MariaDB 10.1.17 this variable was ignored internally and had no effect on the node. From MariaDB Galera 10.0.27 and MariaDB 10.1.17 support for this variable has been added and in order to be backward compatible the default value has been changed to `0` , which essentially allows writesets to be any size.
- **Commandline:** `--wsrep-max-ws-rows=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** Numeric
- **Default Value:**
  - `0` (>= MariaDB Galera 10.0.27, MariaDB 10.1.17)
  - `131072` (<= MariaDB Galera 10.0.26, MariaDB 10.1.16)
- **Range:** `0` to `1048576`

---

wsrep_max_ws_size

- **Description:** Maximum permitted size in bytes per writeset. Writesets exceeding this will be rejected. Note that versions from and before MariaDB 10.1.17 and MariaDB Galera 10.0.27 permitted the maximum to be set beyond 2GB, which was rejected by Galera.
- **Commandline:** `--wsrep-max-ws-size=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** Numeric
- **Default Value:**
  - `2147483647` (2GB, >= MariaDB Galera 10.0.27, MariaDB 10.1.17)
  - `1073741824` (1GB, <= MariaDB Galera 10.0.26, MariaDB 10.1.16)
- **Range:** `1024` to `2147483647`

---

wsrep_mode

- **Description:** Turns on WSREP features which are not part of default behavior.
  - BINLOG_ROW_FORMAT_ONLY: Only ROW binlog format is supported.
  - DISALLOW_LOCAL_GTID: Nodes can have GTIDs for local transactions in a number of scenarios. If DISALLOW_LOCAL_GTID is set, these operations produce an error ERROR HY000: Galera replication not supported. Scenarios include:
    - A DDL statement is executed with wsrep_OSU_method=RSU set.
    - A DML statement writes to a non-InnoDB table.
    - A DML statement writes to an InnoDB table with wsrep_on=OFF set.
  - REPLICATE_ARIA: Whether or not DML updates for Aria tables will be replicated. This functionality is experimental and should not be relied upon in production systems.
  - REPLICATE_MYISAM: Whether or not DML updates for MyISAM tables will be replicated. This functionality is experimental and should not be relied upon in production systems.
  - REQUIRED_PRIMARY_KEY: Table should have PRIMARY KEY defined.
  - STRICT_REPLICATION: Same as the old wsrep_strict_ddl setting.
- **Commandline:** `--wsrep-mode=value`
- **Scope:** Global
- **Dynamic:** Yes

- **Data Type:** Enumeration
- **Default Value:** (Empty)
- **Valid Values:** `BINLOG_ROW_FORMAT_ONLY`, `DISALLOW_LOCAL_GTID`, `REQUIRED_PRIMARY_KEY`, `REPLICATE_ARIA`, `REPLICATE_MYISAM` and `STRICT_REPLICATION`
- **Introduced:** MariaDB 10.6.0

---

wsrep_mysql_replication_bundle

- **Description:** Determines the number of replication events that are grouped together. Experimental implementation aimed to assist with bottlenecks when a single replica faces a large commit time delay. If set to `0` (the default), there is no grouping.
- **Commandline:** `--wsrep-mysql-replication-bundle=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** Numeric
- **Default Value:** `0`
- **Range:** `0` to `1000`

---

wsrep_node_address

- **Description:** Specifies the node's network address, in the format `ip address[:port]`. As of MariaDB 10.1.8, supports IPv6. The default behavior is for the node to pull the address of the first network interface on the system and the default Galera port. This autoguessing can be unreliable, particularly in the following cases:
  - cloud deployments
  - container deployments
  - servers with multiple network interfaces.
  - servers running multiple nodes.
  - network address translation (NAT).
  - clusters with nodes in more than one region.
- **See also** wsrep_provider_options -> gmcast.listen_addr
- **Commandline:** `--wsrep-node-address=value`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** String
- **Default Value:** Primary network address, usually `eth0` with a default port of 4567, or `0.0.0.0` if no IP address.

---

wsrep_node_incoming_address

- **Description:** This is the address from which the node listens for client connections. If an address is not specified or it's set to `AUTO` (default), mysqld uses either --bind-address or --wsrep-node-address, or tries to get one from the list of available network interfaces, in the same order. See also wsrep_provider_options -> gmcast.listen_addr.
- **Commandline:** `--wsrep-node-incoming-address=value`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** String
- **Default Value:** `AUTO`

---

wsrep_node_name

- **Description:** Name of this node. This name can be used in wsrep_sst_donor as a preferred donor. Note that multiple nodes in a cluster can have the same name.
- **Commandline:** `--wsrep-node-name=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** String
- **Default Value:** The server's hostname.

---

wsrep_notify_cmd

- **Description:** Command to be executed each time the node state or the cluster membership changes. Can be used

for raising an alarm, configuring load balancers and so on. See the Codership Notification Script page for more details.

- **Commandline:** `--wsrep-notify-command=value`
- **Scope:** Global
- **Dynamic:**
  - No (>= MariaDB 10.5.9, MariaDB 10.4.18, MariaDB 10.3.28, MariaDB 10.2.37)
  - Yes (<= MariaDB 10.5.8, MariaDB 10.4.17, MariaDB 10.3.27, MariaDB 10.2.36)
- **Data Type:** String
- **Default Value:** Empty

---

wsrep_on

- **Description:** Whether or not wsrep replication is enabled. If the global value is set to `OFF` (the default since MariaDB 10.1), it is not possible to load the provider and join the node in the cluster. If only the session value is set to `OFF`, the operations from that particular session are not replicated in the cluster, but other sessions and applier threads will continue as normal. The session value of the variable does not affect the node's membership and thus, regardless of its value, the node keeps receiving updates from other nodes in the cluster. Before MariaDB 10.1, even though this variable is `ON` by default, its value gets automatically adjusted based on whether mandatory configurations to turn on Galera replication have been specified. Since MariaDB 10.1, it is set to `OFF` by default and must be turned on to enable Galera replication.
- **Commandline:** `--wsrep-on[={0|1}]`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** Boolean
- **Default Value:** `OFF` (>= MariaDB 10.1), `ON` (<= MariaDB Galera Cluster 10.0),
- **Valid Values:** `ON`, `OFF`

---

wsrep_OSU_method

- **Description:** Online schema upgrade method. The default is `TOI`, specifying the setting without the optional parameter will set to `RSU`.
  - `TOI`: Total Order Isolation. In each cluster node, DDL is processed in the same order regarding other transactions, guaranteeing data consistency. However, affected parts of the database will be locked for the whole cluster.
  - `RSU`: Rolling Schema Upgrade. DDL processing is only done locally on the node, and the user needs perform the changes manually on each node. The node is desynced from the rest of the cluster while the processing takes place to avoid the blocking other nodes. Schema changes *must* be backwards compatible in the same way as for ROW based replication to avoid breaking replication when the DDL processing is complete on the single node, and replication recommences.
- **Commandline:** `--wsrep-OSU-method[=value]`
- **Scope:** Global, Session (since MariaDB Galera 10.0.19)
- **Dynamic:** Yes
- **Data Type:** Enum
- **Default Value:** `TOI`
- **Valid Values:** `TOI`, `RSU`

---

wsrep_patch_version

- **Description:** Wsrep patch version, for example `wsrep_25.10`.
- **Commandline:** None
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** String
- **Default Value:** None

---

wsrep_provider

- **Description:** Location of the wsrep library, usually `/usr/lib/libgalera_smm.so` on Debian and Ubuntu, and `/usr/lib64/libgalera_smm.so` on Red Hat/CentOS.
- **Commandline:** `--wsrep-provider=value`
- **Scope:** Global

    - No (>= MariaDB 10.5.9, MariaDB 10.4.18, MariaDB 10.3.28, MariaDB 10.2.37)
    - Yes (<= MariaDB 10.5.8, MariaDB 10.4.17, MariaDB 10.3.27, MariaDB 10.2.36)
- **Data Type:** String
- **Default Value:** None

---

wsrep_provider_options

- **Description:** Semicolon (;) separated list of wsrep options (see wsrep_provider_options).
- **Commandline:** `--wsrep-provider-options=value`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** String
- **Default Value:** Empty

---

wsrep_recover

- **Description:** If set to `ON` when the server starts, the server will recover the sequence number of the most recent write set applied by Galera, and it will be output to `stderr`, which is usually redirected to the error log. At that point, the server will exit. This sequence number can be provided to the `wsrep_start_position` system variable.
- **Commandline:** `--wsrep-recover[={0|1}]`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** Boolean
- **Default Value:** `OFF`

---

wsrep_reject_queries

- **Description:** Variable to set to reject queries from client connections, useful for maintenance. The node continues to apply write-sets, but an `Error 1047: Unknown command` error is generated by a client query.
    - `NONE` - Not set. Queries will be processed as normal.
    - `ALL` - All queries from client connections will be rejected, but existing client connections will be maintained.
    - `ALL_KILL` All queries from client connections will be rejected, and existing client connections, including the current one, will be immediately killed.
- **Commandline:** `--wsrep-reject-queries[=value]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** Enum
- **Default Value:** `NONE`
- **Valid Values:** `NONE`, `ALL`, `ALL_KILL`
- **Introduced:** MariaDB 10.3.6, MariaDB 10.2.14, MariaDB 10.1.32

---

wsrep_replicate_myisam

- **Description:** Whether or not DML updates for MyISAM tables will be replicated. This functionality is still experimental and should not be relied upon in production systems. Deprecated in MariaDB 10.6, and removed in MariaDB 10.7, use wsrep_mode instead.
- **Commandline:** `--wsrep-replicate-myisam[={0|1}]`
- **Scope:** Global
- **Dynamic:** Yes
- **Default Value:** `OFF`
- **Data Type:** Boolean
- **Valid Values:** `ON`, `OFF`
- **Deprecated:** MariaDB 10.6.0
- **Removed:** MariaDB 10.7.0

---

wsrep_restart_slave

- **Description:** If set to ON, the replica is restarted automatically, when node joins back to cluster.
- **Commandline:** `--wsrep-restart-slave[={0|1}]`
- **Scope:** Global

- **Dynamic:** Yes
- **Default Value:** `OFF`
- **Data Type:** Boolean

---

wsrep_retry_autocommit

- **Description:** Number of times autocommited queries will be retried due to cluster-wide conflicts before returning an error to the client. If set to `0`, no retries will be attempted, while a value of `1` (the default) or more specifies the number of retries attempted. Can be useful to assist applications using autocommit to avoid deadlocks.
- **Commandline:** `--wsrep-retry-autocommit=value`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** Numeric
- **Default Value:** `1`
- **Range:** `0` to `10000`

---

wsrep_slave_FK_checks

- **Description:** If set to ON (the default), the applier replica thread performs foreign key constraint checks.
- **Commandline:** `--wsrep-slave-FK-checks[={0|1}]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** Boolean
- **Default Value:** ON

---

wsrep_slave_threads

- **Description:** Number of replica threads used to apply Galera write sets in parallel. The Galera replica threads are able to determine which write sets are safe to apply in parallel. However, if your cluster nodes seem to have frequent consistency problems, then setting the value to `1` will probably fix the problem. See About Galera Replication: Galera Replica Threads for more information.
- **Commandline:** `--wsrep-slave-threads=` #
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** Numeric
- **Default Value:** `1`
- **Range:** `1` to `512`

---

wsrep_slave_UK_checks

- **Description:** If set to ON, the applier replica thread performs secondary index uniqueness checks.
- **Commandline:** `--wsrep-slave-UK-checks[={0|1}]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** Boolean
- **Default Value:** OFF

---

wsrep_sr_store

- **Description:** Storage for streaming replication fragments.
- **Commandline:** `--wsrep-sr-store=val`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** Enum
- **Default Value:** `table`
- **Valid Values:** `table`, `none`
- **Introduced:** MariaDB 10.4.2

---

wsrep_sst_auth

- **Description:** Username and password of the user to use for replication. Unused if wsrep_sst_method is set to `rsync`, while for other methods it should be in the format `<user>:<password>`. The contents are masked in logs and when querying the value with SHOW VARIABLES. See Introduction to State Snapshot Transfers (SSTs) for more information.
- **Commandline:** `--wsrep-sst-auth=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** String
- **Default Value:** (Empty)

---

wsrep_sst_donor

- **Description:** Comma-separated list (from 5.5.33) or name (as per wsrep_node_name) of the servers as donors, or the source of the state transfer, in order of preference. The donor-selection algorithm, in general, prefers a donor capable of transferring only the missing transactions (IST) to the joiner node, instead of the complete state (SST). Thus, it starts by looking for an IST-capable node in the given donor list followed by rest of the nodes in the cluster. In case multiple candidate nodes are found outside the specified donor list, the node in the same segment (gmcast.segment) as the joiner is preferred. If none of the existing nodes in the cluster can serve the missing transactions through IST, the algorithm moves on to look for a suitable node to transfer the entire state (SST). It first looks at the nodes specified in the donor list (irrespective of their segment). If no suitable donor is still found, the rest of the donor nodes are checked for suitability only if the donor list has a "terminating-comma". Note that a stateless node (the Galera arbitrator) can never be a donor. See Introduction to State Snapshot Transfers (SSTs) for more information. [NOTE] Although the variable is dynamic, the node will not use the new value unless the node requiring SST or IST disconnects from the cluster. To force this, set wsrep_cluster_address to an empty string and back to the nodes list. After setting this variable dynamically, on startup the value from the configuration file will be used again.
- **Commandline:** `--wsrep-sst-donor=value`
- **Scope:** Global
- **Dynamic:** Yes (read note above)
- **Data Type:** String
- **Default Value:**

---

wsrep_sst_donor_rejects_queries

- **Description:** If set to `ON` (`OFF` is default), the donor node will reject incoming queries, returning an `UNKNOWN COMMAND` error code. Can be used for informing load balancers that a node is unavailable.
- **Commandline:** `--wsrep-sst-donor-rejects-queries[={0|1}]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** Boolean
- **Default Value:** `OFF`

---

wsrep_sst_method

- **Description:** Method used for taking the state snapshot transfer (SST). See Introduction to State Snapshot Transfers (SSTs): SST Methods for more information.
- **Commandline:** `--wsrep-sst-method=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** String
- **Default Value:** `rsync`
- **Valid Values:** `rsync`, `mysqldump`, `xtrabackup`, `xtrabackup-v2`, `mariabackup`

---

wsrep_sst_receive_address

- **Description:** This is the address where other nodes (donor) in the cluster connect to in order to send the state-transfer updates. If an address is not specified or its set to `AUTO` (default), mysqld uses --wsrep_node_address's value as the receiving address. However, if --wsrep_node_address is not set, it uses address from either --bind-address or tries to get one from the list of available network interfaces, in the same order. Note: setting it to `localhost` will make it impossible for nodes running on other hosts to reach this node. See Introduction to State Snapshot Transfers (SSTs) for more information.
- **Commandline:** `--wsrep-sst-receive-address=value`
- **Scope:** Global

- **Dynamic:** Yes
- **Data Type:** String
- **Default Value:** `AUTO`

---

wsrep_start_position

- **Description:** The start position that the node should use in the format: `UUID:seq_no`. The proper value to use for this position can be recovered with `wsrep_recover`.
- **Commandline:** `--wsrep-start-position=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** String
- **Default Value:** `00000000-0000-0000-0000-000000000000:-1`

---

wsrep_status_file

- **Description:** wsrep status output filename.
- **Commandline:** `--wsrep-status-file=value`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** String
- **Default Value:** None
- **Introduced:** MariaDB 10.9

---

wsrep_strict_ddl

- **Description:** If set, reject DDL statements on affected tables not supporting Galera replication. This is done by checking if the table is InnoDB, which is the only table currently fully supporting Galera replication. MyISAM tables will not trigger the error if the experimental wsrep_replicate_myisam setting is `ON`. If set, should be set on all tables in the cluster. Affected DDL statements include:
  CREATE TABLE (e.g. CREATE TABLE t1(a int) engine=Aria)
  ALTER TABLE
  TRUNCATE TABLE
  CREATE VIEW
  CREATE TRIGGER
  CREATE INDEX
  DROP INDEX
  RENAME TABLE
  DROP TABLE
  Statements in procedures, events, and functions are permitted as the affected tables are only known at execution. Furthermore, the various USER, ROLE, SERVER and DATABASE statements are also allowed as they do not have an affected table. Deprecated in MariaDB 10.6.0 and removed in MariaDB 10.7. Use wsrep_mode=STRICT_REPLICATION instead.
- **Commandline:** `--wsrep-strict-ddl[={0|1}]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB 10.5.1
- **Deprecated:** MariaDB 10.6.0
- **Removed:** MariaDB 10.7.0

---

wsrep_sync_wait

- **Description:** Setting this variable ensures causality checks will take place before executing an operation of the type specified by the value, ensuring that the statement is executed on a fully synced node. While the check is taking place, new queries are blocked on the node to allow the server to catch up with all updates made in the cluster up to the point where the check was begun. Once reached, the original query is executed on the node. This can result in higher latency. Note that when wsrep_dirty_reads is ON, values of wsrep_sync_wait become irrelevant. Sample usage (for a critical read that must have the most up-to-date data) `SET SESSION wsrep_sync_wait=1; SELECT ...; SET SESSION wsrep_sync_wait=0;`

    - `0` - Disabled (default)
    - `1` - READ (SELECT and BEGIN/START TRANSACTION). Up until MariaDB 10.2.8, MariaDB 10.1.26, MariaDB Galera 10.0.31 and MariaDB Galera 5.5.56, also SHOW). This is the same as wsrep_causal_reads=1.
    - `2` - UPDATE and DELETE;
    - `3` - READ, UPDATE and DELETE;
    - `4` - INSERT and REPLACE;
    - `5` - READ, INSERT and REPLACE;
    - `6` - UPDATE, DELETE, INSERT and REPLACE;
    - `7` - READ, UPDATE, DELETE, INSERT and REPLACE;
    - `8` - SHOW (from MariaDB 10.2.9, MariaDB 10.1.27, MariaDB Galera 10.0.32, MariaDB Galera 5.5.57)
    - `9` - READ and SHOW (from MariaDB 10.2.9, MariaDB 10.1.27, MariaDB Galera 10.0.32, MariaDB Galera 5.5.57)
    - `10` - UPDATE, DELETE and SHOW (from MariaDB 10.2.9, MariaDB 10.1.27, MariaDB Galera 10.0.32, MariaDB Galera 5.5.57)
    - `11` - READ, UPDATE, DELETE and SHOW (from MariaDB 10.2.9, MariaDB 10.1.27, MariaDB Galera 10.0.32, MariaDB Galera 5.5.57)
    - `12` - INSERT, REPLACE and SHOW (from MariaDB 10.2.9, MariaDB 10.1.27, MariaDB Galera 10.0.32, MariaDB Galera 5.5.57)
    - `13` - READ, INSERT, REPLACE and SHOW (from MariaDB 10.2.9, MariaDB 10.1.27, MariaDB Galera 10.0.32, MariaDB Galera 5.5.57)
    - `14` - UPDATE, DELETE, INSERT, REPLACE and SHOW (from MariaDB 10.2.9, MariaDB 10.1.27, MariaDB Galera 10.0.32, MariaDB Galera 5.5.57)
    - `15` - READ, UPDATE, DELETE, INSERT, REPLACE and SHOW (from MariaDB 10.2.9, MariaDB 10.1.27, MariaDB Galera 10.0.32, MariaDB Galera 5.5.57)
- **Commandline:** `--wsrep-sync-wait=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** Numeric
- **Default Value:** `0`
- **Range:**
    - `0` to `15`

---

wsrep_trx_fragment_size

- **Description:** Size of transaction fragments for streaming replication (measured in units as specified by wsrep_trx_fragment_unit)
- **Commandline:** `--wsrep-trx-fragment-size=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** numeric
- **Default Value:** `0`
- **Range:** `0` to `2147483647`
- **Introduced:** MariaDB 10.4.2

---

wsrep_trx_fragment_unit

- **Description:** Unit for streaming replication transaction fragments' size:
    - `bytes` : transaction's binlog events buffer size in bytes
    - `rows` : number of rows affected by the transaction
    - `statements` : number of SQL statements executed in the multi-statement transaction
- **Commandline:** `--wsrep-trx-fragment-unit=value`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** enum
- **Default Value:** `bytes`
- **Valid Values:** `bytes`, `rows` or `statements`
- **Introduced:** MariaDB 10.4.2

3.2.11 Building the Galera wsrep Package on Ubuntu and Debian

The instructions on this page were used to create the *galera* package on the Ubuntu and Debian Linux distributions. This package contains the wsrep provider for MariaDB Galera Cluster.

MariaDB Galera Cluster starting with 5.5.35

Starting with MariaDB Galera Cluster 5.5.35, the version of the wsrep provider is **25.3.5**. We also provide **25.2.9** for those that need or want it.

Prior to that, the wsrep version was 23.2.7.

1. Install prerequisites:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get -y install check debhelper libasio-dev libboost-dev libboost-program-options-dev libssl-dev scons
```

2. Clone galera.git from github.com/mariadb and checkout mariadb-3.x banch:

```
git init repo
cd repo
git clone -b mariadb-3.x https://github.com/MariaDB/galera.git
```

3. Build the packages by executing `build.sh` under scripts/ directory with `-p` switch:

```
cd galera
./scripts/build.sh -p
```

When finished, you will have the Debian packages for galera library and arbitrator in the parent directory.

Running galera test suite

If you want to run the `galera` test suite (`mysql-test-run --suite=galera`), you need to install the galera library as either `/usr/lib/galera/libgalera_smm.so` or `/usr/lib64/galera/libgalera_smm.so`

3.2.12 Building the Galera wsrep Package on Fedora

The instructions on this page were used to create the *galera* package on the Fedora Linux distribution. This package contains the wsrep provider for MariaDB Galera Cluster.

The following table lists each version of the Galera 4 wsrep provider, and it lists which version of MariaDB each one was first released in. If you would like to install Galera 4 using yum, apt, or zypper, then the package is called `galera-4` .

Galera Version	Released in MariaDB Version
26.4.14	10.10.3, 10.9.5, 10.8.7, 10.7.8, 10.6.12, 10.5.19, 10.4.28
26.4.13	10.10.2, 10.9.4, 10.8.6, 10.7.7, 10.6.11, 10.5.18, 10.4.27
26.4.12	10.10.1, 10.9.2, 10.8.4, 10.7.5, 10.6.9, 10.5.17, 10.4.26
26.4.11	10.8.1, 10.7.2, 10.6.6, 10.5.14, 10.4.22
26.4.9	10.6.4, 10.5.12, 10.4.21
26.4.8	10.6.1, 10.5.10, 10.4.19
26.4.7	10.5.9, 10.4.18
26.4.6	10.5.7, 10.4.16
26.4.5	10.5.4, 10.4.14
26.4.4	10.5.1, 10.4.13

26.4.3	10.5.0, 10.4.9
26.4.2	10.4.4
26.4.1	10.4.3
26.4.0	10.4.2

The following table lists each version of the Galera 3 wsrep provider, and it lists which version of MariaDB each one was first released in. If you would like to install Galera 3 using yum, apt, or zypper, then the package is called `galera`.

Galera Version	Released in MariaDB Version
25.3.37	MariaDB 10.3.36
25.3.35	MariaDB 10.3.33, MariaDB 10.2.42
25.3.34	MariaDB 10.3.31, MariaDB 10.2.40
25.3.33	MariaDB 10.3.29, MariaDB 10.2.38
25.3.32	MariaDB 10.3.28, MariaDB 10.2.37
25.3.31	MariaDB 10.3.26, MariaDB 10.2.35, MariaDB 10.1.48
25.3.30	MariaDB 10.3.25, MariaDB 10.2.34, MariaDB 10.1.47
25.3.29	MariaDB 10.3.23, MariaDB 10.2.32, MariaDB 10.1.45
25.3.28	MariaDB 10.3.19, MariaDB 10.2.28, MariaDB 10.1.42
25.3.27	MariaDB 10.3.18, MariaDB 10.2.27
25.3.26	MariaDB 10.3.14, MariaDB 10.2.23, MariaDB 10.1.39
25.3.25	MariaDB 10.3.12, MariaDB 10.2.20, MariaDB 10.1.38, MariaDB Galera Cluster 10.0.38, MariaDB Galera Cluster 5.5.63
25.3.24	MariaDB 10.4.0, MariaDB 10.3.10, MariaDB 10.2.18, MariaDB 10.1.37, MariaDB Galera Cluster 10.0.37, MariaDB Galera Cluster 5.5.62
25.3.23	MariaDB 10.3.5, MariaDB 10.2.13, MariaDB 10.1.32, MariaDB Galera Cluster 10.0.35, MariaDB Galera Cluster 5.5.60
25.3.22	MariaDB 10.3.3, MariaDB 10.2.11, MariaDB 10.1.29, MariaDB Galera Cluster 10.0.33, MariaDB Galera Cluster 5.5.59
25.3.21	N/A
25.3.20	MariaDB 10.3.1, MariaDB 10.2.6, MariaDB 10.1.23, MariaDB Galera Cluster 10.0.31, MariaDB Galera Cluster 5.5.56
25.3.19	MariaDB 10.3.0, MariaDB 10.2.3, MariaDB 10.1.20, MariaDB Galera Cluster 10.0.29, MariaDB Galera Cluster 5.5.54
25.3.18	MariaDB 10.2.2, MariaDB 10.1.18, MariaDB Galera Cluster 10.0.28, MariaDB Galera Cluster 5.5.53
25.3.17	MariaDB 10.1.17, MariaDB Galera Cluster 10.0.27, MariaDB Galera Cluster 5.5.51
25.3.16	N/A
25.3.15	MariaDB 10.2.0, MariaDB 10.1.13, MariaDB Galera Cluster 10.0.25, MariaDB Galera Cluster 5.5.49
25.3.14	MariaDB 10.1.12, MariaDB Galera Cluster 10.0.24, MariaDB Galera Cluster 5.5.48
25.3.12	MariaDB 10.1.11
25.3.11	N/A
25.3.10	N/A
25.3.9	MariaDB 10.1.3, MariaDB Galera Cluster 10.0.17, MariaDB Galera Cluster 5.5.42
25.3.8	N/A
25.3.7	N/A
25.3.6	N/A
25.3.5	MariaDB 10.1.1, MariaDB Galera Cluster 10.0.10, MariaDB Galera Cluster 5.5.37

25.3.4	N/A
25.3.3	N/A
25.3.2	MariaDB Galera Cluster 10.0.7, MariaDB Galera Cluster 5.5.35

The following table lists each version of the Galera 2 wsrep provider, and it lists which version of MariaDB each one was first released in.

Galera Version	Released in MariaDB Galera Cluster Version
25.2.9	10.0.10, 5.5.37
25.2.8	10.0.7, 5.5.35
23.2.7	5.5.34

For convenience, a *galera* package containing the **preferred** wsrep provider is included in the MariaDB YUM and APT repositories (the preferred versions are **bolded** in the table above).

See also Deciphering Galera Version Numbers.

1. Install the prerequisites:

```
sudo yum update
sudo yum -y install boost-devel check-devel glibc-devel openssl-devel scons
```

2. Clone galera.git from github.com/mariadb and checkout mariadb-3.x banch:

```
git init repo
cd repo
git clone -b mariadb-3.x https://github.com/MariaDB/galera.git
```

3. Build the packages by executing `build.sh` under scripts/ directory with `-p` switch:

```
cd galera
./scripts/build.sh -p
```

When finished, you will have an RPM package containing galera library, arbitrator and related files in the current directory. Note: The same set of instructions can be applied to other RPM based platforms to generate galera package.

3.2.13 Installing Galera from Source

Contents

There are binary installation packages available for RPM and Debian-based distributions, which will pull in all required Galera dependencies.

If these are not available, you will need to build Galera from source.

MariaDB starting with 10.1

Starting from MariaDB 10.1, the wsrep API for Galera Cluster is included by default. Follow the usual compiling-mariadb-from-source instructions

MariaDB until 10.0

MariaDB 10.0 and below are no longer supported. The instructions below have only historical significance.

Preparation

make cannot manage dependencies for the build process, so the following packages need to be installed first:

RPM-based:

```
yum-builddep MariaDB-server
```

Debian-based:

```
apt-get build-dep mariadb-server
```

If running on an alternative system, or the commands are available, the following packages are required. You will need to check the repositories for the correct package names on your distribution - these may differ between distributions, or require additional packages:

MariaDB Database Server with wsrep API

- Git, CMake (on Fedora, both cmake and cmake-fedora are required), GCC and GCC-C++, Automake, Autoconf, and Bison, as well as development releases of libaio and ncurses.

Building

You can use Git to download the source code, as MariaDB source code is available through GitHub. Clone the repository:

```
git clone https://github.com/mariadb/server mariadb
```

1. Checkout the branch (e.g. 10.0-galera or 5.5-galera), for example:

```
cd mariadb
git checkout 10.0-galera
```

Building the Database Server

The standard and Galera cluster database servers are the same, except that for Galera Cluster, the wsrep API patch is included. Enable the patch with the CMake configuration options `WITH_WSREP` and `WITH_INNODB_DISALLOW_WRITES`. To build the database server, run the following commands:

```
cmake -DWITH_WSREP=ON -DWITH_INNODB_DISALLOW_WRITES=ON .
make
make install
```

There are also some build scripts in the *BUILD/* directory which may be more convenient to use. For example, the following pre-configures the build options discussed above:

```
./BUILD/compile-pentium64-wsrep
```

There are several others as well, so you can select the most convenient.

Besides the server with the Galera support, you will also need a galera provider.

Preparation

make cannot manage dependencies itself, so the following packages need to be installed first:

```
apt-get install -y scons check
```

If running on an alternative system, or the commands are available, the following packages are required. You will need to check the repositories for the correct package names on your distribution - these may differ between distributions, or require additional packages:

Galera Replication Plugin

- SCons, as well as development releases of Boost (libboost_program_options, libboost_headers1), Check and OpenSSL.

Building

Run:

```
git clone -b mariadb-4.x https://github.com/MariaDB/galera.git
```

If you are using MariaDB 10.3 or earlier, you should checkout `mariadb-3.x` instead.

After this, the source files for the Galera provider will be in the `galera` directory.

Building the Galera Provider

The Galera Replication Plugin both implements the wsrep API and operates as the database server's wsrep Provider. To build, cd into the *galera/* directory and do:

```
git submodule init
git submodule update
./scripts/build.sh
mkdir /usr/lib64/galera
cp libgalera_smm.so /usr/lib64/galera
```

The path to `libgalera_smm.so` needs to be defined in the *my.cnf* configuration file.

> Building Galera Replication Plugin from source on FreeBSD runs into issues due to Linux dependencies. To overcome these, either install the binary package: `pkg install galera`, or use the ports build available at `/usr/ports/databases/galera`.

Configuration

After building, a number of other steps are necessary:

- Create the database server user and group:

```
groupadd mysql
useradd -g mysql mysql
```

- Install the database (the path may be different if you specified CMAKE_INSTALL_PREFIX):

```
cd /usr/local/mysql
./scripts/mariadb-install-db --user=mysql
```

- If you want to install the database in a location other than */usr/local/mysql/data*, use the *--basedir* or *--datadir* options.
- Change the user and group permissions for the base directory.

```
chown -R mysql /usr/local/mysql
chgrp -R mysql /usr/local/mysql
```

- Create a system unit for the database server.

```
cp /usr/local/mysql/supported-files/mysql.server /etc/init.d/mysql
chmod +x /etc/init.d/mysql
chkconfig --add mysql
```

- Galera Cluster can now be started using the service command, and is set to start at boot.

3.2.14 Galera Test Repositories

To facilitate development and QA, we have created some test repositories for the Galera wsrep provider.

> These are **test** repositories. There will be periods when they do not work at all, or work incorrectly, or possibly cause earthquakes, typhoons, and tornadoes. You have been warned.

Galera Test Repositories for YUM

Replace `${dist}`
in the code below for the YUM-based distribution you are testing. Valid distributions are:

- centos5-amd64
- centos5-x86
- centos6-amd64
- centos6-x86
- centos7-amd64
- rhel5-amd64
- rhel5-x86
- rhel6-amd64
- rhel6-x86
- rhel6-ppc64
- rhel7-amd64
- rhel7-ppc64
- rhel7-ppc64le
- fedora22-amd64
- fedora22-x86
- fedora23-amd64
- fedora23-x86
- fedora24-amd64
- fedora24-x86
- opensuse13-amd64
- opensuse13-x86
- sles11-amd64
- sles11-x86
- sles12-amd64
- sles12-ppc64le

```
# Place this code block in a file at /etc/yum.repos.d/galera.repo
[galera-test]
name = galera-test
baseurl = http://yum.mariadb.org/galera/repo/rpm/${dist}
gpgkey=https://yum.mariadb.org/RPM-GPG-KEY-MariaDB
gpgcheck=1
```

Galera Test Repositories for APT

Replace `${dist}`
in the code below for the APT-based distribution you are testing. Valid ones are:

- wheezy
- jessie
- sid
- precise
- trusty
- xenial

```
# run the following command:
sudo apt-key adv --recv-keys --keyserver keyserver.ubuntu.com 0xcbcb082a1bb943db 0xF1656F24C74CD1

# Add the following line to your /etc/apt/sources.list file:
deb http://yum.mariadb.org/galera/repo/deb ${dist} main
```

3.2.15 wsrep_provider_options

Contents

6. debug
7. evs.auto_evict
8. evs.causal_keepalive_period
9. evs.debug_log_mask
10. evs.delay_margin
11. evs.delayed_keep_period
12. evs.evict
13. evs.inactive_check_period
14. evs.inactive_timeout
15. evs.info_log_mask
16. evs.install_timeout
17. evs.join_retrans_period
18. evs.keepalive_period
19. evs.max_install_timeouts
20. evs.send_window
21. evs.stats_report_period
22. evs.suspect_timeout
23. evs.use_aggregate
24. evs.user_send_window
25. evs.version
26. evs.view_forget_timeout
27. gcache.dir
28. gcache.keep_pages_size
29. gcache.mem_size
30. gcache.name
31. gcache.page_size
32. gcache.recover
33. gcache.size
34. gcomm.thread_prio
35. gcs.fc_debug
36. gcs.fc_factor
37. gcs.fc_limit
38. gcs.fc_master_slave
39. gcs.fc_single_primary
40. gcs.max_packet_size
41. gcs.max_throttle
42. gcs.recv_q_hard_limit
43. gcs.recv_q_soft_limit
44. gcs.sync_donor
45. gmcast.listen_addr
46. gmcast.mcast_addr
47. gmcast.mcast_ttl
48. gmcast.peer_timeout
49. gmcast.segment
50. gmcast.time_wait
51. gmcast.version
52. ist.recv_addr
53. ist.recv_bind
54. pc.announce_timeout
55. pc.checksum
56. pc.ignore_quorum
57. pc.ignore_sb
58. pc.linger
59. pc.npvo
60. pc.recovery
61. pc.version
62. pc.wait_prim
63. pc.wait_prim_timeout
64. pc.weight
65. protonet.backend
66. protonet.version
67. repl.causal_read_timeout
68. repl.commit_order
69. repl.key_format
70. repl.max_ws_size
71. repl.proto_max
72. socket.checksum
73. socket.dynamic

74. socket.recv_buf_size
75. socket.send_buf_size
76. socket.ssl
77. socket.ssl_ca
78. socket.ssl_cert
79. socket.ssl_cipher
80. socket.ssl_compression
81. socket.ssl_key
82. socket.ssl_password_file

wsrep_provider_options

The following options can be set as part of the Galera wsrep_provider_options variable. Dynamic options can be changed while the server is running.

Options need to be provided as a semicolon (;) separated list on a single line. Options that are not explicitly set are set to their default value.

Note that before Galera 3, the `repl` tag was named `replicator`.

`base_dir`

- **Description:** Specifies the data directory

`base_host`

- **Description:** For internal use. Should not be manually set.
- **Default:** `127.0.0.1` (detected network address)

`base_port`

- **Description:** For internal use. Should not be manually set.
- **Default:** `4567`

`cert.log_conflicts`

- **Description:** Certification failure log details.
- **Dynamic:** Yes
- **Default:** `no`

`cert.optimistic_pa`

- **Description:** Controls parallel application of actions on the replica. If set, the full range of parallelization as determined by the certification algorithm is permitted. If not set, the parallel applying window will not exceed that seen on the primary, and applying will start no sooner than after all actions it has seen on the master are committed.
- **Dynamic:** Yes
- **Default:** `yes`

`debug`

- **Description:** Enable debugging.
- **Dynamic:** Yes
- **Default:** `no`

`evs.auto_evict`

- **Description:** Number of entries the node permits for a given delayed node before triggering the Auto Eviction protocol. An entry is added to a delayed list for each delayed response from a node. If set to `0`, the default, the Auto Eviction protocol is disabled for this node. See Auto Eviction for more.
- **Dynamic:** No

- **Default:** `0`

`evs.causal_keepalive_period`

- **Description:** Used by the developers only, and not manually serviceable.
- **Dynamic:** No
- **Default:** The evs.keepalive_period.

`evs.debug_log_mask`

- **Description:** Controls EVS debug logging. Only effective when wsrep_debug is on.
- **Dynamic:** Yes
- **Default:** `0x1`

`evs.delay_margin`

- **Description:** Time that response times can be delayed before this node adds an entry to the delayed list. See evs.auto_evict. Must be set to a higher value than the round-trip delay time between nodes.
- **Dynamic:** No
- **Default:** `PT1S`

`evs.delayed_keep_period`

- **Description:** Time that this node requires a previously delayed node to remain responsive before being removed from the delayed list. See evs.auto_evict.
- **Dynamic:** No
- **Default:** `PT30S`

`evs.evict`

- **Description:** When set to the gcomm UUID of a node, that node is evicted from the cluster. When set to an empty string, the eviction list is cleared on the node where it is set. See evs.auto_evict.
- **Dynamic:** No
- **Default:** Empty string

`evs.inactive_check_period`

- **Description:** Frequency of checks for peer inactivity (looking for nodes with delayed responses), after which nodes may be added to the delayed list, and later evicted.
- **Dynamic:** No
- **Default:** `PT0.5S`

`evs.inactive_timeout`

- **Description:** Time limit that a node can be inactive before being pronounced as dead.
- **Dynamic:** No
- **Default:** `PT15S`

`evs.info_log_mask`

- **Description:** Controls extra EVS info logging. Bits:
  - 0x1 – extra view change information
  - 0x2 – extra state change information
  - 0x4 – statistics
  - 0x8 – profiling (only available in builds with profiling enabled)
- **Dynamic:** No
- **Default:** `0`

evs.install_timeout

- **Description:** Timeout on waits for install message acknowledgments. Replaces evs.consensus_timeout.
- **Dynamic:** Yes
- **Default:** `PT7.5S`

evs.join_retrans_period

- **Description:** Time period for how often retransmission of EVS join messages when forming cluster membership should occur.
- **Dynamic:** Yes
- **Default:** `PT1S`

evs.keepalive_period

- **Description:** How often keepalive signals should be transmitted when there's no other traffic.
- **Dynamic:** Yes
- **Default:** `PT1S`

evs.max_install_timeouts

- **Description:** Number of membership install rounds to attempt before timing out. The total rounds will be this value plus two.
- **Dynamic:** No
- **Default:** `3`

evs.send_window

- **Description:** Maximum number of packets that can be replicated at a time, Must be more than evs.user_send_window, which applies to data packets only (double is recommended). In WAN environments can be set much higher than the default, for example `512` .
- **Dynamic:** Yes
- **Default:** `4`

evs.stats_report_period

- **Description:** Reporting period for EVS statistics.
- **Dynamic:** No
- **Default:** `PT1M`

evs.suspect_timeout

- **Description:** A node will be suspected to be dead after this period of inactivity. If all nodes agree, the node is dropped from the cluster before evs.inactive_timeout is reached.
- **Dynamic:** No
- **Default:** `PT5S`

evs.use_aggregate

- **Description:** If set to `true` (the default), small packets will be aggregated into one where possible.
- **Dynamic:** No
- **Default:** `true`

evs.user_send_window

- **Description:** Maximum number of data packets that can be replicated at a time. Must be smaller than

evs.send_window (half is recommended). In WAN environments can be set much higher than the default, for example `512`.

- **Dynamic:** Yes
- **Default:** `2`

`evs.version`

- **Description:** EVS protocol version. Defaults to `0` for backward compatibility. Certain EVS features (e.g. auto eviction) require more recent versions.
- **Dynamic:** No
- **Default:** `0`

`evs.view_forget_timeout`

- **Description:** Time after which past views will be dropped from the view history.
- **Dynamic:** No
- **Default:** `P1D`

`gcache.dir`

- **Description:** Directory where GCache files are placed.
- **Dynamic:** No
- **Default:** The working directory

`gcache.keep_pages_size`

- **Description:** Total size of the page storage pages for caching. One page is always present if only page storage is enabled.
- **Dynamic:** No
- **Default:** `0`

`gcache.mem_size`

- **Description:** Maximum size of size of the malloc() store for setups that have spare RAM.
- **Dynamic:** No
- **Default:** `0`

`gcache.name`

- **Description:** Gcache ring buffer storage file name. By default placed in the working directory, changing to another location or partition can reduce disk IO.
- **Dynamic:** No
- **Default:** `./galera.cache` ---

`gcache.page_size`

- **Description:** Size of the page storage page files. These are prefixed by `gcache.page`. Can be set to as large as the disk can handle.
- **Dynamic:** No
- **Default:** `128M`

`gcache.recover`

- **Description:** Whether or not gcache recovery takes place when the node starts up. If it is possible to recover gcache, the node can then provide IST to other joining nodes, which assists when the whole cluster is restarted.
- **Dynamic:** No
- **Default:** `no`
- **Introduced:** MariaDB 10.1.20, MariaDB Galera 10.0.29, MariaDB Galera 5.5.54

gcache.size

- **Description:** Gcache ring buffer storage size (the space the node uses for caching write sets), preallocated on startup.
- **Dynamic:** No
- **Default:** `128M`

gcomm.thread_prio

- **Description:** Gcomm thread policy and priority (in the format `policy:priority`. Priority is an integer, while policy can be one of:
  - `fifo`: First-in, first-out scheduling. Always preempt other, batch or idle threads and can only be preempted by other `fifo` threads of a higher priority or blocked by an I/O request.
  - `rr`: Round-robin scheduling. Always preempt other, batch or idle threads. Runs for a fixed period of time after which the thread is stopped and moved to the end of the list, being replaced by another round-robin thread with the same priority. Otherwise runs until preempted by other `rr` threads of a higher priority or blocked by an I/O request.
  - `other`: Default scheduling on Linux. Threads run until preempted by a thread of a higher priority or a superior scheduling designation, or blocked by an I/O request.
- **Dynamic:** No
- **Default:** Empty string

gcs.fc_debug

- **Description:** If set to a value greater than zero (the default), debug statistics about SST flow control will be posted each time after the specified number of writesets.
- **Dynamic:** No
- **Default:** `0`

gcs.fc_factor

- **Description:**Fraction below gcs.fc_limit which if the recv queue drops below, replication resumes.
- **Dynamic:** Yes
- **Default:** `1.0`

gcs.fc_limit

- **Description:** If the recv queue exceeds this many writesets, replication is paused. Can increase greatly in master-slave setups. Replication will resume again according to the gcs.fc_factor setting.
- **Dynamic:** Yes
- **Default:** `16`

gcs.fc_master_slave

- **Description:** Whether to assume that the cluster only contains one master. Deprecated since Galera 4.10 (MariaDB 10.8.1, MariaDB 10.7.2, MariaDB 10.6.6, MariaDB 10.5.14, MariaDB 10.4.22) - see gcs.fc_single_primary
- **Dynamic:** No
- **Default:** `no`

gcs.fc_single_primary

- **Description:** Defines whether there is more than one source of replication. As the number of nodes in the cluster grows, the larger the calculated gcs.fc_limit gets. At the same time, the number of writes from the nodes increases. When this parameter value is set to NO (multi-primary), the gcs.fc_limit parameter is dynamically modified to give more margin for each node to be a bit further behind applying writes. The gcs.fc_limit parameter is modified by the square root of the cluster size, that is, in a four-node cluster it is two times higher than the base value. This is done to compensate for the increasing replication rate noise.
- **Dynamic:** No
- **Default:** `no`

gcs.max_packet_size

- **Description:** Maximum packet size, after which writesets become fragmented.
- **Dynamic:** No
- **Default:** `64500`

gcs.max_throttle

- **Description:** How much we can throttle replication rate during state transfer (to avoid running out of memory). Set it to 0.0 if stopping replication is acceptable for the sake of completing state transfer.
- **Dynamic:** No
- **Default:** `0.25`

gcs.recv_q_hard_limit

- **Description:** Maximum size of the recv queue. If exceeded, the server aborts. Half of available RAM plus swap is a recommended size.
- **Dynamic:** No
- **Default:** LLONG_MAX

gcs.recv_q_soft_limit

- **Description:** Fraction of gcs.recv_q_hard_limit after which replication rate is throttled. The rate of throttling increases linearly from zero (the regular, varying rate of replication) at and below `csrecv_q_soft_limit` to one (full throttling) at gcs.recv_q_hard_limit
- **Dynamic:** No
- **Default:** `0.25`

gcs.sync_donor

- **Description:** Whether or not the rest of the cluster should stay in sync with the donor. If set to `YES` (`NO` is default), if the donor is blocked by state transfer, the whole cluster is also blocked.
- **Dynamic:** No
- **Default:** `no`

gmcast.listen_addr

- **Description:** Address Galera listens for connections from other nodes. Can be used to override the default port to listen, which is obtained from the connection address.
- **Dynamic:** No
- **Default:** `tcp://0.0.0.0:4567`

gmcast.mcast_addr

- **Description:** Not set by default, but if set, UDP multicast will be used for replication. Must be identical on all nodes.For example, `gmcast.mcast_addr=239.192.0.11`
- **Dynamic:** No
- **Default:** None

gmcast.mcast_ttl

- **Description:** Multicast packet TTL (time to live) value.
- **Dynamic:** No
- **Default:** `1`

gmcast.peer_timeout

- **Description:** Connection timeout for initiating message relaying.
- **Dynamic:** No
- **Default:** `PT3S`

---

gmcast.segment

- **Description:** Defines the segment to which the node belongs. By default, all nodes are placed in the same segment (`0`). Usually, you would place all nodes in the same datacenter in the same segment. Galera protocol traffic is only redirected to one node in each segment, and then relayed to other nodes in that same segment, which saves cross-datacenter network traffic at the expense of some extra latency. State transfers are also, preferably but not exclusively, taken from the same segment. If there are no nodes available in the same segment, state transfer will be taken from a node in another segment.
- **Dynamic:** No
- **Default:** `0`
- **Range:** `0` to `255`

---

gmcast.time_wait

- **Description:** Waiting time before allowing a peer that was declared outside of the stable view to reconnect.
- **Dynamic:** No
- **Default:** `PT5S`

---

gmcast.version

- **Description:** Deprecated option. Gmcast version.
- **Dynamic:** No
- **Default:** `0`

---

ist.recv_addr

- **Description:** Address for listening for Incremental State Transfer.
- **Dynamic:** No
- **Default:** <address>:<port+1> from wsrep_node_address

---

ist.recv_bind

- **Description:**
- **Dynamic:** No
- **Default:** Empty string
- **Introduced:** MariaDB 10.1.17, MariaDB Galera 10.0.27, MariaDB Galera 5.5.51

---

pc.announce_timeout

- **Description:** Period of time for which cluster joining announcements are sent every 1/2 second.
- **Dynamic:** No
- **Default:** `PT3S`

---

pc.checksum

- **Description:** For debug purposes, by default `false` (`true` in earlier releases), indicates whether to checksum replicated messages on PC level. Safe to turn off.
- **Dynamic:** No
- **Default:** `false`

---

pc.ignore_quorum

- **Description:** Whether to ignore quorum calculations, for example when a master splits from several slaves, it will remain in operation if set to `true` (`false is default`). Use with care however, as in master-slave setups, slaves will not automatically reconnect to the master if set.
- **Dynamic:** Yes
- **Default:** `false`

`pc.ignore_sb`

- **Description:** Whether to permit updates to be processed even in the case of split brain (when a node is disconnected from its remaining peers). Safe in master-slave setups, but could lead to data inconsistency in a multi-master setup.
- **Dynamic:** Yes
- **Default:** `false`

`pc.linger`

- **Description:** Time that the PC protocol waits for EVS termination.
- **Dynamic:** No
- **Default:** `PT20S`

`pc.npvo`

- **Description:** If set to `true` (`false` is default), when there are primary component conficts, the most recent component will override the older.
- **Dynamic:** No
- **Default:** `false`

`pc.recovery`

- **Description:** If set to `true` (the default), the Primary Component state is stored on disk and in the case of a full cluster crash (e.g power outages), automatic recovery is then possible. Subsequent graceful full cluster restarts will require explicit bootstrapping for a new Primary Component.
- **Dynamic:** No
- **Default:** `true`

`pc.version`

- **Description:** Deprecated option. PC protocol version.
- **Dynamic:** No
- **Default:** `0`

`pc.wait_prim`

- **Description:** When set to `true` , the default, the node will wait for a primary component for the period of time specified by pc.wait_prim_timeout. Used to bring up non-primary components and make them primary using pc.bootstrap.
- **Dynamic:** No
- **Default:** `true`

`pc.wait_prim_timeout`

- **Description:** Ttime to wait for a primary component. See pc.wait_prim.
- **Dynamic:** No
- **Default:** `PT30S`

`pc.weight`

- **Description:** Node weight, used for quorum calculation. See the Codership article Weighted Quorum.
- **Dynamic:** Yes
- **Default:** `1`

`protonet.backend`

- **Description:** Deprecated option. Transport backend to use. Only ASIO is supported currently.
- **Dynamic:** No
- **Default:** `asio`

`protonet.version`

- **Description:** Deprecated option. Protonet version.
- **Dynamic:** No
- **Default:** `0`

`repl.causal_read_timeout`

- **Description:** Timeout period for causal reads.
- **Dynamic:** Yes
- **Default:** `PT30S`

`repl.commit_order`

- **Description:** Whether or not out-of-order committing is permitted, and under what conditions. By default it is not permitted, but setting this can improve parallel performance.
  - `0` BYPASS: No commit order monitoring is done (useful for measuring the performance penalty).
  - `1` OOOC: Out-of-order committing is permitted for all transactions.
  - `2` LOCAL_OOOC: Out-of-order committing is permitted for local transactions only.
  - `3` NO_OOOC: Out-of-order committing is not permitted at all.
- **Dynamic:** No
- **Default:** `3`

`repl.key_format`

- **Description:** Format for key replication. Can be one of:
  - `FLAT8` - shorter key with a higher probability of false positives when matching
  - `FLAT16` - longer key with a lower probability of false positives when matching
  - `FLAT8A` - shorter key with a higher probability of false positives when matching, includes annotations for debug purposes
  - `FLAT16A` - longer key with a lower probability of false positives when matching, includes annotations for debug purposes
- **Dynamic:** Yes
- **Default:** `FLAT8`

`repl.max_ws_size`

- **Description:**
- **Dynamic:**
- **Default:** `2147483647`

`repl.proto_max`

- **Description:**
- **Dynamic:**
- **Default:** `9`

---

socket.checksum

- **Description:** Method used for generating checksum. Note: If Galera 25.2.x and 25.3.x are both being used in the cluster, MariaDB with Galera 25.3.x must be started with `wsrep_provider_options='socket.checksum=1'` in order to make it backward compatible with Galera v2. Galera wsrep providers other than 25.3.x or 25.2.x are not supported.
- **Dynamic:** No
- **Default:** `2`

---

socket.dynamic

- **Description:** Allow both encrypted and unencrypted connections between nodes. Typically this should be set to `false` (the default), when set to `true` encrypted connections will still be preferred, but will fall back to unencrypted connections when encryption is not possible, e.g. not enabled on all nodes yet. Needs to be `true` on all nodes when wanting to enable or disable encryption via a rolling restart. As this can't be changed at runtime a rolling restart to enable or disable encryption may need three restarts per node in total: one to enable `socket.dynamic` on each node, one to change the actual encryption settings on each node, and a final round to change `socket.dynamic` back to `false`.
- **Dynamic:** No
- **Default:** `false`
- **Introduced:** MariaDB 10.4.19, MariaDB 10.5.10, MariaDB 10.6.0

---

socket.recv_buf_size

- **Description:** Size in bytes of the receive buffer used on the network sockets between nodes, passed on to the kernel via the SO_RCVBUF socket option.
- **Dynamic:** No
- **Default:**
  - >= MariaDB 10.3.23, MariaDB 10.2.32, MariaDB 10.1.45: Auto
  - < MariaDB 10.3.22: MariaDB 10.2.31, MariaDB 10.1.44: `212992`

---

socket.send_buf_size

- **Description:** Size in bytes of the send buffer used on the network sockets between nodes, passed on to the kernel via the SO_SNDBUF socket option.
- **Dynamic:** No
- **Default:**: Auto
- **Introduced:** MariaDB 10.3.23, MariaDB 10.2.32, MariaDB 10.1.45

---

socket.ssl

- **Description:** Explicitly enables TLS usage by the wsrep Provider.
- **Dynamic:** No
- **Default:** `NO`

---

socket.ssl_ca

- **Description:** Path to Certificate Authority (CA) file. Implicitly enables the `socket.ssl` option.
- **Dynamic:** No

---

socket.ssl_cert

- **Description:** Path to TLS certificate. Implicitly enables the `socket.ssl` option.
- **Dynamic:** No

---

socket.ssl_cipher

- **Description:** TLS cipher to use. Implicitly enables the `socket.ssl` option. Since MariaDB 10.2.18 defaults to the

value of the `ssl_cipher` system variable.
- **Dynamic:** No
- **Default:** system default, before MariaDB 10.2.18 defaults to `AES128-SHA`.

`socket.ssl_compression`

- **Description:** Compression to use on TLS connections. Implicitly enables the `socket.ssl` option.
- **Dynamic:** No

`socket.ssl_key`

- **Description:** Path to TLS key file. Implicitly enables the `socket.ssl` option.
- **Dynamic:** No

`socket.ssl_password_file`

- **Description:** Path to password file to use in TLS connections. Implicitly enables the `socket.ssl` option.
- **Dynamic:** No

3.2.16 Galera Cluster Address

URL's in Galera take a particular format:

```
<schema>://<cluster_address>[?option1=value1[&option2=value2]]
```

Contents

Schema

- `gcomm` - This is the option to use for a working implementation.
- `dummy` - Used for running tests and profiling, does not do any actual replication, and all following parameters are ignored.

Cluster address

- The cluster address shouldn't be empty like `gcomm://`. This should never be hardcoded into any configuration files.
- To connect the node to an existing cluster, the cluster address should contain the address of any member of the cluster you want to join.
- The cluster address can also contain a comma-separated list of multiple members of the cluster. It is good practice to list all possible members of the cluster, for example `gcomm:<node1 name or ip>,<node2 name or ip2>,<node3 name or ip>`. Alternately if multicast is use put the multicast address instead of the list of nodes. Each member address or multicast address can specify `<node name or ip>:<port>` if a non-default port is used.

Option list

- The wsrep_provider_options variable is used to set a list of options. These parameters can also be provided (and overridden) as part of the URL. Unlike options provided in a configuration file, they will not endure, and need to be resubmitted with each connection.

A useful option to set is `pc.wait_prim=no` to ensure the server will start running even if it can't determine a primary node. This is useful if all members go down at the same time.

Port

By default, gcomm listens on all interfaces. The port is either provided in the cluster address, or will default to 4567 if not set.

3.2.17 Galera Load Balancer

Galera Load Balancer is a simple Load Balancer specifically designed for Galera Cluster. Like Galera, it only runs on Linux. Galera Load Balancer is developed and mantained by Codership. Documentation is available on fromdual.com.

Galera Load Balancer is inspired by pen, which is a generic TCP load balancer. However, since pen is a generic TCP connections load balancer, the techniques it uses are not well-suited to the particular use case of database servers. Galera Load Balancer is optimized for this type of workload.

Several balancing policies are supported. Each node can be assigned a different weight. Nodes with a higher weight are preferred. Depending on the selected policy, other nodes can even be ignored, until the preferred nodes crash.

A lightweight daemon called glbd receives the connections from clients and it redirects them to nodes. No specific client exists for this demo: a generic TCP client, like nc, can be used to send administrative commands and read the usage statistics.

2.1.3.11 Upgrading Galera Cluster

3.1.25 Using MariaDB Replication with MariaDB Galera Cluster

2.2.1.1.1.5 Securing Communications in Galera Cluster

3.2.21 Installing MariaDB Galera on IBM Cloud

Contents

Get MariaDB Galera on IBM Cloud

You should have an IBM Cloud account, otherwise you can register here. At the end of the tutorial you will have a cluster with MariaDB up and running. IBM Cloud uses Bitnami charts to deploy MariaDB Galera on with helm

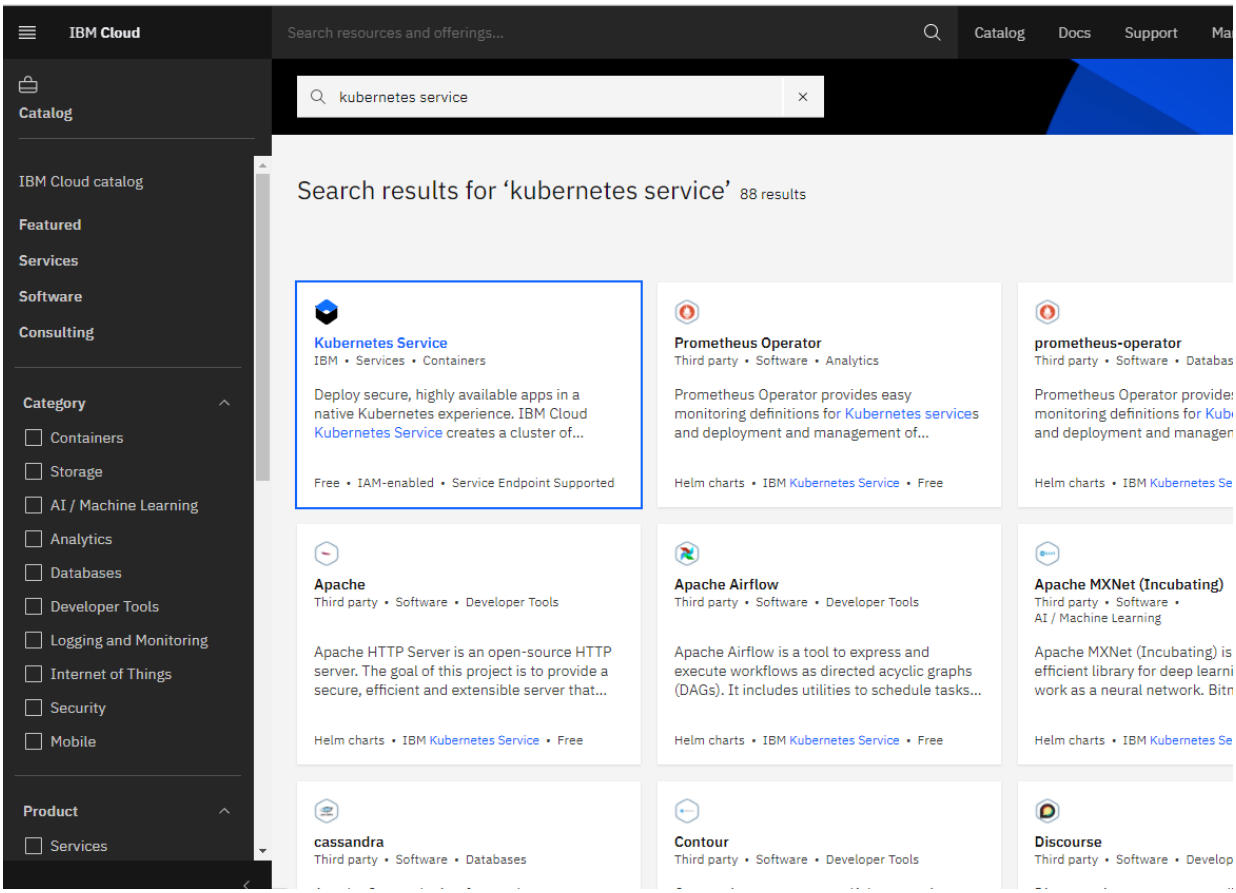
1. We will provision a new Kubernetes Cluster for you if, you already have one skip to step **2**

2. We will deploy the IBM Cloud Block Storage plug-in, if already have it skip to step **3**

3. MariaDB Galera deployment

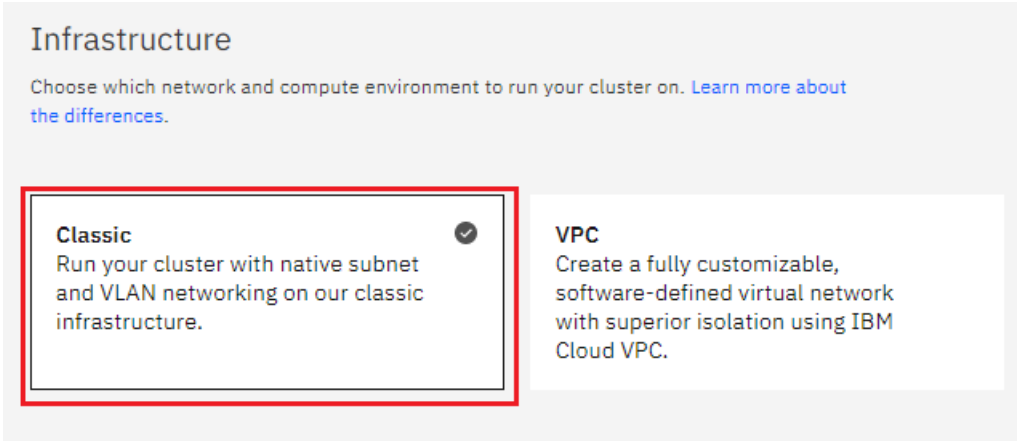
Step 1 provision Kubernetes Cluster

- Click the **Catalog** button on the top
- Select **Service** from the catalog
- Search for **Kubernetes Service** and click on it

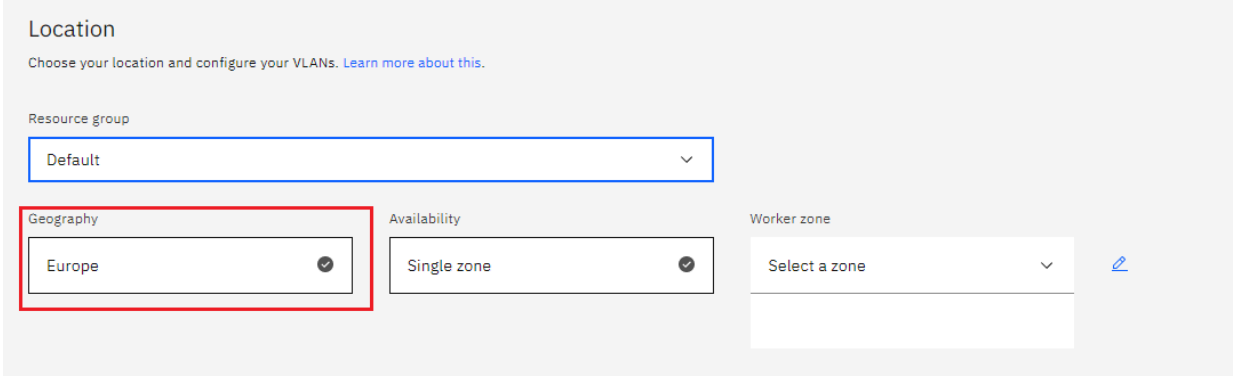


- You are now at the Kubernetes deployment page, you need to specify some details about the cluster
- Choose a plan **standard** or **free**, the free plan only has one worker node and no subnet, to provision a standard cluster, you will need to upgrade you account to Pay-As-You-Go
- To upgrade to a Pay-As-You-Go account, complete the following steps:

- In the console, go to Manage > Account.
- Select Account settings, and click Add credit card.
- Enter your payment information, click Next, and submit your information
- Choose **classic** or **VPC**, read the docs and choose the most suitable type for yourself

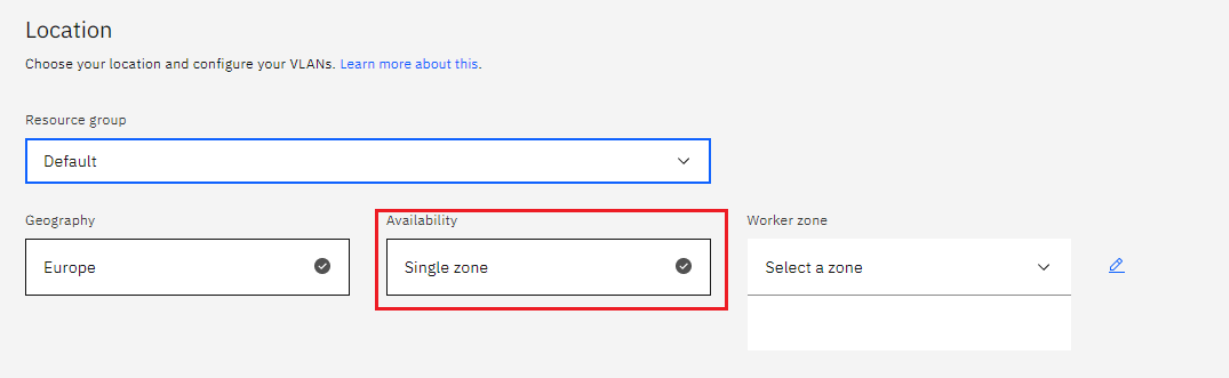


- Now choose your location settings, for more information please visit Locations
- Choose **Geography** (continent)

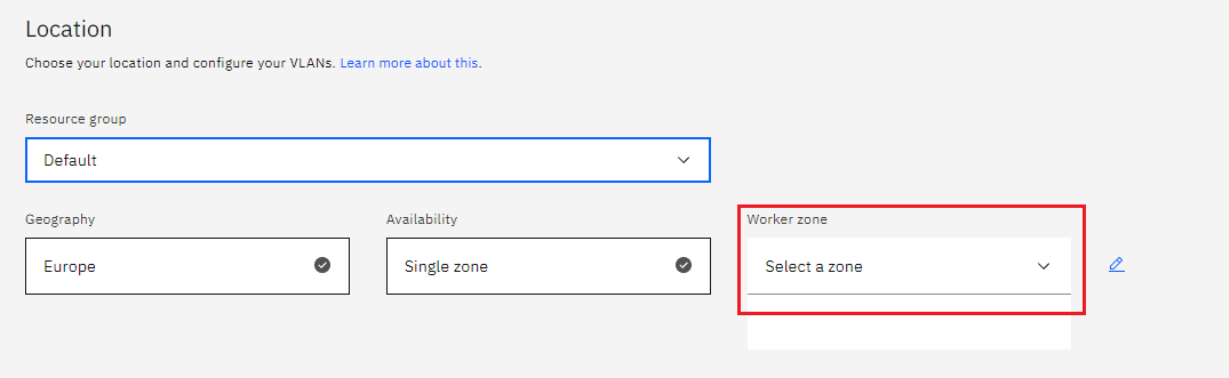


- Choose **Single** or **Multizone**, in single zone your data is only kept in on datacenter, on the other hand with Multizone

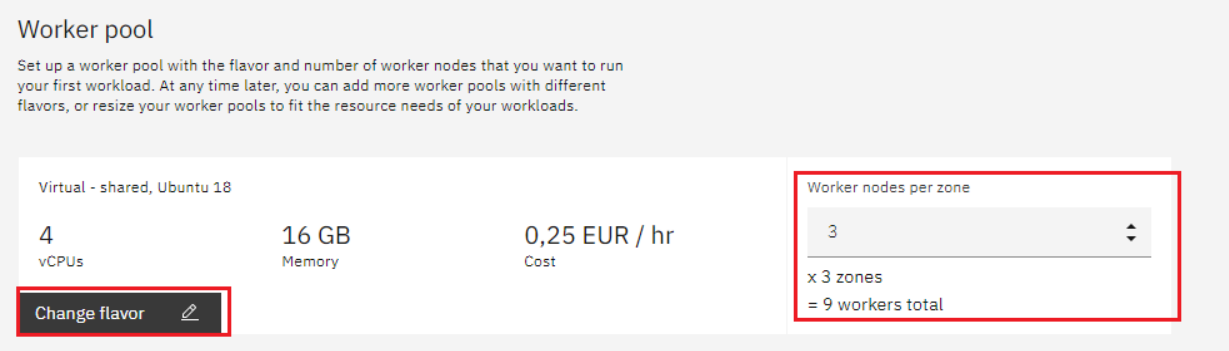
it is distributed to multiple zones, thus safer in an unforseen zone failure



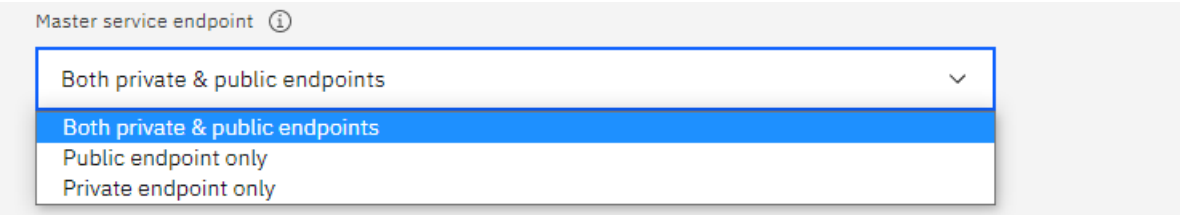
- Choose a **Worker Zone** if using Single zones or **Metro** if Multizone



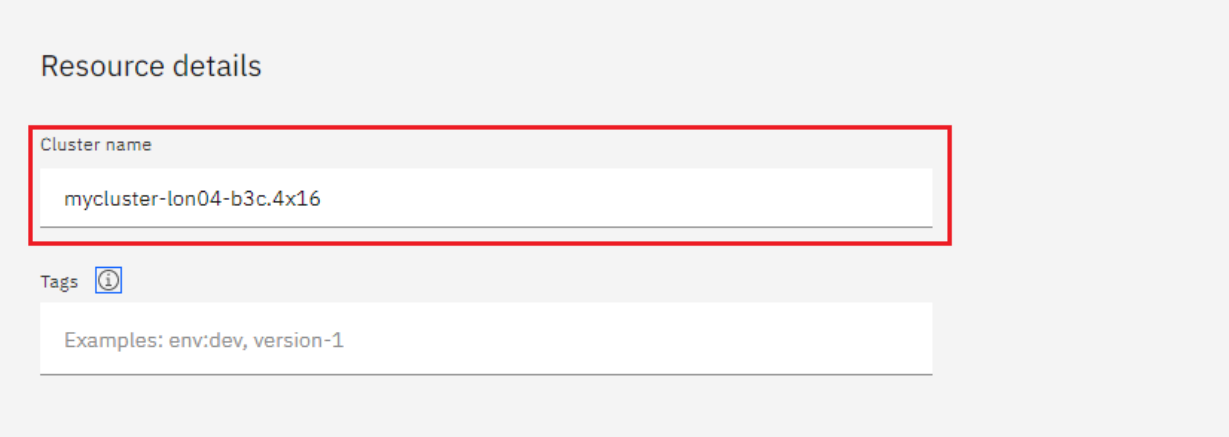
- If you wish to use Multizone please set up your account with VRF or enable Vlan spanning
- If at your current location selection, there is no available Virtual LAN, a new Vlan will be created for you
- Choose a **Worker node setup** or use the preselected one, set **Worker node amount per zone**



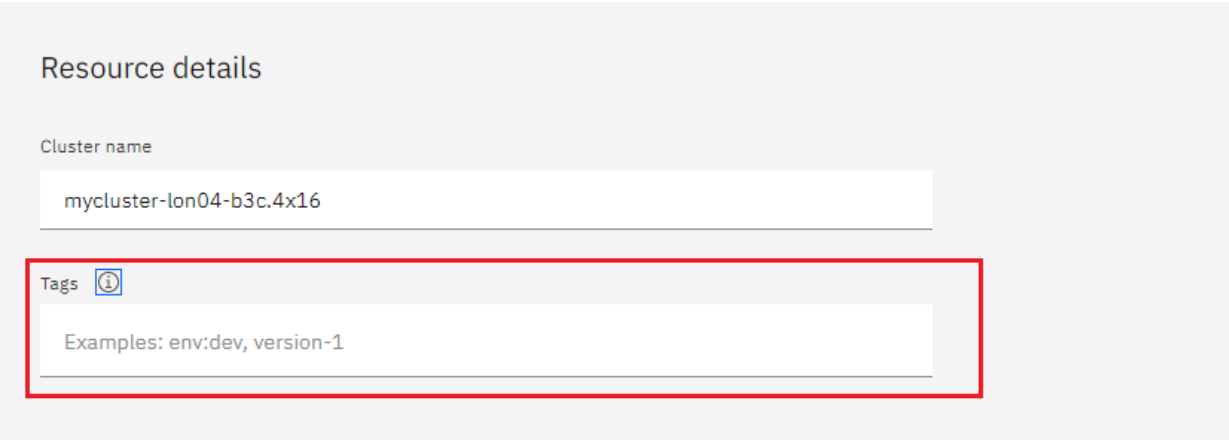
- Choose **Master Service Endpoint**, In VRF-enabled accounts, you can choose private-only to make your master accessible on the private network or via VPN tunnel. Choose public-only to make your master publicly accessible. When you have a VRF-enabled account, your cluster is set up by default to use both private and public endpoints. For more information visit endpoints.



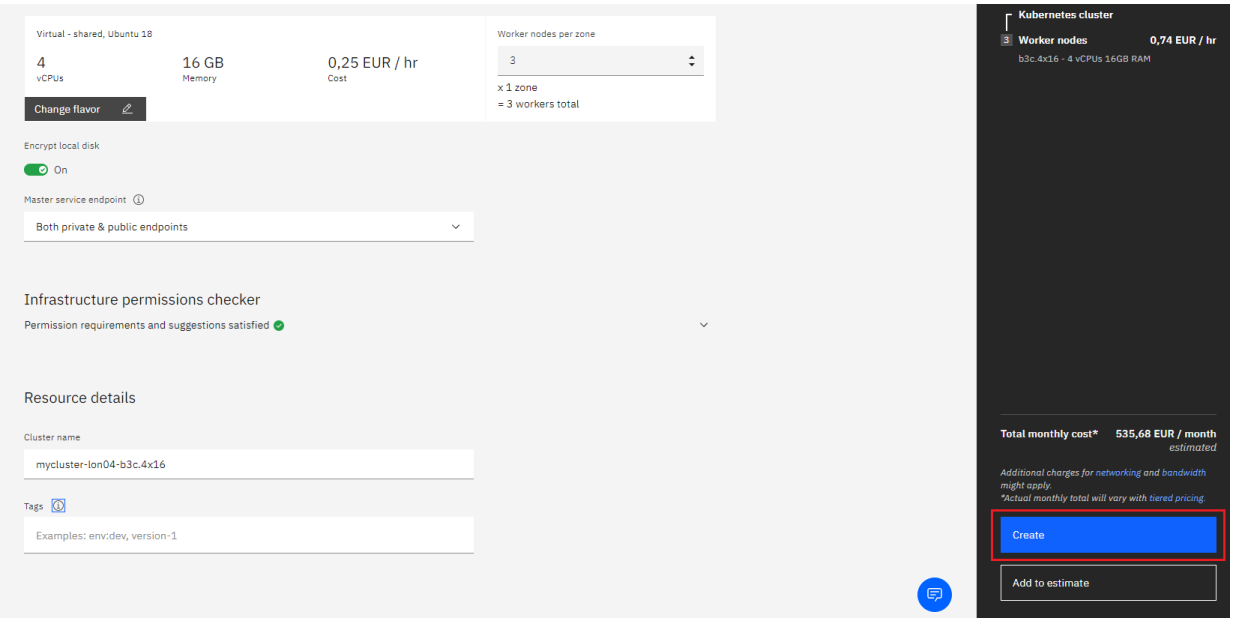
- Give cluster a **name**



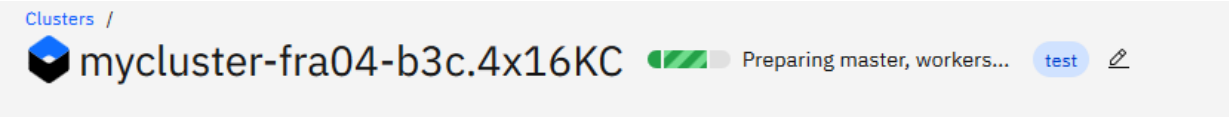
- Give desired **tags** to your cluster, for more information visit tags



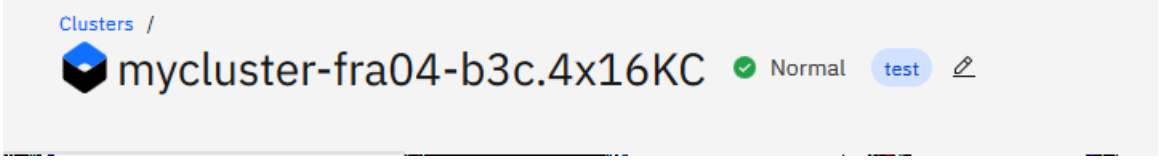
- Click **create**



- Wait for you cluster to be provisioned



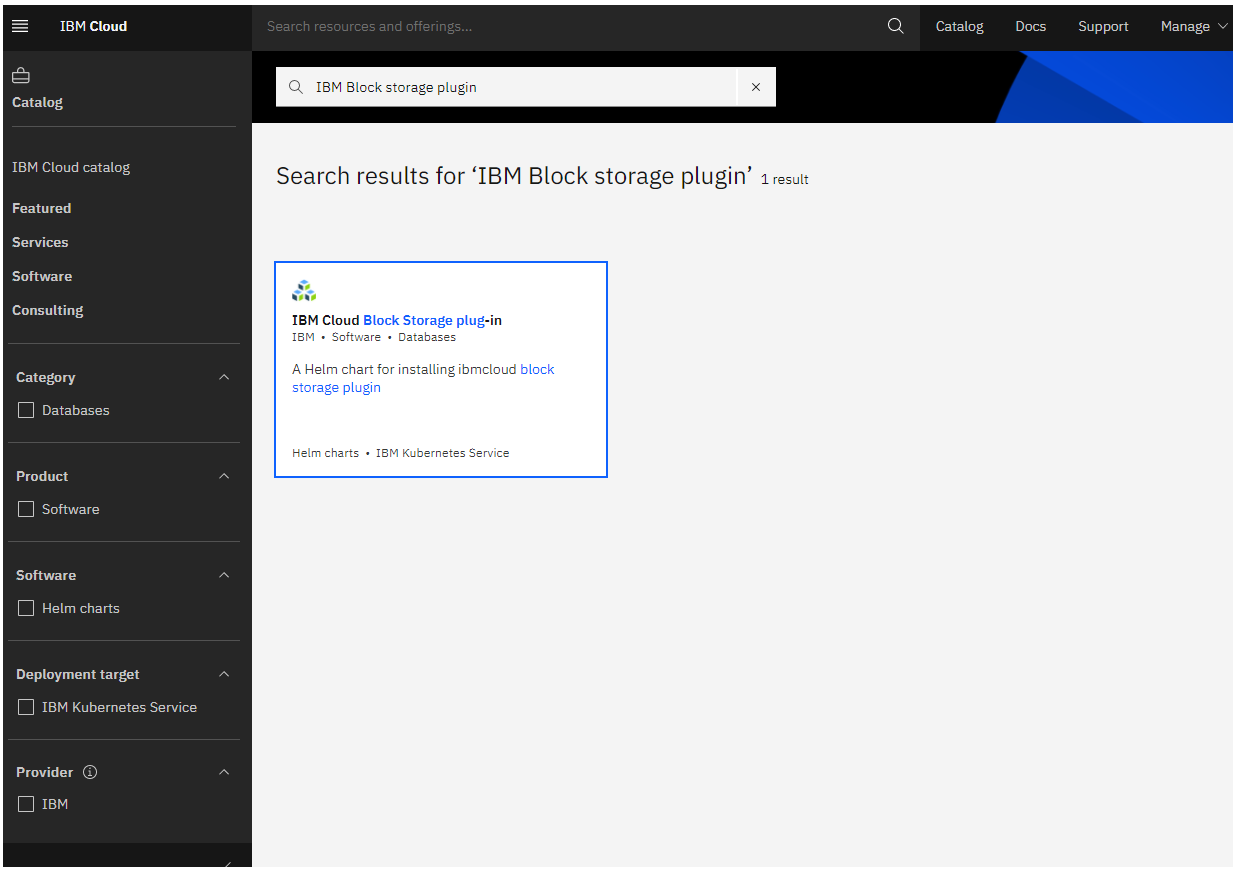
- Your cluster is ready for usage



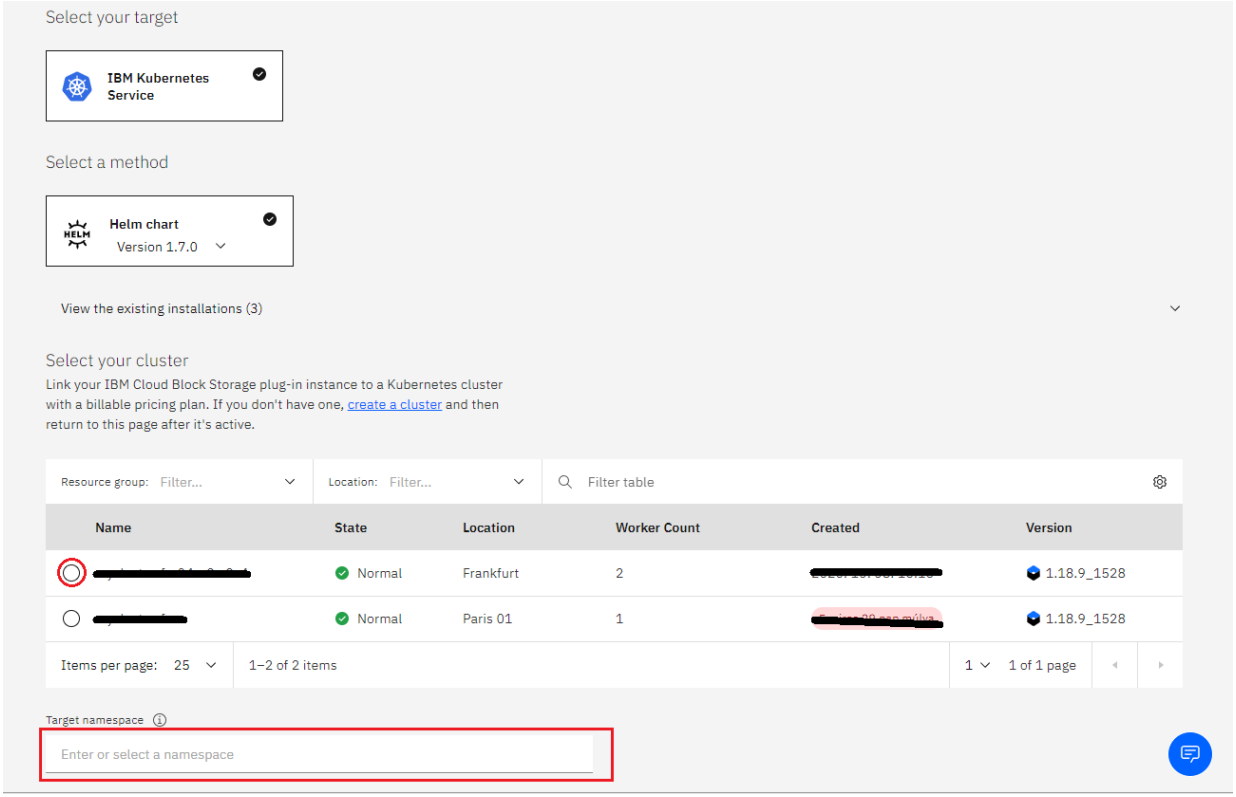
Step 2 deploy IBM Cloud Block Storage plug-in

The Block Storage plug-in is a persistent, high-performance iSCSI storage that you can add to your apps by using Kubernetes Persistent Volumes (PVs).

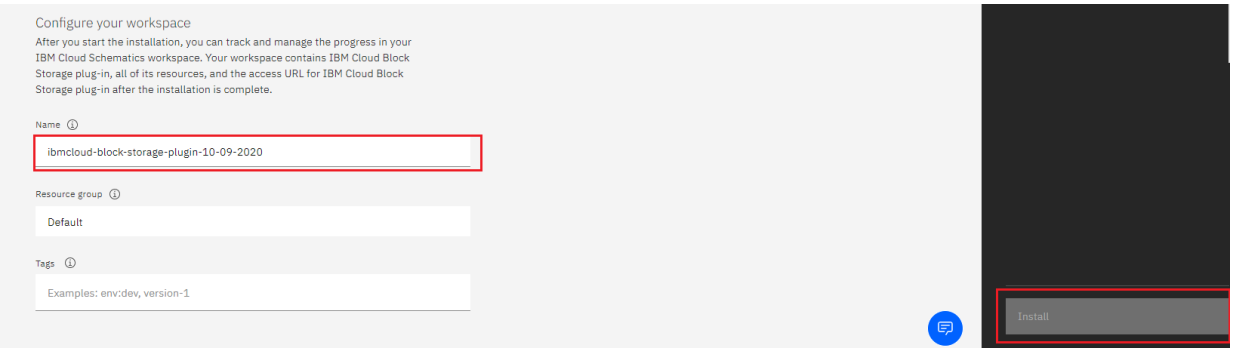
- Click the **Catalog** button on the top
- Select **Software** from the catalog
- Search for **IBM Cloud Block Storage plug-in** and click on it



- On the application page Click in the dot next to the cluster, you wish to use
- Click on **Enter or Select Namespace** and choose the default Namespace or use a custom one (if you get error please wait 30 minutes for the cluster to finalize)



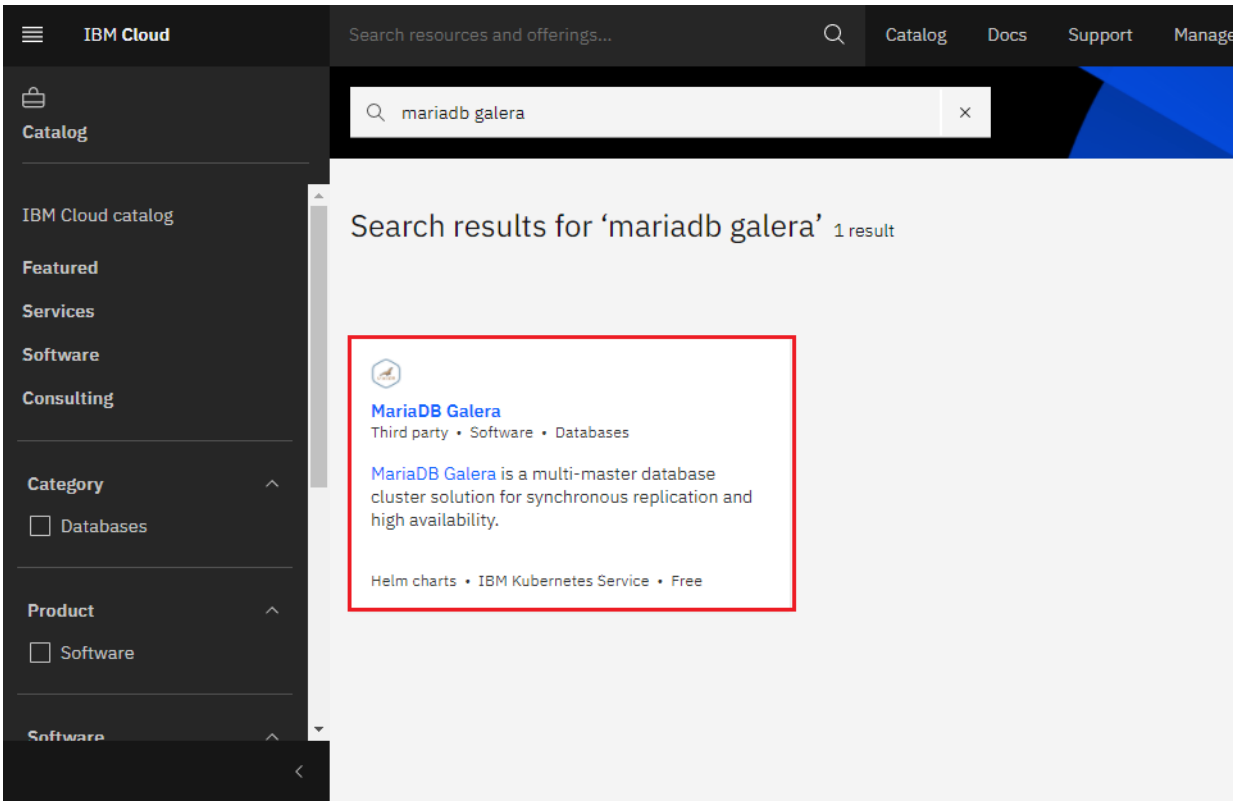
- Give a **name** to this workspace
- Click **install** and wait for the deployment



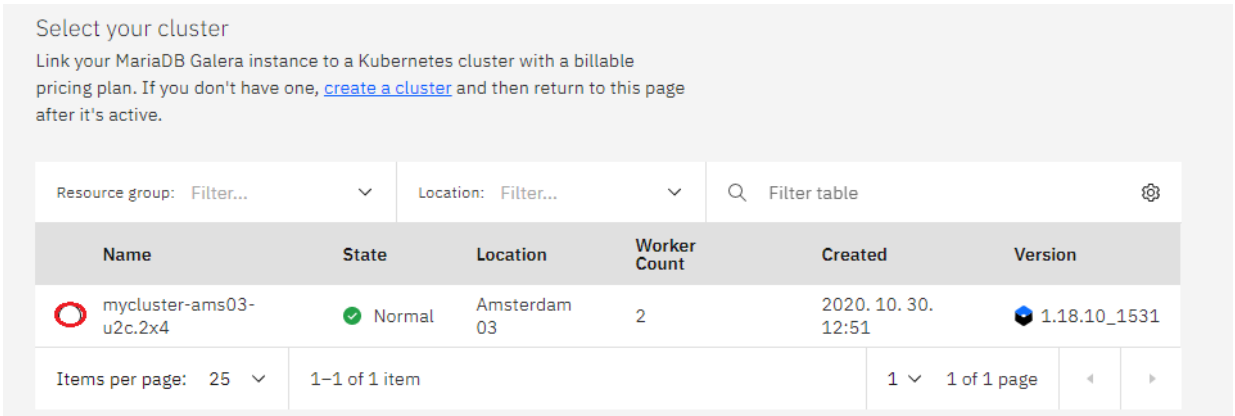
Step 3 deploy MariaDB Galera

We will deploy MariaDB on our cluster

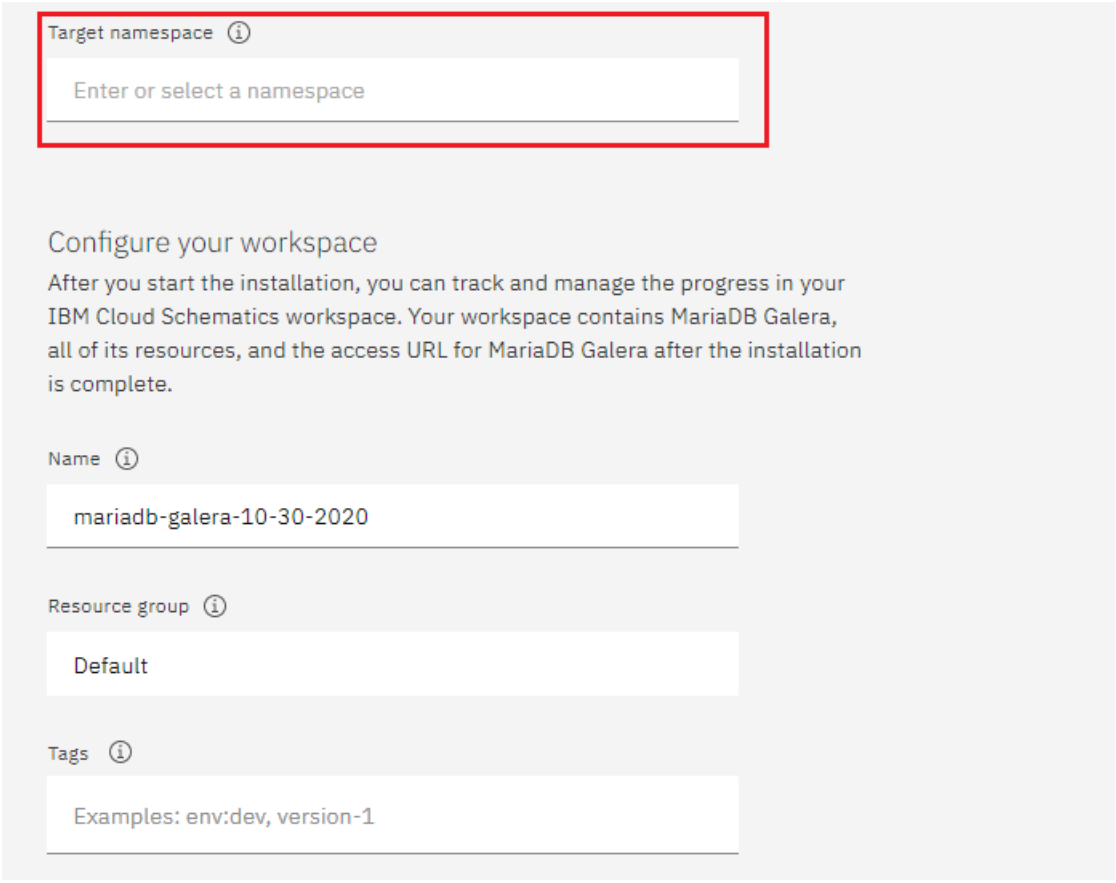
- Click the **Catalog** button on the top
- Select **Software** from the catalog
- Search for **MariaDB** and click on it



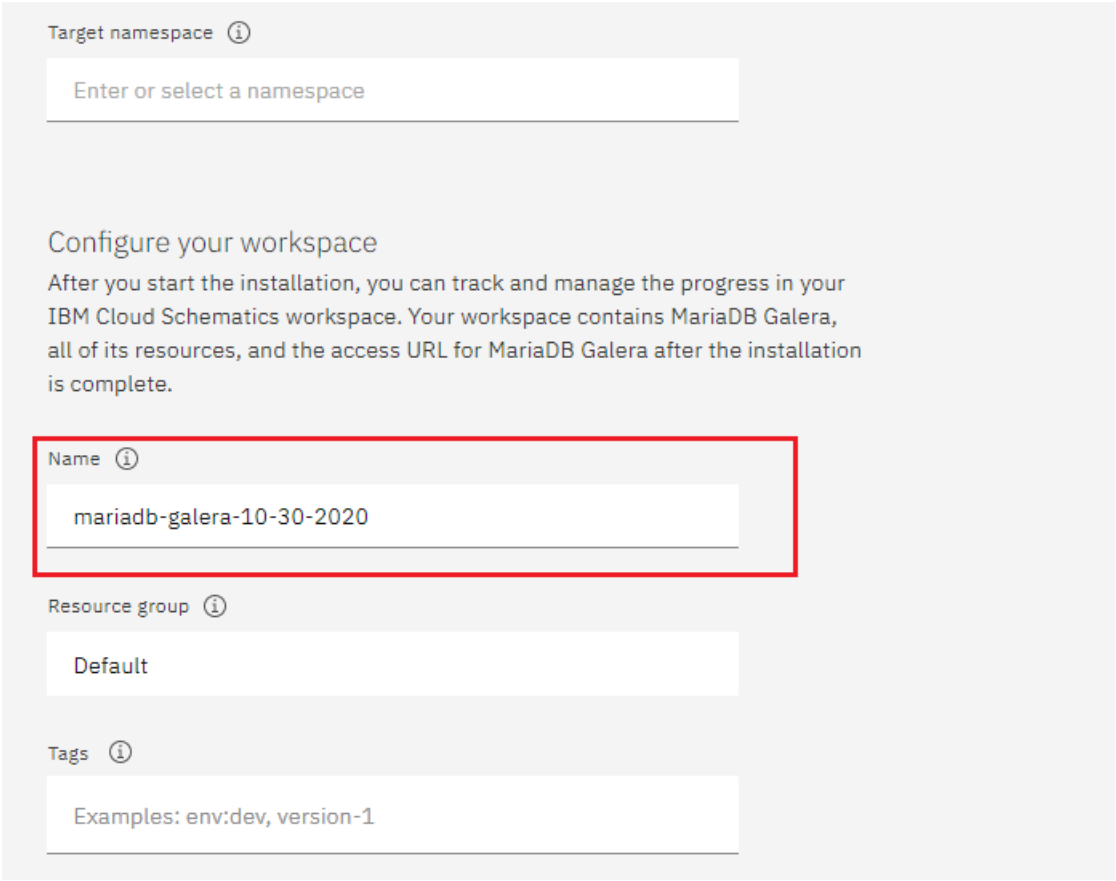
- On the application page Click in the dot next to the cluster, you wish to use



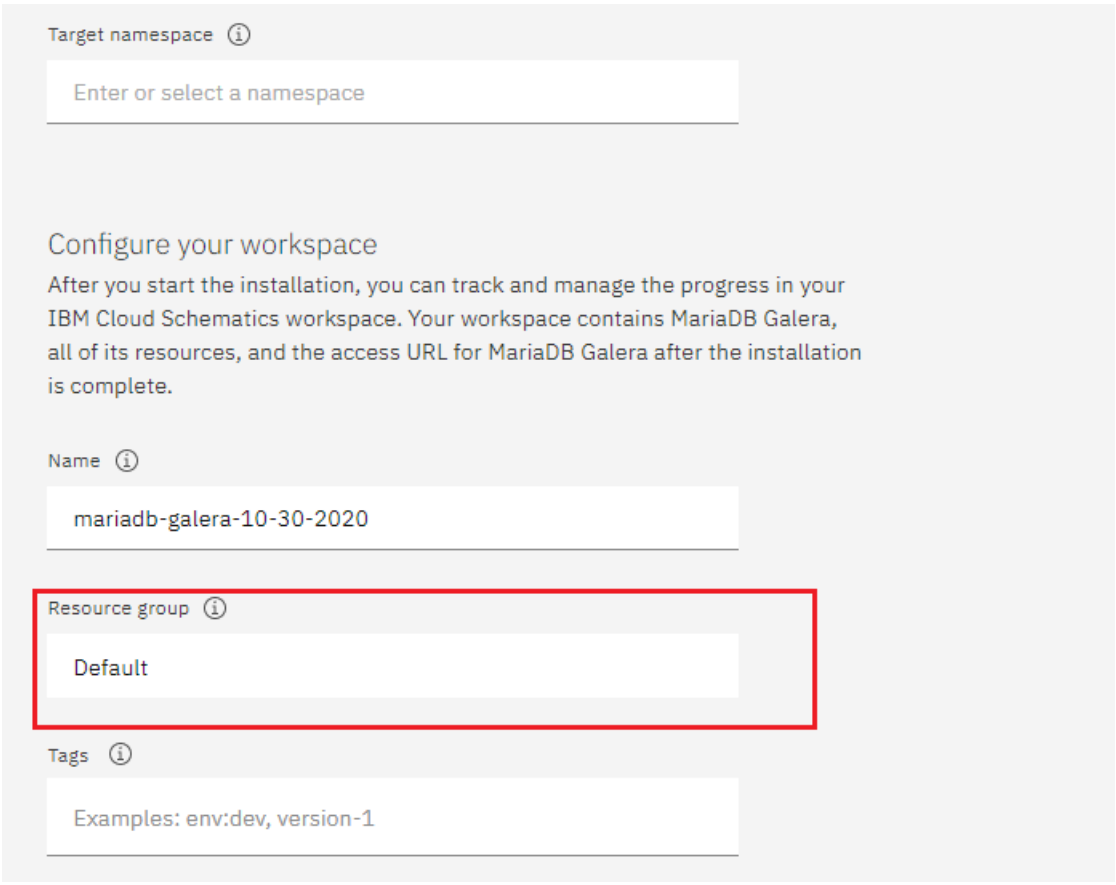
- Click on **Enter or Select Namespace** and choose the default Namespace or use a custom one



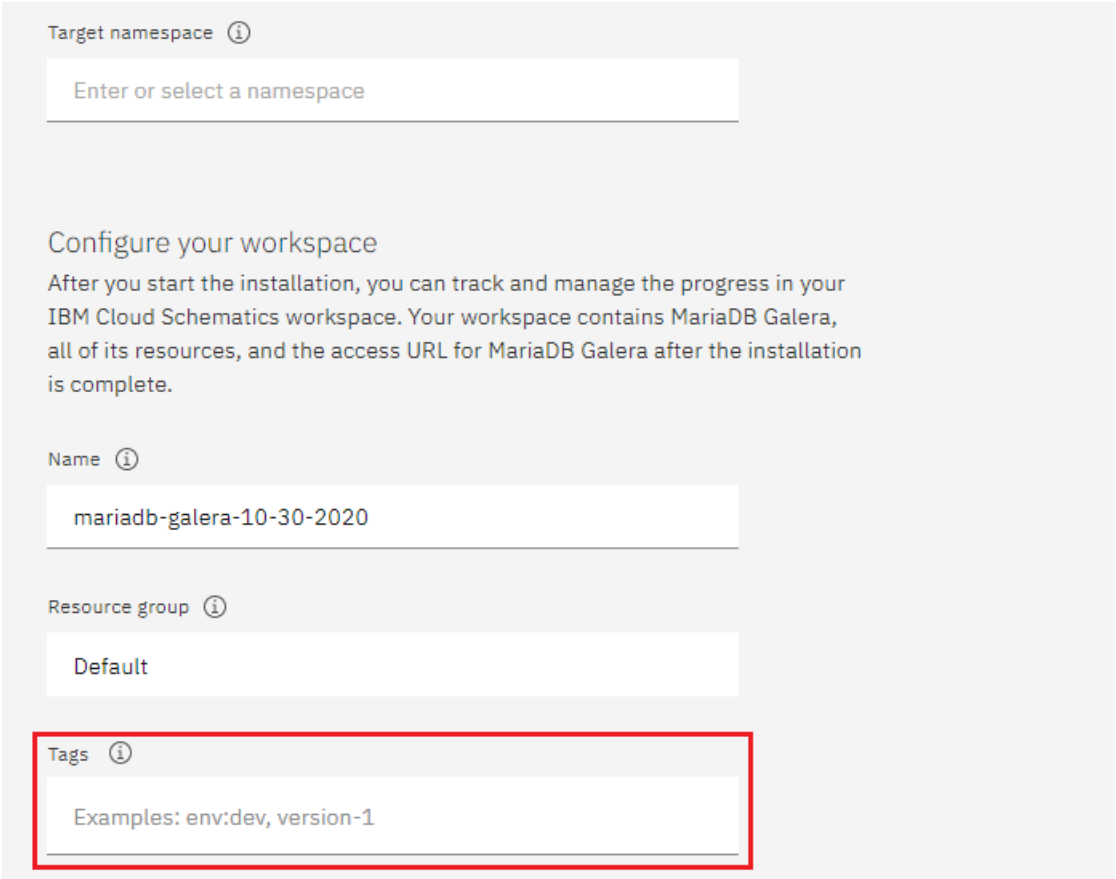
- Give a unique **name** to workspace, which you can easily recognize



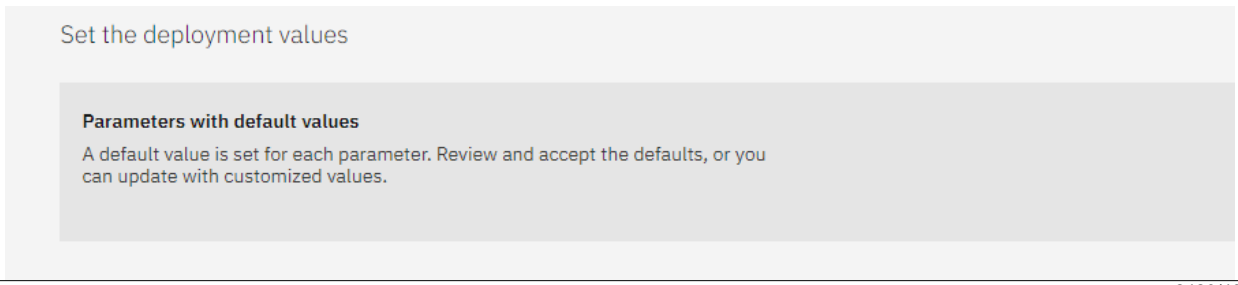
- Select which resource group you want to use, it's for access controll and billing purposes. For more information please visit resource groups



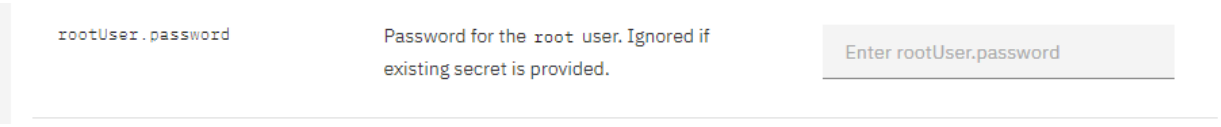
- Give **tags** to your MariaDB Galera, for more information visit tags



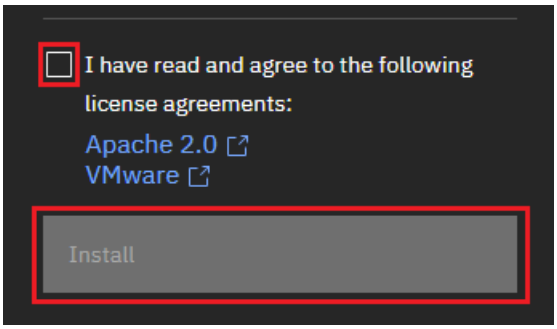
- Click on **Parameters with default values**, You can set deployment values or use the default ones



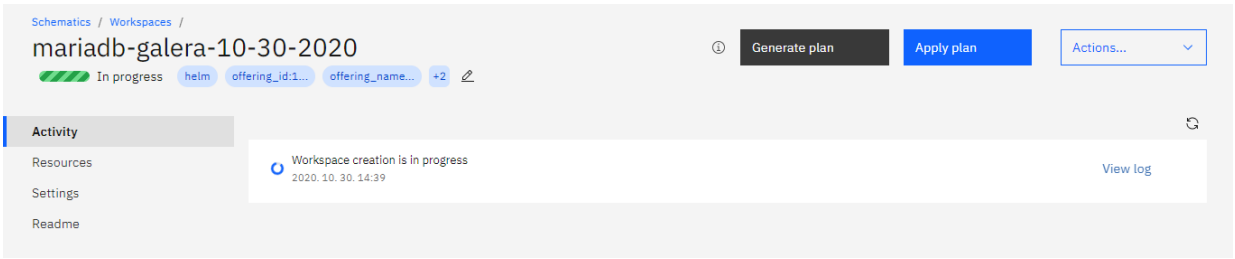
- Please set the MariaDB Galera root password in the parameters



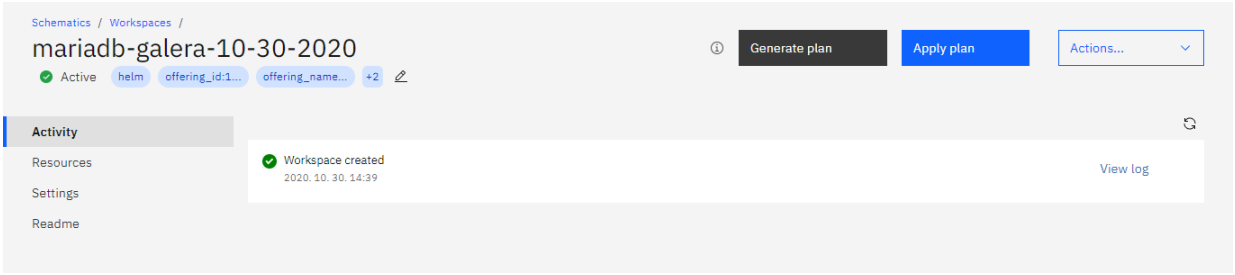
- After finishing everything, **tick** the box next to the agreements and click **install**



- The MariaDB Galera workspace will start installing, wait a couple of minutes

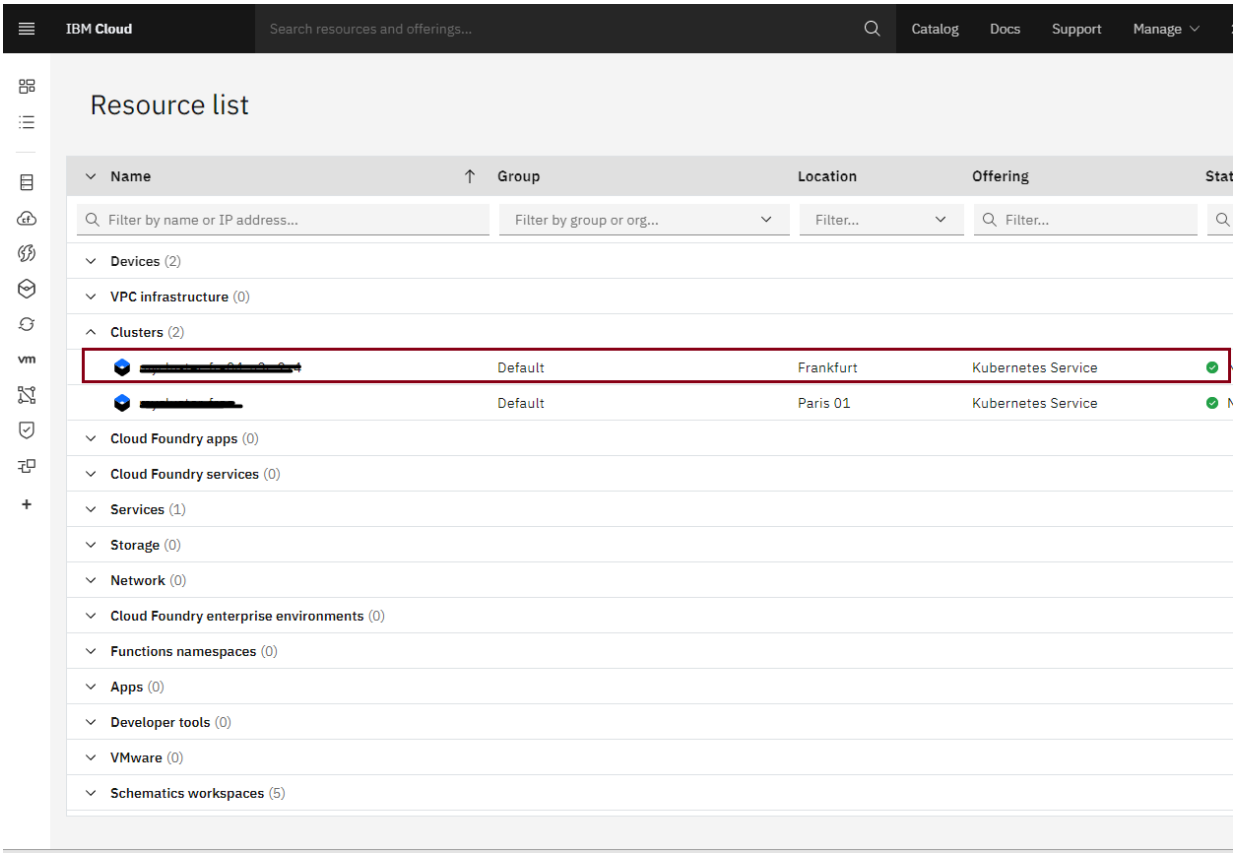


- Your MariaDB Galera workspace has been successfully deployed

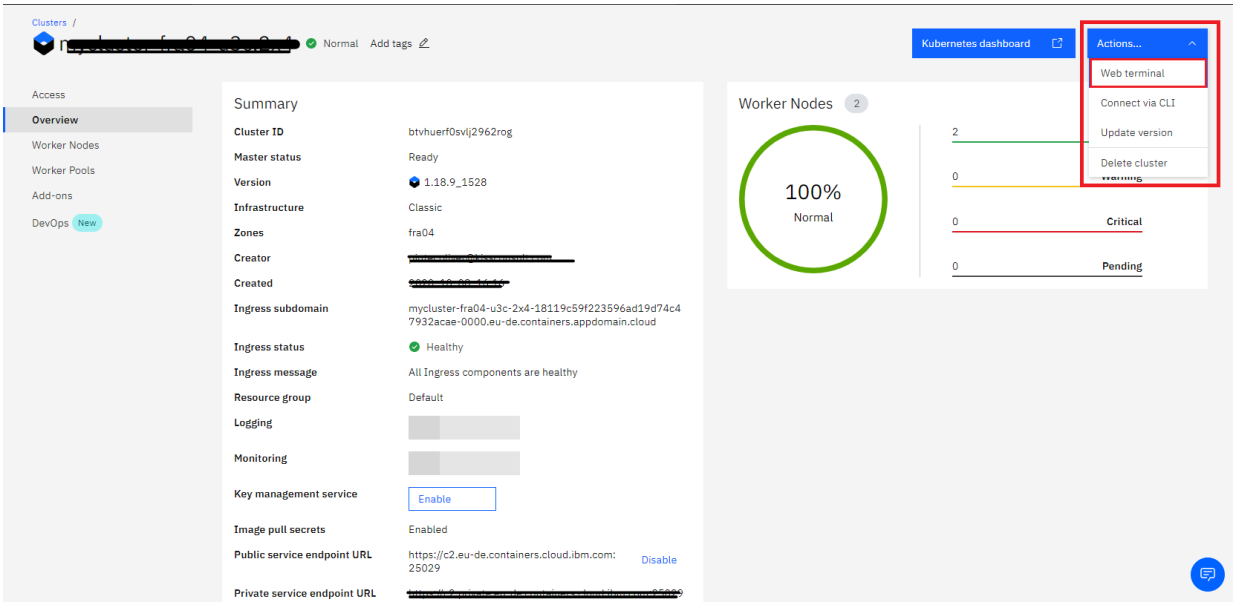


Verify MariaDB Galera installation

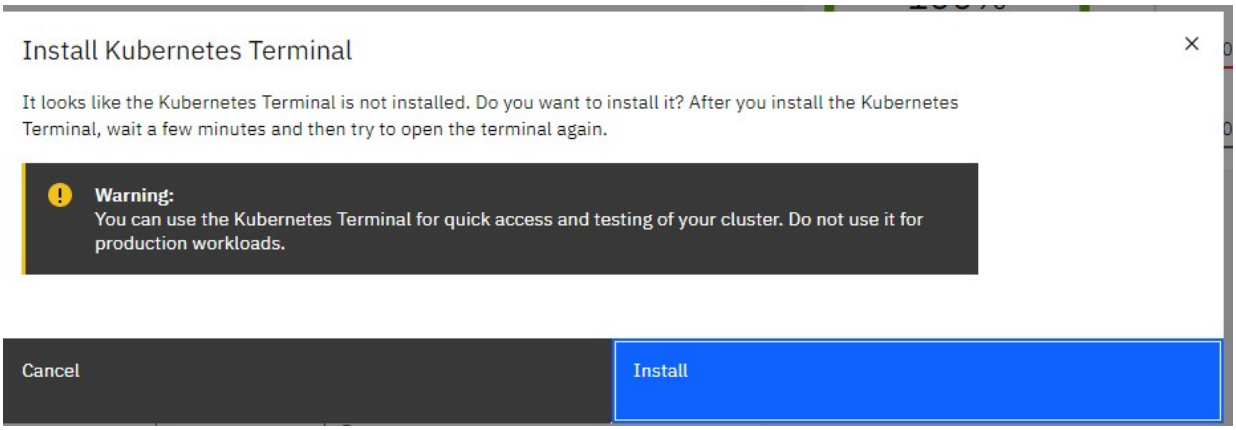
- Go to Resources in your browser
- Click on **Clusters**
- Click on your Cluster



- Now you are at you clusters overview, here Click on **Actions** and **Web terminal** from the dropdown menu



- Click **install** - wait couple of minutes



- Click on **Actions**
- Click **Web terminal** --> a terminal will open up
- **Type** in the terminal, please change NAMESPACE to the namespace you choose at the deployment setup:

```
$ kubectl get ns
```

```
tabatip@k8s-terminal ~ (⎈ mycluster-ams03-u2c.2x4/budvscbf0h42sg93uld0:default)$ kubectl get ns
NAME              STATUS   AGE
default           Active   103m
ibm-cert-store    Active   97m
ibm-operators     Active   98m
ibm-system        Active   102m
kube-node-lease   Active   103m
kube-public       Active   103m
kube-system       Active   103m
mariadb           Active   2m10s
mediawiki         Active   81m
mongodb           Active   67m
storage-plugin    Active   90m
tabatip@k8s-terminal ~ (⎈ mycluster-ams03-u2c.2x4/budvscbf0h42sg93uld0:default)$
```

```
$ kubectl get pod -n NAMESPACE -o wide
```

```
tabatip@k8s-terminal ~ (⎈ mycluster-ams03-u2c.2x4/budvscbf0h42sg93uld0:default)$ kubectl get pods -n mariadb -o wide
NAME                          READY   STATUS    RESTARTS   AGE     IP               NODE             NOMINATED NODE   READINESS GATES
mariadb-galera9a-b9ae-44-0    1/1     Running   0          121m    172.30.108.138   10.136.164.95    <none>           <none>
mariadb-galera9a-b9ae-44-1    1/1     Running   0          118m    172.30.42.205    10.136.164.79    <none>           <none>
mariadb-galera9a-b9ae-44-2    1/1     Running   0          116m    172.30.108.139   10.136.164.95    <none>           <none>
```

```
$ kubectl get service -n NAMESPACE
```

```
tabatip@k8s-terminal ~ (⎈ mycluster-ams03-u2c.2x4/budvscbf0h42sg93uld0:default)$ kubectl get svc -n mariadb
NAME                           TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                      AGE
mariadb-galera9a-b9ae-44       ClusterIP   172.21.86.90    <none>        3306/TCP                     122m
mariadb-galera9a-b9ae-44-headless   ClusterIP   None            <none>        4567/TCP,4568/TCP,4444/TCP   122m
```

- Enter your pod with bash , please replace PODNAME with your mariadb pod's name

```
$ kubectl exec --stdin --tty PODNAME -n NAMESPACE -- /bin/bash
```

```
tabatip@k8s-terminal ~ (⎈ mycluster-ams03-u2c.2x4/budvscbf0h42sg93uld0:default)$ kubectl exec --stdin  --tty mariadb-galera9a-b9ae-44-0 -n mariadb --
/bin/bash
```

- After you are in your pod , please verify that mariadb is running on your pods cluster. Please enter the root password after the prompt

```
mysql -u root -p -e "SHOW STATUS LIKE 'wsrep_cluster_size'"
```

```
I have no name!@mariadb-galera9a-b9ae-44-0:/$ mysql -u root -p -e "SHOW STATUS LIKE 'wsrep_cluster_size'"
Enter password:
+--------------------+-------+
| Variable_name      | Value |
+--------------------+-------+
| wsrep_cluster_size | 3     |
+--------------------+-------+
I have no name!@mariadb-galera9a-b9ae-44-0:/$
```

You have succesfully deployed MariaDB Galera on IBM Cloud!

3.3 Optimization and Tuning

Articles on how to get the most out of MariaDB, including new features.

Hardware Optimization

Better performance with hardware improvements

Operating System Optimizations

Optimizations at the OS level

Optimization and Indexes

Using indexes to improve query performance

Query Optimizations

Getting queries running more optimally

Optimizing Tables

Different ways to optimize tables and data on disk

MariaDB Memory Allocation
Basic issues in RAM allocation for MariaDB.

System Variables
Understanding, optimizing and tuning the server system variables

Buffers, Caches and Threads
Buffering, caching, thread pool to improve performance

Optimizing Data Structure
Designing the most optimal schemas, tables, and columns

MariaDB Internal Optimizations
Different optimizations strategies done internally in MariaDB

Benchmarking
Various benchmark results for MariaDB.

Compression
Types of compression in MariaDB

There are 11 related questions.

3.3.1 Hardware Optimization

Better hardware is one of the easiest ways to improve performance.

As a general rule of thumb, hardware should be improved in the following order:

Memory

Memory is the most important factor as it allows you to adjust the Server System Variables. More memory means larger key and table caches can be stored in memory so that disk access, an order of magnitude slower, is reduced.

Simply adding more memory may not result in drastic improvements if the server variables are not set to make use of the extra available memory.

Using more RAM slots on the motherboard increases the bus frequency, and there will be more latency between the RAM and the CPU. So, using the highest RAM size per slot is preferable.

Disks

Fast disk access is critical, as ultimately it's where the data resides. The key figure is the disk seek time, a measurement of how fast the physical disk can move to access the data, so choose disks with as low a seek time as possible.

You can also add dedicated disks for temporary files and transaction logs.

Fast Ethernet

CPU

Although hardware bottlenecks often fall elsewhere, faster processors allow calculations to be performed more quickly, and the results sent back to the client more quickly. Besides processor speed, the processor's bus speed and cache size are also important factors to consider.

3.3.2 Operating System Optimizations

Between the hardware and MariaDB sits the operating system, and there are a number of optimizations that can be made at this level

Configuring Linux for MariaDB

Linux kernel settings IO scheduler For optimal IO performance running a da...

Configuring Swappiness

Setting Linux swappiness.

Filesystem Optimizations

Which filesystem is best? The filesystem is not the most important aspect ...

There are 1 related questions.

2.1.7.2 Configuring Linux for MariaDB

2.1.7.4 Configuring Swappiness

3.3.2.3 Filesystem Optimizations

Contents

Which filesystem is best?

The filesystem is not the most important aspect of MariaDB performance. Far more important are available RAM, drive speed, the system variable settings (see Hardware Optimization and System Variables).

Optimizing the filesystem can however in some cases make a noticeable difference. Currently, the best Linux filesystems are generally regarded as ext4, XFS and Btrfs. They are all included in the mainline Linux kernel, and are widely supported and available on most Linux distributions. Red Hat though regards Brtfs as a technology preview, not yet ready for production systems.

The following theoretical file size and filesystem size limits apply to the three filesystems:

	ext4	XFS	Brtfs
Max file size	16TB	8EB	16EB
Max filesystem size	1 EB	8EB	16EB

Each has unique characteristics that are worth understanding to get the most from.

Disabling access time

It's unlikely you'll need to record file access time on a database server, and mounting your filesystem with this disabled can give an easy improvement in performance. To do so, use the `noatime` option.

If you want to keep access time for log files or other system files, these can be stored on a separate drive.

3.3.3 Optimization and Indexes

A critical way to improve table performance is by creating indexes on key columns.

The Essentials of an Index

Explains the basics of a table index.

Getting Started with Indexes

Extensive tutorial on creating indexes for tables.

Full-Text Indexes

MariaDB has support for full-text indexing and searching.

ANALYZE TABLE
Store key distributions for a table.

Building the best INDEX for a given SELECT
Cookbook for Creating Indexes

Compound (Composite) Indexes
Compound indexes plus other insights into the mysteries of indexing

EXPLAIN
EXPLAIN returns information about index usage, as well as being a synonym for DESCRIBE.

Foreign Keys
Foreign keys can be used to enforce data integrity.

Ignored Indexes
Indexes that are not used by the optimizer.

Index Statistics
Index statistics and the query optimizer.

Latitude/Longitude Indexing
Efficiently finding the nearest 10 pizza parlors in a huge database

Primary Keys with Nullable Columns
SQL standards in dealing with multi-part primary keys with nullable columns.

SHOW EXPLAIN
Shows an execution plan for a running query.

SPATIAL INDEX
An index type used for geometric columns.

Storage Engine Index Types
The permitted index_types for each storage engine.

There are 4 related questions.

6.2.6 The Essentials of an Index

3.3.3.2 Getting Started with Indexes

Contents

For a very basic overview, see The Essentials of an Index.

There are four main kinds of indexes; primary keys (unique and not null), unique indexes (unique and can be null), plain indexes (not necessarily unique) and full-text indexes (for full-text searching).

The terms 'KEY' and 'INDEX' are generally used interchangeably, and statements should work with either keyword.

Primary Key

A primary key is unique and can never be null. It will always identify only one record, and each record must be represented. Each table can only have one primary key.

In InnoDB tables, all indexes contain the primary key as a suffix. Thus, when using this storage engine, keeping the primary key as small as possible is particularly important. If a primary key does not exist and there are no UNIQUE indexes, InnoDB creates a 6-bytes clustered index which is invisible to the user.

Many tables use a numeric ID field as a primary key. The AUTO_INCREMENT attribute can be used to generate a unique identity for new rows, and is commonly-used with primary keys.

Primary keys are usually added when the table is created with the CREATE TABLE statement. For example, the following creates a primary key on the ID field. Note that the ID field had to be defined as NOT NULL, otherwise the index could not have been created.

```
CREATE TABLE `Employees` (
  `ID` TINYINT(3) UNSIGNED NOT NULL AUTO_INCREMENT,
  `First_Name` VARCHAR(25) NOT NULL,
  `Last_Name` VARCHAR(25) NOT NULL,
  `Position` VARCHAR(25) NOT NULL,
  `Home_Address` VARCHAR(50) NOT NULL,
  `Home_Phone` VARCHAR(12) NOT NULL,
  PRIMARY KEY (`ID`)
) ENGINE=Aria;
```

You cannot create a primary key with the CREATE INDEX command. If you do want to add one after the table has already been created, use ALTER TABLE, for example:

```
ALTER TABLE Employees ADD PRIMARY KEY(ID);
```

Finding Tables Without Primary Keys

Tables in the information_schema database can be queried to find tables that do not have primary keys. For example, here is a query using the TABLES and KEY_COLUMN_USAGE tables that can be used:

```
SELECT t.TABLE_SCHEMA, t.TABLE_NAME
FROM information_schema.TABLES AS t
LEFT JOIN information_schema.KEY_COLUMN_USAGE AS c
ON t.TABLE_SCHEMA = c.CONSTRAINT_SCHEMA
   AND t.TABLE_NAME = c.TABLE_NAME
   AND c.CONSTRAINT_NAME = 'PRIMARY'
WHERE t.TABLE_SCHEMA != 'information_schema'
   AND t.TABLE_SCHEMA != 'performance_schema'
   AND t.TABLE_SCHEMA != 'mysql'
   AND c.CONSTRAINT_NAME IS NULL;
```

Unique Index

A Unique Index must be unique, but it can have columns that may be NULL. So each key value identifies only one record, but not each record needs to be represented.

MariaDB starting with 10.5

Unique, if index type is not specified, is normally a BTREE index that can also be used by the optimizer to find rows. If the key is longer than the max key length for the used storage engine and the storage engine supports long unique index, a HASH key will be created. This enables MariaDB to enforce uniqueness for any type or number of columns.

For example, to create a unique key on the Employee_Code field, as well as a primary key, use:

```
CREATE TABLE `Employees` (
  `ID` TINYINT(3) UNSIGNED NOT NULL,
  `First_Name` VARCHAR(25) NOT NULL,
  `Last_Name` VARCHAR(25) NOT NULL,
  `Position` VARCHAR(25) NOT NULL,
  `Home_Address` VARCHAR(50) NOT NULL,
  `Home_Phone` VARCHAR(12) NOT NULL,
  `Employee_Code` VARCHAR(25) NOT NULL,
  PRIMARY KEY (`ID`),
  UNIQUE KEY (`Employee_Code`)
) ENGINE=Aria;
```

Unique keys can also be added after the table is created with the CREATE INDEX command, or with the ALTER TABLE command, for example:

```
ALTER TABLE Employees ADD UNIQUE `EmpCode`(`Employee_Code`);
```

and

```
CREATE UNIQUE INDEX HomePhone ON Employees(Home_Phone);
```

Indexes can contain more than one column. MariaDB is able to use one or more columns on the leftmost part of the index, if it cannot use the whole index. (except for the HASH index type).

Take another example:

```
CREATE TABLE t1 (a INT NOT NULL, b INT, UNIQUE (a,b));

INSERT INTO t1 values (1,1), (2,2);

SELECT * FROM t1;
+---+------+
| a | b    |
+---+------+
| 1 |    1 |
| 2 |    2 |
+---+------+
```

Since the index is defined as unique over both columns *a* and *b*, the following row is valid, as while neither *a* nor *b* are unique on their own, the combination is unique:

```
INSERT INTO t1 values (2,1);

SELECT * FROM t1;
+---+------+
| a | b    |
+---+------+
| 1 |    1 |
| 2 |    1 |
| 2 |    2 |
+---+------+
```

The fact that a `UNIQUE` constraint can be `NULL` is often overlooked. In SQL any `NULL` is never equal to anything, not even to another `NULL`. Consequently, a `UNIQUE` constraint will not prevent one from storing duplicate rows if they contain null values:

```
INSERT INTO t1 values (3,NULL), (3, NULL);

SELECT * FROM t1;
+---+------+
| a | b    |
+---+------+
| 1 |    1 |
| 2 |    1 |
| 2 |    2 |
| 3 | NULL |
| 3 | NULL |
+---+------+
```

Indeed, in SQL two last rows, even if identical, are not equal to each other:

```
SELECT (3, NULL) = (3, NULL);

+----------------------+
| (3, NULL) = (3, NULL) |
+----------------------+
| 0                    |
+----------------------+
```

In MariaDB you can combine this with virtual columns to enforce uniqueness over a subset of rows in a table:

```
create table Table_1 (
  user_name varchar(10),
  status enum('Active', 'On-Hold', 'Deleted'),
  del char(0) as (if(status in ('Active', 'On-Hold'),'', NULL)) persistent,
  unique(user_name,del)
)
```

This table structure ensures that all *active* or *on-hold* users have distinct names, but as soon as a user is *deleted*, his name is no longer part of the uniqueness constraint, and another user may get the same name.

If a unique index consists of a column where trailing pad characters are stripped or ignored, inserts into that column where values differ only by the number of trailing pad characters will result in a duplicate-key error.

MariaDB starting with 10.5

For some engines, like InnoDB, `UNIQUE` can be used with any type of columns or any number of columns.

```
create table t1 (a int primary key,
b blob,
c1 varchar(1000),
c2 varchar(1000),
c3 varchar(1000),
c4 varchar(1000),
c5 varchar(1000),
c6 varchar(1000),
c7 varchar(1000),
c8 varchar(1000),
c9 varchar(1000),
unique key `b` (b),
unique key `all_c` (c1,c2,c3,c4,c6,c7,c8,c9)) engine=myisam;
```

If the key length is longer than the max key length supported by the engine, a HASH key will be created. This can be seen with `SHOW CREATE TABLE table_name` or `SHOW INDEX FROM table_name`:

```
show create table t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE `t1` (
  `a` int(11) NOT NULL,
  `b` blob DEFAULT NULL,
  `c1` varchar(1000) DEFAULT NULL,
  `c2` varchar(1000) DEFAULT NULL,
  `c3` varchar(1000) DEFAULT NULL,
  `c4` varchar(1000) DEFAULT NULL,
  `c5` varchar(1000) DEFAULT NULL,
  `c6` varchar(1000) DEFAULT NULL,
  `c7` varchar(1000) DEFAULT NULL,
  `c8` varchar(1000) DEFAULT NULL,
  `c9` varchar(1000) DEFAULT NULL,
  PRIMARY KEY (`a`),
  UNIQUE KEY `b` (`b`) USING HASH,
  UNIQUE KEY `all_c` (`c1`,`c2`,`c3`,`c4`,`c6`,`c7`,`c8`,`c9`) USING HASH
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_swedish_ci
```

Plain Indexes

Indexes do not necessarily need to be unique. For example:

```
CREATE TABLE t2 (a INT NOT NULL, b INT, INDEX (a,b));

INSERT INTO t2 values (1,1), (2,2), (2,2);

SELECT * FROM t2;
+---+------+
| a | b    |
+---+------+
| 1 |    1 |
| 2 |    2 |
| 2 |    2 |
+---+------+
```

Full-Text Indexes

Full-text indexes support full-text indexing and searching. See the Full-Text Indexes section.

Choosing Indexes

In general you should only add indexes to match the queries your application uses. Any extra will waste resources. In an application with very small tables, indexes will not make much difference but as soon as your tables are larger than your buffer sizes the indexes will start to speed things up dramatically.

Using the EXPLAIN statement on your queries can help you decide which columns need indexing.

If you query contains something like `LIKE '%word%'`, without a fulltext index you are using a full table scan every time, which is very slow.

If your table has a large number of reads and writes, consider using delayed writes. This uses the db engine in a "batch" write mode, which cuts down on disk io, therefore increasing performance.

Use the CREATE INDEX command to create an index.

If you are building a large table then for best performance add the index after the table is populated with data. This is to increase the insert performance and remove the index overhead during inserts.

Viewing Indexes

You can view which indexes are present on a table, as well as details about them, with the SHOW INDEX statement.

If you want to know how to re-create an index, run `SHOW CREATE TABLE`.

When to Remove an Index

If an index is rarely used (or not used at all) then remove it to increase INSERT, and UPDATE performance.

If user statistics are enabled, the Information Schema INDEX_STATISTICS table stores the index usage.

If the slow query log is enabled and the `log_queries_not_using_indexes` server system variable is `ON`, the queries which do not use indexes are logged.

The initial version of this article was copied, with permission, from http://hashmysql.org/wiki/Proper_Indexing_Strategy on 2012-10-30.

3.3.3.3 Full-Text Indexes

MariaDB has support for full-text indexing and searching.

Full-Text Index Overview
Full-text indexing and searching overview.

Full-Text Index Stopwords
Default list of full-text stopwords used by MATCH...AGAINST.

MATCH AGAINST
Perform a fulltext search on a fulltext index.

3.3.3.3.1 Full-Text Index Overview

Contents

MariaDB has support for full-text indexing and searching:

- A full-text index in MariaDB is an index of type FULLTEXT, and it allows more options when searching for portions of text from a field.
- Full-text indexes can be used only with MyISAM, Aria, InnoDB and Mroonga tables, and can be created only for CHAR, VARCHAR, or TEXT columns.
- Partitioned tables cannot contain fulltext indexes, even if the storage engine supports them.
- A FULLTEXT index definition can be given in the CREATE TABLE statement when a table is created, or added later using ALTER TABLE or CREATE INDEX.
- For large data sets, it is much faster to load your data into a table that has no FULLTEXT index and then create the index after that, than to load data into a table that has an existing FULLTEXT index.

Full-text searching is performed using MATCH() ... AGAINST syntax. MATCH() takes a comma-separated list that names the columns to be searched. AGAINST takes a string to search for, and an optional modifier that indicates what type of search to perform. The search string must be a literal string, not a variable or a column name.

```
MATCH (col1,col2,...) AGAINST (expr [search_modifier])
```

Excluded Results

- Partial words are excluded.
- Words less than 4 (MyISAM) or 3 (InnoDB) characters in length will not be stored in the fulltext index. This value can be adjusted by changing the ft_min_word_length system variable (or, for InnoDB, innodb_ft_min_token_size).
- Words longer than 84 characters in length will also not be stored in the fulltext index. This values can be adjusted by changing the ft_max_word_length system variable (or, for InnoDB, innodb_ft_max_token_size).
- Stopwords are a list of common words such as "once" or "then" that do not reflect in the search results unless IN BOOLEAN MODE is used. The stopword list for MyISAM/Aria tables and InnoDB tables can differ. See stopwords for details and a full list, as well as for details on how to change the default list.
- For MyISAM/Aria fulltext indexes only, if a word appears in more than half the rows, it is also excluded from the results of a fulltext search.
- For InnoDB indexes, only committed rows appear - modifications from the current transaction do not apply.

Relevance

MariaDB calculates a relevance for each result, based on a number of factors, including the number of words in the index, the number of unique words in a row, the total number of words in both the index and the result, and the weight of the word. In English, 'cool' will be weighted less than 'dandy', at least at present! The relevance can be returned as part of a query simply by using the MATCH function in the field list.

Types of Full-Text search

IN NATURAL LANGUAGE MODE

IN NATURAL LANGUAGE MODE is the default type of full-text search, and the keywords can be omitted. There are no special operators, and searches consist of one or more comma-separated keywords.

Searches are returned in descending order of relevance.

IN BOOLEAN MODE

Boolean search permits the use of a number of special operators:

Operator	Description
+	The word is mandatory in all rows returned.
-	The word cannot appear in any row returned.
<	The word that follows has a lower relevance than other words, although rows containing it will still match
>	The word that follows has a higher relevance than other words.
()	Used to group words into subexpressions.
~	The word following contributes negatively to the relevance of the row (which is different to the '-' operator, which specifically excludes the word, or the '<' operator, which still causes the word to contribute positively to the relevance of the row.
*	The wildcard, indicating zero or more characters. It can only appear at the end of a word.
"	Anything enclosed in the double quotes is taken as a whole (so you can match phrases, for example).

Searches are not returned in order of relevance, and nor does the 50% limit apply. Stopwords and word minimum and maximum lengths still apply as usual.

WITH QUERY EXPANSION

A query expansion search is a modification of a natural language search. The search string is used to perform a regular natural language search. Then, words from the most relevant rows returned by the search are added to the search string and the search is done again. The query returns the rows from the second search. The IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION or WITH QUERY EXPANSION modifier specifies a query expansion search. It can be useful when relying on implied knowledge within the data, for example that MariaDB is a database.

Examples

Creating a table, and performing a basic search:

```
CREATE TABLE ft_myisam(copy TEXT,FULLTEXT(copy)) ENGINE=MyISAM;

INSERT INTO ft_myisam(copy) VALUES ('Once upon a time'),
  ('There was a wicked witch'), ('Who ate everybody up');

SELECT * FROM ft_myisam WHERE MATCH(copy) AGAINST('wicked');
+--------------------------+
| copy                     |
+--------------------------+
| There was a wicked witch |
+--------------------------+
```

Multiple words:

```
SELECT * FROM ft_myisam WHERE MATCH(copy) AGAINST('wicked,witch');
+--------------------------------+
| copy                           |
+--------------------------------+
| There was a wicked witch       |
+--------------------------------+
```

Since 'Once' is a stopword, no result is returned:

```
SELECT * FROM ft_myisam WHERE MATCH(copy) AGAINST('Once');
Empty set (0.00 sec)
```

Inserting the word 'wicked' into more than half the rows excludes it from the results:

```
INSERT INTO ft_myisam(copy) VALUES ('Once upon a wicked time'),
  ('There was a wicked wicked witch'), ('Who ate everybody wicked up');

SELECT * FROM ft_myisam WHERE MATCH(copy) AGAINST('wicked');
Empty set (0.00 sec)
```

Using IN BOOLEAN MODE to overcome the 50% limitation:

```
SELECT * FROM ft_myisam WHERE MATCH(copy) AGAINST('wicked' IN BOOLEAN MODE);
+-------------------------------+
| copy                          |
+-------------------------------+
| There was a wicked witch      |
| Once upon a wicked time       |
| There was a wicked wicked witch |
| Who ate everybody wicked up   |
+-------------------------------+
```

Returning the relevance:

```
SELECT copy,MATCH(copy) AGAINST('witch') AS relevance
  FROM ft_myisam WHERE MATCH(copy) AGAINST('witch');
+-------------------------------+--------------------+
| copy                          | relevance          |
+-------------------------------+--------------------+
| There was a wicked witch      | 0.6775632500648499 |
| There was a wicked wicked witch | 0.5031757950782776 |
+-------------------------------+--------------------+
```

WITH QUERY EXPANSION. In the following example, 'MariaDB' is always associated with the word 'database', so it is returned when query expansion is used, even though not explicitly requested.

```
CREATE TABLE ft2(copy TEXT,FULLTEXT(copy)) ENGINE=MyISAM;

INSERT INTO ft2(copy) VALUES
 ('MySQL vs MariaDB database'),
 ('Oracle vs MariaDB database'),
 ('PostgreSQL vs MariaDB database'),
 ('MariaDB overview'),
 ('Foreign keys'),
 ('Primary keys'),
 ('Indexes'),
 ('Transactions'),
 ('Triggers');

SELECT * FROM ft2 WHERE MATCH(copy) AGAINST('database');
+-------------------------------+
| copy                          |
+-------------------------------+
| MySQL vs MariaDB database     |
| Oracle vs MariaDB database    |
| PostgreSQL vs MariaDB database |
+-------------------------------+
3 rows in set (0.00 sec)

SELECT * FROM ft2 WHERE MATCH(copy) AGAINST('database' WITH QUERY EXPANSION);
+-------------------------------+
| copy                          |
+-------------------------------+
| MySQL vs MariaDB database     |
| Oracle vs MariaDB database    |
| PostgreSQL vs MariaDB database |
| MariaDB overview              |
+-------------------------------+
4 rows in set (0.00 sec)
```

Partial word matching with IN BOOLEAN MODE:

```
SELECT * FROM ft2 WHERE MATCH(copy) AGAINST('Maria*' IN BOOLEAN MODE);
+-------------------------------+
| copy                          |
+-------------------------------+
| MySQL vs MariaDB database     |
| Oracle vs MariaDB database    |
| PostgreSQL vs MariaDB database |
| MariaDB overview              |
+-------------------------------+
```

Using boolean operators

```
SELECT * FROM ft2 WHERE MATCH(copy) AGAINST('+MariaDB -database'
  IN BOOLEAN MODE);
+------------------+
| copy             |
+------------------+
| MariaDB overview |
+------------------+
```

3.3.3.3.2 Full-Text Index Stopwords

Contents

Stopwords are used to provide a list of commonly-used words that can be ignored for the purposes of Full-text-indexes.

Full-text indexes built in MyISAM and InnoDB have different stopword lists by default.

MyISAM Stopwords

For full-text indexes on MyISAM tables, by default, the list is built from the file `storage/myisam/ft_static.c`, and searched using the server's character set and collation. The ft_stopword_file system variable allows the default list to be overridden with words from another file, or for stopwords to be ignored altogether.

If the stopword list is changed, any existing full-text indexes need to be rebuilt

The following table shows the default list of stopwords, although you should always treat `storage/myisam/ft_static.c` as the definitive list. See the Fulltext Index Overview for more details, and Full-text-indexes for related articles.

a's	able	about	above
according	accordingly	across	actually
after	afterwards	again	against
ain't	all	allow	allows
almost	alone	along	already
also	although	always	am
among	amongst	an	and
another	any	anybody	anyhow
anyone	anything	anyway	anyways
anywhere	apart	appear	appreciate
appropriate	are	aren't	around
as	aside	ask	asking
associated	at	available	away
awfully	be	became	because
become	becomes	becoming	been
before	beforehand	behind	being
believe	below	beside	besides
best	better	between	beyond
both	brief	but	by
c'mon	c's	came	can
can't	cannot	cant	cause
causes	certain	certainly	changes
clearly	co	com	come

comes	concerning	consequently	consider
considering	contain	containing	contains
corresponding	could	couldn't	course
currently	definitely	described	despite
did	didn't	different	do
does	doesn't	doing	don't
done	down	downwards	during
each	edu	eg	eight
either	else	elsewhere	enough
entirely	especially	et	etc
even	ever	every	everybody
everyone	everything	everywhere	ex
exactly	example	except	far
few	fifth	first	five
followed	following	follows	for
former	formerly	forth	four
from	further	furthermore	get
gets	getting	given	gives
go	goes	going	gone
got	gotten	greetings	had
hadn't	happens	hardly	has
hasn't	have	haven't	having
he	he's	hello	help
hence	her	here	here's
hereafter	hereby	herein	hereupon
hers	herself	hi	him
himself	his	hither	hopefully
how	howbeit	however	i'd
i'll	i'm	i've	ie
if	ignored	immediate	in
inasmuch	inc	indeed	indicate
indicated	indicates	inner	insofar
instead	into	inward	is
isn't	it	it'd	it'll
it's	its	itself	just
keep	keeps	kept	know
knows	known	last	lately
later	latter	latterly	least
less	lest	let	let's
like	liked	likely	little
look	looking	looks	ltd
mainly	many	may	maybe
me	mean	meanwhile	merely

might	more	moreover	most
mostly	much	must	my
myself	name	namely	nd
near	nearly	necessary	need
needs	neither	never	nevertheless
new	next	nine	no
nobody	non	none	noone
nor	normally	not	nothing
novel	now	nowhere	obviously
of	off	often	oh
ok	okay	old	on
once	one	ones	only
onto	or	other	others
otherwise	ought	our	ours
ourselves	out	outside	over
overall	own	particular	particularly
per	perhaps	placed	please
plus	possible	presumably	probably
provides	que	quite	qv
rather	rd	re	really
reasonably	regarding	regardless	regards
relatively	respectively	right	said
same	saw	say	saying
says	second	secondly	see
seeing	seem	seemed	seeming
seems	seen	self	selves
sensible	sent	serious	seriously
seven	several	shall	she
should	shouldn't	since	six
so	some	somebody	somehow
someone	something	sometime	sometimes
somewhat	somewhere	soon	sorry
specified	specify	specifying	still
sub	such	sup	sure
t's	take	taken	tell
tends	th	than	thank
thanks	thanx	that	that's
thats	the	their	theirs
them	themselves	then	thence
there	there's	thereafter	thereby
therefore	therein	theres	thereupon
these	they	they'd	they'll
they're	they've	think	third

this	thorough	thoroughly	those
though	three	through	throughout
thru	thus	to	together
too	took	toward	towards
tried	tries	truly	try
trying	twice	two	un
under	unfortunately	unless	unlikely
until	unto	up	upon
us	use	used	useful
uses	using	usually	value
various	very	via	viz
vs	want	wants	was
wasn't	way	we	we'd
we'll	we're	we've	welcome
well	went	were	weren't
what	what's	whatever	when
whence	whenever	where	where's
whereafter	whereas	whereby	wherein
whereupon	wherever	whether	which
while	whither	who	who's
whoever	whole	whom	whose
why	will	willing	wish
with	within	without	won't
wonder	would	wouldn't	yes
yet	you	you'd	you'll
you're	you've	your	yours
yourself	yourselves	zero	

InnoDB Stopwords

Stopwords on full-text indexes are only enabled if the innodb_ft_enable_stopword system variable is set (by default it is) at the time the index was created.

The stopword list is determined as follows:

- If the innodb_ft_user_stopword_table system variable is set, that table is used as a stopword list.
- If `innodb_ft_user_stopword_table` is not set, the table set by innodb_ft_server_stopword_table is used.
- If neither variable is set, the built-in list is used, which can be viewed by querying the INNODB_FT_DEFAULT_STOPWORD table in the Information Schema.

In the first two cases, the specified table must exist at the time the system variable is set and the full-text index created. It must be an InnoDB table with a single column, a VARCHAR named VALUE.

The default InnoDB stopword list differs from the default MyISAM list, being much shorter, and contains the following words:

a	about	an	are
as	at	be	by
com	de	en	for
from	how	i	in
is	it	la	of

on	or	that	the
this	to	was	what
when	where	who	will
with	und	the	www

1.2.2.36 MATCH AGAINST

1.3.8.6 myisam_ftdump

1.1.1.2.1.2 ANALYZE TABLE

3.3.3.5 Building the best INDEX for a given SELECT

Contents

The problem

You have a SELECT and you want to build the best INDEX for it. This blog is a "cookbook" on how to do that task.

- A short algorithm that works for many simpler SELECTs and helps in complex queries.
- Examples of the algorithm, plus digressions into exceptions and variants
- Finally a long list of "other cases".

The hope is that a newbie can quickly get up to speed, and his/her INDEXes will no longer smack of "newbie".

Many edge cases are explained, so even an expert may find something useful here.

Algorithm

Here's the way to approach creating an INDEX, given a SELECT. Follow the steps below, gathering columns to put in the INDEX in order. When the steps give out, you usually have the 'perfect' index.

1. Given a WHERE with a bunch of expressions connected by AND: Include the columns (if any), in any order, that are compared to a constant and not hidden in a function. 2. You get one more chance to add to the INDEX; do the first of these that applies:

- 2a. One column used in a 'range' -- BETWEEN, '>', LIKE w/o leading wildcard, etc.
- 2b. All columns, in order, of the GROUP BY.
- 2c. All columns, in order, of the ORDER BY if there is no mixing of ASC and DESC.

Digression

This blog assumes you know the basic idea behind having an INDEX. Here is a refresher on some of the key points.

Virtually all INDEXes in MySQL are structured as BTrees BTrees allow very efficient for

- Given a key, find the corresponding row(s);
- "Range scans" -- That is start at one value for the key and repeatedly find the "next" (or "previous") row.

A PRIMARY KEY is a UNIQUE KEY; a UNIQUE KEY is an INDEX. ("KEY" == "INDEX".)

InnoDB "clusters" the PRIMARY KEY with the data. Hence, given the value of the PK ("PRIMARY KEY"), after drilling down the BTree to find the index entry, you have all the columns of the row when you get there. A "secondary key" (any UNIQUE or INDEX other than the PK) in InnoDB first drills down the BTree for the secondary index, where it finds a copy of the PK. Then it drills down the PK to find the row.

Every InnoDB table has a PRIMARY KEY. While there is a default if you do not specify one, it is best to explicitly provide a PK.

For completeness: MyISAM works differently. All indexes (including the PK) are in separate BTrees. The leaf node of such BTrees have a pointer (usually a byte offset) into the data file.

All discussion here assumes InnoDB tables, however most statements apply to other Engines.

First, some examples

Think of a list of names, sorted by last_name, then first_name. You have undoubtedly seen such lists, and they often have other information such as address and phone number. Suppose you wanted to look me up. If you remember my full name ('James' and 'Rick'), it is easy to find my entry. If you remembered only my last name ('James') and first initial ('R'). You would quickly zoom in on the Jameses and find the Rs in them. There, you might remember 'Rick' and ignore 'Ronald'. But, suppose you remembered my first name ('Rick') and only my last initial ('J'). Now you are in trouble. You would be scanning all the Js -- Jones, Rick; Johnson, Rick; Jamison, Rick; etc, etc. That's much less efficient.

Those equate to

```
INDEX(last_name, first_name) -- the order of the list.
WHERE last_name = 'James' AND first_name = 'Rick'  -- best case
WHERE last_name = 'James' AND first_name LIKE 'R%' -- pretty good
WHERE last_name LIKE 'J%' AND first_name = 'Rick'  -- pretty bad
```

Think about this example as I talk about "=" versus "range" in the Algorithm, below.

Algorithm, step 1 (WHERE "column = const")

- `WHERE aaa = 123 AND ...` : an INDEX starting with aaa is good.
- `WHERE aaa = 123 AND bbb = 456 AND ...` : an INDEX starting with aaa and bbb is good. In this case, it does not matter whether aaa or bbb comes first in the INDEX.
- `xxx IS NULL` : this acts like "= const" for this discussion.
- `WHERE t1.aa = 123 AND t2.bb = 456` -- You must only consider columns in the current table.

Note that the expression must be of the form of `column_name` = (constant). These do not apply to this step in the Algorithm: DATE(dt) = '...', LOWER(s) = '...', CAST(s ...) = '...', x='...' COLLATE...

(If there are no "=" parts AND'd in the WHERE clause, move on to step 2 without any columns in your putative INDEX.)

Algorithm, step 2

Find the first of 2a / 2b / 2c that applies; use it; then quit. If none apply, then you are through gathering columns for the index.

In some cases it is optimal to do step 1 (all equals) plus step 2c (ORDER BY).

Algorithm, step 2a (one range)

A "range" shows up as

- `aaa >= 123` -- any of <, <=, >=, >; but not <>, !=
- `aaa BETWEEN 22 AND 44`
- `sss LIKE 'blah%'` -- but not `sss LIKE '%blah'`
- `xxx IS NOT NULL` Add the column in the range to your putative INDEX.

If there are more parts to the WHERE clause, you must stop now.

Complete examples (assume nothing else comes after the snippet)

- `WHERE aaa >= 123 AND bbb = 1 ⇒ INDEX(bbb, aaa)` (WHERE order does not matter; INDEX order does)
- `WHERE aaa >= 123 ⇒ INDEX(aaa)`
- `WHERE aaa >= 123 AND ccc > 'xyz' ⇒ INDEX(aaa) or INDEX(ccc)` (only one range)
- `WHERE aaa >= 123 ORDER BY aaa ⇒ INDEX(aaa)` -- Bonus: The ORDER BY will use the INDEX.
- `WHERE aaa >= 123 ORDER BY aaa ⇒ INDEX(aaa) DESC` -- Same Bonus.

Algorithm, step 2b (GROUP BY)

If there is a GROUP BY, all the columns of the GROUP BY should now be added, in the specified order, to the INDEX you are building. (I do not know what happens if one of the columns is already in the INDEX.)

If you are GROUPing BY an expression (including function calls), you cannot use the GROUP BY; stop.

Complete examples (assume nothing else comes after the snippet)

- `WHERE aaa = 123 AND bbb = 1 GROUP BY ccc ⇒ INDEX(bbb, aaa, ccc) or INDEX(aaa, bbb, ccc)` (='s first, in any order; then the GROUP BY)
- `WHERE aaa >= 123 GROUP BY xxx ⇒ INDEX(aaa)` (You should have stopped with Step 2a)
- `GROUP BY x,y ⇒ INDEX(x,y)` (no WHERE)
- `WHERE aaa = 123 GROUP BY xxx, (a+b) ⇒ INDEX(aaa)` -- expression in GROUP BY, so no use including even xxx.

Algorithm, step 2c (ORDER BY)

If there is a ORDER BY, all the columns of the ORDER BY should now be added, in the specified order, to the INDEX you are building.

If there are multiple columns in the ORDER BY, and there is a mixture of ASC and DESC, do not add the ORDER BY columns; they won't help; stop.

If you are ORDERing BY an expression (including function calls), you cannot use the ORDER BY; stop.

Complete examples (assume nothing else comes after the snippet)

- `WHERE aaa = 123 GROUP BY ccc ORDER BY ddd ⇒ INDEX(aaa, ccc)` -- should have stopped with Step 2b
- `WHERE aaa = 123 GROUP BY ccc ORDER BY ccc ⇒ INDEX(aaa, ccc)` -- the ccc will be used for both GROUP BY and ORDER BY
- `WHERE aaa = 123 ORDER BY xxx ASC, yyy DESC ⇒ INDEX(aaa)` -- mixture of ASC and DESC.

The following are especially good. Normally a LIMIT cannot be applied until after lots of rows are gathered and then sorted according to the ORDER BY. But, if the INDEX gets all they way through the ORDER BY, only (OFFSET + LIMIT) rows need to be gathered. So, in these cases, you win the lottery with your new index:

- `WHERE aaa = 123 GROUP BY ccc ORDER BY ccc LIMIT 10 ⇒ INDEX(aaa, ccc)`
- `WHERE aaa = 123 ORDER BY ccc LIMIT 10 ⇒ INDEX(aaa, ccc)`
- `ORDER BY ccc LIMIT 10 ⇒ INDEX(ccc)`
- `WHERE ccc > 432 ORDER BY ccc LIMIT 10 ⇒ INDEX(ccc)` -- This "range" is compatible with ORDER BY

(It does not make much sense to have a LIMIT without an ORDER BY, so I do not discuss that case.)

Algorithm end

You have collected a few columns; put them in INDEX and ADD that to the table. That will often produce a "good" index for the SELECT you have. Below are some other suggestions that may be relevant.

An example of the Algorithm being 'wrong':

```
    SELECT ... FROM t WHERE flag = true;
```

This would (according to the Algorithm) call for INDEX(flag). However, indexing a column that has two (or a small number of) values is almost always useless. This is called 'low cardinality'. The Optimizer would prefer to do a table scan than to bounce between the index BTree and the data.

On the other hand, the Algorithm is 'right' with

```
    SELECT ... FROM t WHERE flag = true AND date >= '2015-01-01';
```

That would call for a compound index starting with a flag: INDEX(flag, date). Such an index is likely to be very beneficial. And it is likely to be more beneficial than INDEX(date).

If your resulting INDEX include column(s) that are likely to be UPDATEd, note that the UPDATE will have extra work to remove a 'row' from one place in the INDEX's BTree and insert a 'row' back into the BTree. For example:

```
INDEX(x)
UPDATE t SET x = ... WHERE ...;
```

There are too many variables to say whether it is better to keep the index or to toss it.

In this case, shortening the index may may be beneficial:

```
INDEX(z, x)
UPDATE t SET x = ... WHERE ...;
```

Changing to INDEX(z) would make for less work for the UPDATE, but might hurt some SELECT. It depends on the frequency of each, plus many more factors.

Limitations

(There are exceptions to some of these.)

- You may not create an index bigger than 3KB.
- You may not include a column that equates to bigger than some value (767 bytes -- VARCHAR(255) CHARACTER SET utf8).
- You can deal with big fields using "prefix" indexing; but see below.
- You should not have more than 5 columns in an index. (This is just a Rule of Thumb; nothing prevents having more.)
- You should not have redundant indexes. (See below.)

Flags and low cardinality

`INDEX(flag)` is almost never useful if `flag` has very few values. More specifically, when you say WHERE flag = 1 and "1" occurs more than 20% of the time, such an index will be shunned. The Optimizer would prefer to scan the table instead of bouncing back and forth between the index and the data for more than 20% of the rows.

("20%" is really somewhere between 10% and 30%, depending on the phase of the moon.)

"Covering" indexes

A "Covering" index is an index that contains all the columns in the SELECT. It is special in that the SELECT can be completed by looking only at the INDEX BTree. (Since InnoDB's PRIMARY KEY is clustered with the data, "covering" is of no benefit when considering at the PRIMARY KEY.)

Mini-cookbook: 1. Gather the list of column(s) according to the "Algorithm", above. 2. Add to the end of the list the rest of the columns seen in the SELECT, in any order.

Examples:

- `SELECT x FROM t WHERE y = 5;` ⇒ `INDEX(y,x)` -- The algorithm said just INDEX(y)
- `SELECT x,z FROM t WHERE y = 5 AND q = 7;` ⇒ `INDEX(y,q,x,z)` -- y and q in either order (Algorithm), then x and z in either order (covering).
- `SELECT x FROM t WHERE y > 5 AND q > 7;` ⇒ `INDEX(y,q,x)` -- y or q first (that's as far as the Algorithm goes), then the other two fields afterwards.

The speedup you get might be minor, or it might be spectacular; it is hard to predict.

But...

- It is not wise to build an index with lots of columns. Let's cut it off at 5 (Rule of Thumb).
- Prefix indexes cannot 'cover', so don't use them anywhere in a 'covering' index.
- There are limits (3KB?) on how 'wide' an index can be, so "covering" may not be possible.

Redundant/excessive indexes

INDEX(a,b) can find anything that INDEX(a) could find. So you don't need both. Get rid of the shorter one.

If you have lots of SELECTs and they generate lots of INDEXes, this may cause a different problem. Each index must be updated (sooner or later) for each INSERT. More indexes ⇒ slower INSERTs. Limit the number of indexes on a table to about 6 (Rule of Thumb).

Notice in the cookbook how it says "in any order" in a few places. If, for example, you have both of these (in different SELECTs):

- `WHERE a=1 AND b=2` begs for either INDEX(a,b) or INDEX(b,a)
- `WHERE a>1 AND b=2` begs only for INDEX(b,a) Include only INDEX(b,a) since it handles both cases with only one INDEX.

Suppose you have a lot of indexes, including (a,b,c,dd) and (a,b,c,ee). Those are getting rather long. Consider either picking one of them, or having simply (a,b,c). Sometimes the selectivity of (a,b,c) is so good that tacking on 'dd' or 'ee' does make enough difference to matter.

Optimizer picks ORDER BY

The main cookbook skips over an important optimization that is sometimes used. The optimizer will sometimes ignore the WHERE and, instead, use an INDEX that matches the ORDER BY. This, of course, needs to be a perfect match -- all columns, in the same order. And all ASC or all DESC.

This becomes especially beneficial if there is a LIMIT.

But there is a problem. There could be two situations, and the Optimizer is sometimes not smart enough to see which case applies:

- If the WHERE does very little filtering, fetching the rows in ORDER BY order avoids a sort and has little wasted effort (because of 'little filtering'). Using the INDEX matching the ORDER BY is better in this case.
- If the WHERE does a lot of filtering, the ORDER BY is wasting a lot of time fetching rows only to filter them out. Using an INDEX matching the WHERE clause is better.

What should you do? If you think the "little filtering" is likely, then create an index with the ORDER BY columns in order and hope that the Optimizer uses it when it should.

OR

Cases...

- `WHERE a=1 OR a=2` -- This is turned into WHERE a IN (1,2) and optimized that way.
- `WHERE a=1 OR b=2` usually cannot be optimized.
- `WHERE x.a=1 OR y.b=2` This is even worse because of using two different tables.

A workaround is to use UNION. Each part of the UNION is optimized separately. For the second case:

```
( SELECT ... WHERE a=1 )   -- and have INDEX(a)
UNION DISTINCT -- "DISTINCT" is assuming you need to get rid of dups
( SELECT ... WHERE b=2 )   -- and have INDEX(b)
GROUP BY ... ORDER BY ...  -- whatever you had at the end of the original query
```

Now the query can take good advantage of two different indexes. Note: "Index merge" might kick in on the original query, but it is not necessarily any faster than the UNION. Sister blog on compound indexes, including 'Index Merge'

The third case (OR across 2 tables) is similar to the second.

If you originally had a LIMIT, UNION gets complicated. If you started with ORDER BY z LIMIT 190, 10, then the UNION needs to be

```
( SELECT ... LIMIT 200 )   -- Note: OFFSET 0, LIMIT 190+10
UNION DISTINCT -- (or ALL)
( SELECT ... LIMIT 200 )
LIMIT 190, 10              -- Same as originally
```

TEXT / BLOB

You cannot directly index a TEXT or BLOB or large VARCHAR or large BINARY column. However, you can use a "prefix" index: INDEX(foo(20)). This says to index the first 20 characters of `foo`. But... It is rarely useful.

Example of a prefix index:

```
    INDEX(last_name(2), first_name)
```

The index for me would contain 'Ja', 'Rick'. That's not useful for distinguishing between 'Jamison', 'Jackson', 'James', etc., so the index is so close to useless that the optimizer often ignores it.

Probably never do UNIQUE(foo(20)) because this applies a uniqueness constraint on the first 20 characters of the column, not the whole column!

More on prefix indexing

Dates

DATE, DATETIME, etc. are tricky to compare against.

Some tempting, but inefficient, techniques:

`date_col LIKE '2016-01%'` -- must convert date_col to a string, so acts like a function `LEFT(date_col, 4) = '2016-01'` -- hiding the column in function `DATE(date_col) = 2016` -- hiding the column in function

All must do a full scan. (On the other hand, it can handy to use GROUP BY LEFT(date_col, 7) for monthly grouping, but that is not an INDEX issue.)

This is efficient, and can use an index:

```
        date_col >= '2016-01-01'
    AND date_col  < '2016-01-01' + INTERVAL 3 MONTH
```

This case works because both right-hand values are converted to constants, then it is a "range". I like the design pattern with INTERVAL because it avoids computing the last day of the month. And it avoids tacking on '23:59:59', which is wrong if you have microsecond times. (And other cases.)

EXPLAIN Key_len

Perform EXPLAIN SELECT... (and EXPLAIN FORMAT=JSON SELECT... if you have 5.6.5). Look at the Key that it chose, and the Key_len. From those you can deduce how many columns of the index are being used for filtering. (JSON makes it easier to get the answer.) From that you can decide whether it is using as much of the INDEX as you thought. Caveat: Key_len only covers the WHERE part of the action; the non-JSON output won't easily say whether GROUP BY or ORDER BY was handled by the index.

IN

`IN (1,99,3)` is sometimes optimized as efficiently as "=", but not always. Older versions of MySQL did not optimize it as well as newer versions. (5.6 is possibly the main turning point.)

IN (SELECT ...)

From version 4.1 through 5.5, IN (SELECT ...) was very poorly optimized. The SELECT was effectively re-evaluated every time. Often it can be transformed into a JOIN, which works much faster. Heres is a pattern to follow:

```
SELECT  ...
    FROM  a
    WHERE  test_a
      AND  x IN (
        SELECT  x
            FROM  b
            WHERE  test_b
                );
⇒
SELECT  ...
    FROM  a
    JOIN  b USING(x)
    WHERE  test_a
      AND  test_b;
```

The SELECT expressions will need "a." prefixing the column names.

Alas, there are cases where the pattern is hard to follow.

5.6 does some optimizing, but probably not as good as the JOIN.

If there is a JOIN or GROUP BY or ORDER BY LIMIT in the subquery, that complicates the JOIN in new format. So, it might be better to use this pattern:

```
SELECT  ...
    FROM  a
    WHERE  test_a
      AND  x IN ( SELECT  x  FROM ... );
⇒
SELECT  ...
    FROM  a
    JOIN        ( SELECT  x  FROM ... ) b
        USING(x)
    WHERE  test_a;
```

Caveat: If you end up with two subqueries JOINed together, note that neither has any indexes, hence performance can be very bad. (5.6 improves on it by dynamically creating indexes for subqueries.)

There is work going on in MariaDB and Oracle 5.7, in relation to "NOT IN", "NOT EXISTS", and "LEFT JOIN..IS NULL"; here is an old discussion on the topic So, what I say here may not be the final word.

Explode/Implode

When you have a JOIN and a GROUP BY, you may have the situation where the JOIN exploded more rows than the original query (due to many:many), but you wanted only one row from the original table, so you added the GROUP BY to implode back to the desired set of rows.

This explode + implode, itself, is costly. It would be better to avoid them if possible.

Sometimes the following will work.

Using DISTINCT or GROUP BY to counteract the explosion

```
SELECT  DISTINCT
        a.*,
        b.y
    FROM a
    JOIN b
⇒
SELECT  a.*,
        ( SELECT GROUP_CONCAT(b.y) FROM b WHERE b.x = a.x ) AS ys
    FROM a
```

When using second table just to check for existence:

```
SELECT  a.*
    FROM a
    JOIN b  ON b.x = a.x
    GROUP BY a.id
⇒
SELECT  a.*,
    FROM a
    WHERE EXISTS ( SELECT *  FROM b  WHERE b.x = a.x )
```

Another variant

Many-to-many mapping table

Do it this way.

```
    CREATE TABLE XtoY (
        # No surrogate id for this table
        x_id MEDIUMINT UNSIGNED NOT NULL,   -- For JOINing to one table
        y_id MEDIUMINT UNSIGNED NOT NULL,   -- For JOINing to the other table
        # Include other fields specific to the 'relation'
        PRIMARY KEY(x_id, y_id),            -- When starting with X
        INDEX      (y_id, x_id)             -- When starting with Y
    ) ENGINE=InnoDB;
```

Notes:

- Lack of an AUTO_INCREMENT id for this table -- The PK given is the 'natural' PK; there is no good reason for a surrogate.
- "MEDIUMINT" -- This is a reminder that all INTs should be made as small as is safe (smaller ⇒ faster). Of course the declaration here must match the definition in the table being linked to.
- "UNSIGNED" -- Nearly all INTs may as well be declared non-negative
- "NOT NULL" -- Well, that's true, isn't it?
- "InnoDB" -- More effecient than MyISAM because of the way the PRIMARY KEY is clustered with the data in InnoDB.
- "INDEX(y_id, x_id)" -- The PRIMARY KEY makes it efficient to go one direction; this index makes the other direction efficient. No need to say UNIQUE; that would be extra effort on INSERTs.
- In the secondary index, saying justINDEX(y_id) would work because it would implicit include x_id. But I would rather make it more obvious that I am hoping for a 'covering' index.

To conditionally INSERT new links, use IODKU

Note that if you had an AUTO_INCREMENT in this table, IODKU would "burn" ids quite rapidly.

Subqueries and UNIONs

Each subquery SELECT and each SELECT in a UNION can be considered separately for finding the optimal INDEX.

Exception: In a "correlated" ("dependent") subquery, the part of the WHERE that depends on the outside table is not easily factored into the INDEX generation. (Cop out!)

JOINs

The first step is to decide what order the optimizer will go through the tables. If you cannot figure it out, then you may need to be pessimistic and create two indexes for each table -- one assuming the table will be used first, one assiming that it will come later in the table order.

The optimizer usually starts with one table and extracts the data needed from it. As it finds a useful (that is, matches the WHERE clause, if any) row, it reaches into the 'next' table. This is called NLJ ("Nested Loop Join"). The process of filtering and reaching to the next table continues through the rest of the tables.

The optimizer usually picks the "first" table based on these hints:

- STRAIGHT_JOIN forces the the table order.
- The WHERE clause limits which rows needed (whether indexed or not).
- The table to the "left" in a LEFT JOIN usually comes before the "right" table. (By looking at the table definitions, the optimizer may decide that "LEFT" is irrelevant.)
- The current INDEXes will encourage an order.
- etc.

Running EXPLAIN tells you the table order that the Optimizer is very likely to use today. After adding a new INDEX, the

optimizer may pick a different table order. You should anticipate the order changing, guess at what order makes the most sense, and build the INDEXes accordingly. Then rerun EXPLAIN to see if the Optimizer's brain was on the same wavelength you were on.

You should build the INDEX for the "first" table based on any parts of the WHERE, GROUP BY, and ORDER BY clauses that are relevant to it. If a GROUP/ORDER BY mentions a different table, you should ignore that clause.

The second (and subsequent) table will be reached into based on the ON clause. (Instead of using commajoin, please write JOINs with the JOIN keyword and ON clause!) In addition, there could be parts of the WHERE clause that are relevant. GROUP/ORDER BY are not to be considered in writing the optimal INDEX for subsequent tables.

PARTITIONing

PARTITIONing is rarely a substitute for a good INDEX.

PARTITION BY RANGE is a technique that is sometimes useful when indexing fails to be good enough. In a two-dimensional situation such as nearness in a geographical sense, one dimension can partially be handled by partition pruning; then the other dimension can be handled by a regular index (preferrably the PRIMARY KEY). This goes into more detail: Find nearest 10 pizza parlors.

FULLTEXT

FULLTEXT is now implemented in InnoDB as well as MyISAM. It provides a way to search for "words" in TEXT columns. This is much faster (when it is applicable) than col LIKE '%word%'.

```
    WHERE x = 1
      AND MATCH (...) AGAINST (...)
```

always(?) uses the FULLTEXT index first. That is, the whole Algorithm is invalidated when one of the ANDs is a MATCH.

Signs of a Newbie

- No "compound" (aka "composite") indexes
- No PRIMARY KEY
- Redundant indexes (especially blatant is PRIMARY KEY(id), KEY(id))
- Most or all columns individually indexes ("But I indexed everything")
- "Commajoin" -- That is `FROM a, b WHERE a.x=b.x` instead of `FROM a JOIN b ON a.x=b.x`

Speeding up wp_postmeta

The published table (see Wikipedia) is

```
    CREATE TABLE wp_postmeta (
      meta_id bigint(20) unsigned NOT NULL AUTO_INCREMENT,
      post_id bigint(20) unsigned NOT NULL DEFAULT '0',
      meta_key varchar(255) DEFAULT NULL,
      meta_value longtext,
      PRIMARY KEY (meta_id),
      KEY post_id (post_id),
      KEY meta_key (meta_key)
    ) ENGINE=InnoDB  DEFAULT CHARSET=utf8;
```

The problems:

- The AUTO_INCREMENT provides no benefit; in fact it slows down most queries and clutters disk.
- Much better is PRIMARY KEY(post_id, meta_key) -- clustered, handles both parts of usual JOIN.
- BIGINT is overkill, but that can't be fixed without changing other tables.
- VARCHAR(255) can be a problem in 5.6 with utf8mb4; see workarounds below.
- When would `meta_key` or `meta_value` ever be NULL?

The solutions:

```
    CREATE TABLE wp_postmeta (
        post_id BIGINT UNSIGNED NOT NULL,
        meta_key VARCHAR(255) NOT NULL,
        meta_value LONGTEXT NOT NULL,
        PRIMARY KEY(post_id, meta_key),
        INDEX(meta_key)
        ) ENGINE=InnoDB;
```

Postlog

Initial posting: March, 2015; Refreshed Feb, 2016; Add DATE June, 2016; Add WP example May, 2017.

The tips in this document apply to MySQL, MariaDB, and Percona.

3.3.3.6 Compound (Composite) Indexes

A mini-lesson in "compound indexes" ("composite indexes")

This document starts out trivial and perhaps boring, but builds up to more interesting information, perhaps things you did not realize about how MariaDB and MySQL indexing works.

This also explains EXPLAIN (to some extent).

(Most of this applies to other databases, too.)

The query to discuss

The question is "When was Andrew Johnson president of the US?".

The available table `Presidents` looks like:

```
+-----+------------+----------------+-----------+
| seq | last_name  | first_name     | term      |
+-----+------------+----------------+-----------+
|   1 | Washington | George         | 1789-1797 |
|   2 | Adams      | John           | 1797-1801 |
...
|   7 | Jackson    | Andrew         | 1829-1837 |
...
|  17 | Johnson    | Andrew         | 1865-1869 |
...
|  36 | Johnson    | Lyndon B.      | 1963-1969 |
...
```

("Andrew Johnson" was picked for this lesson because of the duplicates.)

What index(es) would be best for that question? More specifically, what would be best for

```
    SELECT  term
        FROM  Presidents
        WHERE  last_name = 'Johnson'
          AND  first_name = 'Andrew';
```

Some INDEXes to try...

- No indexes
- INDEX(first_name), INDEX(last_name) (two separate indexes)
- "Index Merge Intersect"
- INDEX(last_name, first_name) (a "compound" index)
- INDEX(last_name, first_name, term) (a "covering" index)
- Variants

No indexes

Well, I am fudging a little here. I have a PRIMARY KEY on `seq`, but that has no advantage on the query we are studying.

```
SHOW CREATE TABLE Presidents \G
CREATE TABLE `presidents` (
  `seq` tinyint(3) unsigned NOT NULL AUTO_INCREMENT,
  `last_name` varchar(30) NOT NULL,
  `first_name` varchar(30) NOT NULL,
  `term` varchar(9) NOT NULL,
  PRIMARY KEY (`seq`)
) ENGINE=InnoDB AUTO_INCREMENT=45 DEFAULT CHARSET=utf8

EXPLAIN  SELECT  term
   FROM  Presidents
   WHERE  last_name = 'Johnson'
   AND  first_name = 'Andrew';
+----+-------------+------------+------+---------------+------+---------+------+------+----------
| id | select_type | table      | type | possible_keys | key  | key_len | ref  | rows | Extra
+----+-------------+------------+------+---------------+------+---------+------+------+----------
|  1 | SIMPLE      | Presidents | ALL  | NULL          | NULL | NULL    | NULL |   44 | Using whe
+----+-------------+------------+------+---------------+------+---------+------+------+----------

# Or, using the other form of display:  EXPLAIN ... \G
           id: 1
  select_type: SIMPLE
        table: Presidents
         type: ALL        <-- Implies table scan
possible_keys: NULL
          key: NULL       <-- Implies that no index is useful, hence table scan
      key_len: NULL
          ref: NULL
         rows: 44         <-- That's about how many rows in the table, so table scan
        Extra: Using where
```

Implementation Details

First, let's describe how InnoDB stores and uses indexes.

- The data and the PRIMARY KEY are "clustered" together in on BTree.
- A BTree lookup is quite fast and efficient. For a million-row table there might be 3 levels of BTree, and the top two levels are probably cached.
- Each secondary index is in another BTree, with the PRIMARY KEY at the leaf.
- Fetching 'consecutive' (according to the index) items from a BTree is very efficient because they are stored consecutively.
- For the sake of simplicity, we can count each BTree lookup as 1 unit of work, and ignore scans for consecutive items. This approximates the number of disk hits for a large table in a busy system.

For MyISAM, the PRIMARY KEY is not stored with the data, so think of it as being a secondary key (over-simplified).

INDEX(first_name), INDEX(last_name)

The novice, once he learns about indexing, decides to index lots of columns, one at a time. But...

MariaDB rarely uses more than one index at a time in a query. So, it will analyze the possible indexes.

- first_name -- there are 2 possible rows (one BTree lookup, then scan consecutively)
- last_name -- there are 2 possible rows Let's say it picks last_name. Here are the steps for doing the SELECT: 1. Using INDEX(last_name), find 2 index entries with last_name = 'Johnson'. 2. Get the PRIMARY KEY (implicitly added to each secondary index in InnoDB); get (17, 36). 3. Reach into the data using seq = (17, 36) to get the rows for Andrew Johnson and Lyndon B. Johnson. 4. Use the rest of the WHERE clause filter out all but the desired row. 5. Deliver the answer (1865-1869).

```
EXPLAIN  SELECT  term
  FROM  Presidents
  WHERE  last_name = 'Johnson'
  AND  first_name = 'Andrew'  \G
  select_type: SIMPLE
        table: Presidents
         type: ref
possible_keys: last_name, first_name
          key: last_name
      key_len: 92                <-- VARCHAR(30) utf8 may need 2+3*30 bytes
          ref: const
         rows: 2                 <-- Two 'Johnson's
        Extra: Using where
```

"Index Merge Intersect"

OK, so you get really smart and decide that MariaDB should be smart enough to use both name indexes to get the answer. This is called "Intersect". 1. Using INDEX(last_name), find 2 index entries with last_name = 'Johnson'; get (7, 17) 2. Using INDEX(first_name), find 2 index entries with first_name = 'Andrew'; get (17, 36) 3. "And" the two lists together (7,17) & (17,36) = (17) 4. Reach into the data using seq = (17) to get the row for Andrew Johnson. 5. Deliver the answer (1865-1869).

```
           id: 1
  select_type: SIMPLE
        table: Presidents
         type: index_merge
possible_keys: first_name,last_name
          key: first_name,last_name
      key_len: 92,92
          ref: NULL
         rows: 1
        Extra: Using intersect(first_name,last_name); Using where
```

The EXPLAIN fails to give the gory details of how many rows collected from each index, etc.

INDEX(last_name, first_name)

This is called a "compound" or "composite" index since it has more than one column. 1. Drill down the BTree for the index to get to exactly the index row for Johnson+Andrew; get seq = (17). 2. Reach into the data using seq = (17) to get the row for Andrew Johnson. 3. Deliver the answer (1865-1869). This is much better. In fact this is usually the "best".

```
    ALTER TABLE Presidents
        (drop old indexes and...)
        ADD INDEX compound(last_name, first_name);

           id: 1
  select_type: SIMPLE
        table: Presidents
         type: ref
possible_keys: compound
          key: compound
      key_len: 184             <-- The length of both fields
          ref: const,const     <-- The WHERE clause gave constants for both
         rows: 1               <-- Goodie!  It homed in on the one row.
        Extra: Using where
```

"Covering": INDEX(last_name, first_name, term)

Surprise! We can actually do a little better. A "Covering" index is one in which _all_ of the fields of the SELECT are found in the index. It has the added bonus of not having to reach into the "data" to finish the task. 1. Drill down the BTree for the index to get to exactly the index row for Johnson+Andrew; get seq = (17). 2. Deliver the answer (1865-1869). The "data" BTree is not touched; this is an improvement over "compound".

```
    ... ADD INDEX covering(last_name, first_name, term);

           id: 1
  select_type: SIMPLE
        table: Presidents
         type: ref
possible_keys: covering
          key: covering
      key_len: 184
          ref: const,const
         rows: 1
        Extra: Using where; Using index   <-- Note
```

Everything is similar to using "compound", except for the addition of "Using index".

Variants

- What would happen if you shuffled the fields in the WHERE clause? Answer: The order of ANDed things does not matter.
- What would happen if you shuffled the fields in the INDEX? Answer: It may make a huge difference. More in a minute.
- What if there are extra fields on the the end? Answer: Minimal harm; possibly a lot of good (eg, 'covering').
- Reduncancy? That is, what if you have both of these: INDEX(a), INDEX(a,b)? Answer: Reduncy costs something on INSERTs; it is rarely useful for SELECTs.
- Prefix? That is, INDEX(last_name(5). first_name(5)) Answer: Don't bother; it rarely helps, and often hurts. (The details are another topic.)

More examples:

```
    INDEX(last, first)
    ... WHERE last = '...' -- good (even though `first` is unused)
    ... WHERE first = '...' -- index is useless

    INDEX(first, last), INDEX(last, first)
    ... WHERE first = '...' -- 1st index is used
    ... WHERE last = '...' -- 2nd index is used
    ... WHERE first = '...' AND last = '...' -- either could be used equally well

    INDEX(last, first)
    Both of these are handled by that one INDEX:
    ... WHERE last = '...'
    ... WHERE last = '...' AND first = '...'

    INDEX(last), INDEX(last, first)
    In light of the above example, don't bother including INDEX(last).
```

Postlog

Refreshed -- Oct, 2012; more links -- Nov 2016

1.1.1.2.2.4 EXPLAIN

3.3.3.8 Foreign Keys

Contents

Overview

A foreign key is a constraint which can be used to enforce data integrity. It is composed by a column (or a set of columns) in a table called the child table, which references to a column (or a set of columns) in a table called the parent table. If foreign keys are used, MariaDB performs some checks to enforce that some integrity rules are always enforced. For a more exhaustive explanation, see Relational databases: Foreign Keys.

Foreign keys can only be used with storage engines that support them. The default InnoDB and the obsolete PBXT support foreign keys.

Partitioned tables cannot contain foreign keys, and cannot be referenced by a foreign key.

Syntax

Note: Until MariaDB 10.4, MariaDB accepts the shortcut format with a REFERENCES clause only in ALTER TABLE and CREATE TABLE statements, but that syntax does nothing. For example:

```
CREATE TABLE b(for_key INT REFERENCES a(not_key));
```

MariaDB simply parses it without returning any error or warning, for compatibility with other DBMS's. However, only the syntax described below creates foreign keys.

From MariaDB 10.5, MariaDB will attempt to apply the constraint. See the Examples below.

Foreign keys are created with CREATE TABLE or ALTER TABLE. The definition must follow this syntax:

```
[CONSTRAINT [symbol]] FOREIGN KEY
    [index_name] (index_col_name, ...)
    REFERENCES tbl_name (index_col_name,...)
    [ON DELETE reference_option]
    [ON UPDATE reference_option]

reference_option:
    RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT
```

The `symbol` clause, if specified, is used in error messages and must be unique in the database.

The columns in the child table must be a BTREE (not HASH, RTREE, or FULLTEXT — see SHOW INDEX) index, or the leftmost part of a BTREE index. Index prefixes are not supported (thus, TEXT and BLOB columns cannot be used as foreign keys). If MariaDB automatically creates an index for the foreign key (because it does not exist and is not explicitly created), its name will be `index_name`.

The referenced columns in the parent table must be a an index or a prefix of an index.

The foreign key columns and the referenced columns must be of the same type, or similar types. For integer types, the size and sign must also be the same.

Both the foreign key columns and the referenced columns can be PERSISTENT columns. However, the ON UPDATE CASCADE, ON UPDATE SET NULL, ON DELETE SET NULL clauses are not allowed in this case.

The parent and the child table must use the same storage engine, and must not be `TEMPORARY` or partitioned tables. They can be the same table.

Constraints

If a foreign keys exists, each row in the child table must match a row in the parent table. Multiple child rows can match the same parent row. A child row *matches* a parent row if all its foreign key values are identical to a parent row's values in the parent table. However, if at least one of the foreign key values is `NULL`, the row has no parents, but it is still allowed.

MariaDB performs certain checks to guarantee that the data integrity is enforced:

- Trying to insert non-matching rows (or update matching rows in a way that makes them non-matching rows) in the child table produces a 1452 error (SQLSTATE '23000').
- When a row in the parent table is deleted and at least one child row exists, MariaDB performs an action which depends on the `ON DELETE` clause of the foreign key.
- When a value in the column referenced by a foreign key changes and at least one child row exists, MariaDB performs an action which depends on the `ON UPDATE` clause of the foreign key.
- Trying to drop a table that is referenced by a foreign key produces a 1217 error (SQLSTATE '23000').
- A TRUNCATE TABLE against a table containing one or more foreign keys is executed as a DELETE without

WHERE, so that the foreign keys are enforced for each row.

The allowed actions for `ON DELETE` and `ON UPDATE` are:

- `RESTRICT` : The change on the parent table is prevented. The statement terminates with a 1451 error (SQLSTATE '2300'). This is the default behavior for both `ON DELETE` and `ON UPDATE` .
- `NO ACTION` : Synonym for `RESTRICT` .
- `CASCADE` : The change is allowed and propagates on the child table. For example, if a parent row is deleted, the child row is also deleted; if a parent row's ID changes, the child row's ID will also change.
- `SET NULL` : The change is allowed, and the child row's foreign key columns are set to `NULL` .
- `SET DEFAULT` : Only worked with PBXT. Similar to `SET NULL` , but the foreign key columns were set to their default values. If default values do not exist, an error is produced.

The delete or update operations triggered by foreign keys do not activate triggers and are not counted in the Com_delete and Com_update status variables.

Foreign key constraints can be disabled by setting the foreign_key_checks server system variable to 0. This speeds up the insertion of large quantities of data.

Metadata

The Information Schema `REFERENTIAL_CONSTRAINTS` table contains information about foreign keys. The individual columns are listed in the `KEY_COLUMN_USAGE` table.

The InnoDB-specific Information Schema tables also contain information about the InnoDB foreign keys. The foreign key information is stored in the `INNODB_SYS_FOREIGN` . Data about the individual columns are stored in `INNODB_SYS_FOREIGN_COLS` .

The most human-readable way to get information about a table's foreign keys sometimes is the `SHOW CREATE TABLE` statement.

Limitations

Foreign keys have the following limitations in MariaDB:

- Currently, foreign keys are only supported by InnoDB.
- Cannot be used with views.
- The `SET DEFAULT` action is not supported.
- Foreign keys actions do not activate triggers.
- If ON UPDATE CASCADE recurses to update the same table it has previously updated during the cascade, it acts like RESTRICT.

Examples

Let's see an example. We will create an `author` table and a `book` table. Both tables have a primary key called `id` . `book` also has a foreign key composed by a field called `author_id` , which refers to the `author` primary key. The foreign key constraint name is optional, but we'll specify it because we want it to appear in error messages: `fk_book_author` .

```
CREATE TABLE author (
  id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(100) NOT NULL
) ENGINE = InnoDB;

CREATE TABLE book (
  id MEDIUMINT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  title VARCHAR(200) NOT NULL,
  author_id SMALLINT UNSIGNED NOT NULL,
  CONSTRAINT `fk_book_author`
    FOREIGN KEY (author_id) REFERENCES author (id)
    ON DELETE CASCADE
    ON UPDATE RESTRICT
) ENGINE = InnoDB;
```

Now, if we try to insert a book with a non-existing author, we will get an error:

```
INSERT INTO book (title, author_id) VALUES ('Necronomicon', 1);
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails
 (`test`.`book`, CONSTRAINT `fk_book_author` FOREIGN KEY (`author_id`)
  REFERENCES `author` (`id`) ON DELETE CASCADE)
```

The error is very descriptive.

Now, let's try to properly insert two authors and their books:

```
INSERT INTO author (name) VALUES ('Abdul Alhazred');
INSERT INTO book (title, author_id) VALUES ('Necronomicon', LAST_INSERT_ID());

INSERT INTO author (name) VALUES ('H.P. Lovecraft');
INSERT INTO book (title, author_id) VALUES
  ('The call of Cthulhu', LAST_INSERT_ID()),
  ('The colour out of space', LAST_INSERT_ID());
```

It worked!

Now, let's delete the second author. When we created the foreign key, we specified `ON DELETE CASCADE`. This should propagate the deletion, and make the deleted author's books disappear:

```
DELETE FROM author WHERE name = 'H.P. Lovecraft';

SELECT * FROM book;
+----+--------------+-----------+
| id | title        | author_id |
+----+--------------+-----------+
|  3 | Necronomicon |         1 |
+----+--------------+-----------+
```

We also specified `ON UPDATE RESTRICT`. This should prevent us from modifying an author's `id` (the column referenced by the foreign key) if a child row exists:

```
UPDATE author SET id = 10 WHERE id = 1;
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails
 (`test`.`book`, CONSTRAINT `fk_book_author` FOREIGN KEY (`author_id`)
  REFERENCES `author` (`id`) ON DELETE CASCADE)
```

REFERENCES

Until MariaDB 10.4

```
CREATE TABLE a(a_key INT primary key, not_key INT);

CREATE TABLE b(for_key INT REFERENCES a(not_key));

SHOW CREATE TABLE b;
+-------+-----------------------------------------------------------------------------------------+
| Table | Create Table                                                                            |
+-------+-----------------------------------------------------------------------------------------+
| b     | CREATE TABLE `b` (
  `for_key` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+-------+-----------------------------------------------------------------------------------------+

INSERT INTO a VALUES (1,10);
Query OK, 1 row affected (0.005 sec)

INSERT INTO b VALUES (10);
Query OK, 1 row affected (0.004 sec)

INSERT INTO b VALUES (1);
Query OK, 1 row affected (0.004 sec)

SELECT * FROM b;
+---------+
| for_key |
+---------+
|      10 |
|       1 |
+---------+
```

From MariaDB 10.5

```
CREATE TABLE a(a_key INT primary key, not_key INT);

CREATE TABLE b(for_key INT REFERENCES a(not_key));
ERROR 1005 (HY000): Can't create table `test`.`b`
  (errno: 150 "Foreign key constraint is incorrectly formed")

CREATE TABLE c(for_key INT REFERENCES a(a_key));

SHOW CREATE TABLE c;
+-------+-----------------------------------------------------------------------------------------+
| Table | Create Table                                                                            |
+-------+-----------------------------------------------------------------------------------------+
| c     | CREATE TABLE `c` (
  `for_key` int(11) DEFAULT NULL,
  KEY `for_key` (`for_key`),
  CONSTRAINT `c_ibfk_1` FOREIGN KEY (`for_key`) REFERENCES `a` (`a_key`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+-------+-----------------------------------------------------------------------------------------+

INSERT INTO a VALUES (1,10);
Query OK, 1 row affected (0.004 sec)

INSERT INTO c VALUES (10);
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails
  (`test`.`c`, CONSTRAINT `c_ibfk_1` FOREIGN KEY (`for_key`) REFERENCES `a` (`a_key`))

INSERT INTO c VALUES (1);
Query OK, 1 row affected (0.004 sec)

SELECT * FROM c;
+---------+
| for_key |
+---------+
|       1 |
+---------+
```

3.3.3.9 Ignored Indexes

MariaDB starting with 10.6.0

MariaDB starting with [illegible]
Ignored indexes were added in MariaDB 10.6.

Ignored indexes are indexes that are visible and maintained, but which are not used by the optimizer. MySQL 8 has a similar feature which they call "invisible indexes".

Syntax

By default, an index is not ignored. One can mark existing index as ignored (or not ignored) with an ALTER TABLE statement:

```
ALTER TABLE table_name ALTER {KEY|INDEX} [IF EXISTS] key_name [NOT] IGNORED;
```

It is also possible to specify IGNORED attribute when creating an index with a CREATE TABLE, or CREATE INDEX statement:

```
CREATE TABLE table_name (
  ...
  INDEX index_name ( ...) [NOT] IGNORED
  ...
```

```
CREATE INDEX index_name (...) [NOT] IGNORED ON tbl_name (...);
```

table's primary key cannot be ignored. This applies to both explicitly defined primary key, as well as implicit primary key - if there is no explicit primary key defined but the table has a unique key containing only NOT NULL columns, the first of such keys becomes the implicitly defined primary key.

Handling for ignored indexes

The optimizer will treats ignored indexes as if they didn't exist. They will not be used in the query plans, or as a source of statistical information. Also, an attempt to use an ignored index in a `USE INDEX`, `FORCE INDEX`, or `IGNORE INDEX` hint will result in an error - the same what would have if one used a name of a non-existent index.

Information about whether or not indexes are ignored can be viewed in the IGNORED column in the Information Schema STATISTICS table or the SHOW INDEX statement.

Intended Usage

The primary use case is as follows: a DBA sees an index that seems to have little or no usage and considers whether to remove it. Dropping the index is a risk as it may still be needed in a few cases. For example, the optimizer may rely on the estimates provided by the index without using the index in query plans. If dropping an index causes an issue, it will take a while to re-create the index. On the other hand, marking the index as ignored (or not ignored) is instant, so the suggested workflow is:

1. Mark the index as ignored
2. Check if everything continues to work
3. If not, mark the index as not ignored.
4. If everything continues to work, one can safely drop the index.

Examples

```
CREATE TABLE t1 (id INT PRIMARY KEY, b INT, KEY k1(b) IGNORED);
```

```
CREATE OR REPLACE TABLE t1 (id INT PRIMARY KEY, b INT, KEY k1(b));
ALTER TABLE t1 ALTER INDEX k1 IGNORED;
```

```
CREATE OR REPLACE TABLE t1 (id INT PRIMARY KEY, b INT);
CREATE INDEX k1 ON t1(b) IGNORED;
```

```
SELECT * FROM INFORMATION_SCHEMA.STATISTICS WHERE TABLE_NAME = 't1'\G
*************************** 1. row ***************************
TABLE_CATALOG: def
 TABLE_SCHEMA: test
   TABLE_NAME: t1
   NON_UNIQUE: 0
 INDEX_SCHEMA: test
   INDEX_NAME: PRIMARY
 SEQ_IN_INDEX: 1
  COLUMN_NAME: id
    COLLATION: A
  CARDINALITY: 0
     SUB_PART: NULL
       PACKED: NULL
     NULLABLE:
   INDEX_TYPE: BTREE
      COMMENT:
INDEX_COMMENT:
      IGNORED: NO
*************************** 2. row ***************************
TABLE_CATALOG: def
 TABLE_SCHEMA: test
   TABLE_NAME: t1
   NON_UNIQUE: 1
 INDEX_SCHEMA: test
   INDEX_NAME: k1
 SEQ_IN_INDEX: 1
  COLUMN_NAME: b
    COLLATION: A
  CARDINALITY: 0
     SUB_PART: NULL
       PACKED: NULL
     NULLABLE: YES
   INDEX_TYPE: BTREE
      COMMENT:
INDEX_COMMENT:
      IGNORED: YES
```

```
SHOW INDEXES FROM t1\G
*************************** 1. row ***************************
        Table: t1
   Non_unique: 0
     Key_name: PRIMARY
 Seq_in_index: 1
  Column_name: id
    Collation: A
  Cardinality: 0
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
      Ignored: NO
*************************** 2. row ***************************
        Table: t1
   Non_unique: 1
     Key_name: k1
 Seq_in_index: 1
  Column_name: b
    Collation: A
  Cardinality: 0
     Sub_part: NULL
       Packed: NULL
         Null: YES
   Index_type: BTREE
      Comment:
Index_comment:
      Ignored: YES
```

The optimizer does not make use of an index when it is ignored, while if the index is not ignored (the default), the optimizer will consider it in the optimizer plan, as shown in the EXPLAIN output.

```
CREATE OR REPLACE TABLE t1 (id INT PRIMARY KEY, b INT, KEY k1(b) IGNORED);

EXPLAIN SELECT * FROM t1 ORDER BY b;
+------+-------------+-------+------+---------------+------+---------+------+------+-------------
| id   | select_type | table | type | possible_keys | key  | key_len | ref  | rows | Extra
+------+-------------+-------+------+---------------+------+---------+------+------+-------------
|    1 | SIMPLE      | t1    | ALL  | NULL          | NULL | NULL    | NULL | 1    | Using filesc
+------+-------------+-------+------+---------------+------+---------+------+------+-------------

ALTER TABLE t1 ALTER INDEX k1 NOT IGNORED;

EXPLAIN SELECT * FROM t1 ORDER BY b;
+------+-------------+-------+-------+---------------+------+---------+------+------+-------------
| id   | select_type | table | type  | possible_keys | key  | key_len | ref  | rows | Extra
+------+-------------+-------+-------+---------------+------+---------+------+------+-------------
|    1 | SIMPLE      | t1    | index | NULL          | k1   | 5       | NULL | 1    | Using index
+------+-------------+-------+-------+---------------+------+---------+------+------+-------------
```

3.3.3.10 Index Statistics

Contents

How Index Statistics Help the Query Optimizer

The MariaDB query optimizer decides how best to execute each query based largely on the details of the indexes involved.

The index statistics help inform these decisions. Imagine yourself choosing whether to look up a number in a phone book, or in your personal address book. You'd choose the personal phone book if at all possible, as it would (usually!) contain far fewer records and be quicker to search.

Now imagine getting to your personal address book and finding it has twice the number of entries as the phone book. Your search would be slower. The same process applies to the query optimizer, so having access to up-to-date and accurate statistics is critical.

Value Groups

The statistics are mainly based on groups of index elements of the same value. In a primary key, every index is unique, so every group size is one. In a non-unique index, you may have multiple keys with the same value. A worst-case example would be having large groups with the same value, for example an index on a boolean field.

MariaDB makes heavy use of the average group size statistic. For example, if there are 100 rows, and twenty groups with the same index values, the average group size would be five.

However, averages can be skewed by extremes, and the usual culprit is NULL values. The row of 100 may have 19 groups with an average size of 1, while the other 81 values are all NULL. MariaDB may think five is a good average size and choose to use that index, and then end up having to read through 81 rows with identical keys, taking longer than an alternative.

Dealing with NULLs

There are three main approaches to the problem of NULLs. NULL index values can be treated as a single group (nulls_equal). This is usually fine, but if you have large numbers of NULLs the average group size is slanted higher, and the optimizer may miss using the index for ref accesses when it would be useful. This is the default used by XtraDB/InnoDB and MyISAM. Nulls_unequal is the opposite approach, with each NULL forming its own group of one. Conversely, the average group size is slanted lower, and the optimizer may use the index for ref accesses when not suitable. This is the default used by the Aria storage engine. A third options sees NULL's ignored altogether from index group calculations.

The default approaches can be changed by setting the aria_stats_method, myisam_stats_method and innodb_stats_method server variables.

Null-Safe and Regular Comparisons

The comparison operator used plays an important role. If two values are compared with <=> (see the null-safe-equal comparison operator), and both are null, 1 is returned. If the same values are compared with = (see the equal comparison operator) null is returned. For example:

```
SELECT 1 <=> 1, NULL <=> NULL, 1 <=> NULL;
+---------+---------------+-------------+
| 1 <=> 1 | NULL <=> NULL | 1 <=> NULL |
+---------+---------------+-------------+
|       1 |             1 |           0 |
+---------+---------------+-------------+

SELECT 1 = 1, NULL = NULL, 1 = NULL;
+-------+-------------+----------+
| 1 = 1 | NULL = NULL | 1 = NULL |
+-------+-------------+----------+
|     1 |        NULL |     NULL |
+-------+-------------+----------+
```

Engine-Independent Statistics

MariaDB 10.0.1 introduced a way to gather statistics independently of the storage engine. See Engine-independent table statistics.

Histogram-Based Statistics

Histogram-Based Statistics were introduced in MariaDB 10.0.2, and are collected by default from MariaDB 10.4.3.

3.3.3.11 Latitude/Longitude Indexing

Contents

The problem

You want to find the nearest 10 pizza parlors, but you cannot figure out how to do it efficiently in your huge database. Database indexes are good at one-dimensional indexing, but poor at two-dimensions.

You might have tried

- INDEX(lat), INDEX(lon) -- but the optimizer used only one
- INDEX(lat,lon) -- but it still had to work too hard
- Sometimes you ended up with a full table scan -- Yuck.

WHERE SQRT(...)< ... -- No chance of using any index.

WHERE lat BETWEEN ... AND lng BETWEEN... -- This has some chance of using such indexes.

The goal is to look only at records "close", in both directions, to the target lat/lng.

A solution -- first, the principles

PARTITIONs in MariaDB and MySQL sort of give you a way to have two clustered indexes. So, if we could slice up (partition) the globe in one dimension and use ordinary indexing in the other dimension, maybe we can get something approximating a 2D index. This 2D approach keeps the number of disk hits significantly lower than 1D approaches, thereby speeding up "find nearest" queries.

It works. Not perfectly, but better than the alternatives.

What to PARTITION on? It seems like latitude or longitude would be a good idea. Note that longitudes vary in width, from 69 miles (111 km) at the equator, to 0 at the poles. So, latitude seems like a better choice.

How many PARTITIONs? It does not matter a lot. Some thoughts:

- 90 partitions - 2 degrees each. (I don't like tables with too many partitions; 90 seems like plenty.)
- 50-100 - evenly populated. (This requires code. For 2.7M placenames, 85 partitions varied from 0.5 degrees to very wide partitions at the poles.)
- Don't have more than 100 partitions, there are inefficiencies in the partition implementation.

How to PARTITION? Well, MariaDB and MySQL are very picky. So FLOAT/DOUBLE are out. DECIMAL is out. So, we are stuck with some kludge. Essentially, we need to convert Lat/Lng to some size of INT and use PARTITION BY RANGE.

Representation choices

To get to a datatype that can be used in PARTITION, you need to "scale" the latitude and longitude. (Consider only the *INTs; the other datatypes are included for comparison)

```
Datatype           Bytes       resolution
------------------ -----  --------------------------------
Deg*100 (SMALLINT)     4  1570 m    1.0 mi  Cities
DECIMAL(4,2)/(5,2)     5  1570 m    1.0 mi  Cities
SMALLINT scaled        4   682 m    0.4 mi  Cities
Deg*10000 (MEDIUMINT)  6    16 m     52 ft  Houses/Businesses
DECIMAL(6,4)/(7,4)     7    16 m     52 ft  Houses/Businesses
MEDIUMINT scaled       6   2.7 m    8.8 ft
FLOAT                  8   1.7 m    5.6 ft
DECIMAL(8,6)/(9,6)     9    16cm    1/2 ft  Friends in a mall
Deg*10000000 (INT)     8    16mm    5/8 in  Marbles
DOUBLE                16   3.5nm     ...    Fleas on a dog
```

(Sorted by resolution)

What these mean...

Deg*100 (SMALLINT) -- you take the lat/lng, multiply by 100, round, and store into a SMALLINT. That will take 2 bytes for each dimension, for a total of 4 bytes. Two items might be 1570 meters apart, but register as having the same latitude and longitude.

DECIMAL(4,2) for latitude and DECIMAL(5,2) for longitude will take 2+3 bytes and have no better resolution than Deg*100.

SMALLINT scaled -- Convert latitude into a SMALLINT SIGNED by doing (degrees / 90 * 32767) and rounding; longitude by (degrees / 180 * 32767).

FLOAT has 24 significant bits; DOUBLE has 53. (They don't work with PARTITIONing but are included for completeness. Often people use DOUBLE without realizing how much an overkill it is, and how much space it takes.)

Sure, you could do DEG*1000 and other "in between" cases, but there is no advantage. DEG*1000 takes as much space as DEG*10000, but has less resolution.

So, go down the list to see how much resolution you need, then pick an encoding you are comfortable with. However, since we are about to use latitude as a "partition key", it must be limited to one of the INTs. For the sample code, I will use Deg*10000 (MEDIUMINT).

GCDist -- compute "great circle distance"

GCDist is a helper FUNCTION that correctly computes the distance between two points on the globe.

The code has been benchmarked at about 20 microseconds per call on a 2011-vintage PC. If you had to check a million points, that would take 20 seconds -- far too much for a web application. So, one goal of the Procedure that uses it will be to minimize the usage of this function. With the code presented here, the function need be called only a few dozen or few hundred times, except in pathological cases.

Sure, you could use the Pythagorean formula. And it would work for most applications. But it does not take extra effort to do the GC. Furthermore, GC works across a pole and across the dateline. And, a Pythagorean function is not that much faster.

For efficiency, GCDist understands the scaling you picked and has that stuff hardcoded. I am picking "Deg*10000", so the

function expects 350000 for representing 35 degrees. If you choose a different scaling, you will need to change the code.

GCDist() takes 4 scaled DOUBLEs -- lat1, lon1, lat2, lon2 -- and returns a scaled number of "degrees" representing the distance.

The table of representation choices says 52 feet of resolution for Deg*10000 and DECIMAL(x,4). Here is how it was calculated: To measuring a diagonal between lat/lng (0,0) and (0.0001,00001) (one 'unit in the last place'): GCDist(0,0,1,1) * 69.172 / 10000 * 5280 = 51.65, where

- 69.172 miles/degree of latitude
- 10000 units per degree for the scaling chosen
- 5280 feet / mile.

(No, this function does not compensate for the Earth being an oblate spheroid, etc.)

Required table structure

There will be one table (plus normalization tables as needed). The one table must be partitioned and indexed as indicated below.

Fields and indexes

- PARTITION BY RANGE(lat)
- lat -- scaled latitude (see above)
- lon -- scaled longitude
- PRIMARY KEY(lon, lat, ...) -- lon must be first; something must be added to make it UNIQUE
- id -- (optional) you may need to identify the rows for your purposes; AUTO_INCREMENT if you like
- INDEX(id) -- if `id` is AUTO_INCREMENT, then this plain INDEX (not UNIQUE, not PRIMARY KEY) is necessary
- ENGINE=InnoDB -- so the PRIMARY KEY will be "clustered"
- Other indexes -- keep to a minimum (this is a general performance rule for large tables)

For most of this discussion, lat is assumed to be MEDIUMINT -- scaled from -90 to +90 by multiplying by 10000. Similarly for lon and -180 to +180.

The PRIMARY KEY must

- start with `lon` since the algorithm needs the "clustering" that InnoDB will provide, and
- include `lat` somewhere, since it is the PARTITION key, and
- contain something to make the key UNIQUE (lon+lat is unlikely to be sufficient).

The FindNearest PROCEDURE will do multiple SELECTs something like this:

```
    WHERE lat    BETWEEN @my_lat - @dlat
                     AND @my_lat + @dlat   -- PARTITION Pruning and bounding box
      AND lon    BETWEEN @my_lon - @dlon
                     AND @my_lon + @dlon   -- first part of PK
      AND condition                        -- filter out non-pizza parlors
```

The query planner will

- Do PARTITION "pruning" based on the latitude; then
- Within a PARTITION (which is effectively a table), use lon do a 'clustered' range scan; then
- Use the "condition" to filter down to the rows you desire, plus recheck lat. This design leads to very few disk blocks needing to be read, which is the main goal of the design.

Note that this does not even call GCDist. That comes in the last pass when the ORDER BY and LIMIT are used.

The stored procedure has a loop. At least two SELECTs will be executed, but with proper tuning; usually no more than about 6 SELECTs will be performed. Because of searching by the PRIMARY KEY, each SELECT hits only one block, sometimes more, of the table. Counting the number of blocks hit is a crude, but effective way, of comparing the performance of multiple designs. By comparison, a full table scan will probably touch thousands of blocks. A simple INDEX(lat) probably leads to hitting hundreds of blocks.

Filtering... An argument to the FindNearest procedure includes a boolean expression ("condition") for a WHERE clause. If you don't need any filtering, pass in "1". To avoid "SQL injection", do not let web users put arbitrary expressions; instead, construct the "condition" from inputs they provide, thereby making sure it is safe.

The algorithm

The algorithm is embodied in a stored procedure because of its complexity.

- You feed it a starting width for a "square" and a number of items to find.
- It builds a "square" around where you are.

- A SELECT is performed to see how many items are in the square.
- Loop, doubling the width of the square, until enough items are found.
- Now, a 'last' SELECT is performed to get the exact distances, sort them (ORDER BY) and LIMIT to the desired number.
- If spanning a pole or the dateline, a more complex SELECT is used.

The next section ("Performance") should make this a bit clearer as it walks through some examples.

Performance

Because of all the variations, it is hard to get a meaningful benchmark. So, here is some hand-waving instead.

Each SELECT is constrained by a "square" defined by a latitude range and a longitude range. (See the WHERE clause mentioned above, or in the sample code below.) Because of the way longitude lines warp, the longitude range of the "square" will be more degrees than the latitude range. Let's say the latitude partitioning is 3 degrees wide in the area where you are searching. That is over 200 miles (over 300km), so you are very likely to have a latitude range smaller than the partition width. Still, if you are reaching from the edge of a latitude stripe, the square could span two partitions. After partition pruning down to one (sometimes more) partition, the query is then constrained by a longitude range. (Remember, the PRIMARY KEY starts with `lon`.) If an InnoDB data block contains 100 rows (a handy Rule of Thumb), the select will touch one (or a few) block. If the square spans two (or more) partitions, then the same logic applies to each partition.

So, scanning the square will involve as little as one block; rarely more than a few blocks. The number of blocks is mostly independent of the dataset size.

The primary use case for this algorithm is when the data is significantly larger than will fit into cache (the buffer_pool). Hence, the main goal is to minimize the number of disk hits.

Now let's look at some edge cases, and argue that the number of blocks is still better (usually) than with traditional indexing techniques.

What if you are looking for Starbucks in a dense city? There would be dozens, maybe hundreds per square mile. If you start the guess at 100 miles, the SELECTs would be hitting lots of blocks -- not efficient. In this case, the "starting distance" should be small, say, 2 miles. Let's say your app wants the closest 10 stores. In this example, you would probably find more than 10 Starbucks within 2 miles in 1 InnoDB block in one partition. Even though there is a second SELECT to finish off the query, it would be hitting the same block. Total: One block hit == cheap.

Let's say you start with a 5 mile square. Since there are upwards of 200 Starbucks within a 5-miles radius in some dense cities of the world, that might imply 300 in our "square". That maps to about 4 disk blocks, and a modest amount of CPU to chew through the 300 records. Still not bad.

Now, suppose you are on an ocean liner somewhere in the Pacific. And there is one Starbucks onboard, but you are looking for the nearest 10. If you again start with 2 miles, it will take several iterations to find 10 sites. But, let's walk through it anyway. The first probe will hit one partition (maybe 2), and find just one hit. The second probe doubles the width of the square; 4 miles will still give you one hit -- the same hit in the same block, which is now cached, so we won't count it as a second disk I/O. Eventually the square will be wide enough to span multiple partitions. Each extra partition will be one new disk hit to discover no sites in the square. Finally, the square will hit Chile or Hawaii or Fiji and find some more sites, perhaps enough to stop the iteration. Since the main criteria in determining the number of disk hits is the number of partitions hit, we do not want to split the world into too many partitions. If there are, say, 40 partitions, then I have just described a case where there might be 20 disk hits.

2-degree partitions might be good for a global table of stores or restaurants. A 5-mile starting distance might be good when filtering for Starbucks. 20 miles might be better for a department store.

Now, let's discuss the 'last' SELECT, wherein the square is expanded by SQRT(2) and it uses the Great Circle formula to precisely order the N results. The SQRT(2) is in case that the N items were all at the corners of the 'square'. Growing the square by this much allows us to catch any other sites that were just outside the old square.

First, note that this 'last' SELECT is hitting the same block(s) that the iteration hit, plus possibly hitting some more blocks. It is hard to predict how many extra blocks might be hit. Here's a pathological case. You are in the middle of a desert; the square grows and grows. Eventually it finds N sites. There is a big city just outside the final square from the iterating. Now the 'last' SELECT kicks in, and it includes lots of sites in this big city. "Lots of sites" --> lots of blocks --> lots of disk hits.

Discussion of reference code

Here's the gist of the stored procedure FindNearest().

- Make a guess at how close to "me" to look.
- See how many items are in a 'square' around me, after filtering.
- If not enough, repeat, doubling the width of the square.
- After finding enough, or giving up because we are looking "too far", make one last pass to get all the data, ORDERed and LIMITed

Note that the loop merely uses 'squares' of lat/lng ranges. This is crude, but works well with the partitioning and indexing, and avoids calling to GCDist (until the last step). In the sample code, I picked 15 miles as starting value. Adjusting this will have some impact on the Procedure's performance, but the impact will vary with the use cases. A rough way to set the radius is to guess what will find the desired LIMIT about half the time. (This value is hardcoded in the PROCEDURE.)

Parameters passed into FindNearest():

- your Latitude -- -90..90 (not scaled -- see hardcoded conversion in PROCEDURE)
- your Longitude -- -180..180 (not scaled)
- Start distance -- (miles or km) -- see discussion below
- Max distance -- in miles or km -- see hardcoded conversion in PROCEDURE
- Limit -- maximum number of items to return
- Condition -- something to put after 'AND' (more discussion above)

The function will find the nearest items, up to Limit that meet the Condition. But it will give up at Max distance. (If you are at the South Pole, why bother searching very far for the tenth pizza parlor?)

Because of the "scaling", "hardcoding", "Condition", the table name, etc, this PROCEDURE is not truly generic; the code must be modified for each application. Yes, I could have designed it to pass all that stuff in. But what a mess.

The "_start_dist" gives some control over the performance. Making this too small leads to extra iterations; too big leads to more rows being checked. If you choose to tune the Stored Procedure, do the following. "SELECT @iterations" after calling the SP for a number of typical values. If the value is usually 1, then decrease _start_dist. If it is usually 2 or more, then increase it.

Timing: Under 10ms for "typical" usage; any dataset size. Slower for pathological cases (low min distance, high max distance, crossing dateline, bad filtering, cold cache, etc)

End-cases:

- By using GC distance, not Pythagoras, distances are 'correct' even near poles.
- Poles -- Even if the "nearest" is almost 360 degrees away (longitude), it can find it.
- Dateline -- There is a small, 'contained', piece of code for crossing the Dateline. Example: you are at +179 deg longitude, and the nearest item is at -179.

The procedure returns one resultset, SELECT *, distance.

- Only rows that meet your Condition, within Max distance are returned
- At most Limit rows are returned
- The rows will be ordered, "closest" first.
- "dist" will be in miles or km (based on a hardcoded constant in the SP)

Reference code, assuming deg*10000 and 'miles'

This version is based on scaling "Deg*10000 (MEDIUMINT)".

```
DELIMITER //

drop function if exists GCDist //
CREATE FUNCTION GCDist (
        _lat1 DOUBLE,  -- Scaled Degrees north for one point
        _lon1 DOUBLE,  -- Scaled Degrees west for one point
        _lat2 DOUBLE,  -- other point
        _lon2 DOUBLE
    ) RETURNS DOUBLE
    DETERMINISTIC
    CONTAINS SQL  -- SQL but does not read or write
    SQL SECURITY INVOKER  -- No special privileges granted
-- Input is a pair of latitudes/longitudes multiplied by 10000.
--    For example, the south pole has latitude -900000.
-- Multiply output by .0069172 to get miles between the two points
--    or by .0111325 to get kilometers
BEGIN
    -- Hardcoded constant:
    DECLARE _deg2rad DOUBLE DEFAULT PI()/1800000;  -- For scaled by 1e4 to MEDIUMINT
    DECLARE _rlat1 DOUBLE DEFAULT _deg2rad * _lat1;
    DECLARE _rlat2 DOUBLE DEFAULT _deg2rad * _lat2;
    -- compute as if earth's radius = 1.0
    DECLARE _rlond DOUBLE DEFAULT _deg2rad * (_lon1 - _lon2);
    DECLARE _m     DOUBLE DEFAULT COS(_rlat2);
    DECLARE _x     DOUBLE DEFAULT COS(_rlat1) - _m * COS(_rlond);
    DECLARE _y     DOUBLE DEFAULT                _m * SIN(_rlond);
    DECLARE _z     DOUBLE DEFAULT SIN(_rlat1) - SIN(_rlat2);
    DECLARE _n     DOUBLE DEFAULT SQRT(
                        _x * _x +
```

```
                        _x * _x +
                        _y * _y +
                        _z * _z    );
    RETURN  2 * ASIN(_n / 2) / _deg2rad;   -- again--scaled degrees
END;
//
DELIMITER ;

DELIMITER //
-- FindNearest (about my 6th approach)
drop procedure if exists FindNearest6 //
CREATE
PROCEDURE FindNearest (
        IN _my_lat DOUBLE,  -- Latitude of me [-90..90] (not scaled)
        IN _my_lon DOUBLE,  -- Longitude [-180..180]
        IN _START_dist DOUBLE,  -- Starting estimate of how far to search: miles or km
        IN _max_dist DOUBLE,  -- Limit how far to search: miles or km
        IN _limit INT,     -- How many items to try to get
        IN _condition VARCHAR(1111)   -- will be ANDed in a WHERE clause
    )
    DETERMINISTIC
BEGIN
    -- lat and lng are in degrees -90..+90 and -180..+180
    -- All computations done in Latitude degrees.
    -- Thing to tailor
    --   *Locations* -- the table
    --   Scaling of lat, lon; here using *10000 in MEDIUMINT
    --   Table name
    --   miles versus km.

    -- Hardcoded constant:
    DECLARE _deg2rad DOUBLE DEFAULT PI()/1800000;  -- For scaled by 1e4 to MEDIUMINT

    -- Cannot use params in PREPARE, so switch to @variables:
    -- Hardcoded constant:
    SET @my_lat := _my_lat * 10000,
        @my_lon := _my_lon * 10000,
        @deg2dist := 0.0069172,  -- 69.172 for miles; 111.325 for km  *** (mi vs km)
        @start_deg := _start_dist / @deg2dist,  -- Start with this radius first (eg, 15 miles)
        @max_deg := _max_dist / @deg2dist,
        @cutoff := @max_deg / SQRT(2),  -- (slightly pessimistic)
        @dlat := @start_deg,  -- note: must stay positive
        @lon2lat := COS(_deg2rad * @my_lat),
        @iterations := 0;        -- just debugging

    -- Loop through, expanding search
    --   Search a 'square', repeat with bigger square until find enough rows
    --   If the inital probe found _limit rows, then probably the first
    --   iteration here will find the desired data.
    -- Hardcoded table name:
    -- This is the "first SELECT":
    SET @sql = CONCAT(
        "SELECT COUNT(*) INTO @near_ct
            FROM Locations
            WHERE lat    BETWEEN @my_lat - @dlat
                             AND @my_lat + @dlat   -- PARTITION Pruning and bounding box
              AND lon    BETWEEN @my_lon - @dlon
                             AND @my_lon + @dlon   -- first part of PK
              AND ", _condition);
    PREPARE _sql FROM @sql;
    MainLoop: LOOP
        SET @iterations := @iterations + 1;
        -- The main probe: Search a 'square'
        SET @dlon := ABS(@dlat / @lon2lat);  -- good enough for now  -- note: must stay
positive
        -- Hardcoded constants:
        SET @dlon := IF(ABS(@my_lat) + @dlat >= 900000, 3600001, @dlon);  -- near a Pole
        EXECUTE _sql;
        IF ( @near_ct >= _limit OR         -- Found enough
             @dlat >= @cutoff ) THEN       -- Give up (too far)
            LEAVE MainLoop;
        END IF;
        -- Expand 'square':
        SET @dlat := LEAST(2 * @dlat, @cutoff);   -- Double the radius to search
    END LOOP MainLoop;
```

```
    DEALLOCATE PREPARE _sql;

    -- Out of loop because found _limit items, or going too far.
    -- Expand range by about 1.4 (but not past _max_dist),
    -- then fetch details on nearest 10.

    -- Hardcoded constant:
    SET @dlat := IF( @dlat >= @max_deg OR @dlon >= 1800000,
                @max_deg,
                GCDist(ABS(@my_lat), @my_lon,
                       ABS(@my_lat) - @dlat, @my_lon - @dlon) );
            -- ABS: go toward equator to find farthest corner (also avoids poles)
            -- Dateline: not a problem (see GCDist code)

    -- Reach for longitude line at right angle:
    -- sin(dlon)*cos(lat) = sin(dlat)
    -- Hardcoded constant:
    SET @dlon := IFNULL(ASIN(SIN(_deg2rad * @dlat) /
                             COS(_deg2rad * @my_lat))
                           / _deg2rad -- precise
                       , 3600001);    -- must be too near a pole

    -- This is the "last SELECT":
    -- Hardcoded constants:
    IF (ABS(@my_lon) + @dlon < 1800000 OR    -- Usual case - not crossing dateline
        ABS(@my_lat) + @dlat <  900000) THEN -- crossing pole, so dateline not an issue
        -- Hardcoded table name:
        SET @sql = CONCAT(
            "SELECT *,
                    @deg2dist * GCDist(@my_lat, @my_lon, lat, lon) AS dist
                FROM Locations
                WHERE lat BETWEEN @my_lat - @dlat
                              AND @my_lat + @dlat   -- PARTITION Pruning and bounding box
                  AND lon BETWEEN @my_lon - @dlon
                              AND @my_lon + @dlon   -- first part of PK
                  AND ", _condition, "
                HAVING dist <= ", _max_dist, "
                ORDER BY dist
                LIMIT ", _limit
                       );
    ELSE
        -- Hardcoded constants and table name:
        -- Circle crosses dateline, do two SELECTs, one for each side
        SET @west_lon := IF(@my_lon < 0, @my_lon, @my_lon - 3600000);
        SET @east_lon := @west_lon + 3600000;
        -- One of those will be beyond +/- 180; this gets points beyond the dateline
        SET @sql = CONCAT(
            "( SELECT *,
                    @deg2dist * GCDist(@my_lat, @west_lon, lat, lon) AS dist
                FROM Locations
                WHERE lat BETWEEN @my_lat - @dlat
                              AND @my_lat + @dlat   -- PARTITION Pruning and bounding box
                  AND lon BETWEEN @west_lon - @dlon
                              AND @west_lon + @dlon   -- first part of PK
                  AND ", _condition, "
                HAVING dist <= ", _max_dist, " )
            UNION ALL
            ( SELECT *,
                    @deg2dist * GCDist(@my_lat, @east_lon, lat, lon) AS dist
                FROM Locations
                WHERE lat BETWEEN @my_lat - @dlat
                              AND @my_lat + @dlat   -- PARTITION Pruning and bounding box
                  AND lon BETWEEN @east_lon - @dlon
                              AND @east_lon + @dlon   -- first part of PK
                  AND ", _condition, "
                HAVING dist <= ", _max_dist, " )
            ORDER BY dist
            LIMIT ", _limit
                       );
    END IF;

    PREPARE _sql FROM @sql;
    EXECUTE _sql;
    DEALLOCATE PREPARE _sql;
END;
```

```
//
DELIMITER ;
<<code>>

== Sample

Find the 5 cities with non-zero population (out of 3 million) nearest to (+35.15, -90.15).
Start with a 10-mile bounding box and give up at 100 miles.

<<code>>
CALL FindNearestLL(35.15, -90.05, 10, 100, 5, 'population > 0');
+---------+--------+---------+---------+-------------+-------------+-------+------------+--
-----------+---------------------+------------------------+
| id      | lat    | lon     | country | ascii_city   | city         | state | population |
@gcd_ct := 0 | dist                | @gcd_ct := @gcd_ct + 1 |
+---------+--------+---------+---------+-------------+-------------+-------+------------+--
-----------+---------------------+------------------------+
| 3023545 | 351494 | -900489 | us      | memphis      | Memphis      | TN    |     641608 |
0 | 0.07478733189367963 |                     3 |
| 2917711 | 351464 | -901844 | us      | west memphis | West Memphis | AR    |      28065 |
0 |   7.605683607627499 |                     2 |
| 2916457 | 352144 | -901964 | us      | marion       | Marion       | AR    |       9227 |
0 |     9.3994963998986 |                     1 |
| 3020923 | 352044 | -898739 | us      | bartlett     | Bartlett     | TN    |      43264 |
0 |  10.643941157860604 |                     7 |
| 2974644 | 349889 | -900125 | us      | southaven    | Southaven    | MS    |      38578 |
0 |  11.344042217329935 |                     5 |
+---------+--------+---------+---------+-------------+-------------+-------+------------+--
-----------+---------------------+------------------------+
5 rows in set (0.00 sec)
Query OK, 0 rows affected (0.04 sec)

SELECT COUNT(*) FROM ll_table;
+----------+
| COUNT(*) |
+----------+
|  3173958 |
+----------+
1 row in set (5.04 sec)

FLUSH STATUS;
CALL...
SHOW SESSION STATUS LIKE 'Handler%';

show session status like 'Handler%';
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| Handler_read_first         | 1     |
| Handler_read_key           | 3     |
| Handler_read_next          | 1307  |  -- some index, some tmp, but far less than 3 million.
| Handler_read_rnd           | 5     |
| Handler_read_rnd_next      | 13    |
| Handler_write              | 12    |  -- it needed a tmp
+----------------------------+-------+
```

Postlog

There is a "Haversine" algorithm that is twice as fast as the GCDist function here. But it has a fatal flaw of sometimes returning NULL for the distance between a point and itself. (This is because of computing a number slightly bigger than 1.0, then trying to take the ACOS of it.)

3.3.3.12 Primary Keys with Nullable Columns

MariaDB deals with primary keys over nullable columns according to the SQL standards.

Take the following table structure:

```
CREATE TABLE t1(
  c1 INT NOT NULL AUTO_INCREMENT,
  c2 INT NULL DEFAULT NULL,
  PRIMARY KEY(c1,c2)
);
```

Column c2 is part of a primary key, and thus it cannot be NULL.

Before MariaDB 10.1.7, MariaDB (as well as versions of MySQL before MySQL 5.7) would silently convert it into a NOT NULL column with a default value of *0*.

Since MariaDB 10.1.7, the column is converted to NOT NULL, but without a default value. If we then attempt to insert a record without explicitly setting *c2*, a warning (or, in strict mode, an error), will be thrown, for example:

```
INSERT INTO t1() VALUES();
Query OK, 1 row affected, 1 warning (0.00 sec)
Warning (Code 1364): Field 'c2' doesn't have a default value

SELECT * FROM t1;
+----+----+
| c1 | c2 |
+----+----+
|  1 |  0 |
+----+----+
```

MySQL, since 5.7, will abort such a CREATE TABLE with an error.

The MariaDB 10.1.7 behavior adheres to the SQL 2003 standard.

SQL-2003, Part II, “Foundation” says:

11.7 <unique constraint definition>
Syntax Rules

…

5) If the <unique specification> specifies PRIMARY KEY, then for each <column name> in the explicit or implicit <unique column list> for which NOT NULL is not specified, NOT NULL is implicit in the <column definition>.

Essentially this means that all PRIMARY KEY columns are automatically converted to NOT NULL. Furthermore:

11.5 <default clause>
General Rules

…

3) When a site S is set to its default value,

…

b) If the data descriptor for the site includes a <default option>, then S is set to the value specified by that <default option>.

…

e) Otherwise, S is set to the null value.

There is no concept of “no default value” in the standard. Instead, a column always has an implicit default value of NULL. On insertion it might however fail the NOT NULL constraint. MariaDB and MySQL instead mark such a column as “not having a default value”. The end result is the same — a value must be specified explicitly or an INSERT will fail.

MariaDB since 10.1.7 behaves in a standard compatible manner — being part of a PRIMARY KEY, the nullable column gets an automatic NOT NULL constraint, on insertion one must specify a value for such a column. MariaDB before 10.1.7 was automatically assigning a default value of 0 — this behavior was non-standard. Issuing an error at CREATE TABLE time is also non-standard.

1.1.1.2.2.7 SHOW EXPLAIN

1.1.3.15 Spatial Index

3.3.3.15 Storage Engine Index Types

Contents

This refers to the index_type definition when creating an index, i.e. BTREE, HASH or RTREE.

For more information on general types of indexes, such as primary keys, unique indexes etc, go to Getting Started with Indexes.

Storage Engine	Permitted Indexes
Aria	BTREE, RTREE
MyISAM	BTREE, RTREE
InnoDB	BTREE
MEMORY/HEAP	HASH, BTREE

BTREE is generally the default index type. For MEMORY tables, HASH is the default. TokuDB uses a particular data structure called *fractal trees*, which is optimized for data that do not entirely fit memory.

Understanding the B-tree and hash data structures can help predict how different queries perform on different storage engines that use these data structures in their indexes, particularly for the MEMORY storage engine that lets you choose B-tree or hash indexes. B-Tree Index Characteristics

B-tree Indexes

B-tree indexes are used for column comparisons using the >, >=, =, >=, < or BETWEEN operators, as well as for LIKE comparisons that begin with a constant.

For example, the query `SELECT * FROM Employees WHERE First_Name LIKE 'Maria%';` can make use of a B-tree index, while `SELECT * FROM Employees WHERE First_Name LIKE '%aria';` cannot.

B-tree indexes also permit leftmost prefixing for searching of rows.

If the number or rows doesn't change, hash indexes occupy a fixed amount of memory, which is lower than the memory occupied by BTREE indexes.

Hash Indexes

Hash indexes, in contrast, can only be used for equality comparisons, so those using the = or <=> operators. They cannot be used for ordering, and provide no information to the optimizer on how many rows exist between two values.

Hash indexes do not permit leftmost prefixing - only the whole index can be used.

R-tree Indexes

See SPATIAL for more information.

3.3.4 Query Optimizations

Different query optimizations and how you can use and tune them to get better performance.

Index Hints: How to Force Query Plans

Using hints to get the optimizer to use another query plan.

Subquery Optimizations

Articles about subquery optimizations in MariaDB.

Optimization Strategies

Various optimization strategies used by the query optimizer.

Optimizations for Derived Tables

Optimizations for derived tables, or subqueries in the FROM clause

Table Elimination
Resolving queries without accessing some of the tables the query refers to

Statistics for Optimizing Queries
Different statistics provided by MariaDB to help you optimize your queries

Filesort with Small LIMIT Optimization
Filesort with small LIMIT optimization.

LIMIT ROWS EXAMINED
Means to terminate execution of SELECTs that examine too many rows.

index_merge sort_intersection
Operation to allow the use of index_merge in a broader number of cases.

MariaDB 5.3 Optimizer Debugging
MariaDB 5.3's optimizer debugging patch

optimizer_switch
Server variable for enabling specific optimizations.

How to Quickly Insert Data Into MariaDB
Techniques for inserting data quickly into MariaDB.

Index Condition Pushdown
Index Condition Pushdown optimization.

Query Limits and Timeouts
Different methods MariaDB provides to limit/timeout a query.

Aborting Statements that Exceed a Certain Time to Execute
Aborting statements that take longer than a certain time to execute.

Partition Pruning and Selection
Partition pruning is when the optimizer knows which partitions are relevant for the query.

Big DELETEs
How to DELETE lots of rows from a large table

Data Sampling: Techniques for Efficiently Finding a Random Row
Fetching random rows from a table (beyond ORDER BY RAND())

Data Warehousing High Speed Ingestion
Ingesting lots of data and performance is bottlenecked in the INSERT area. What to do?

Data Warehousing Summary Tables
Creation and maintenance of summary tables

Data Warehousing Techniques
Improving performance for data-warehouse-like tables

Equality propagation optimization
Basic idea Consider a query with a WHERE clause: WHERE col1=col2 AND ... t...

FORCE INDEX
Similar to USE INDEX, but tells the optimizer to regard a table scan as very expensive.

Groupwise Max in MariaDB
Finding the largest row for each group.

GUID/UUID Performance
GUID/UUID performance (type 1 only).

hash_join_cardinality optimizer_switch Flag

MariaDB starting with <a href="#https://mariadb.com/kb/en/mariadb-10-6-13-release-notes/">10.6.13...

IGNORE INDEX

Tell the optimizer to not consider a particular index.

not_null_range_scan Optimization

Enables constructing range scans from NOT NULL conditions inferred from the WHERE clause.

Optimizing for "Latest News"-style Queries

Optimizing the schema and code for "Latest News"-style queries

Pagination Optimization

Pagination, not with OFFSET, LIMIT

Pivoting in MariaDB

Pivoting data so a linear list of values with two keys becomes a spreadsheet-like array.

Rollup Unique User Counts

Technique for counting unique users

Rowid Filtering Optimization

Rowid filtering is an optimization available from MariaDB 10.4.

Sargable UPPER

Starting from MariaDB 11.3, expressions in the form UPPER(key_col) = expr ...

USE INDEX

Find rows in the table using only one of the named indexes.

There are 2 related questions.

3.3.4.1 Index Hints: How to Force Query Plans

Contents

The optimizer is largely cost-based and will try to choose the optimal plan for any query. However in some cases it does not have enough information to choose a perfect plan and in these cases you may have to provide hints to force the optimizer to use another plan.

You can examine the query plan for a SELECT by writing EXPLAIN before the statement. SHOW EXPLAIN shows the output of a running query. In some cases, its output can be closer to reality than `EXPLAIN`.

For the following queries, we will use the world database for the examples.

Setting up the World Example Database

Download it from ftp://ftp.askmonty.org/public/world.sql.gz

Install it with:

```
mariadb-admin create world
zcat world.sql.gz | ../client/mysql world
```

or

```
mariadb-admin create world
gunzip world.sql.gz
../client/mysql world < world.sql
```

Forcing Join Order

You can force the join order by using STRAIGHT_JOIN either in the SELECT or JOIN part.

The simplest way to force the join order is to put the tables in the correct order in the `FROM` clause and use `SELECT STRAIGHT_JOIN` like so:

```
SELECT STRAIGHT_JOIN SUM(City.Population) FROM Country,City WHERE
City.CountryCode=Country.Code AND Country.HeadOfState="Volodymyr Zelenskyy";
```

If you only want to force the join order for a few tables, use `STRAIGHT_JOIN` in the `FROM` clause. When this is done, only tables connected with `STRAIGHT_JOIN` will have their order forced. For example:

```
SELECT SUM(City.Population) FROM Country STRAIGHT_JOIN City WHERE
City.CountryCode=Country.Code AND Country.HeadOfState="Volodymyr Zelenskyy";
```

In both of the above cases `Country` will be scanned first and for each matching country (one in this case) all rows in `City` will be checked for a match. As there is only one matching country this will be faster than the original query.

The output of EXPLAIN for the above cases is:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	Country	ALL	PRIMARY	NULL	NULL	NULL	239	Using where
1	SIMPLE	City	ALL	NULL	NULL	NULL	NULL	4079	Using where; Using join buffer (flat, BNL join)

This is one of the few cases where `ALL` is ok, as the scan of the `Country` table will only find one matching row.

Forcing Usage of a Specific Index for the WHERE Clause

In some cases the optimizer may choose a non-optimal index or it may choose to not use an index at all, even if some index could theoretically be used.

In these cases you have the option to either tell the optimizer to only use a limited set of indexes, ignore one or more indexes, or force the usage of some particular index.

USE INDEX: Use a Limited Set of Indexes

You can limit which indexes are considered with the USE INDEX option.

```
USE INDEX [{FOR {JOIN|ORDER BY|GROUP BY}] ([index_list])
```

The default is '`FOR JOIN`', which means that the hint only affects how the `WHERE` clause is optimized.

`USE INDEX` is used after the table name in the `FROM` clause.

Example:

```
CREATE INDEX Name ON City (Name);
CREATE INDEX CountryCode ON City (Countrycode);
EXPLAIN SELECT Name FROM City USE INDEX (CountryCode)
WHERE name="Helsingborg" AND countrycode="SWE";
```

This produces:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	City	ref	CountryCode	CountryCode	3	const	14	Using where

If we had not used USE INDEX, the `Name` index would have been in `possible keys`.

IGNORE INDEX: Don't Use a Particular Index

You can tell the optimizer to not consider some particular index with the IGNORE INDEX option.

```
IGNORE INDEX [{FOR {JOIN|ORDER BY|GROUP BY}] ([index_list])
```

This is used after the table name in the `FROM` clause:

```
CREATE INDEX Name ON City (Name);
CREATE INDEX CountryCode ON City (Countrycode);
EXPLAIN SELECT Name FROM City IGNORE INDEX (Name)
WHERE name="Helsingborg" AND countrycode="SWE";
```

This produces:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	City	ref	CountryCode	CountryCode	3	const	14	Using where

The benefit of using `IGNORE_INDEX` instead of `USE_INDEX` is that it will not disable a new index which you may add later.

Also see Ignored Indexes for an option to specify in the index definition that indexes should be ignored.

FORCE INDEX: Forcing an Index

Forcing an index to be used is mostly useful when the optimizer decides to do a table scan even if you know that using an index would be better. (The optimizer could decide to do a table scan even if there is an available index when it believes that most or all rows will match and it can avoid the overhead of using the index).

```
CREATE INDEX Name ON City (Name);
EXPLAIN SELECT Name,CountryCode FROM City FORCE INDEX (Name)
WHERE name>="A" and CountryCode >="A";
```

This produces:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	City	range	Name	Name	35	NULL	4079	Using where

`FORCE_INDEX` works by only considering the given indexes (like with `USE_INDEX`) but in addition it tells the optimizer to regard a table scan as something very expensive. However if none of the 'forced' indexes can be used, then a table scan will be used anyway.

Index Prefixes

When using index hints (USE, FORCE or IGNORE INDEX), the index name value can also be an unambiguous prefix of an index name.

Forcing an Index to be Used for ORDER BY or GROUP BY

The optimizer will try to use indexes to resolve ORDER BY and GROUP BY.

You can use USE INDEX, IGNORE INDEX and FORCE INDEX as in the `WHERE` clause above to ensure that some specific index used:

```
USE INDEX [{FOR {JOIN|ORDER BY|GROUP BY}] ([index_list])
```

This is used after the table name in the `FROM` clause.

Example:

```
CREATE INDEX Name ON City (Name);
EXPLAIN SELECT Name,Count(*) FROM City
FORCE INDEX FOR GROUP BY (Name)
WHERE population >= 10000000 GROUP BY Name;
```

This produces:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	City	index	NULL	Name	35	NULL	4079	Using where

Without the FORCE INDEX option we would have ' `Using where; Using temporary; Using filesort` ' in the 'Extra' column, which means that the optimizer would created a temporary table and sort it.

Help the Optimizer Optimize GROUP BY and ORDER BY

The optimizer uses several strategies to optimize GROUP BY and ORDER BY:

- Resolve with an index:
  - Scan the table in index order and output data as we go. (This only works if the ORDER BY / GROUP BY can be resolved by an index after constant propagation is done).
- Filesort:
  - Scan the table to be sorted and collect the sort keys in a temporary file.
  - Sort the keys + reference to row (with filesort)
  - Scan the table in sorted order
- Use a temporary table for ORDER BY:
  - Create a temporary (in memory) table for the 'to-be-sorted' data. (If this gets bigger than `max_heap_table_size` or contains blobs then an Aria or MyISAM disk based table will be used)
  - Sort the keys + reference to row (with filesort)
  - Scan the table in sorted order

A temporary table will always be used if the fields which will be sorted are not from the first table in the JOIN order.

- Use a temporary table for GROUP BY:
  - Create a temporary table to hold the GROUP BY result with an index that matches the GROUP BY fields.
  - Produce a result row
  - If a row with the GROUP BY key exists in the temporary table, add the new result row to it. If not, create a new row.
  - Before sending the results to the user, sort the rows with filesort to get the results in the GROUP BY order.

Forcing/Disallowing TemporaryTables to be Used for GROUP BY:

Using an in-memory table (as described above) is usually the fastest option for GROUP BY if the result set is small. It is not optimal if the result set is very big. You can tell the optimizer this by using `SELECT SQL_SMALL_RESULT` or `SELECT SQL_BIG_RESULT`.

For example:

```
EXPLAIN SELECT SQL_SMALL_RESULT Name,Count(*) AS Cities FROM City GROUP BY Name HAVING Cities > 2
```

produces:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	City	ALL	NULL	NULL	NULL	NULL	4079	Using temporary; Using filesort

while:

```
EXPLAIN SELECT SQL_BIG_RESULT Name,Count(*) AS Cities FROM City
GROUP BY Name HAVING Cities > 2;
```

produces:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra

1	SIMPLE	City	ALL	NULL	NULL	NULL	NULL	4079	Using filesort

The difference is that with `SQL_SMALL_RESULT` a temporary table is used.

Forcing Usage of Temporary Tables

In some cases you may want to force the use of a temporary table for the result to free up the table/row locks for the used tables as quickly as possible.

You can do this with the `SQL_BUFFER_RESULT` option:

```
CREATE INDEX Name ON City (Name);
EXPLAIN SELECT SQL_BUFFER_RESULT Name,Count(*) AS Cities FROM City
GROUP BY Name HAVING Cities > 2;
```

This produces:

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	City	index	NULL	Name	35	NULL	4079	Using index; Using temporary

Without `SQL_BUFFER_RESULT`, the above query would not use a temporary table for the result set.

Optimizer Switch

In MariaDB 5.3 we added an optimizer switch which allows you to specify which algorithms will be considered when optimizing a query.

See the optimizer section for more information about the different algorithms which are used.

3.3.4.2 Subquery Optimizations

Articles about subquery optimizations in MariaDB.

Subquery Optimizations Map
Map showing types of subqueries and the optimizer strategies available to handle them

Semi-join Subquery Optimizations
MariaDB has a set of optimizations specifically targeted at semi-join subqueries.

Table Pullout Optimization
Table pullout is an optimization for Semi-join subqueries.

Non-semi-join Subquery Optimizations
Alternative strategies for IN-subqueries that cannot be flattened into semi-joins

Subquery Cache
Subquery cache for optimizing the evaluation of correlated subqueries.

Condition Pushdown Into IN subqueries
This article describes Condition Pushdown into IN subqueries as implemented...

Conversion of Big IN Predicates Into Subqueries
The optimizer will convert big IN predicates into subqueries.

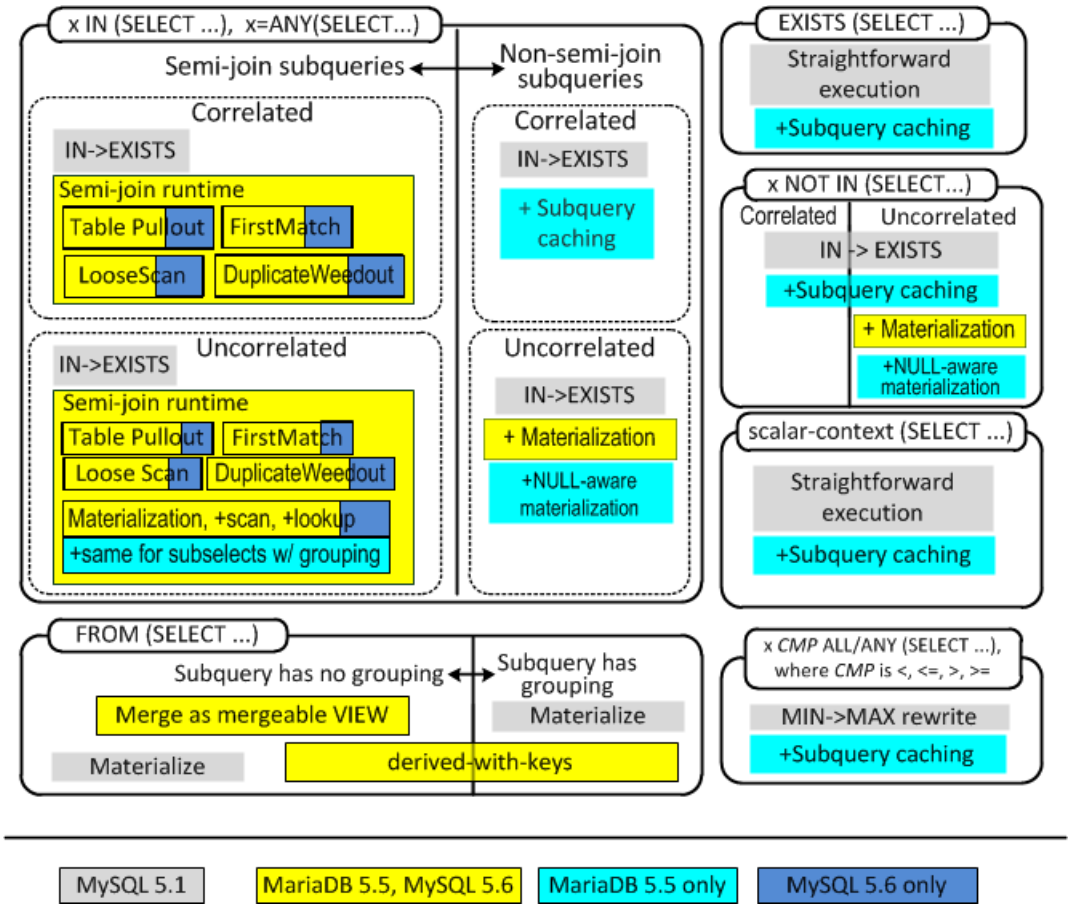
EXISTS-to-IN Optimization
Optimizations for IN subqueries.

Optimizing GROUP BY and DISTINCT Clauses in Subqueries
MariaDB removes DISTINCT and GROUP BY without HAVING in certain cases

3.3.4.2.1 Subquery Optimizations Map

Below is a map showing all types of subqueries allowed in the SQL language, and the optimizer strategies available to handle them.

- Uncolored areas represent different kinds of subqueries, for example:
  - Subqueries that have form `x IN (SELECT ...)`
  - Subqueries that are in the `FROM` clause
  - .. and so forth
- The size of each uncolored area roughly corresponds to how important (i.e. frequently used) that kind of subquery is. For example, `x IN (SELECT ...)` queries are the most important, and `EXISTS (SELECT ...)` are relatively unimportant.
- Colored areas represent optimizations/execution strategies that are applied to handle various kinds of subqueries.
- The color of optimization indicates which version of MySQL/MariaDB it was available in (see legend below)



Some things are not on the map:

- MariaDB doesn't evaluate expensive subqueries when doing optimization (this means, EXPLAIN is always fast). MySQL 5.6 has made a progress in this regard but its optimizer will still evaluate certain kinds of subqueries (for example, scalar-context subqueries used in range predicates)

Links to pages about individual optimizations:

- IN->EXISTS
- Subquery Caching
- Semi-join optimizations
  - Table pullout
  - FirstMatch
  - Materialization, +scan, +lookup
  - LooseScan
  - DuplicateWeedout execution strategy
- Non-semi-join Materialization (including NULL-aware and partial matching)
- Derived table optimizations
  - Derived table merge
  - Derived table with keys

3.3.4.2.2 Semi-join Subquery Optimizations

Contents

MariaDB has a set of optimizations specifically targeted at *semi-join subqueries*.

What is a Semi-Join Subquery

A semi-join subquery has a form of

```
SELECT ... FROM outer_tables WHERE expr IN (SELECT ... FROM inner_tables ...) AND ...
```

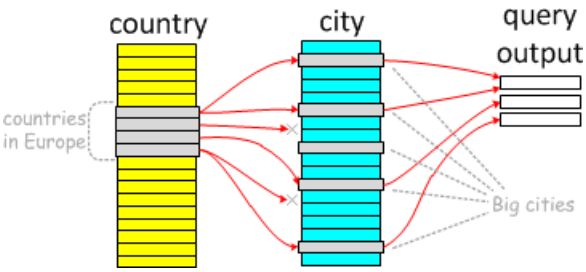
that is, the subquery is an IN-subquery and it is located in the WHERE clause. The most important part here is

with semi-join subquery, we're only interested in records of outer_tables that have matches in the subquery

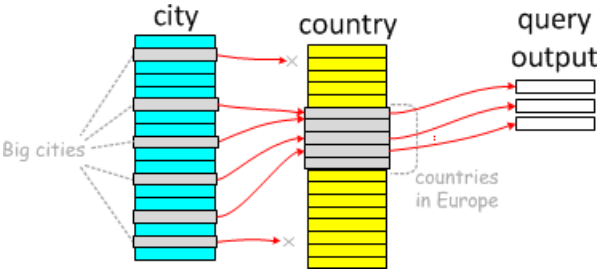
Let's see why this is important. Consider a semi-join subquery:

```
select * from Country
where
  Continent='Europe' and
  Country.Code in (select City.country
                   from City
                   where City.Population>1*1000*1000);
```

One can execute it "naturally", by starting from countries in Europe and checking if they have populous Cities:



The semi-join property also allows "backwards" execution: we can start from big cities, and check which countries they are in:

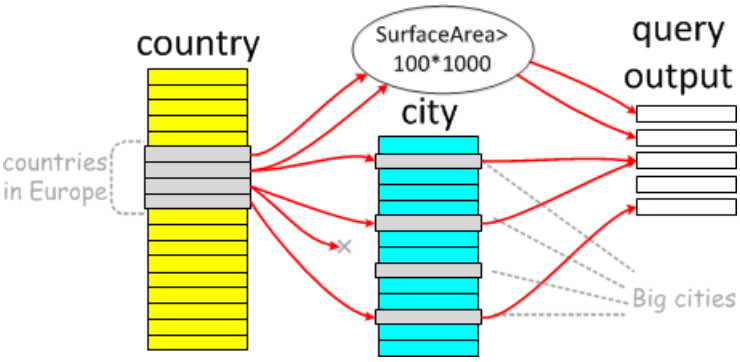


To contrast, let's change the subquery to be non-semi-join:

```
select * from Country
where
   Country.Continent='Europe' and
   (Country.Code in (select City.country
                     from City where City.Population>1*1000*1000)
    or Country.SurfaceArea > 100*1000  -- Added this part
   );
```

It is still possible to start from countries, and then check

- if a country has any big cities
- if it has a large surface area:



The opposite, city-to-country way is not possible. This is not a semi-join.

Difference from Inner Joins

Semi-join operations are similar to regular relational joins. There is a difference though: with semi-joins, you don't care how many matches an inner table has for an outer row. In the above countries-with-big-cities example, Germany will be returned once, even if it has three cities with populations of more than one million each.

Semi-Join Optimizations in MariaDB

MariaDB uses semi-join optimizations to run IN subqueries.The optimizations are enabled by default. You can disable them by turning off their optimizer_switch like so:

```
SET optimizer_switch='semijoin=off'
```

MariaDB has five different semi-join execution strategies:

- Table pullout optimization
- FirstMatch execution strategy
- Semi-join Materialization execution strategy
- LooseScan execution strategy
- DuplicateWeedout execution strategy

3.3.4.2.3 Table Pullout Optimization

Contents

Table pullout is an optimization for Semi-join subqueries.

The idea of Table Pullout

Sometimes, a subquery can be re-written as a join. For example:

```
select *
from City
where City.Country in (select Country.Code
                       from Country
                       where Country.Population < 100*1000);
```

If we know that there can be, at most, one country with with a given value of `Country.Code` (we can tell that if we see that table Country has a primary key or unique index over that column), we can re-write this query as:

```
select City.*
from
  City, Country
where
 City.Country=Country.Code AND Country.Population < 100*1000;
```

Table pullout in action

If one runs EXPLAIN for the above query in MySQL 5.1-5.6 or MariaDB 5.1-5.2, they'll get this plan:

```
MySQL [world]> explain select * from City where City.Country in (select Country.Code from
Country where Country.Population < 100*1000);
+----+--------------------+---------+----------------+-------------------+---------+-------
--+------+-------------+
| id | select_type        | table   | type           | possible_keys     | key     | key_len
| ref  | rows | Extra       |
+----+--------------------+---------+----------------+-------------------+---------+-------
--+------+-------------+
|  1 | PRIMARY            | City    | ALL            | NULL              | NULL    | NULL
| NULL | 4079 | Using where |
|  2 | DEPENDENT SUBQUERY | Country | unique_subquery | PRIMARY,Population | PRIMARY | 3
| func |    1 | Using where |
+----+--------------------+---------+----------------+-------------------+---------+-------
--+------+-------------+
2 rows in set (0.00 sec)
```

It shows that the optimizer is going to do a full scan on table `City`, and for each city it will do a lookup in table `Country`.

If one runs the same query in MariaDB 5.3, they will get this plan:

```
MariaDB [world]> explain select * from City where City.Country in (select Country.Code from
Country where Country.Population < 100*1000);
+----+-------------+---------+-------+-------------------+------------+---------+-----------
---------+------+-----------------------+
| id | select_type | table   | type  | possible_keys     | key        | key_len | ref
| rows | Extra                 |
+----+-------------+---------+-------+-------------------+------------+---------+-----------
---------+------+-----------------------+
|  1 | PRIMARY     | Country | range | PRIMARY,Population | Population | 4       | NULL
|   37 | Using index condition |
|  1 | PRIMARY     | City    | ref   | Country           | Country    | 3       |
world.Country.Code |   18 |                       |
+----+-------------+---------+-------+-------------------+------------+---------+-----------
---------+------+-----------------------+
2 rows in set (0.00 sec)
```

The interesting parts are:

- Both tables have `select_type=PRIMARY`, and `id=1` as if they were in one join.
- The `Country` table is first, followed by the `City` table.

Indeed, if one runs EXPLAIN EXTENDED; SHOW WARNINGS, they will see that the subquery is gone and it was replaced with a join:

```
MariaDB [world]> show warnings\G
*************************** 1. row ***************************
  Level: Note
   Code: 1003
Message: select `world`.`City`.`ID` AS `ID`,`world`.`City`.`Name` AS
`Name`,`world`.`City`.`Country` AS `Country`,`world`.`City`.`Population` AS
`Population`


   from `world`.`City` join `world`.`Country` where


((`world`.`City`.`Country` = `world`.`Country`.`Code`) and (`world`.`Country`.
`Population` < (100 * 1000)))
1 row in set (0.00 sec)
```

Changing the subquery into a join allows feeding the join to the join optimizer, which can make a choice between two possible join orders:

1. City -> Country
2. Country -> City

as opposed to the single choice of

1. City->Country

which we had before the optimization.

In the above example, the choice produces a better query plan. Without pullout, the query plan with a subquery would read `(4079 + 1*4079)=8158` table records. With table pullout, the join plan would read `(37 + 37 * 18) = 703` rows. Not all row reads are equal, but generally, reading `10` times fewer table records is faster.

Table pullout fact sheet

- Table pullout is possible only in semi-join subqueries.
- Table pullout is based on `UNIQUE` / `PRIMARY` key definitions.
- Doing table pullout does not cut off any possible query plans, so MariaDB will always try to pull out as much as possible.
- Table pullout is able to pull individual tables out of subqueries to their parent selects. If all tables in a subquery have been pulled out, the subquery (i.e. its semi-join) is removed completely.
- One common bit of advice for optimizing MySQL has been "If possible, rewrite your subqueries as joins". Table pullout does exactly that, so manual rewrites are no longer necessary.

Controlling table pullout

There is no separate @@optimizer_switch flag for table pullout. Table pullout can be disabled by switching off all semi-join optimizations with `SET @@optimizer_switch='semijoin=off'` command.

3.3.4.2.4 Non-semi-join Subquery Optimizations

Contents

Certain kinds of IN-subqueries cannot be flattened into semi-joins. These subqueries can be both correlated or non-correlated. In order to provide consistent performance in all cases, MariaDB provides several alternative strategies for these types of subqueries. Whenever several strategies are possible, the optimizer chooses the optimal one based on cost estimates.

The two primary non-semi-join strategies are materialization (also called outside-in materialization), and in-to-exists transformation. Materialization is applicable only for non-correlated subqueries, while in-to-exist can be used both for correlated and non-correlated subqueries.

Applicability

An IN subquery cannot be flattened into a semi-join in the following cases. The examples below use the *World* database from the MariaDB regression test suite.

Subquery in a disjunction (OR)

The subquery is located directly or indirectly under an OR operation in the WHERE clause of the outer query.

Query pattern:

```
SELECT ... FROM ... WHERE (expr1, ..., exprN) [NOT] IN (SELECT ... ) OR expr;
```

Example:

```
SELECT Name FROM Country
WHERE (Code IN (select Country from City where City.Population > 100000) OR
       Name LIKE 'L%') AND
      surfacearea > 1000000;
```

Negated subquery predicate (NOT IN)

The subquery predicate itself is negated.

Query pattern:

```
SELECT ... FROM ... WHERE ... (expr1, ..., exprN) NOT IN (SELECT ... ) ...;
```

Example:

```
SELECT Country.Name
FROM Country, CountryLanguage
WHERE Code NOT IN (SELECT Country FROM CountryLanguage WHERE Language = 'English')
  AND CountryLanguage.Language = 'French'
  AND Code = Country;
```

Subquery in the SELECT or HAVING clause

The subquery is located in the SELECT or HAVING clauses of the outer query.

Query pattern:

```
SELECT field1, ..., (SELECT ...)  WHERE ...;
SELECT ...  WHERE ... HAVING (SELECT ...);
```

Example:

```
select Name, City.id in (select capital from Country where capital is not null) as is_capital
from City
where City.population > 10000000;
```

Subquery with a UNION

The subquery itself is a UNION, while the IN predicate may be anywhere in the query where IN is allowed.

Query pattern:

```
... [NOT] IN (SELECT ... UNION SELECT ...)
```

Example:

```
SELECT * from City where (Name, 91) IN
(SELECT Name, round(Population/1000) FROM City WHERE Country = "IND" AND Population > 2500000
UNION
 SELECT Name, round(Population/1000) FROM City WHERE Country = "IND" AND Population < 100000);
```

Materialization for non-correlated IN-subqueries

Materialization basics

The basic idea of subquery materialization is to execute the subquery and store its result in an internal temporary table indexed on all its columns. Naturally, this is possible only when the subquery is non-correlated. The IN predicate tests whether its left operand is present in the subquery result. Therefore it is not necessary to store duplicate subquery result rows in the temporary table. Storing only unique subquery rows provides two benefits - the size of the temporary table is smaller, and the index on all its columns can be unique.

If the size of the temporary table is less than the tmp_table_size system variable, the table is a hash-indexed in-memory HEAP table. In the rare cases when the subquery result exceeds this limit, the temporary table is stored on disk in an ARIA

or MyISAM B-tree indexed table (ARIA is the default).

Subquery materialization happens on demand during the first execution of the IN predicate. Once the subquery is materialized, the IN predicate is evaluated very efficiently by index lookups of the outer expression into the unique index of the materialized temporary table. If there is a match, IN is TRUE, otherwise IN is FALSE.

NULL-aware efficient execution

An IN predicate may produce a NULL result if there is a NULL value in either of its arguments. Depending on its location in a query, a NULL predicate value is equivalent to FALSE. These are the cases when substituting NULL with FALSE would reject exactly the same result rows. A NULL result of IN is indistinguishable from a FALSE if the IN predicate is:

- not negated,
- not a function argument,
- inside a WHERE or ON clause.

In all these cases the evaluation of IN is performed as described in the previous paragraph via index lookups into the materialized subquery. In all remaining cases when NULL cannot be substituted with FALSE, it is not possible to use index lookups. This is not a limitation in the server, but a consequence of the NULL semantics in the ANSI SQL standard.

Suppose an IN predicate is evaluated as

```
NULL IN (select
not_null_col from t1)
```

, that is, the left operand of IN is a NULL value, and there are no NULLs in the subquery. In this case the value of IN is neither FALSE, nor TRUE. Instead it is NULL. If we were to perform an index lookup with the NULL as a key, such a value would not be found in not_null_col, and the IN predicate would incorrectly produce a FALSE.

In general, an NULL value on either side of an IN acts as a "wildcard" that matches any value, and if a match exists, the result of IN is NULL. Consider the following example:

If the left argument of IN is the row: `(7, NULL, 9)`
, and the result of the right subquery operand of IN is the table:

```
(7, 8, 10)
(6, NULL, NULL)
(7, 11, 9)
```

The the IN predicate matches the row `(7, 11, 9)`
, and the result of IN is NULL. Matches where the differing values on either side of the IN arguments are matched by a NULL in the other IN argument, are called *partial matches*.

In order to efficiently compute the result of an IN predicate in the presence of NULLs, MariaDB implements two special algorithms for partial matching, described here in detail.

- Rowid-merge partial matching
  This technique is used when the number of rows in the subquery result is above a certain limit. The technique creates special indexes on some of the columns of the temporary table, and merges them by alternative scanning of each index thus performing an operation similar to set-intersection.
- Table scan partial matching
  This algorithm is used for very small tables when the overhead of the rowid-merge algorithm is not justifiable. Then the server simply scans the materialized subquery, and checks for partial matches. Since this strategy doesn't need any in-memory buffers, it is also used when there is not enough memory to hold the indexes of the rowid-merge strategy.

Limitations

In principle the subquery materialization strategy is universal, however, due to some technical limitations in the MariaDB server, there are few cases when the server cannot apply this optimization.

- BLOB fields
  Either the left operand of an IN predicate refers to a BLOB field, or the subquery selects one or more BLOBs.
- Incomparable fields
  TODO

In the above cases, the server reverts to the IN-TO-EXISTS transformation.

The IN-TO-EXISTS transformation

This optimization is the only subquery execution strategy that existed in older versions of MariaDB and MySQL prior to MariaDB 5.3. We have made various changes and fixed a number of bugs in this code as well, but in essence it remains the same.

Performance discussion

Example speedup over MySQL 5.x and MariaDB 5.1/5.2

Depending on the query and data, either of the two strategies described here may result in orders of magnitude better/worse plan than the other strategy.

Older versions of MariaDB and any current MySQL version (including MySQL 5.5, and MySQL 5.6 DMR as of July 2011) implement only the IN-TO-EXISTS transformation. As illustrated below, this strategy is inferior in many common cases to subquery materialization.

Consider the following query over the data of the DBT3 benchmark scale 10. Find customers with top balance in their nations:

```
SELECT * FROM part
WHERE p_partkey IN
      (SELECT l_partkey FROM lineitem
       WHERE l_shipdate between '1997-01-01' and '1997-02-01')
ORDER BY p_retailprice DESC LIMIT 10;
```

The times to run this query is as follows:

- Execution time in MariaDB 5.2/MySQL 5.x (any MySQL): **> 1 h**
  The query takes more than one hour (we didn't wait longer), which makes it impractical to use subqueries in such cases. The EXPLAIN below shows that the subquery was transformed into a correlated one, which indicates an IN-TO-EXISTS transformation.

```
+--+------------------+--------+--------------+------------------+----+------+--------------
-------------+
|id|select_type       |table   |type          |key               |ref |rows  |Extra
|
+--+------------------+--------+--------------+------------------+----+------+--------------
-------------+
| 1|PRIMARY           |part    |ALL           |NULL              |NULL|199755|Using where;
Using filesort|
| 2|DEPENDENT SUBQUERY|lineitem|index_subquery|i_l_suppkey_partkey|func|    14|Using where
|
+--+------------------+--------+--------------+------------------+----+------+--------------
-------------+
```

- Execution time in MariaDB 5.3: **43 sec**
  In MariaDB 5.3 it takes less than a minute to run the same query. The EXPLAIN shows that the subquery remains uncorrelated, which is an indication that it is being executed via subquery materialization.

```
+--+------------+-----------+------+------------------+----+------+------------------------------
|id|select_type |table      |type  |key               |ref |rows  |Extra
+--+------------+-----------+------+------------------+----+------+------------------------------
| 1|PRIMARY     |part       |ALL   |NULL              |NULL|199755|Using temporary; Using filesor
| 1|PRIMARY     |<subquery2>|eq_ref|distinct_key      |func|     1|
| 2|MATERIALIZED|lineitem   |range |l_shipdate_partkey|NULL|160060|Using where; Using index
+--+------------+-----------+------+------------------+----+------+------------------------------
```

The speedup here is practically infinite, because both MySQL and older MariaDB versions cannot complete the query in any reasonable time.

In order to show the benefits of partial matching we extended the *customer* table from the DBT3 benchmark with two extra columns:

- c_pref_nationkey - preferred nation to buy from,
- c_pref_brand - preferred brand.

Both columns are prefixed with the percent NULL values in the column, that is, c_pref_nationkey_05 contains 5% NULL

values.

Consider the query "Find all customers that didn't buy from a preferred country, and from a preferred brand withing some date ranges":

```
SELECT count(*)
FROM customer
WHERE (c_custkey, c_pref_nationkey_05, c_pref_brand_05) NOT IN
  (SELECT o_custkey, s_nationkey, p_brand
   FROM orders, supplier, part, lineitem
   WHERE l_orderkey = o_orderkey and
         l_suppkey = s_suppkey and
         l_partkey = p_partkey and
         p_retailprice < 1200 and
         l_shipdate >= '1996-04-01' and l_shipdate < '1996-04-05' and
         o_orderdate >= '1996-04-01' and o_orderdate < '1996-04-05');
```

- Execution time in MariaDB 5.2/MySQL 5.x (any MySQL): **40 sec**
- Execution time in MariaDB 5.3: **2 sec**

The speedup for this query is 20 times.

Performance guidelines

TODO

Optimizer control

In certain cases it may be necessary to override the choice of the optimizer. Typically this is needed for benchmarking or testing purposes, or to mimic the behavior of an older version of the server, or if the optimizer made a poor choice.

All the above strategies can be controlled via the following switches in optimizer_switch system variable.

- materialization=on/off
  In some very special cases, even if materialization was forced, the optimizer may still revert to the IN-TO-EXISTS strategy if materialization is not applicable. In the cases when materialization requres partial matching (because of the presense of NULL values), there are two subordinate switches that control the two partial matching strategies:
  - partial_match_rowid_merge=on/off
    This switch controls the Rowid-merge strategy. In addition to this switch, the system variable rowid_merge_buff_size controls the maximum memory available to the Rowid-merge strategy.
  - partial_match_table_scan=on/off
    Controls the alternative partial match strategy that performs matching via a table scan.
- in_to_exists=on/off
  This switch controls the IN-TO-EXISTS transformation.
- tmp_table_size and max_heap_table_size system variables
  The *tmp_table_size* system variable sets the upper limit for internal MEMORY temporary tables. If an internal temporary table exceeds this size, it is converted automatically into a Aria or MyISAM table on disk with a B-tree index. Notice however, that a MEMORY table cannot be larger than *max_heap_table_size*.

The two main optimizer switches - *materialization* and *in_to_exists* cannot be simultaneously off. If both are set to off, the server will issue an error.

3.3.4.2.5 Subquery Cache

Contents

The goal of the subquery cache is to optimize the evaluation of correlated subqueries by storing results together with correlation parameters in a cache and avoiding re-execution of the subquery in cases where the result is already in the cache.

Administration

The cache is on by default. One can switch it off using the optimizer_switch `subquery_cache` setting, like so:

```
SET optimizer_switch='subquery_cache=off';
```

The efficiency of the subquery cache is visible in 2 statistical variables:

- Subquery_cache_hit - Global counter for all subquery cache hits.
- Subquery_cache_miss - Global counter for all subquery cache misses.

The session variables tmp_table_size and max_heap_table_size influence the size of in-memory temporary tables in the table used for caching. It cannot grow more than the minimum of the above variables values (see the Implementation section for details).

Visibility

Your usage of the cache is visible in `EXTENDED EXPLAIN` output (warnings) as `"<expr_cache><//list of parameters//>(//cached expression//)"`. For example:

```
EXPLAIN EXTENDED SELECT * FROM t1 WHERE a IN (SELECT b FROM t2);
+----+--------------------+-------+------+---------------+------+---------+------+------+----
------+-------------+
| id | select_type        | table | type | possible_keys | key  | key_len | ref  | rows |
filtered | Extra       |
+----+--------------------+-------+------+---------------+------+---------+------+------+----
------+-------------+
|  1 | PRIMARY            | t1    | ALL  | NULL          | NULL | NULL    | NULL |    2 |
100.00 | Using where |
|  2 | DEPENDENT SUBQUERY | t2    | ALL  | NULL          | NULL | NULL    | NULL |    2 |
100.00 | Using where |
+----+--------------------+-------+------+---------------+------+---------+------+------+----
------+-------------+
2 rows in set, 1 warning (0.00 sec)

SHOW WARNINGS;
+-------+------+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
---------------------------------+
| Level | Code | Message
|
+-------+------+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
---------------------------------+
| Note  | 1003 | SELECT `test`.`t1`.`a` AS `a` from `test`.`t1` WHERE <expr_cache>
<`test`.`t1`.`a`>(<in_optimizer>(`test`.`t1`.`a`,<exists>(SELECT 1 FROM `test`.`t2` WHERE
(<cache>(`test`.`t1`.`a`) = `test`.`t2`.`b`))) |
+-------+------+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
---------------------------------+
1 row in set (0.00 sec)
```

In the example above the presence of `"<expr_cache><`test`.`t1`.`a`>(...)"` is how you know you are using the subquery cache.

Implementation

Every subquery cache creates a temporary table where the results and all parameters are stored. It has a unique index over all parameters. First the cache is created in a MEMORY table (if doing this is impossible the cache becomes disabled for that expression). When the table grows up to the minimum of `tmp_table_size` and `max_heap_table_size`, the hit rate will be checked:

- if the hit rate is really small (<0.2) the cache will be disabled.
- if the hit rate is moderate (<0.7) the table will be cleaned (all records deleted) to keep the table in memory
- if the hit rate is high the table will be converted to a disk table (for 5.3.0 it can only be converted to a disk table).

```
hit rate = hit / (hit + miss)
```

Performance Impact

Here are some examples that show the performance impact of the subquery cache (these tests were made on a 2.53 GHz Intel Core 2 Duo MacBook Pro with dbt-3 scale 1 data set).

example	cache on	cache off	gain	hit	miss	hit rate
1	1.01sec	1 hour 31 min 43.33sec	5445x	149975	25	99.98%
2	0.21sec	1.41sec	6.71x	6285	220	96.6%
3	2.54sec	2.55sec	1.00044x	151	461	24.67%
4	1.87sec	1.95sec	0.96x	0	23026	0%

Example 1

Dataset from DBT-3 benchmark, a query to find customers with balance near top in their nation:

```
select count(*) from customer
where
   c_acctbal > 0.8 * (select max(c_acctbal)
                      from customer C
                      where C.c_nationkey=customer.c_nationkey
                      group by c_nationkey);
```

Example 2

DBT-3 benchmark, Query #17

```
select sum(l_extendedprice) / 7.0 as avg_yearly
from lineitem, part
where
  p_partkey = l_partkey and
  p_brand = 'Brand#42' and p_container = 'JUMBO BAG' and
  l_quantity < (select 0.2 * avg(l_quantity) from lineitem
                where l_partkey = p_partkey);
```

Example 3

DBT-3 benchmark, Query #2

```
select
        s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address, s_phone, s_comment
from
        part, supplier, partsupp, nation, region
where
        p_partkey = ps_partkey and s_suppkey = ps_suppkey and p_size = 33
        and p_type like '%STEEL' and s_nationkey = n_nationkey
        and n_regionkey = r_regionkey and r_name = 'MIDDLE EAST'
        and ps_supplycost = (
                select
                        min(ps_supplycost)
                from
                        partsupp, supplier, nation, region
                where
                        p_partkey = ps_partkey and s_suppkey = ps_suppkey
                        and s_nationkey = n_nationkey and n_regionkey = r_regionkey
                        and r_name = 'MIDDLE EAST'
        )
order by
        s_acctbal desc, n_name, s_name, p_partkey;
```

Example 4

DBT-3 benchmark, Query #20

```
select
        s_name, s_address
from
        supplier, nation
where
        s_suppkey in (
                select
                        distinct (ps_suppkey)
                from
                        partsupp, part
                where
                        ps_partkey=p_partkey
                        and p_name like 'indian%'
                        and ps_availqty > (
                                select
                                        0.5 * sum(l_quantity)
                                from
                                        lineitem
                                where
                                        l_partkey = ps_partkey
                                        and l_suppkey = ps_suppkey
                                        and l_shipdate >= '1995-01-01'
                                        and l_shipdate < date_ADD('1995-01-01',interval 1 year)
                                )
        )
        and s_nationkey = n_nationkey and n_name = 'JAPAN'
order by
        s_name;
```

3.3.4.2.6 Condition Pushdown Into IN subqueries

This article describes Condition Pushdown into IN subqueries as implemented in MDEV-12387.

optimizer_switch flag name: `condition_pushdown_for_subquery`.

3.3.4.2.7 Conversion of Big IN Predicates Into Subqueries

Starting from MariaDB 10.3, the optimizer converts certain big IN predicates into IN subqueries.

That is, an IN predicate in the form

```
column [NOT] IN (const1, const2, .... )
```

is converted into an equivalent IN-subquery:

```
column [NOT] IN (select ... from temporary_table)
```

which opens new opportunities for the query optimizer.

The conversion happens if the following conditions are met:

- the IN list has more than 1000 elements (One can control it through the in_predicate_conversion_threshold parameter).
- the [NOT] IN condition is at the top level of the WHERE/ON clause.

Controlling the Optimization

- The optimization is on by default. MariaDB 10.3.18 (and debug builds prior to that) introduced the in_predicate_conversion_threshold variable. Set to `0` to disable the optimization.

Benefits of the Optimization

If `column` is a key-prefix, MariaDB optimizer will process the condition

```
column [NOT] IN (const1, const2, .... )
```

by trying to construct a range access. If the list is large, the analysis may take a lot of memory and CPU time. The problem gets worse when `column` is a part of a multi-column index and the query has conditions on other parts of the index.

Conversion of IN predicates into a subqueries bypass the range analysis, which means the query optimization phase will use less CPU and memory.

Possible disadvantages of the conversion are are:

- The optimization may convert 'IN LIST elements' key accesses to a table scan (if there is no other usable index for the table)
- The estimates for the number of rows matching the `IN (...)` are less precise.

3.3.4.2.8 EXISTS-to-IN Optimization

Contents

MySQL (including MySQL 5.6) has only one execution strategy for EXISTS subqueries. The strategy is essentially the straightforward, "naive" execution, without any rewrites.

MariaDB 5.3 introduced a rich set of optimizations for IN subqueries. Since then, it makes sense to convert an EXISTS subquery into an IN so that the new optimizations can be used.

`EXISTS` will be converted into `IN` in two cases:

1. Trivially correlated EXISTS subqueries
2. Semi-join EXISTS

We will now describe these two cases in detail

Trivially-correlated EXISTS subqueries

Often, EXISTS subquery is correlated, but the correlation is trivial. The subquery has form

```
EXISTS (SELECT ...  FROM ... WHERE outer_col= inner_col AND inner_where)
```

and "outer_col" is the only place where the subquery refers to outside fields. In this case, the subquery can be re-written into uncorrelated IN:

```
outer_col IN (SELECT inner_col FROM ... WHERE inner_where)
```

(`NULL` values require some special handling, see below). For uncorrelated IN subqueries, MariaDB is able a cost-based choice between two execution strategies:

- IN-to-EXISTS (basically, convert back into EXISTS)
- Materialization

That is, converting trivially-correlated `EXISTS` into uncorrelated `IN` gives query optimizer an option to use Materialization strategy for the subquery.

Currently, EXISTS->IN conversion works only for subqueries that are at top level of the WHERE clause, or are under NOT operation which is directly at top level of the WHERE clause.

Semi-join EXISTS subqueries

If `EXISTS` subquery is an AND-part of the `WHERE` clause:

```
SELECT ... FROM outer_tables WHERE EXISTS (SELECT ...) AND ...
```

then it satisfies the main property of semi-join subqueries:

with semi-join subquery, we're only interested in records of outer_tables that have matches in the subquery

Semi-join optimizer offers a rich set of execution strategies for both correlated and uncorrelated subqueries. The set includes FirstMatch strategy which is an equivalent of how EXISTS suqueries are executed, so we do not lose any opportunities when converting an EXISTS subquery into a semi-join.

In theory, it makes sense to convert all kinds of EXISTS subqueries: convert both correlated and uncorrelated ones, convert irrespectively of whether the subquery has inner=outer equality.

In practice, the subquery will be converted only if it has inner=outer equality. Both correlated and uncorrelated subqueries are converted.

Handling of NULL values

TODO: rephrase this:

- IN has complicated NULL-semantics. NOT EXISTS doesn't.
- EXISTS-to-IN adds IS NOT NULL before the subquery predicate, when required

Control

The optimization is controlled by the `exists_to_in` flag in optimizer_switch. Before MariaDB 10.0.12, the optimization was OFF by default. Since MariaDB 10.0.12, it has been ON by default.

Limitations

EXISTS-to-IN doesn't handle

- subqueries that have GROUP BY, aggregate functions, or HAVING clause
- subqueries are UNIONs
- a number of degenerate edge cases

3.3.4.2.9 Optimizing GROUP BY and DISTINCT Clauses in Subqueries

A DISTINCT clause and a GROUP BY without a corresponding HAVING clause have no meaning in IN/ALL/ANY/SOME/EXISTS subqueries. The reason is that IN/ALL/ANY/SOME/EXISTS only check if an outer row satisfies some condition with respect to all or any row in the subquery result. Therefore is doesn't matter if the subquery has duplicate result rows or not - if some condition is true for some row of the subquery, this condition will be true for all duplicates of this row. Notice that GROUP BY without a corresponding HAVING clause is equivalent to a DISTINCT.

MariaDB 5.3 and later versions automatically remove DISTINCT and GROUP BY without HAVING if these clauses appear in an IN/ALL/ANY/SOME/EXISTS subquery. For instance:

```
select * from t1
where t1.a > ALL(select distinct b from t2 where t2.c > 100)
```

is transformed to:

```
select * from t1
where t1.a > ALL(select b from t2 where t2.c > 100)
```

Removing these unnecessary clauses allows the optimizer to find more efficient query plans because it doesn't need to take care of post-processing the subquery result to satisfy DISTINCT / GROUP BY.

3.3.4.3 Optimization Strategies

Various optimization strategies used by the query optimizer.

DuplicateWeedout Strategy

DuplicateWeedout is an execution strategy for Semi-join subqueries.

FirstMatch Strategy
FirstMatch is an execution strategy for Semi-join subqueries

LooseScan Strategy
LooseScan is an execution strategy for Semi-join subqueries

Semi-join Materialization Strategy
Semi-join Materialization is a subquery materialization used for Semi-join subqueries.

Improvements to ORDER BY Optimization
Several Improvements to the ORDER BY Optimizer in Version 10.1 of MariaDB.

3.3.4.3.1 DuplicateWeedout Strategy

`DuplicateWeedout` is an execution strategy for Semi-join subqueries.

Contents

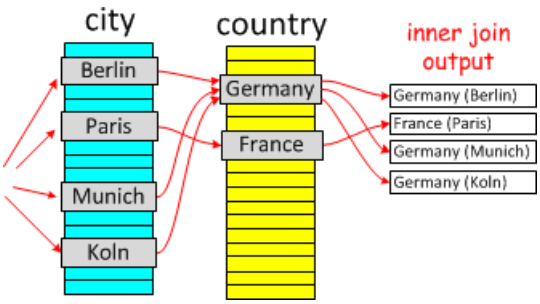
The idea

The idea is to run the semi-join (a query with uses `WHERE X IN (SELECT Y FROM ...)`) as if it were a regular inner join, and then eliminate the duplicate record combinations using a temporary table.

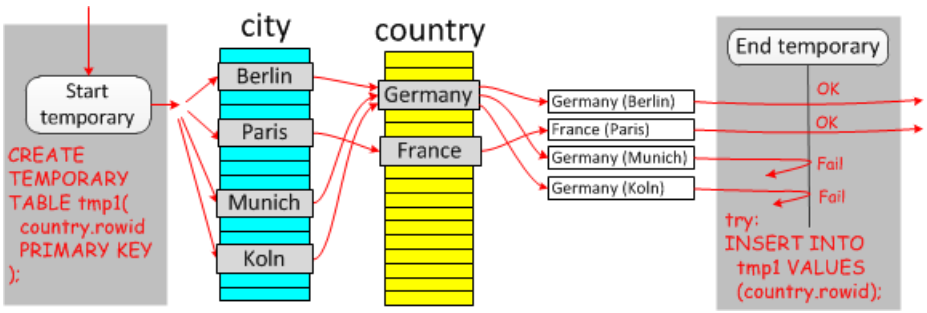
Suppose, you have a query where you're looking for countries which have more than 33% percent of their population in one big city:

```
select *
from Country
where
   Country.code IN (select City.Country
                    from City
                    where
                      City.Population > 0.33 * Country.Population and
                      City.Population > 1*1000*1000);
```

First, we run a regular inner join between the `City` and `Country` tables:



The Inner join produces duplicates. We have Germany three times, because it has three big cities. Now, lets put `DuplicateWeedout` into the picture:



Here one can see that a temporary table with a primary key was used to avoid producing multiple records with 'Germany'.

DuplicateWeedout in action

The `Start temporary` and `End temporary` from the last diagram are shown in the `EXPLAIN` output:

```
explain select * from Country where Country.code IN
  (select City.Country from City where City.Population > 0.33 * Country.Population
   and City.Population > 1*1000*1000)\G
*************************** 1. row ***************************
           id: 1
  select_type: PRIMARY
        table: City
         type: range
possible_keys: Population,Country
          key: Population
      key_len: 4
          ref: NULL
         rows: 238
        Extra: Using index condition; Start temporary
*************************** 2. row ***************************
           id: 1
  select_type: PRIMARY
        table: Country
         type: eq_ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 3
          ref: world.City.Country
         rows: 1
        Extra: Using where; End temporary
2 rows in set (0.00 sec)
```

This query will read 238 rows from the `City` table, and for each of them will make a primary key lookup in the `Country` table, which gives another 238 rows. This gives a total of 476 rows, and you need to add 238 lookups in the temporary table (which are typically *much* cheaper since the temporary table is in-memory).

If we run the same EXPLAIN in MySQL, we'll get:

```
explain select * from Country where Country.code IN
  (select City.Country from City where City.Population > 0.33 * Country.Population
    and City.Population > 1*1000*1000)\G
*************************** 1. row ***************************
           id: 1
  select_type: PRIMARY
        table: Country
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 239
        Extra: Using where
*************************** 2. row ***************************
           id: 2
  select_type: DEPENDENT SUBQUERY
        table: City
         type: index_subquery
possible_keys: Population,Country
          key: Country
      key_len: 3
          ref: func
         rows: 18
        Extra: Using where
2 rows in set (0.00 sec)
```

This plan will read `(239 + 239*18) = 4541` rows, which is much slower.

Factsheet

- `DuplicateWeedout` is shown as "Start temporary/End temporary" in `EXPLAIN`.
- The strategy can handle correlated subqueries.
- But it cannot be applied if the subquery has meaningful `GROUP BY` and/or aggregate functions.
- `DuplicateWeedout` allows the optimizer to freely mix a subquery's tables and the parent select's tables.

- There is no separate @@optimizer_switch flag for `DuplicateWeedout`. The strategy can be disabled by switching off all semi-join optimizations with `SET @@optimizer_switch='optimizer_semijoin=off'` command.

3.3.4.3.2 FirstMatch Strategy

Contents

`FirstMatch` is an execution strategy for Semi-join subqueries.

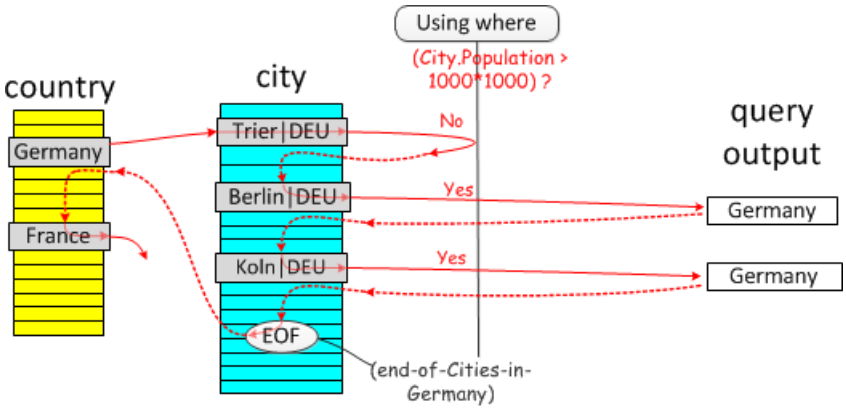
The idea

It is very similar to how `IN/EXISTS` subqueries were executed in MySQL 5.x.

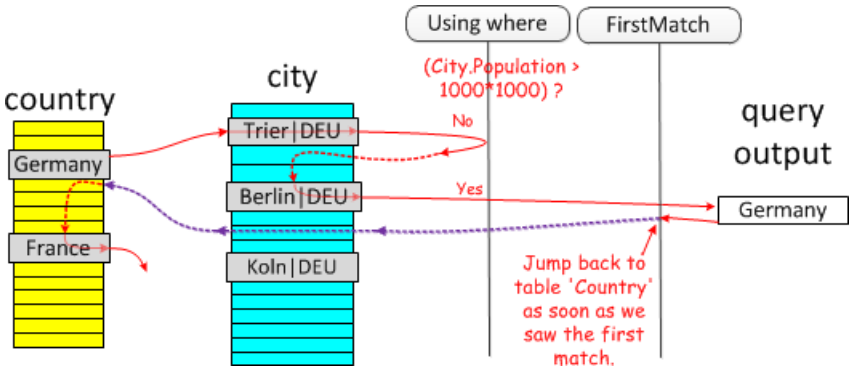
Let's take the usual example of a search for countries with big cities:

```
select * from Country
where Country.code IN (select City.Country
                       from City
                       where City.Population > 1*1000*1000)
      and Country.continent='Europe'
```

Suppose, our execution plan is to find countries in Europe, and then, for each found country, check if it has any big cities. Regular inner join execution will look as follows:



Since Germany has two big cities (in this diagram), it will be put into the query output twice. This is not correct, `SELECT ... FROM Country` should not produce the same country record twice. The `FirstMatch` strategy avoids the production of duplicates by short-cutting execution as soon as the first genuine match is found:



Note that the short-cutting has to take place after "Using where" has been applied. It would have been wrong to short-cut after we found Trier.

FirstMatch in action

The `EXPLAIN` for the above query will look as follows:

```
MariaDB [world]> explain select * from Country where Country.code IN
  (select City.Country from City where City.Population > 1*1000*1000)
   and Country.continent='Europe';
+----+-------------+---------+------+-------------------+-----------+---------+-------------
-------+------+-----------------------------------+
| id | select_type | table   | type | possible_keys     | key       | key_len | ref
| rows | Extra                             |
+----+-------------+---------+------+-------------------+-----------+---------+-------------
-------+------+-----------------------------------+
|  1 | PRIMARY     | Country | ref  | PRIMARY,continent | continent | 17      | const
|   60 | Using index condition            |
|  1 | PRIMARY     | City    | ref  | Population,Country | Country   | 3       |
world.Country.Code |   18 | Using where; FirstMatch(Country) |
+----+-------------+---------+------+-------------------+-----------+---------+-------------
-------+------+-----------------------------------+
2 rows in set (0.00 sec)
```

`FirstMatch(Country)` in the Extra column means that *as soon as we have produced one matching record combination, short-cut the execution and jump back to the Country* table.

`FirstMatch` 's query plan is very similar to one you would get in MySQL:

```
MySQL [world]> explain select * from Country  where Country.code IN
  (select City.Country from City where City.Population > 1*1000*1000)
   and Country.continent='Europe';
+----+--------------------+---------+----------------+--------------------+-----------+------
---+--------+------+-----------------------------------------+
| id | select_type        | table   | type           | possible_keys      | key       |
key_len | ref    | rows | Extra                                   |
+----+--------------------+---------+----------------+--------------------+-----------+------
---+--------+------+-----------------------------------------+
|  1 | PRIMARY            | Country | ref            | continent          | continent | 17
| const |   60 | Using index condition; Using where |
|  2 | DEPENDENT SUBQUERY | City    | index_subquery | Population,Country | Country   | 3
| func  |   18 | Using where                        |
+----+--------------------+---------+----------------+--------------------+-----------+------
---+--------+------+-----------------------------------------+
2 rows in set (0.01 sec)
```

and these two particular query plans will execute in the same time.

Difference between FirstMatch and IN->EXISTS

The general idea behind the `FirstMatch` strategy is the same as the one behind the `IN->EXISTS` transformation, however, `FirstMatch` has several advantages:

- Equality propagation works across semi-join bounds, but not subquery bounds. Therefore, converting a subquery to semi-join and using `FirstMatch` can still give a better execution plan. (TODO example)
- There is only one way to apply the `IN->EXISTS` strategy and MySQL will do it unconditionally. With `FirstMatch` , the optimizer can make a choice between whether it should run the `FirstMatch` strategy as soon as all tables used in the subquery are in the join prefix, or at some later point in time. (TODO: example)

FirstMatch factsheet

- The `FirstMatch` strategy works by executing the subquery and short-cutting its execution as soon as the first match is found.
- This means, subquery tables must be after all of the parent select's tables that are referred from the subquery predicate.
- `EXPLAIN` shows `FirstMatch` as " `FirstMatch(tableN)` ".
- The strategy can handle correlated subqueries.
- But it cannot be applied if the subquery has meaningful `GROUP BY` and/or aggregate functions.
- Use of the `FirstMatch` strategy is controlled with the `firstmatch=on|off` flag in the optimizer_switch variable.

3.3.4.3.3 LooseScan Strategy

LooseScan is an execution strategy for Semi-join subqueries.

Contents

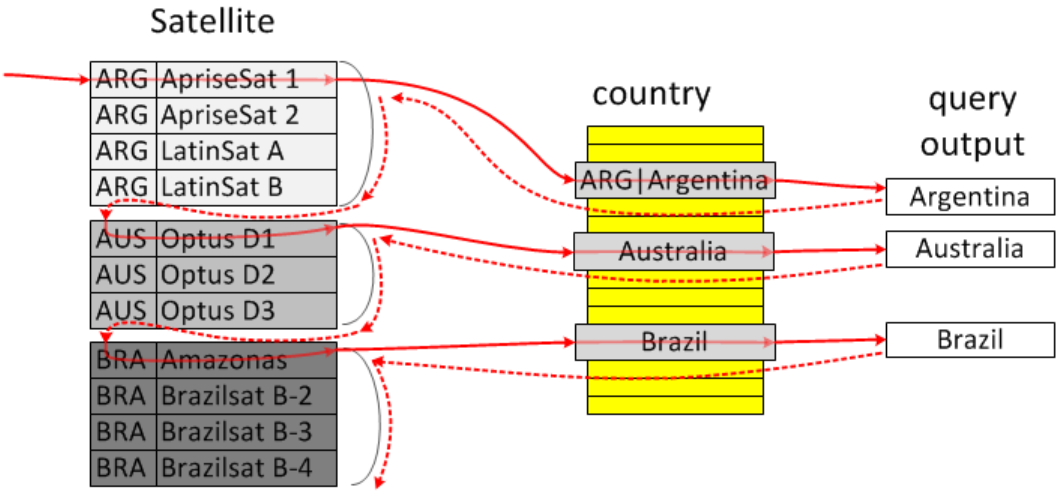
The idea

We will demonstrate the `LooseScan` strategy by example. Suppose, we're looking for countries that have satellites. We can get them using the following query (for the sake of simplicity we ignore satellites that are owned by consortiums of multiple countries):

```
select * from Country
where
  Country.code in (select country_code from Satellite)
```

Suppose, there is an index on `Satellite.country_code`. If we use that index, we will get satellites in the order of their owner country:

The `LooseScan` strategy doesn't really need ordering, what it needs is grouping. In the above figure, satellites are grouped by country. For instance, all satellites owned by Australia come together, without being mixed with satellites of other countries. This makes it easy to select just one satellite from each group, which you can join with its country and get a list of countries without duplicates:



LooseScan in action

The `EXPLAIN` output for the above query looks as follows:

```
MariaDB [world]> explain select * from Country where Country.code in
  (select country_code from Satellite);
+----+-------------+-----------+--------+---------------+-------------+---------+-----------
------------------+------+---------------------------------------+
| id | select_type | table     | type   | possible_keys | key         | key_len | ref
| rows | Extra                                 |
+----+-------------+-----------+--------+---------------+-------------+---------+-----------
------------------+------+---------------------------------------+
|  1 | PRIMARY     | Satellite | index  | country_code  | country_code | 9       | NULL
|  932 | Using where; Using index; LooseScan |
|  1 | PRIMARY     | Country   | eq_ref | PRIMARY       | PRIMARY      | 3       |
world.Satellite.country_code |    1 | Using index condition                 |
+----+-------------+-----------+--------+---------------+-------------+---------+-----------
------------------+------+---------------------------------------+
```

Factsheet

- LooseScan avoids the production of duplicate record combinations by putting the subquery table first and using its index to select one record from multiple duplicates
- Hence, in order for LooseScan to be applicable, the subquery should look like:

```
expr IN (SELECT tbl.keypart1 FROM tbl ...)
```

or

```
expr IN (SELECT tbl.keypart2 FROM tbl WHERE tbl.keypart1=const AND ...)
```

- LooseScan can handle correlated subqueries
- LooseScan can be switched off by setting the `loosescan=off` flag in the optimizer_switch variable.

3.3.4.3.4 Semi-join Materialization Strategy

Contents

Semi-join Materialization is a special kind of subquery materialization used for Semi-join subqueries. It actually includes two strategies:

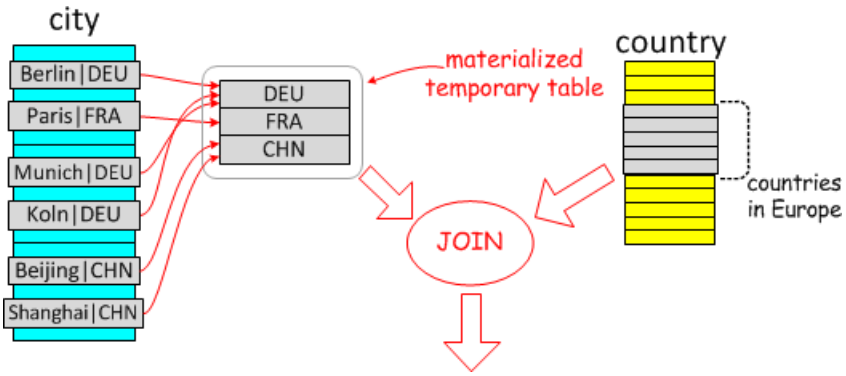
- Materialization/lookup
- Materialization/scan

The idea

Consider a query that finds countries in Europe which have big cities:

```
select * from Country
where Country.code IN (select City.Country
                       from City
                       where City.Population > 7*1000*1000)
      and Country.continent='Europe'
```

The subquery is uncorrelated, that is, we can run it independently of the upper query. The idea of semi-join materialization is to do just that, and fill a temporary table with possible values of the City.country field of big cities, and then do a join with countries in Europe:



The join can be done in two directions:

1. From the materialized table to countries in Europe
2. From countries in Europe to the materialized table

The first way involves doing a full scan on the materialized table, so we call it "Materialization-scan".

If you run a join from Countries to the materialized table, the cheapest way to find a match in the materialized table is to make a lookup on its primary key (it has one: we used it to remove duplicates). Because of that, we call the strategy "Materialization-lookup".

Semi-join materialization in action

Materialization-Scan

If we chose to look for cities with a population greater than 7 million, the optimizer will use Materialization-Scan and `EXPLAIN` will show this:

```
MariaDB [world]> explain select * from Country where Country.code IN
  (select City.Country from City where  City.Population > 7*1000*1000);
+----+--------------+-------------+--------+--------------------+------------+---------+-----
--------------+------+-----------------------+
| id | select_type  | table       | type   | possible_keys      | key        | key_len | ref
| rows | Extra                  |
+----+--------------+-------------+--------+--------------------+------------+---------+-----
--------------+------+-----------------------+
|  1 | PRIMARY      | <subquery2> | ALL    | distinct_key       | NULL       | NULL    | NULL
|   15 |                        |
|  1 | PRIMARY      | Country     | eq_ref | PRIMARY            | PRIMARY    | 3       |
world.City.Country |    1 |                        |
|  2 | MATERIALIZED | City        | range  | Population,Country | Population | 4       | NULL
|   15 | Using index condition |
+----+--------------+-------------+--------+--------------------+------------+---------+-----
--------------+------+-----------------------+
3 rows in set (0.01 sec)
```

Here, you can see:

- There are still two `SELECT` s (look for columns with `id=1` and `id=2`)
- The second select (with `id=2`) has `select_type=MATERIALIZED` . This means it will be executed and its results will be stored in a temporary table with a unique key over all columns. The unique key is there to prevent the table from containing any duplicate records.
- The first select received the table name `<subquery2>` . This is the table that we got as a result of the materialization of the select with `id=2` .

The optimizer chose to do a full scan over the materialized table, so this is an example of a use of the Materialization-Scan strategy.

As for execution costs, we're going to read 15 rows from table City, write 15 rows to materialized table, read them back (the optimizer assumes there won't be any duplicates), and then do 15 eq_ref accesses to table Country. In total, we'll do 45 reads and 15 writes.

By comparison, if you run the `EXPLAIN` in MySQL, you'll get this:

```
MySQL [world]> explain select * from Country where Country.code IN
  (select City.Country from City where  City.Population > 7*1000*1000);
+----+--------------------+---------+-------+--------------------+------------+---------+----
--+------+-----------------------------------+
| id | select_type        | table   | type  | possible_keys      | key        | key_len | ref
| rows | Extra                              |
+----+--------------------+---------+-------+--------------------+------------+---------+----
--+------+-----------------------------------+
|  1 | PRIMARY            | Country | ALL   | NULL               | NULL       | NULL    | NULL
|  239 | Using where                        |
|  2 | DEPENDENT SUBQUERY | City    | range | Population,Country | Population | 4       | NULL
|   15 | Using index condition; Using where |
+----+--------------------+---------+-------+--------------------+------------+---------+----
--+------+-----------------------------------+
```

...which is a plan to do `(239 + 239*15) = 3824` table reads.

Materialization-Lookup

Let's modify the query slightly and look for countries which have cities with a population over one millon (instead of seven):

```
MariaDB [world]> explain select * from Country where Country.code IN
  (select City.Country from City where  City.Population > 1*1000*1000) ;
+----+-------------+-------------+--------+--------------------+-------------+---------+---
---+------+-----------------------+
| id | select_type  | table       | type   | possible_keys      | key         | key_len | ref
| rows | Extra                 |
+----+-------------+-------------+--------+--------------------+-------------+---------+---
---+------+-----------------------+
|  1 | PRIMARY      | Country     | ALL    | PRIMARY            | NULL        | NULL    |
NULL |  239 |                       |
|  1 | PRIMARY      | <subquery2> | eq_ref | distinct_key       | distinct_key | 3       |
func |    1 |                       |
|  2 | MATERIALIZED | City        | range  | Population,Country | Population  | 4       |
NULL |  238 | Using index condition |
+----+-------------+-------------+--------+--------------------+-------------+---------+---
---+------+-----------------------+
3 rows in set (0.00 sec)
```

The `EXPLAIN` output is similar to the one which used Materialization-scan, except that:

- the `<subquery2>` table is accessed with the `eq_ref` access method
- the access uses an index named `distinct_key`

This means that the optimizer is planning to do index lookups into the materialized table. In other words, we're going to use the Materialization-lookup strategy.

In MySQL (or with `optimizer_switch='semijoin=off,materialization=off'`), one will get this `EXPLAIN` :

```
MySQL [world]> explain select * from Country where Country.code IN
  (select City.Country from City where  City.Population > 1*1000*1000) ;
+----+--------------------+---------+----------------+--------------------+---------+--------
-+------+------+-------------+
| id | select_type        | table   | type           | possible_keys      | key     | key_len
| ref  | rows | Extra       |
+----+--------------------+---------+----------------+--------------------+---------+--------
-+------+------+-------------+
|  1 | PRIMARY            | Country | ALL            | NULL               | NULL    | NULL
| NULL |  239 | Using where |
|  2 | DEPENDENT SUBQUERY | City    | index_subquery | Population,Country | Country | 3       |
func |   18 | Using where |
+----+--------------------+---------+----------------+--------------------+---------+--------
-+------+------+-------------+
```

One can see that both plans will do a full scan on the `Country` table. For the second step, MariaDB will fill the materialized table (238 rows read from table City and written to the temporary table) and then do a unique key lookup for each record in table `Country` , which works out to 238 unique key lookups. In total, the second step will cost `(239+238) = 477` reads and `238` temp.table writes.

MySQL's plan for the second step is to read 18 rows using an index on `City.Country` for each record it receives for table

`Country`. This works out to a cost of `(18*239) = 4302` reads. Had there been fewer subquery invocations, this plan would have been better than the one with Materialization. By the way, MariaDB has an option to use such a query plan, too (see FirstMatch Strategy), but it did not choose it.

Subqueries with grouping

MariaDB is able to use Semi-join materialization strategy when the subquery has grouping (other semi-join strategies are not applicable in this case).

This allows for efficient execution of queries that search for the best/last element in a certain group.

For example, let's find cities that have the biggest population on their continent:

```
explain
select * from City
where City.Population in (select max(City.Population) from City, Country
                          where City.Country=Country.Code
                          group by Continent)
+------+--------------+-------------+------+---------------+------------+---------+----------
------------------------+------+-----------------+
| id   | select_type  | table       | type | possible_keys | key        | key_len | ref
| rows | Extra           |
+------+--------------+-------------+------+---------------+------------+---------+----------
------------------------+------+-----------------+
|    1 | PRIMARY      | <subquery2> | ALL  | distinct_key  | NULL       | NULL    | NULL
|  239 |                 |
|    1 | PRIMARY      | City        | ref  | Population    | Population | 4       |
<subquery2>.max(City.Population) |    1 |                |
|    2 | MATERIALIZED | Country     | ALL  | PRIMARY       | NULL       | NULL    | NULL
|  239 | Using temporary |
|    2 | MATERIALIZED | City        | ref  | Country       | Country    | 3       |
world.Country.Code               |   18 |                |
+------+--------------+-------------+------+---------------+------------+---------+----------
------------------------+------+-----------------+
4 rows in set (0.00 sec)
```

the cities are:

```
+------+-------------------+---------+------------+
| ID   | Name              | Country | Population |
+------+-------------------+---------+------------+
| 1024 | Mumbai (Bombay)   | IND     |   10500000 |
| 3580 | Moscow            | RUS     |    8389200 |
| 2454 | Macao             | MAC     |     437500 |
|  608 | Cairo             | EGY     |    6789479 |
| 2515 | Ciudad de México | MEX     |    8591309 |
|  206 | São Paulo        | BRA     |    9968485 |
|  130 | Sydney            | AUS     |    3276207 |
+------+-------------------+---------+------------+
```

Factsheet

Semi-join materialization

- Can be used for uncorrelated IN-subqueries. The subselect may use grouping and/or aggregate functions.
- Is shown in `EXPLAIN` as `type=MATERIALIZED` for the subquery, and a line with `table=<subqueryN>` in the parent subquery.
- Is enabled when one has both `materialization=on` and `semijoin=on` in the optimizer_switch variable.
- The `materialization=on|off` flag is shared with Non-semijoin materialization.

3.3.4.3.5 Improvements to ORDER BY Optimization

MariaDB starting with 10.1

MariaDB 10.1 includes several improvements to the ORDER BY optimizer.

The fixes were made as a response to complaints by MariaDB customers, so they fix real-world optimization problems. The fixes are a bit hard to describe (as the `ORDER BY` optimizer is complicated), but here's a short description:

The ORDER BY optimizer in MariaDB 10.1:

- Doesn't make stupid choices when several multi-part keys and potential range accesses are present (MDEV-6402).
  - This also fixes MySQL Bug#12113.
- Always uses "range" and (not full "index" scan) when it switches to an index to satisfy `ORDER BY … LIMIT` (MDEV-6657).
- Tries hard to be smart and use cost/number of records estimates from other parts of the optimizer (MDEV-6384, MDEV-465).
  - This change also fixes MySQL Bug#36817.
- Takes full advantage of InnoDB's Extended Keys feature when checking if filesort() can be skipped (MDEV-6796).

Extra optimizations

Starting from MariaDB 10.1.15

- The ORDER BY optimizer takes multiple-equalities into account (MDEV-8989). This optimization is not enabled by default in MariaDB 10.1. You need to explicitly switch it ON by setting the optimizer_switch system variable, as follows:

```
optimizer_switch='orderby_uses_equalities=on'
```

Setting the switch ON is considered safe. It is off by default in MariaDB 10.1 in order to avoid changing query plans in a stable release. It is on by default from MariaDB 10.2

Comparison with MySQL 5.7

In MySQL 5.7 changelog, one can find this passage:

> Make switching of index due to small limit cost-based (WL#6986) : We have made the decision in make_join_select() of whether to switch to a new index in order to support "ORDER BY ... LIMIT N" cost-based. This work fixes Bug#73837.

MariaDB is not using Oracle's fix (we believe `make_join_select` is not the right place to do ORDER BY optimization), but the effect is the same: this case is covered by MariaDB 10.1's optimizer.

3.3.4.4 Optimizations for Derived Tables

Derived tables are subqueries in the `FROM` clause. Prior to MariaDB 5.3/MySQL 5.6, they were too slow to be usable. In MariaDB 5.3/MySQL 5.6, there are two optimizations which provide adequate performance:

Condition Pushdown into Derived Table Optimization

If a query uses a derived table (or a view), the first action that the que...

Derived Table Merge Optimization

MariaDB 5.3 introduced the derived table merge optimization.

Derived Table with Key Optimization

Since MariaDB 5.3, the optimizer can create an index and use it for joins with other tables.

Lateral Derived Optimization

Lateral Derived optimization, also referred to as "Split Grouping Optimization".

3.3.4.4.1 Condition Pushdown into Derived Table Optimization

Contents

If a query uses a derived table (or a view), the first action that the query optimizer will attempt is to apply the derived-table-merge-optimization and merge the derived table into its parent select. However, that optimization is only applicable when

the select inside the derived table has a join as the top-level operation. If it has a GROUP-BY, DISTINCT, or uses window functions, then derived-table-merge-optimization is not applicable.

In that case, the Condition Pushdown optimization is applicable.

Introduction to Condition Pushdown

Consider an example

```
create view OCT_TOTALS as
select
  customer_id,
  SUM(amount) as TOTAL_AMT
from orders
where  order_date BETWEEN '2017-10-01' and '2017-10-31'
group by customer_id;

select * from OCT_TOTALS where customer_id=1
```

The naive way to execute the above is to

1. Compute the OCT_TOTALS contents (for all customers).
2. The, select the line with customer_id=1

This is obviously inefficient, if there are 1000 customers, then one will be doing up to 1000 times more work than necessary.

However, the optimizer can take the condition `customer_id=1` and push it down into the OCT_TOTALS view.

(TODO: elaborate here)

Condition Pushdown Properties

- Condition Pushdown has been available since MariaDB 10.2.
- The Jira task for it was MDEV-9197.
- The optimization is enabled by default. One can disable it by setting `@@optimizer_switch` flag `condition_pushdown_for_derived` to OFF.

3.3.4.4.2 Derived Table Merge Optimization

Contents

Background

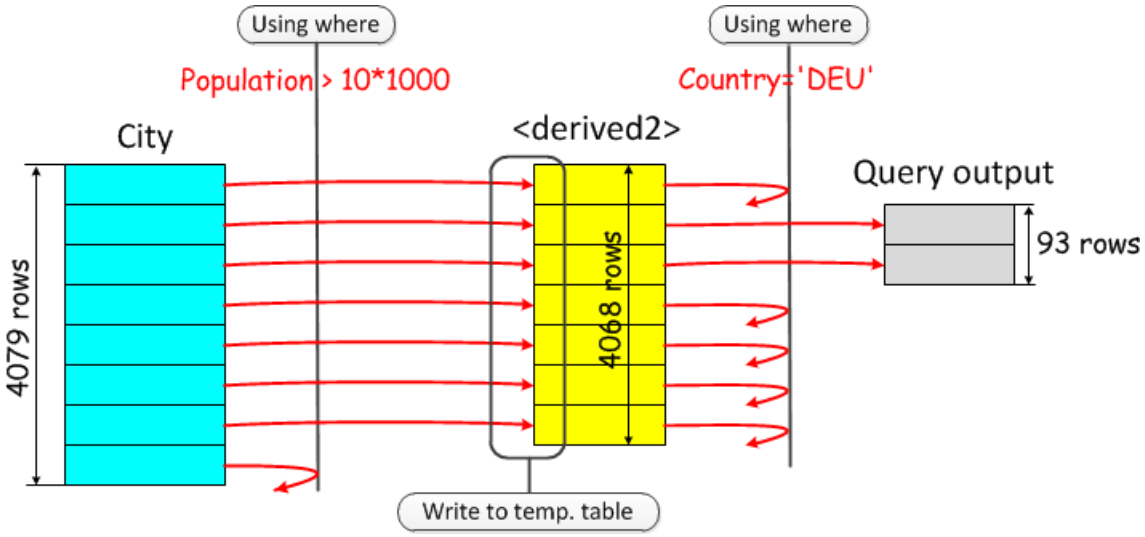
Users of "big" database systems are used to using `FROM` subqueries as a way to structure their queries. For example, if one's first thought was to select cities with population greater than 10,000 people, and then that from these cities to select those that are located in Germany, one could write this SQL:

```
SELECT *
FROM
  (SELECT * FROM City WHERE Population > 10*1000) AS big_city
WHERE
  big_city.Country='DEU'
```

For MySQL, using such syntax was taboo. If you run EXPLAIN for this query, you can see why:

```
mysql> EXPLAIN SELECT * FROM (SELECT * FROM City WHERE Population > 1*1000)
  AS big_city WHERE big_city.Country='DEU' ;
+----+-------------+------------+------+---------------+------+---------+------+------+------
-------+
| id | select_type | table      | type | possible_keys | key  | key_len | ref  | rows | Extra
|
+----+-------------+------------+------+---------------+------+---------+------+------+------
-------+
|  1 | PRIMARY     | <derived2> | ALL  | NULL          | NULL | NULL    | NULL | 4068 | Using
where |
|  2 | DERIVED     | City       | ALL  | Population    | NULL | NULL    | NULL | 4079 | Using
where |
+----+-------------+------------+------+---------------+------+---------+------+------+------
-------+
2 rows in set (0.60 sec)
```

It plans to do the following actions:



From left to right:

1. Execute the subquery: `(SELECT * FROM City WHERE Population > 1*1000)` , exactly as it was written in the query.
2. Put result of the subquery into a temporary table.
3. Read back, and apply a `WHERE` condition from the upper select, `big_city.Country='DEU'`

Executing a subquery like this is very inefficient, because the highly-selective condition from the parent select, (`Country='DEU'`) is not used when scanning the base table `City` . We read too many records from the `City` table, and then we have to write them into a temporary table and read them back again, before finally filtering them out.

Derived table merge in action

If one runs this query in MariaDB/MySQL 5.6, they get this:

```
MariaDB [world]> EXPLAIN SELECT * FROM (SELECT * FROM City WHERE Population > 1*1000)
  AS big_city WHERE big_city.Country='DEU';
+----+-------------+------+------+-----------------+---------+---------+-------+------+--
-----------------------------------+
| id | select_type | table | type | possible_keys   | key     | key_len | ref   | rows |
Extra                              |
+----+-------------+------+------+-----------------+---------+---------+-------+------+--
-----------------------------------+
|  1 | SIMPLE      | City  | ref  | Population,Country | Country | 3       | const |   90 |
Using index condition; Using where |
+----+-------------+------+------+-----------------+---------+---------+-------+------+--
-----------------------------------+
1 row in set (0.00 sec)
```

From the above, one can see that:

1. The output has only one line. This means that the subquery has been merged into the top-level `SELECT` .
2. Table `City` is accessed through an index on the `Country` column. Apparently, the `Country='DEU'` condition was used to construct `ref` access on the table.

3. The query will read about 90 rows, which is a big improvement over the 4079 row reads plus 4068 temporary table reads/writes we had before.

Factsheet

- Derived tables (subqueries in the `FROM` clause) can be merged into their parent select when they have no grouping, aggregates, or `ORDER BY ... LIMIT` clauses. These requirements are the same as requirements for `VIEW` s to allow `algorithm=merge` .
- The optimization is enabled by default. It can be disabled with:

```
set @@optimizer_switch='derived_merge=OFF'
```

- Versions of MySQL and MariaDB which do not have support for this optimization will execute subqueries even when running `EXPLAIN` . This can result in a well-known problem (see e.g. MySQL Bug #44802) of `EXPLAIN` statements taking a very long time. Starting from MariaDB 5.3+ and MySQL 5.6+ `EXPLAIN` commands execute instantly, regardless of the `derived_merge` setting.

3.3.4.4.3 Derived Table with Key Optimization

Contents

The idea

If a derived table cannot be merged into its parent SELECT, it will be materialized in a temporary table, and then parent select will treat it as a regular base table.

Before MariaDB 5.3/MySQL 5.6, the temporary table would never have any indexes, and the only way to read records from it would be a full table scan. Starting from the mentioned versions of the server, the optimizer has an option to create an index and use it for joins with other tables.

Example

Consider a query: we want to find countries in Europe, that have more than one million people living in cities. This is accomplished with this query:

```
select *
from
   Country,
   (select
       sum(City.Population) as urban_population,
       City.Country
    from City
    group by City.Country
    having
    urban_population > 1*1000*1000
   ) as cities_in_country
where
  Country.Code=cities_in_country.Country and Country.Continent='Europe';
```

The EXPLAIN output for it will show:

```
+----+-------------+------------+------+-----------------+-----------+---------+-----------
---------+------+-----------------------------------+
| id | select_type | table      | type | possible_keys     | key       | key_len | ref
| rows | Extra                             |
+----+-------------+------------+------+-----------------+-----------+---------+-----------
---------+------+-----------------------------------+
|  1 | PRIMARY     | Country    | ref  | PRIMARY,continent | continent | 17      | const
|   60 | Using index condition           |
|  1 | PRIMARY     | <derived2> | ref  | key0              | key0      | 3       |
world.Country.Code |   17 |                                  |
|  2 | DERIVED     | City       | ALL  | NULL              | NULL      | NULL    | NULL
| 4079 | Using temporary; Using filesort |
+----+-------------+------------+------+-----------------+-----------+---------+-----------
---------+------+-----------------------------------+
```

One can see here that

- table `<derived2>` is accessed through `key0`.
- `ref` column shows `world.Country.Code`
- if we look that up in the original query, we find the equality that was used to construct `ref` access: `Country.Code=cities_in_country.Country`.

Factsheet

- The idea of "derived table with key" optimization is to let the materialized derived table have one key which is used for joins with other tables.
- The optimization is applied then the derived table could not be merged into its parent SELECT
  - which happens when the derived table doesn't meet criteria for mergeable VIEW
- The optimization is ON by default, it can be switched off like so:

```
set optimizer_switch='derived_with_keys=off'
```

3.3.4.4.4 Lateral Derived Optimization

Contents

MariaDB supports the Lateral Derived optimization, also referred to as "Split Grouping Optimization" or "Split Materialized Optimization" in some sources.

Description

The optimization's use case is

- The query uses a derived table (or a VIEW, or a non-recursive CTE)
- The derived table/View/CTE has a GROUP BY operation as its top-level operation
- The query only needs data from a few GROUP BY groups

An example of this: consider a VIEW that computes totals for each customer in October:

```
create view OCT_TOTALS as
select
  customer_id,
  SUM(amount) as TOTAL_AMT
from orders
where
  order_date BETWEEN '2017-10-01' and '2017-10-31'
group by
  customer_id;
```

And a query that does a join with the customer table to get October totals for "Customer#1" and Customer#2:

```
select *
from
  customer, OCT_TOTALS
where
  customer.customer_id=OCT_TOTALS.customer_id and
  customer.customer_name IN ('Customer#1', 'Customer#2')
```

Before Lateral Derived optimization, MariaDB would execute the query as follows:

1. Materialize the view OCT_TOTALS. This essentially computes OCT_TOTALS for all customers.
2. Join it with table customer.

The EXPLAIN would look like so:

```
+------+-------------+------------+-------+---------------+-----------+---------+------------
--------------+-------+--------------------------+
| id   | select_type | table      | type  | possible_keys | key       | key_len | ref
| rows  | Extra                     |
+------+-------------+------------+-------+---------------+-----------+---------+------------
--------------+-------+--------------------------+
|    1 | PRIMARY     | customer   | range | PRIMARY,name  | name      | 103     | NULL
| 2     | Using where; Using index |
|    1 | PRIMARY     | <derived2> | ref   | key0          | key0      | 4       |
test.customer.customer_id | 36    |                           |
|    2 | DERIVED     | orders     | index | NULL          | o_cust_id | 4       | NULL
| 36738 | Using where              |
+------+-------------+------------+-------+---------------+-----------+---------+------------
--------------+-------+--------------------------+
```

It is obvious that Step #1 is very inefficient: we compute totals for all customers in the database, while we will only need them for two customers. (If there are 1000 customers, we are doing 500x more work than needed here)

Lateral Derived optimization addresses this case. It turns the computation of OCT_TOTALS into what SQL Standard refers to as "LATERAL subquery": a subquery that may have dependencies on the outside tables. This allows pushing the equality `customer.customer_id=OCT_TOTALS.customer_id` down into the derived table/view, where it can be used to limit the computation to compute totals only for the customer of interest.

The query plan will look as follows:

1. Scan table `customer` and find `customer_id` for Customer#1 and Customer#2.
2. For each customer_id, compute the October totals, for this specific customer.

The EXPLAIN output will look like so:

```
+------+-----------------+------------+-------+---------------+-----------+---------+--------
------------------+------+--------------------------+
| id   | select_type     | table      | type  | possible_keys | key       | key_len | ref
| rows | Extra                     |
+------+-----------------+------------+-------+---------------+-----------+---------+--------
------------------+------+--------------------------+
|    1 | PRIMARY         | customer   | range | PRIMARY,name  | name      | 103     | NULL
| 2    | Using where; Using index |
|    1 | PRIMARY         | <derived2> | ref   | key0          | key0      | 4       |
test.customer.customer_id | 2    |                           |
|    2 | LATERAL DERIVED | orders     | ref   | o_cust_id     | o_cust_id | 4       |
test.customer.customer_id | 1    | Using where               |
+------+-----------------+------------+-------+---------------+-----------+---------+--------
------------------+------+--------------------------+
```

Note the line with `id=2` : select_type is `LATERAL DERIVED` . And table customer uses ref access referring to `customer.customer_id` , which is normally not allowed for derived tables.

In `EXPLAIN FORMAT=JSON` output, the optimization is shown like so:

```
...
        "table": {
          "table_name": "<derived2>",
          "access_type": "ref",
...
          "materialized": {
            "lateral": 1,
```

Note the `"lateral": 1` member.

Controlling the Optimization

Lateral Derived is enabled by default, the optimizer will make a cost-based decision whether the optimization should be used.

If you need to disable the optimization, it has an optimizer_switch flag. It can be disabled like so:

```
set optimizer_switch='split_materialized=off'
```

References

- Jira task: https://jira.mariadb.org/browse/MDEV-13369
- Commit: https://github.com/MariaDB/server/commit/b14e2b044b

3.3.4.5 Table Elimination

Articles about Table Elimination, the idea that it is sometimes possible to resolve a query without accessing some of the tables the query refers to.

What is Table Elimination?
Resolving a query without accessing some of the tables that the query refers to.

Table Elimination in MariaDB
Table elimination in the MariaDB optimizer.

Table Elimination User Interface
Table Elimination User Interface and EXPLAIN.

Table Elimination in Other Databases
Table Elimination in SQL Server and Oracle.

Table Elimination External Resources
an example of how to do this in MariaDB

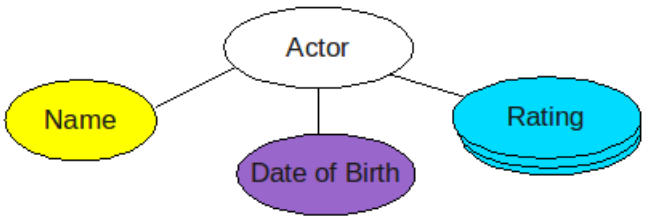
There are 2 related questions.

3.3.4.5.1 What is Table Elimination?

The basic idea behind table elimination is that sometimes it is possible to resolve a query without even accessing some of the tables that the query refers to. One can invent many kinds of such cases, but in Table Elimination we targeted only a certain class of SQL constructs that one ends up writing when they are querying highly-normalized data.

The sample queries were drawn from “Anchor Modeling”, a database modeling technique which takes normalization to the extreme. The slides at the anchor modeling website have an in-depth explanation of Anchor modeling and its merits, but the part that's important for table elimination can be shown with an example.

Suppose the database stores information about actors, together with their names, birthdays, and ratings, where ratings can change over time:



According to anchor modeling, each attribute should go into its own table:

- the 'anchor' table which only has a synthetic primary key:

```
create table  ac_anchor(AC_ID int primary key);
```

- a table for the 'name' attribute:

```
create table ac_name(AC_ID int, ACNAM_name char(N),
                     primary key(AC_ID));
```

- a table for the 'birthdate' attribute:

```
create table ac_dob(AC_ID int,
                    ACDOB_birthdate date,
                    primary key(AC_ID));
```

- a table for the ‘rating’ attribute, which is historized:

```
create table ac_rating(AC_ID int,
                       ACRAT_rating int,
                       ACRAT_fromdate date,
                       primary key(AC_ID, ACRAT_fromdate));
```

With this approach it becomes easy to add/change/remove attributes, but this comes at a cost of added complexity in querying the data: in order to answer the simplest, select-star question of displaying actors and their current ratings one has to write outer joins:

Display actors, with their names and current ratings:

```
select
  ac_anchor.AC_ID, ACNAM_Name, ACDOB_birthdate, ACRAT_rating
from
  ac_anchor
  left join ac_name on ac_anchor.AC_ID=ac_name.AC_ID
  left join ac_dob on ac_anchor.AC_ID=ac_dob.AC_ID
  left join ac_rating on (ac_anchor.AC_ID=ac_rating.AC_ID and
                          ac_rating.ACRAT_fromdate =
                            (select max(sub.ACRAT_fromdate)
                             from ac_rating sub where sub.AC_ID = ac_rating.AC_ID))
```

We don't want to write the joins every time we need to access an actor's properties, so we’ll create a view:

```
create view actors as
  select  ac_anchor.AC_ID, ACNAM_Name,  ACDOB_birthdate, ACRAT_rating
  from <see the select above>
```

This will allow us to access the data as if it was stored in a regular way:

```
select ACRAT_rating from actors where ACNAM_name='Gary Oldman'
```

And this is where table elimination will be needed.

3.3.4.5.2 Table Elimination in MariaDB

The first thing the MariaDB optimizer does is to merge the `VIEW` definition into the query to obtain:

```
select ACRAT_rating
from
  ac_anchor
  left join ac_name on ac_anchor.AC_ID=ac_name.AC_ID
  left join ac_dob on ac_anchor.AC_ID=ac_dob.AC_ID
  left join ac_rating on (ac_anchor.AC_ID=ac_rating.AC_ID and
                          ac_rating.ACRAT_fromdate =
                            (select max(sub.ACRAT_fromdate)
                             from ac_rating sub where sub.AC_ID = ac_rating.AC_ID))
where
 ACNAM_name='Gary Oldman'
```

It's important to realize that the obtained query has a useless part:

- `left join ac_dob on ac_dob.AC_ID=...` will produce exactly one matching record:
  - `primary key(ac_dob.AC_ID)` guarantees that there will be at most one match for any value of `ac_anchor.AC_ID`,
  - and if there won't be a match, `LEFT JOIN` will generate a NULL-complemented "row"
- and we don't care what the matching record is, as table `ac_dob` is not used anywhere else in the query.

This means that the `left join ac_dob on ...` part can be removed from the query and this is what Table Elimination module does. The detection logic is rather smart, for example it would be able to remove the `left join ac_rating on ...` part as well, together with the subquery (in the above example it won't be removed because ac_rating used in the selection list of the query). The Table Elimination module can also handle nested outer joins and multi-table outer joins.

3.3.4.5.3 Table Elimination User Interface

One can check that table elimination is working by looking at the output of `EXPLAIN [EXTENDED]` and not finding there the tables that were eliminated:

```
explain select ACRAT_rating from actors where ACNAM_name='Gary Oldman';
+----+--------------------+-----------+--------+---------------+---------+---------+---------
------------+------+-------------+
| id | select_type        | table     | type   | possible_keys | key     | key_len | ref
| rows | Extra       |
+----+--------------------+-----------+--------+---------------+---------+---------+---------
------------+------+-------------+
|  1 | PRIMARY            | ac_anchor | index  | PRIMARY       | PRIMARY | 4       | NULL
|    2 | Using index |
|  1 | PRIMARY            | ac_name   | eq_ref | PRIMARY       | PRIMARY | 4       |
test.ac_anchor.AC_ID |    1 | Using where |
|  1 | PRIMARY            | ac_rating | ref    | PRIMARY       | PRIMARY | 4       |
test.ac_anchor.AC_ID |    1 |             |
|  3 | DEPENDENT SUBQUERY | sub       | ref    | PRIMARY       | PRIMARY | 4       |
test.ac_rating.AC_ID |    1 | Using index |
+----+--------------------+-----------+--------+---------------+---------+---------+---------
------------+------+-------------+
```

Note that `ac_dob` table is not in the output. Now let's try getting birthdate instead:

```
explain select ACDOB_birthdate from actors where ACNAM_name='Gary Oldman';
+----+-------------+-----------+--------+---------------+---------+---------+----------------
------+------+-------------+
| id | select_type | table     | type   | possible_keys | key     | key_len | ref
| rows | Extra       |
+----+-------------+-----------+--------+---------------+---------+---------+----------------
------+------+-------------+
|  1 | PRIMARY     | ac_anchor | index  | PRIMARY       | PRIMARY | 4       | NULL
|    2 | Using index |
|  1 | PRIMARY     | ac_name   | eq_ref | PRIMARY       | PRIMARY | 4       |
test.ac_anchor.AC_ID |    1 | Using where |
|  1 | PRIMARY     | ac_dob    | eq_ref | PRIMARY       | PRIMARY | 4       |
test.ac_anchor.AC_ID |    1 |             |
+----+-------------+-----------+--------+---------------+---------+---------+----------------
------+------+-------------+
3 rows in set (0.01 sec)
```

The `ac_dob` table is there while `ac_rating` and the subquery are gone. Now, if we just want to check the name of the actor:

```
explain select count(*) from actors where ACNAM_name='Gary Oldman';
+----+-------------+-----------+--------+---------------+---------+---------+---------------
------+------+-------------+
| id | select_type | table     | type   | possible_keys | key     | key_len | ref
| rows | Extra       |
+----+-------------+-----------+--------+---------------+---------+---------+---------------
------+------+-------------+
|  1 | PRIMARY     | ac_anchor | index  | PRIMARY       | PRIMARY | 4       | NULL
|    2 | Using index |
|  1 | PRIMARY     | ac_name   | eq_ref | PRIMARY       | PRIMARY | 4       |
test.ac_anchor.AC_ID |    1 | Using where |
+----+-------------+-----------+--------+---------------+---------+---------+---------------
------+------+-------------+
2 rows in set (0.01 sec)
```

In this case it will eliminate both the `ac_dob` and `ac_rating` tables.

Removing tables from a query does not make the query slower, and it does not cut off any optimization opportunities, so table elimination is unconditional and there are no plans on having any kind of query hints for it.

For debugging purposes there is a `table_elimination=on|off` switch in debug builds of the server.

3.3.4.5.4 Table Elimination in Other Databases

In addition to MariaDB, Table Elimination is found in both Microsoft SQL Server 2005/2008 and Oracle 11g. Of the two, Microsoft SQL Server 2005/2008 seems to have the most advanced implementation. Oracle 11g has been confirmed to use table elimination but not to the same extent.

To compare the two, we will look at the following query:

```
select
 A.colA
from
 tableA A
left outer join
 tableB B
on
 B.id = A.id;
```

When using A as the left table we ensure that the query will return at least as many rows as there are in that table. For rows where the join condition (B.id = A.id) is not met the selected column (A.colA) will still contain its original value. The not seen B.* row would contain all NULL:s.

However, the result set could actually contain more rows than what is found in tableA if there are duplicates of the column B.id in tableB. If A contains a row [1, "val1"] and B the rows [1, "other1a"],[1, "other1b"] then two rows will match in the join condition. The only way to know what the result will look like is to actually touch both tables during execution.

Instead, let's say tableB contains rows that make it possible to place a unique constraint on the column B.id, for example, which is often the case with a primary key. In this situation we know that we will get exactly as many rows as there are in tableA, since joining with tableB cannot introduce any duplicates. Furthermore, as in the example query, if we do not select any columns from tableB, touching that table during execution is unnecessary. We can remove the whole join operation from the execution plan.

Both SQL Server 2005/2008 and Oracle 11g deploy table elimination in the case described above. Let us look at a more advanced query, where Oracle fails.

```
select
 A.colA
from
 tableA A
left outer join
 tableB B
on
 B.id = A.id
and
 B.fromDate = (
   select
     max(sub.fromDate)
   from
     tableB sub
   where
     sub.id = A.id
 );
```

In this example we have added another join condition, which ensures that we only pick the matching row from tableB having the latest fromDate. In this case tableB will contain duplicates of the column B.id, so in order to ensure uniqueness the primary key has to contain the fromDate column as well. In other words the primary key of tableB is (B.id, B.fromDate).

Furthermore, since the subselect ensures that we only pick the latest B.fromDate for a given B.id we know that at most one row will match the join condition. We will again have the situation where joining with tableB cannot affect the number of rows in the result set. Since we do not select any columns from tableB, the whole join operation can be eliminated from the execution plan.

SQL Server 2005/2008 will deploy table elimination in this situation as well. We have not found a way to make Oracle 11g use it for this type of query. Queries like these arise in two situations. Either when you have a denormalized model consisting of a fact table with several related dimension tables, or when you have a highly normalized model where each attribute is stored in its own table. The example with the subselect is common whenever you store historized/versioned data.

3.3.4.5.5 Table Elimination External Resources

- an example of how to do this in MariaDB

3.3.4.6 Statistics for Optimizing Queries

Different statistics provided by MariaDB to help you optimize your queries

Engine-Independent Table Statistics

Table statistics independent of the storage engine.

Histogram-Based Statistics

Histogram-based statistics can improve the optimizer query plan in certain situations.

Index Statistics

Index statistics and the query optimizer.

InnoDB Persistent Statistics

InnoDB persistent statistics are stored on disk, leading to more consistent query plans.

Slow Query Log Extended Statistics

The slow query log makes extended statistics available.

User Statistics

User Statistics.

3.3.4.6.1 Engine-Independent Table Statistics

MariaDB starting with 10.4

The engine-independent table statistics feature was first implemented in MariaDB 10.0.1 and was first enabled for queries by default in MariaDB 10.4.

Contents

Introduction

Before MariaDB 10.0, the MySQL/MariaDB optimizer relied on storage engines (e.g. InnoDB) to provide statistics for the query optimizer. This approach worked; however it had some deficiencies:

- Storage engines provided poor statistics (this was fixed to some degree with the introduction of Persistent Statistics).
- The statistics were supplied through the MySQL Storage Engine Interface, which puts a lot of restrictions on what kind of data is supplied (for example, there is no way to get any data about value distribution in a non-indexed column)
- There was little control of the statistics. There was no way to "pin" current statistic values, or provide some values on your own, etc.

Engine-independent table statistics lift these limitations.

- Statistics are stored in regular tables in the `mysql` database.
  - it is possible for a DBA to read and update the values.
- More data is collected/used.

Statistics are stored in three tables, mysql.table_stats, mysql.column_stats and mysql.index_stats.

Use or update of data from these tables is controlled by use_stat_tables variable. Possible values are listed below:

Value	Meaning
'never'	The optimizer doesn't use data from statistics tables. Default for MariaDB 10.4.0 and below.
'complementary'	The optimizer uses data from statistics tables if the same kind of data is not provided by the storage engine.
'preferably'	Prefer the data from statistics tables, if it's not available there, use the data from the storage engine.
'complementary_for_queries'	Same as `complementary`, but for queries only (to avoid needlessly collecting for ANALYZE TABLE). From MariaDB 10.4.1.
'preferably_for_queries'	Same as `preferably`, but for queries only (to avoid needlessly collecting for ANALYZE TABLE). Available and default from MariaDB 10.4.1.

Collecting Statistics with the ANALYZE TABLE Statement

The ANALYZE TABLE statement can be used to collect table statistics. For example:

```
ANALYZE TABLE table_name;
```

When the ANALYZE TABLE statement is executed, MariaDB makes a call to the table's storage engine, and the storage engine collects its own statistics for the table. The specific behavior depends on the storage engine. For InnoDB, see InnoDB Persistent Statistics for more information.

When the ANALYZE TABLE statement is executed, MariaDB may also collect engine-independent statistics for the table. The specific behavior depends on the value of the use_stat_tables system variable. Engine-independent statistics will only be collected by the ANALYZE TABLE statement if one of the following is true:

- The use_stat_tables system variable is set to `complementary` or `preferably`.
- The ANALYZE TABLE statement includes the `PERSISTENT FOR` clause.

MariaDB starting with 10.4

In MariaDB 10.4 and later, the use_stat_tables system variable is set to `preferably_for_queries` by default. With this value, engine-independent statistics are used by default, but they are not collected by default. If you want to use engine-independent statistics with the default configuration, then you will have to collect them by executing the ANALYZE TABLE statement and by specifying the `PERSISTENT FOR` clause. It is recommended to collect engine-independent statistics on as-needed basis, so typically one will not have engine-independent statistics for all indexes/all columns.

Engine-independent statistics are collected by doing full table and full index scans, and this process can be quite expensive.

Collecting Statistics for Specific Columns or Indexes

The syntax for the ANALYZE TABLE statement has been extended with the `PERSISTENT FOR` clause. This clause allows one to collect engine-independent statistics only for particular columns or indexes. This clause also allows one to collect engine-independent statistics, regardless of the value of the use_stat_tables system variable. For example:

```
ANALYZE TABLE table_name PERSISTENT FOR ALL;
```

Statistics for columns using the BLOB and TEXT data types are not collected. If a column using one of these types is explicitly specified, then a warning is returned.

Examples of Statistics Collection

```
-- update all engine-independent statistics for all columns and indexes
ANALYZE TABLE tbl PERSISTENT FOR ALL;

-- update specific columns and indexes:
ANALYZE TABLE tbl PERSISTENT FOR COLUMNS (col1,col2,...) INDEXES (idx1,idx2,...);

-- empty lists are allowed:
ANALYZE TABLE tbl PERSISTENT FOR COLUMNS (col1,col2,...) INDEXES ();
ANALYZE TABLE tbl PERSISTENT FOR COLUMNS () INDEXES (idx1,idx2,...);

-- the following will only update mysql.table_stats fields:
ANALYZE TABLE tbl PERSISTENT FOR COLUMNS () INDEXES ();

-- when use_stat_tables is set to 'COMPLEMENTARY' or 'PREFERABLY',
-- a simple ANALYZE TABLE  collects engine-independent statistics for all columns and indexes.
SET SESSION use_stat_tables='COMPLEMENTARY';
ANALYZE TABLE tbl;
```

Manual Updates to Statistics Tables

Statistics are stored in three tables, mysql.table_stats, mysql.column_stats and mysql.index_stats.

It is possible to update statistics tables manually. One should modify the table(s) with regular INSERT/UPDATE/DELETE statements. Statistics data will be re-read when the tables are re-opened. One way to force all tables to be re-opened is to issue FLUSH TABLES command.

A few scenarios where one might need to update statistics tables manually:

- Deleting the statistics. Currently, the ANALYZE TABLE command will collect the statistics, but there is no special command to delete statistics.
- Running ANALYZE on a different server. ANALYZE TABLE does a full table scan, which can put too much load on the server. It is possible to run ANALYZE on the slave, and then take the data from statistics tables on the slave and apply it on the master.
- In some cases, knowledge of the database allows one to compute statistics manually in a more efficient way than ANALYZE does. One can compute the statistics manually and put it into the database.

3.3.4.6.2 Histogram-Based Statistics

MariaDB starting with 10.4.3

Histograms are collected by default from MariaDB 10.4.3.

Contents

Histogram-based statistics are a mechanism to improve the query plan chosen by the optimizer in certain situations. Before their introduction, all conditions on non-indexed columns were ignored when searching for the best execution plan. Histograms can be collected for both indexed and non-indexed columns, and are made available to the optimizer.

Histogram statistics are stored in the mysql.column_stats table, which stores data for engine-independent table statistics, and so are essentially a subset of engine-independent table statistics.

Consider this example, using the following query:

```
SELECT * FROM t1,t2 WHERE t1.a=t2.a and t2.b BETWEEN 1 AND 3;
```

Let's assume that

- table t1 contains 100 records
- table t2 contains 1000 records
- there is a primary index on t1(a)
- there is a secondary index on t2(a)
- there is no index defined on column t2.b
- the selectivity of the condition t2.b BETWEEN (1,3) is high (~ 1%)

Before histograms were introduced, the optimizer would choose the plan that:

- accesses t1 using a table scan
- accesses t2 using index t2(a)
- checks the condition t2.b BETWEEN 1 AND 3

This plan examines all rows of both tables and performs 100 index look-ups.

With histograms available, the optimizer can choose the following, more efficient plan:

- accesses table t2 in a table scan
- checks the condition t2.b BETWEEN 1 AND 3
- accesses t1 using index t1(a)

This plan also examine all rows from t2, but it performs only 10 look-ups to access 10 rows of table t1.

System Variables

There are a number of system variables that affect histograms.

histogram_size

The histogram_size variable determines the size, in bytes, from 0 to 255, used for a histogram. This is effectively the number of bins for `histogram_type=SINGLE_PREC_HB` or number of bins/2 for `histogram_type=DOUBLE_PREC_HB`. If it is set to 0 (the default for MariaDB 10.4.2 and below), no histograms are created when running an ANALYZE TABLE.

histogram_type

The histogram_type variable determines whether single precision (`SINGLE_PREC_HB`) or double precision (`DOUBLE_PREC_HB`) height-balanced histograms are created. From MariaDB 10.4.3, double precision is the default. For MariaDB 10.4.2 and below, single precision is the default.

From MariaDB 10.8, `JSON_HB` , JSON-format histograms, are accepted.

optimizer_use_condition_selectivity

The optimizer_use_condition_selectivity controls which statistics can be used by the optimizer when looking for the best query execution plan.

- `1` Use selectivity of predicates as in MariaDB 5.5.
- `2` Use selectivity of all range predicates supported by indexes.
- `3` Use selectivity of all range predicates estimated without histogram.

- `4` Use selectivity of all range predicates estimated with histogram.
- `5` Additionally use selectivity of certain non-range predicates calculated on record sample.

From MariaDB 10.4.1, the default is `4`. Until MariaDB 10.4.0, the default is `1`.

3.3.3.10 Index Statistics

3.3.4.6.4 InnoDB Persistent Statistics

Before MariaDB 10.0, InnoDB statistics were not stored on disk, meaning that on server restarts the statistics would need to be recalculated, which is both needless computation, as well as leading to inconsistent query plans.

There are a number of variables that control persistent statistics:

- innodb_stats_persistent - when set (the default) enables InnoDB persistent statistics.
- innodb_stats_auto_recalc - when set (the default), persistent statistics are automatically recalculated when the table changes significantly (more than 10% of the rows)
- innodb_stats_persistent_sample_pages - Number of index pages sampled (default 20) when estimating cardinality and statistics for indexed columns. Increasing this value will increases index statistics accuracy, but use more I/O resources when running ANALYZE TABLE.

These settings can be overwritten on a per-table basis by use of the STATS_PERSISTENT, STATS_AUTO_RECALC and STATS_SAMPLE_PAGES clauses in a CREATE TABLE or ALTER TABLE statement.

Details of the statistics are stored in two system tables in the mysql database:

- innodb_table_stats
- innodb_index_stats

3.3.4.6.5 Slow Query Log Extended Statistics

Contents

Overview

- Added extra logging to slow log of 'Thread_id, Schema, Query Cache hit, Rows sent and Rows examined'
- Added optional logging to slow log, through log_slow_verbosity, of query plan statistics
- Added new session variables log_slow_rate_limit, log_slow_verbosity, log_slow_filter
- Added log-slow-file as synonym for 'slow-log-file', as most slow-log variables starts with 'log-slow'
- Added log-slow-time as synonym for long-query-time.

Session Variables

log_slow_verbosity

You can set the verbosity of what's logged to the slow query log by setting the the log_slow_verbosity variable to a combination of the following values:

- `All` (From MariaDB 10.6.16)
  - Enable all verbosity options.
- `Query_plan`
  - For select queries, log information about the query plan. This includes "Full_scan", "Full_join", "Tmp_table", "Tmp_table_on_disk", "Filesort", "Filesort_on_disk" and number of "Merge_passes during sorting"
- `explain`
  - EXPLAIN output is logged in the slow query log. See explain-in-the-slow-query-log for details.
- `Innodb` (From MariaDB 10.6.15. Before that this option did nothing)
  - Kept for compatibility. Same as `engine`.
- `engine` (From MariaDB 10.6.15)

- Writes statistics from the storage engine. This includes:

Option	Description	Engine
`Pages_accessed`	Number of pages accessed from page buffer (innodb-buffer-pool / key cache)	InnoDB
`Pages_updated`	Number of pages updated in memory	InnoDB
`Pages_read_time`	Milliseconds spend reading pages from storage	InnoDB
`Old_rows_read`	Number of retrieval of old versions of rows in the engine (versioning)	InnoDB
`Engine_time`	Milliseconds spent inside engine calls (read_row / read_next_row etc)	All

- `Warnings` (From MariaDB 10.6.16)
  - Print all errors, warnings and notes related to statement, up to `log_slow_max_warnings` lines.
- `full`.
  - Old shortcut to enable all the verbosity options

The default value for `log_slow_verbosity` is ' ', to be compatible with MySQL 5.1.

The possible values for `log_slow_verbosity are` innodb,query_plan,explain,engine,warnings. `Multiple options are separated by ','.`

log_slow_verbosity is not supported when log_output='TABLE'.

In the future we will add more `engine` statistics and also support for other engines.

log_slow_filter

You can define which queries to log to the slow query log by setting the variable log_slow_filter to a combination of the following values:

- `admin`
  - Log administrative statements (create, optimize, drop etc...)
- `filesort`
  - Log statement if it uses filesort
- `filesort_on_disk`
  - Log statement if it uses filesort that needs temporary tables on disk
- `full_join`
  - Log statements that doesn't uses indexes to join tables
- `full_scan`
  - Log statements that uses full table scans
- `query_cache`
  - Log statements that are resolved by the query cache
- `query_cache_miss`
  - Log statements that are not resolved by the query cache
- `tmp_table`
  - Log statements that uses in memory temporary tables
- `tmp_table_on_disk`
  - Log statements that uses temporary tables on disk

Multiple options are separated by ','. If you don't specify any options everything will be logged.

log_slow_rate_limit

The log_slow_rate_limit variable limits logging to the slow query log by not logging every query (only one query / log_slow_rate_limit is logged). This is mostly useful when debugging and you get too much information to the slow query log.

Note that in any case, only queries that takes longer than **log_slow_time** or **long_query_time'** are logged (as before).

log_slow_max_warnings

MariaDB starting with 10.6.16

If one enables the warning option for `log_slow_verbosity`, all notes and warnings for a slow query will also be added to the slow query log. This is very usable when one has enabled warnings for Notes when an index cannot be used. `log_slow_max_warnings` limits the number of warnings printed to the slow query log per query. The default value is 10.

Credits

Part of this addition is based on the microslow patch from Percona.

3.3.4.6.6 User Statistics

The User Statistics feature was first released in MariaDB 5.2.0, and moved to the `userstat` plugin in MariaDB 10.1.1.

The `userstat` plugin creates the USER_STATISTICS, CLIENT_STATISTICS, the INDEX_STATISTICS, and the TABLE_STATISTICS tables in the INFORMATION_SCHEMA database. As an alternative to these tables, the plugin also adds the SHOW USER_STATISTICS, the SHOW CLIENT_STATISTICS, the SHOW INDEX_STATISTICS, and the SHOW TABLE_STATISTICS statements.

These tables and commands can be used to understand the server activity better and to identify the sources of your database's load.

The plugin also adds the FLUSH USER_STATISTICS, FLUSH CLIENT_STATISTICS, FLUSH INDEX_STATISTICS, and FLUSH TABLE_STATISTICS statements.

The MariaDB implementation of this plugin is based on the userstatv2 patch from Percona and Ourdelta. The original code comes from Google (Mark Callaghan's team) with additional work from Percona, Ourdelta, and Weldon Whipple. The MariaDB implementation provides the same functionality as the userstatv2 patch but a lot of changes have been made to make it faster and to better fit the MariaDB infrastructure.

Contents

How it Works

The `userstat` plugin works by keeping several hash tables in memory. All variables are incremented while the query is running. At the end of each statement the global values are updated.

Enabling the Plugin

By default statistics are not collected. This is to ensure that statistics collection does not cause any extra load on the server unless desired.

Set the userstat=ON system variable in a relevant server option group in an option file to enable the plugin. For example:

```
[mariadb]
...
userstat = 1
```

The value can also be changed dynamically. For example:

```
SET GLOBAL userstat=1;
```

Using the Plugin

Using the Information Schema Table

The `userstat` plugin creates the USER_STATISTICS, CLIENT_STATISTICS, the INDEX_STATISTICS, and the TABLE_STATISTICS tables in the INFORMATION_SCHEMA database.

```
SELECT * FROM INFORMATION_SCHEMA.USER_STATISTICS\G
*************************** 1. row ***************************
                  USER: root
     TOTAL_CONNECTIONS: 1
CONCURRENT_CONNECTIONS: 0
        CONNECTED_TIME: 297
             BUSY_TIME: 0.001725
              CPU_TIME: 0.001982
        BYTES_RECEIVED: 388
            BYTES_SENT: 2327
  BINLOG_BYTES_WRITTEN: 0
             ROWS_READ: 0
             ROWS_SENT: 12
          ROWS_DELETED: 0
         ROWS_INSERTED: 13
          ROWS_UPDATED: 0
       SELECT_COMMANDS: 4
       UPDATE_COMMANDS: 0
        OTHER_COMMANDS: 3
   COMMIT_TRANSACTIONS: 0
 ROLLBACK_TRANSACTIONS: 0
    DENIED_CONNECTIONS: 0
      LOST_CONNECTIONS: 0
         ACCESS_DENIED: 0
         EMPTY_QUERIES: 1
```

```
SELECT * FROM INFORMATION_SCHEMA.CLIENT_STATISTICS\G
*************************** 1. row ***************************
                CLIENT: localhost
     TOTAL_CONNECTIONS: 3
CONCURRENT_CONNECTIONS: 0
        CONNECTED_TIME: 4883
             BUSY_TIME: 0.009722
              CPU_TIME: 0.0102131
        BYTES_RECEIVED: 841
            BYTES_SENT: 13897
  BINLOG_BYTES_WRITTEN: 0
             ROWS_READ: 0
             ROWS_SENT: 214
          ROWS_DELETED: 0
         ROWS_INSERTED: 207
          ROWS_UPDATED: 0
       SELECT_COMMANDS: 10
       UPDATE_COMMANDS: 0
        OTHER_COMMANDS: 13
   COMMIT_TRANSACTIONS: 0
 ROLLBACK_TRANSACTIONS: 0
    DENIED_CONNECTIONS: 0
      LOST_CONNECTIONS: 0
         ACCESS_DENIED: 0
         EMPTY_QUERIES: 1
1 row in set (0.00 sec)
```

```
SELECT * FROM INFORMATION_SCHEMA.INDEX_STATISTICS WHERE TABLE_NAME = "author";
+--------------+------------+------------+-----------+
| TABLE_SCHEMA | TABLE_NAME | INDEX_NAME | ROWS_READ |
+--------------+------------+------------+-----------+
| books        | author     | by_name    |        15 |
+--------------+------------+------------+-----------+
```

```
SELECT * FROM INFORMATION_SCHEMA.TABLE_STATISTICS WHERE TABLE_NAME='user';
+--------------+------------+-----------+--------------+-----------------------+
| TABLE_SCHEMA | TABLE_NAME | ROWS_READ | ROWS_CHANGED | ROWS_CHANGED_X_INDEXES |
+--------------+------------+-----------+--------------+-----------------------+
| mysql        | user       |         5 |            2 |                     2 |
+--------------+------------+-----------+--------------+-----------------------+
```

Using the SHOW Statements

As an alternative to the INFORMATION_SCHEMA tables, the `userstat` plugin also adds the SHOW USER_STATISTICS, the SHOW CLIENT_STATISTICS, the SHOW INDEX_STATISTICS, and the SHOW TABLE_STATISTICS statements.

These commands are another way to display the information stored in the information schema tables. WHERE clauses are accepted. LIKE clauses are accepted but ignored.

```
SHOW USER_STATISTICS
SHOW CLIENT_STATISTICS
SHOW INDEX_STATISTICS
SHOW TABLE_STATISTICS
```

Flushing Plugin Data

The `userstat` plugin also adds the FLUSH USER_STATISTICS, FLUSH CLIENT_STATISTICS, FLUSH INDEX_STATISTICS, and FLUSH TABLE_STATISTICS statements, which discard the information stored in the specified information schema table.

```
FLUSH USER_STATISTICS
FLUSH CLIENT_STATISTICS
FLUSH INDEX_STATISTICS
FLUSH TABLE_STATISTICS
```

Versions

USER_STATISTICS

Version	Status	Introduced
2.0	Stable	MariaDB 10.1.18
2.0	Gamma	MariaDB 10.1.1

CLIENT_STATISTICS

Version	Status	Introduced
2.0	Stable	MariaDB 10.1.13
2.0	Gamma	MariaDB 10.1.1

INDEX_STATISTICS

Version	Status	Introduced
2.0	Stable	MariaDB 10.1.13
2.0	Gamma	MariaDB 10.1.1

TABLE_STATISTICS

Version	Status	Introduced
2.0	Stable	MariaDB 10.1.18
2.0	Gamma	MariaDB 10.1.1

System Variables

`userstat`

- **Description:** If set to `1`, user statistics will be activated.
- **Commandline:** `--userstat=1`
- **Scope:** Global
- **Dynamic:** Yes

- **Data Type:** `boolean`
- **Default Value:** `OFF`

3.3.4.7 MIN/MAX optimization

Contents

Min/Max optimization without GROUP BY

MariaDB and MySQL can optimize the MIN() and MAX() functions to be a single row lookup in the following cases:

- There is only one table used in the `SELECT`.
- You only have constants, `MIN()` and `MAX()` in the `SELECT` part.
- The argument to `MIN()` and `MAX()` is a simple column reference that is part of a key.
- There is no `WHERE` clause or the `WHERE` is used with a constant for all prefix parts of the key before the argument to `MIN()` / `MAX()`.
- If the argument is used in the `WHERE` clause, it can be be compared to a constant with `<` or `<=` in case of `MAX()` and with `>` or `>=` in case of `MIN()`.

Here are some examples to clarify this. In this case we assume there is an index on columns `(a,b,c)`

```
SELECT MIN(a),MAX(a) from t1
SELECT MIN(b) FROM t1 WHERE a=const
SELECT MIN(b),MAX(b) FROM t1 WHERE a=const
SELECT MAX(c) FROM t1 WHERE a=const AND b=const
SELECT MAX(b) FROM t1 WHERE a=const AND b<const
SELECT MIN(b) FROM t1 WHERE a=const AND b>const
SELECT MIN(b) FROM t1 WHERE a=const AND b BETWEEN const AND const
SELECT MAX(b) FROM t1 WHERE a=const AND b BETWEEN const AND const
```

- Instead of `a=const` the condition `a IS NULL` can be used.

The above optimization also works for subqueries:

```
SELECT x from t2 where y= (SELECT MIN(b) FROM t1 WHERE a=const)
```

Cross joins, where there is no join condition for a table, can also be optimized to a few key lookups:

```
select min(t1.key_part_1), max(t2.key_part_1) from t1, t2
```

Min/Max optimization with GROUP BY

MariaDB and MySQL support loose index scan, which can speed up certain `GROUP BY` queries. The basic idea is that when scanning a `BTREE` index (the most common index type for the MariaDB storage engines) we can jump over identical values for any prefix of a key and thus speed up the scan significantly.

Loose scan is possible in the following cases:

- The query uses only one table.
- The `GROUP BY` part only uses indexed columns in the same order as in the index.
- The only aggregated functions in the `SELECT` part are `MIN()` and `MAX()` functions and all of them using the same column which is the next index part after the used `GROUP BY` columns.
- Partial indexed columns cannot be used (like only indexing 10 characters of a `VARCHAR(20)` column).

Loose scan will apply for your query if `EXPLAIN` shows `Using index for group-by` in the `Extra` column. In this case the optimizer will do only one extra row fetch to calculate the value for `MIN()` or `MAX()` for every unique key prefix.

The following examples assume that the table `t1` has an index on `(a,b,c)`.

```
SELECT a, b, MIN(c),MAX(c) FROM t1 GROUP BY a,b
```

3.3.4.8 Filesort with Small LIMIT Optimization

Contents

Optimization Description

When `n` is sufficiently small, the optimizer will use a priority queue for sorting. Before the optimization's porting to MariaDB 10.0, the alternative was, roughly speaking, to sort the entire output and then pick only first `n` rows.

Optimization Visibility in MariaDB

There are two ways to check whether filesort has used a priority queue.

Status Variable

The first way is to check the Sort_priority_queue_sorts status variable. It shows the number of times that sorting was done through a priority queue. (The total number of times sorting was done is a sum Sort_range and Sort_scan).

Slow Query Log

The second way is to check the slow query log. When one uses Extended statistics in the slow query log and specifies log_slow_verbosity=query_plan, slow query log entries look like this

```
# Time: 140714 18:30:39
# User@Host: root[root] @ localhost []
# Thread_id: 3  Schema: test  QC_hit: No
# Query_time: 0.053857  Lock_time: 0.000188  Rows_sent: 11  Rows_examined: 100011
# Full_scan: Yes  Full_join: No  Tmp_table: No  Tmp_table_on_disk: No
# Filesort: Yes  Filesort_on_disk: No  Merge_passes: 0  Priority_queue: Yes
SET timestamp=1405348239;SET timestamp=1405348239;
select * from t1 where col1 between 10 and 20 order by col2 limit 100;
```

Note the "Priority_queue: Yes" on the last comment line. (`pt-query-digest` is able to parse slow query logs with the Priority_queue field)

As for `EXPLAIN` , it will give no indication whether filesort uses priority queue or the generic quicksort and merge algorithm. `Using filesort` will be shown in both cases, by both MariaDB and MySQL.

3.3.4.9 LIMIT ROWS EXAMINED

Syntax

```
SELECT ... FROM ... WHERE ...
[group_clause] [order_clause]
LIMIT [[offset,] row_count] ROWS EXAMINED rows_limit;
```

Similar to the parameters of `LIMIT` , *`rows_limit`* can be both a prepared statement parameter, or a stored program parameter.

Description

The purpose of this optimization is to provide the means to terminate the execution of `SELECT` statements which examine too many rows, and thus use too many resources. This is achieved through an extension of the `LIMIT` clause — `LIMIT ROWS EXAMINED number_of_rows` . Whenever possible the semantics of `LIMIT ROWS EXAMINED` is the same as that of normal `LIMIT` (for instance for aggregate functions).

The `LIMIT ROWS EXAMINED` clause is taken into account by the query engine only during query execution. Thus the clause

is ignored in the following cases:

- If a query is `EXPLAIN` -ed.
- During query optimization.
- During auxiliary operations such as writing to system tables (e.g. logs).

The clause is not applicable to `DELETE` or `UPDATE` statements, and if used in those statements produces a syntax error.

The effects of this clause are as follows:

- The server counts the number of read, inserted, modified, and deleted rows during query execution. This takes into account the use of temporary tables, and sorting for intermediate query operations.
- Once the counter exceeds the value specified in the LIMIT ROWS EXAMINED clause, query execution is terminated as soon as possible.
- The effects of terminating the query because of LIMIT ROWS EXAMINED are as follows:
  - The result of the query is a subset of the complete query, depending on when the query engine detected that the limit was reached. The result may be empty if no result rows could be computed before reaching the limit.
  - A warning is generated of the form: "Query execution was interrupted. The query examined at least 100 rows, which exceeds LIMIT ROWS EXAMINED (20). The query result may be incomplete."
  - If query processing was interrupted during filesort, an error is returned in addition to the warning.
  - If a UNION was interrupted during execution of one of its queries, the last step of the UNION is still executed in order to produce a partial result.
  - Depending on the join and other execution strategies used for a query, the same query may produce no result at all, or a different subset of the complete result when terminated due to LIMIT ROWS EXAMINED.
  - If the query contains a GROUP BY clause, the last group where the limit was reached will be discarded.

The `LIMIT ROWS EXAMINED` clause cannot be specified on a per-subquery basis. There can be only one `LIMIT ROWS EXAMINED` clause for the whole `SELECT` statement. If a `SELECT` statement contains several subqueries with `LIMIT ROWS EXAMINED`, the one that is parsed last is taken into account.

Examples

A simple example of the clause is:

```
SELECT * from t1, t2 LIMIT 10 ROWS EXAMINED 10000;
```

The `LIMIT ROWS EXAMINED` clause is global for the whole statement.

If a composite query (such as `UNION`, or query with derived tables or with subqueries) contains more than one `LIMIT ROWS EXAMINED`, the last one parsed is taken into account. In this manner either the last or the outermost one is taken into account. For instance, in the query:

```
SELECT * FROM t1
WHERE c1 IN (SELECT * FROM t2 WHERE c2 > ' ' LIMIT ROWS EXAMINED 0)
LIMIT ROWS EXAMINED 11;
```

The limit that is taken into account is 11, not 0.

3.3.4.10 Block-Based Join Algorithms

Contents

In the versions of MariaDB/MySQL before 5.3 only one block-based join algorithm was implemented: the Block Nested Loops (BNL) join algorithm which could only be used for inner joins.

MariaDB 5.3 enhanced the implementation of BNL joins and provides a variety of block-based join algorithms that can be used for inner joins, outer joins, and semi-joins. Block-based join algorithms in MariaDB employ a join buffer to accumulate records of the first join operand before they start looking for matches in the second join operand.

This page documents the various block-based join algorithms.

- Block Nested Loop (BNL) join
- Block Nested Loop Hash (BNLH) join
- Block Index join known as Batch Key Access (BKA) join
- Block Index Hash join known as Batch Key Access Hash (BKAH) join

Block Nested Loop Join

The major difference between the implementation of BNL join in MariaDB 5.3 compared to earlier versions of MariaDB/MySQL is that the former uses a new format for records written into join buffers. This new format allows:

- More efficient use of buffer space for null field values and field values of flexible length types (like the varchar type)
- Support for so-called *incremental* join buffers saving buffer space for multi-way joins
- Use of the algorithm for outer joins and semi-joins

How Block Nested Loop Join Works

The algorithm performs a join operation of tables t1 and t2 according to the following schema.
The records of the first operand are written into the join buffer one by one until the buffer is full.
The records of the second operand are read from the base/temporary table one by one. For every read record r2 of table t2 the join buffer is scanned, and, for any record r1 from the buffer such that r2 matches r1 the concatenation of the interesting fields of r1 and r2 is sent to the result stream of the corresponding partial join.
To read the records of t2 a full table scan, a full index scan or a range index scan is performed. Only the records that meet the condition pushed to table t2 are checked for a match of the records from the join buffer.
When the scan of the table t2 is finished a new portion of the records of the first operand fills the buffer and matches for these records are looked for in t2.
The buffer refills and scans of the second operand that look for matches in the join buffer are performed again and again until the records of first operand are exhausted.
In total the algorithm scans the second operand as many times as many refills of the join buffer occur.

More Efficient Usage of Join Buffer Space

No join buffer space is used for null field values.
Any field value of a flexible length type is not padded by 0 up to the maximal field size anymore.

Incremental Join Buffers

If we have a query with a join of three tables t1, t2, t3 such that table t1 is joined with table t2 and the result of this join operation is joined with table t3 then two join buffers can be used to execute the query. The first join buffer B1 is used to store the records comprising interesting fields of table t1, while the second join buffer B2 contains the records with fields from the partial join of t1 and t2. The interesting fields of any record r1 from B1 are copied into B2 for any record record r1,r2 from the partial join of t1 and t2. One could suggest storing in B2 just a pointer to the position of the r1 fields in B1 together with the interesting fields from t2. So for any record r2 matching the record r1 the buffer B2 would contain a reference to the fields of r1 in B1 and the fields of r2. In this case the buffer B2 is called incremental. Incremental buffers allow to avoid copying field values from one buffer into another. They also allow to save a significant amount of buffer space if for a record from t1 several matches from t2 are expected.

Using Join Buffers for Simple Outer Joins and Semi-joins

If a join buffer is used for a simple left outer join of tables t1 and t1 t1 LEFT JOIN t2 ON P(t1,t2) then each record r1 stored in the buffer is provided with a match flag. Initially this flag is set off. As soon as the first match for r1 is found this flag is set on. When all matching candidates from t2 have been check, the record the join buffer are scanned and for those of them that still have there match flags off null-complemented rows are generated. The same match flag is used for any record in the join buffer is a semi-join operation t1 SEMI JOIN t2 ON P(t1,t2) is performed with a block based join algorithm. When this match flag is set to on for a record r1 in the buffer no matches from table t2 for record r1 are looked for anymore.

Block Hash Join

Block based hash join algorithm is a new option to be used for join operations in MariaDB 5.3. It can be employed in the cases when there are equi-join sub-condition for the joined tables, in the other words when equalities of the form t2.f1=

e1(t1),...,t2.fn=en(t1) can be extracted from the full join condition. As any block based join algorithm this one used a join buffer filled with the records of the first operand and looks through the records of the second operand to find matches for the records in the buffer.

How Block Hash Join Works

For each refill of the join buffer and each record r1 from it the algorithm builds a hash table with the keys constructed over the values e1(r1),...en(r1). Then the records of t2 are looked through. For each record r2 from t2 that the condition pushed to the table t2 a hash key over the fields r2.f1,..., r2.fn is calculated to probe into the hash table. The probing returns those records from the buffer to which r2 matches. As for BNL join algorithm this algorithm scans the second operand as many time as many refills of the buffer occur. Yet it has to look only through the records of one bucket in the hash table when looking for the records to which a record from t2 matches, not through all records in the join buffer as BNL join algorithm does. The implementation of this algorithm in MariaDB builds the hash table with hash keys at the very end of the join buffer. That's why the number of records written into the buffer at one refill is less then for BNL join algorithms. However a much shorter list of possible matching candidates makes this the block hash join algorithm usually much faster then BNL join.

Batch Key Access Join

Batch Keys Access join algorithm performs index look-ups when looking for possible matching candidates provided by the second join operand. With this respect the algorithm behave itself as the regular join algorithm. Yet BKA performs index look-ups for a batch of the records from the join buffer. For conventional database engines like InnoDB/MyISAM it allows to fetch matching candidates in an optimal way. For the engines with remote data store such as FederateX/Spider the algorithm allows to save on transfers between the MySQL node and the data store nodes.

How Batch Keys Access Join Works

The implementation of the algorithm in 5.3 heavily exploits the multi-range-read interface and its properties. The interface hides the actual mechanism of fetching possible candidates for matching records from the table to be joined. As any block based join algorithm the BKA join repeatedly fills the join buffer with records of the first operand and for each refill it finds records from the join table that could match the records in the buffer. To find such records it asks the MRR interface to perform index look-ups with the keys constructed over all records from the buffer. Together with each key the interface receives a return address - a reference to the record over which this key has been constructed. The actual implementation functions of the MRR interface organize and optimize somehow the process of fetching the records of the joined table by the received keys. Each fetched record r2 is appended with the return address associated with the key by which the record has been found and the result is passed to the BKA join procedure. The procedure takes the record r1 from the join buffer by the return address, joins it with r2 and checks the join condition. If the condition is evaluated to true the joined records is sent to the result stream of the join operation. So for each record returned by the MRR interface only one record from the join buffer is accessed. The number of records from table t2 fetched by the BKA join is exactly the same as for the regular nested loops join algorithm. Yet BKA join allows to optimize the order in which the records are fetched.

Interaction of BKA Join With the MRR Functions

BKA join interacts with the MRR functions respecting the following contract. The join procedure calls the MRR function multi_range_read_init passing it the callback functions that allows to initialize reading keys for the records in the join buffer and to iterate over these keys. It also passes the parameters of the buffer for MRR needs allocated within the join buffer space. Then BKA join repeatedly calls the MRR function multi_range_read_next. The function works as an iterator function over the records fetched by index look-ups with the keys produced by a callback function set in the call of multi_range_read_init. A call of the function multi_range_read_next returns the next fetched record through the dedicated record buffer, and the associated reference to the matched record from the join buffer as the output parameter of the function.

Managing Usage of Block-Based Join Algorithms

Currently 4 different types of block-based join algorithms are supported. For a particular join operation each of them can be employed with a regular (flat) join buffer or with an incremental join buffer.

Three optimizer switches - `join_cache_incremental` , `join_cache_hashed` , `join_cache_bka` – and the system variable join_cache_level control which of the 8 variants of the block-based algorithms will be used for join operations.

If `join_cache_bka` is off then BKA and BKAH join algorithms are not allowed. If `join_cache_hashed` is off then BNLH and BKAH join algorithms are not allowed. If `join_cache_incremental` is off then no incremental variants of the block-based join algorithms are allowed.

By default the switches `join_cache_incremental` , `join_cache_hashed` , `join_cache_bka` are set to 'on'. However it

does not mean that by default any of block-based join algorithms is allowed to be used. All of them are allowed only if the system variable join_cache_level is set to 8. This variable can take an integer value in the interval from 0 to 8.

If the value is set to 0 no block-based algorithm can be used for a join operation. The values from 1 to 8 correspond to the following variants of block-based join algorithms :

- 1 – Flat BNL
- 2 – Incremental BNL
- 3 – Flat BNLH
- 4 – Incremental BNLH
- 5 – Flat BKA
- 6 – Incremental BKA
- 7 – Flat BKAH
- 8 – Incremental BKAH

If the value of join_cache_level is set to N, any of block-based algorithms with the level greater than N is disallowed.

So if join_cache_level is set to 5, no usage of BKAH is allowed and usage of incremental BKA is not allowed either while usage of all remaining variants are controlled by the settings of the optimizer switches `join_cache_incremental`, `join_cache_hashed`, `join_cache_bka`.

By default join_cache_level is set to 2. In other words only usage of flat or incremental BNL is allowed.

By default block-based algorithms can be used only for regular (inner) join operations. To allow them for outer join operations (left outer joins and right outer joins) the optimizer switch `outer_join_with_cache` has to be set to 'on'. Setting the optimizer switch `semijoin_with_cache` to 'on' allows using these algorithms for semi-join operations.

Currently, only incremental variants of the block-based join algorithms can be used for nested outer joins and nested semi-joins.

Size of Join Buffers

The maximum size of join buffers used by block-based algorithms is controlled by setting the join_buffer_size system variable. This value must be large enough in order for the join buffer employed for a join operation to contain all relevant fields for at least one joined record.

MariaDB 5.3 introduced the system variable join_buffer_space_limit that limits the total memory used for join buffers in a query.

To optimize the usage of the join buffers within the limit set by `join_buffer_space_limit`, one should use the optimizer switch `optimize_join_buffer_size=on`. When this flag is set to 'off' (default until MariaDB 10.4.2), the size of the used join buffer is taken directly from the join_buffer_size system variable. When this flag is set to 'on' (default from MariaDB 10.4.3) then the size of the buffer depends on the estimated number of rows in the partial join whose records are to be stored in the buffer.

Related MRR Settings

To use BKA/BKAH join algorithms for InnoDB/MyISAM, one must set the optimizer switch `mrr` to 'on'. When using these algorithms for InnoDB/MyISAM the overall performance of the join operations can be dramatically improved if the optimizer switch `mrr_sort_keys` is set 'on'.

3.3.4.11 index_merge sort_intersection

Prior to MariaDB 5.3, the `index_merge` access method supported `union`, `sort-union`, and `intersection` operations. Starting from MariaDB 5.3, the `sort-intersection` operation is also supported. This allows the use of `index_merge` in a broader number of cases.

This feature is disabled by default. To enable it, turn on the optimizer switch `index_merge_sort_intersection` like so:

```
SET optimizer_switch='index_merge_sort_intersection=on'
```

Limitations of index_merge/intersection

Prior to MariaDB 5.3, the `index_merge` access method had one intersection strategy called `intersection`. That strategy can only be used when merged index scans produced rowid-ordered streams. In practice this means that an `intersection` could only be constructed from equality (=) conditions.

For example, the following query will use `intersection`:

```
MySQL [ontime]> EXPLAIN SELECT AVG(arrdelay) FROM ontime WHERE depdel15=1 AND OriginState
='CA';
+--+-----------+------+-----------+-------------------+-------------------+-------+----+---
--+-----------------------------------------------+
|id|select_type|table |type       |possible_keys      |key                |key_len|ref |rows
|Extra                                          |
+--+-----------+------+-----------+-------------------+-------------------+-------+----+---
--+-----------------------------------------------+
| 1|SIMPLE     |ontime|index_merge|OriginState,DepDel15|OriginState,DepDel15|3,5
|NULL|76952|Using intersect(OriginState,DepDel15);Using where|
+--+-----------+------+-----------+-------------------+-------------------+-------+----+---
--+-----------------------------------------------+
```

but if you replace `OriginState ='CA'` with `OriginState IN ('CA', 'GB')` (which matches the same number of records), then `intersection` is not usable anymore:

```
MySQL [ontime]> explain select avg(arrdelay) from ontime where depdel15=1 and OriginState IN
('CA', 'GB');
+--+-----------+------+----+-------------------+--------+-------+-----+-----------+
|id|select_type|table |type|possible_keys      |key     |key_len|ref  |rows |Extra      |
+--+-----------+------+----+-------------------+--------+-------+-----+-----------+
| 1|SIMPLE     |ontime|ref |OriginState,DepDel15|DepDel15|5      |const|36926|Using where|
+--+-----------+------+----+-------------------+--------+-------+-----+-----+-----------+
```

The latter query would also run 5.x times slower (from 2.2 to 10.8 seconds) in our experiments.

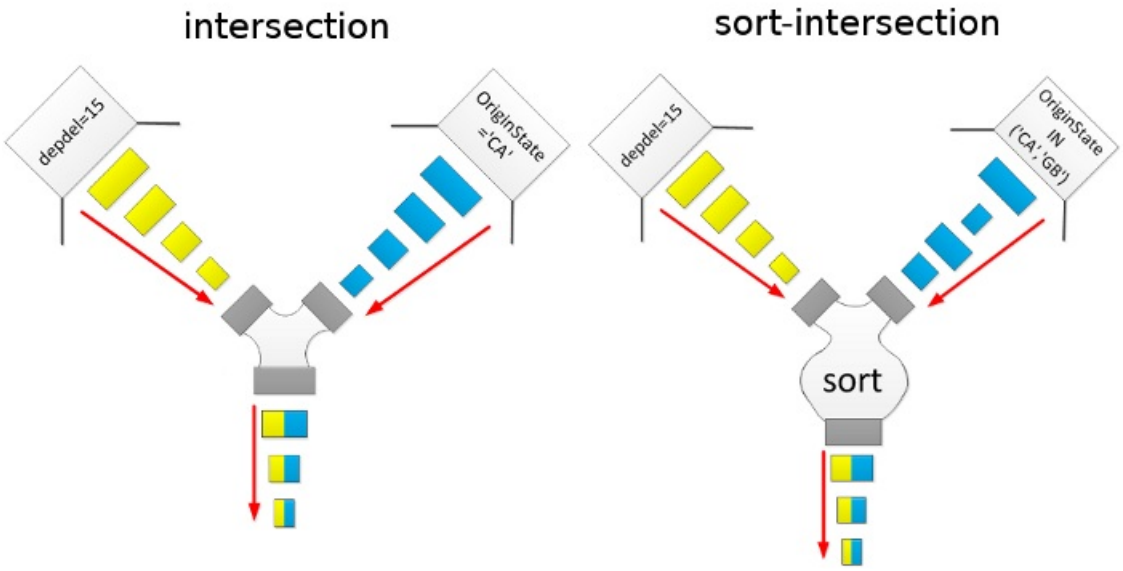
How index_merge/sort_intersection improves the situation

In MariaDB 5.3, when `index_merge_sort_intersection` is enabled, `index_merge` intersection plans can be constructed from non-equality conditions:

```
MySQL [ontime]> explain select avg(arrdelay) from ontime where depdel15=1 and OriginState IN
('CA', 'GB');
+--+-----------+------+-----------+-------------------+-------------------+-------+----+---
--+-------------------------------------------------------+
|id|select_type|table |type       |possible_keys      |key                |key_len|ref |rows
|Extra                                                    |
+--+-----------+------+-----------+-------------------+-------------------+-------+----+---
--+-------------------------------------------------------+
| 1|SIMPLE     |ontime|index_merge|OriginState,DepDel15|DepDel15,OriginState|5,3
|NULL|60754|Using sort_intersect(DepDel15,OriginState); Using where |
+--+-----------+------+-----------+-------------------+-------------------+-------+----+---
--+-------------------------------------------------------+
```

In our tests, this query ran in 3.2 seconds, which is not as good as the case with two equalities, but still much better than 10.8 seconds we were getting without `sort_intersect`.

The `sort_intersect` strategy has higher overhead than `intersect` but is able to handle a broader set of `WHERE` conditions.



When to Use

`index_merge/sort_intersection` works best on tables with lots of records and where intersections are sufficiently large (but still small enough to make a full table scan overkill).

The benefit is expected to be bigger for io-bound loads.

3.3.4.12 optimizer_switch

Contents

optimizer_switch is a server variable that one can use to enable/disable specific optimizations.

Syntax

To set or unset the various optimizations, use the following syntax:

```
SET [GLOBAL|SESSION] optimizer_switch='cmd[,cmd]...';
```

The *cmd* takes the following format:

Syntax	Description
default	Reset all optimizations to their default values.
optimization_name=default	Set the specified optimization to its default value.
optimization_name=on	Enable the specified optimization.
optimization_name=off	Disable the specified optimization.

There is no need to list all flags - only those that are specified in the command will be affected.

Available Flags

Below is a list of all *optimizer_switch* flags available in MariaDB:

Flag and MariaDB default	Supported in MariaDB since	Supported in MySQL since
condition_pushdown_for_derived=on	MariaDB 10.2.2	-
condition_pushdown_for_subquery=on	MariaDB 10.4.0	-

condition_pushdown_from_having=on	MariaDB 10.4.3	-
derived_merge=on	MariaDB 5.3	MySQL 5.7
derived_with_keys=on	MariaDB 5.3	-
default	MariaDB 5.1	MySQL 5.1
engine_condition_pushdown=off	MariaDB 5.5 (deprecated in 10.1)	MySQL 5.5
exists_to_in=on	MariaDB 10.0	-
extended_keys=on	MariaDB 5.5.21	-
firstmatch=on	MariaDB 5.3	MySQL 5.6
index_condition_pushdown=on	MariaDB 5.3	MySQL 5.6
hash_join_cardinality=off	MariaDB 10.6.13 (MDEV-30812)	-
index_merge=on	MariaDB 5.1	MySQL 5.1
index_merge_intersection=on	MariaDB 5.1	MySQL 5.1
index_merge_sort_intersection=off	MariaDB 5.3	-
index_merge_sort_union=on	MariaDB 5.1	MySQL 5.1
index_merge_union=on#	MariaDB 5.1	MySQL 5.1
in_to_exists=on	MariaDB 5.3	-
join_cache_bka=on	MariaDB 5.3	-
join_cache_hashed=on	MariaDB 5.3	-
join_cache_incremental=on	MariaDB 5.3	-
loosescan=on	MariaDB 5.3	MySQL 5.6
materialization=on (semi-join, non-semi-join)	MariaDB 5.3	MySQL 5.6
mrr=off	MariaDB 5.3	MySQL 5.6
mrr_cost_based=off	MariaDB 5.3	MySQL 5.6
mrr_sort_keys=off	MariaDB 5.3	-
not_null_range_scan=off	MariaDB 10.5	-
optimize_join_buffer_size=on	MariaDB 5.3, Defaults to ON from MariaDB 10.4.3	-
orderby_uses_equalities=on	MariaDB 10.1.15	-
outer_join_with_cache=on	MariaDB 5.3	-
partial_match_rowid_merge=on	MariaDB 5.3	-
partial_match_table_scan=on	MariaDB 5.3	-
rowid_filter=on	MariaDB 10.4.3	-
sargable_casefold=on	MariaDB 11.3.0	-
semijoin=on	MariaDB 5.3	MySQL 5.6
semijoin_with_cache=on	MariaDB 5.3	-
split_materialized=on [1]	MariaDB 10.3.4	-
subquery_cache=on	MariaDB 5.3	-
table_elimination=on	MariaDB 5.1	-

1. ↑ replaced split_grouping_derived, introduced in MariaDB 10.3.1

Defaults

From version	Default optimizer_switch setting

MariaDB 11.3.0	index_merge=on, index_merge_union=on, index_merge_sort_union=on, index_merge_intersection=on, index_merge_sort_intersection=off, index_condition_pushdown=on, derived_merge=on, derived_with_keys=on, firstmatch=on, loosescan=on, materialization=on, in_to_exists=on, semijoin=on, partial_match_rowid_merge=on, partial_match_table_scan=on, subquery_cache=on, mrr=off, mrr_cost_based=off, mrr_sort_keys=off, outer_join_with_cache=on, semijoin_with_cache=on, join_cache_incremental=on, join_cache_hashed=on, join_cache_bka=on, optimize_join_buffer_size=on, table_elimination=on, extended_keys=on, exists_to_in=on, orderby_uses_equalities=on, condition_pushdown_for_derived=on, split_materialized=on, condition_pushdown_for_subquery=on, rowid_filter=on, condition_pushdown_from_having=on, not_null_range_scan=off, hash_join_cardinality=on, sargable_casefold=on
MariaDB 11.0.2	index_merge=on, index_merge_union=on, index_merge_sort_union=on, index_merge_intersection=on, index_merge_sort_intersection=off, engine_condition_pushdown=off, index_condition_pushdown=on, derived_merge=on, derived_with_keys=on, firstmatch=on, loosescan=on, materialization=on, in_to_exists=on, semijoin=on, partial_match_rowid_merge=on, partial_match_table_scan=on, subquery_cache=on, mrr=off, mrr_cost_based=off, mrr_sort_keys=off, outer_join_with_cache=on, semijoin_with_cache=on, join_cache_incremental=on, join_cache_hashed=on, join_cache_bka=on, optimize_join_buffer_size=on, table_elimination=on, extended_keys=on, exists_to_in=on, orderby_uses_equalities=on, condition_pushdown_for_derived=on, split_materialized=on, condition_pushdown_for_subquery=on, rowid_filter=on,condition_pushdown_from_having=on, not_null_range_scan=off, hash_join_cardinality=on
MariaDB 10.6.13, MariaDB 10.11.3	index_merge=on, index_merge_union=on, index_merge_sort_union=on, index_merge_intersection=on, index_merge_sort_intersection=off, engine_condition_pushdown=off, index_condition_pushdown=on, derived_merge=on, derived_with_keys=on, firstmatch=on, loosescan=on, materialization=on, in_to_exists=on, semijoin=on, partial_match_rowid_merge=on, partial_match_table_scan=on, subquery_cache=on, mrr=off, mrr_cost_based=off, mrr_sort_keys=off, outer_join_with_cache=on, semijoin_with_cache=on, join_cache_incremental=on, join_cache_hashed=on, join_cache_bka=on, optimize_join_buffer_size=on, table_elimination=on, extended_keys=on, exists_to_in=on, orderby_uses_equalities=on, condition_pushdown_for_derived=on, split_materialized=on, condition_pushdown_for_subquery=on, rowid_filter=on,condition_pushdown_from_having=on, not_null_range_scan=off, hash_join_cardinality=off
MariaDB 10.5.0	index_merge=on, index_merge_union=on, index_merge_sort_union=on, index_merge_intersection=on, index_merge_sort_intersection=off, engine_condition_pushdown=off, index_condition_pushdown=on, derived_merge=on, derived_with_keys=on, firstmatch=on, loosescan=on, materialization=on, in_to_exists=on, semijoin=on, partial_match_rowid_merge=on, partial_match_table_scan=on, subquery_cache=on, mrr=off, mrr_cost_based=off, mrr_sort_keys=off, outer_join_with_cache=on, semijoin_with_cache=on, join_cache_incremental=on, join_cache_hashed=on, join_cache_bka=on, optimize_join_buffer_size=on, table_elimination=on, extended_keys=on, exists_to_in=on, orderby_uses_equalities=on, condition_pushdown_for_derived=on, split_materialized=on, condition_pushdown_for_subquery=on, rowid_filter=on,condition_pushdown_from_having=on, not_null_range_scan=off
MariaDB 10.4.3	index_merge=on, index_merge_union=on, index_merge_sort_union=on, index_merge_intersection=on, index_merge_sort_intersection=off, engine_condition_pushdown=off, index_condition_pushdown=on, derived_merge=on, derived_with_keys=on, firstmatch=on, loosescan=on, materialization=on, in_to_exists=on, semijoin=on, partial_match_rowid_merge=on, partial_match_table_scan=on, subquery_cache=on, mrr=off, mrr_cost_based=off, mrr_sort_keys=off, outer_join_with_cache=on, semijoin_with_cache=on, join_cache_incremental=on, join_cache_hashed=on, join_cache_bka=on, optimize_join_buffer_size=on, table_elimination=on, extended_keys=on, exists_to_in=on, orderby_uses_equalities=on, condition_pushdown_for_derived=on, split_materialized=on, condition_pushdown_for_subquery=on, rowid_filter=on,condition_pushdown_from_having=on
MariaDB 10.4.0	index_merge=on, index_merge_union=on, index_merge_sort_union=on, index_merge_intersection=on, index_merge_sort_intersection=off, engine_condition_pushdown=off, index_condition_pushdown=on, derived_merge=on, derived_with_keys=on, firstmatch=on, loosescan=on, materialization=on, in_to_exists=on, semijoin=on, partial_match_rowid_merge=on, partial_match_table_scan=on, subquery_cache=on, mrr=off, mrr_cost_based=off, mrr_sort_keys=off, outer_join_with_cache=on, semijoin_with_cache=on, join_cache_incremental=on, join_cache_hashed=on, join_cache_bka=on, optimize_join_buffer_size=off, table_elimination=on, extended_keys=on, exists_to_in=on, orderby_uses_equalities=on, condition_pushdown_for_derived=on, split_materialized=on, condition_pushdown_for_subquery=on
MariaDB 10.3.4	index_merge=on, index_merge_union=on, index_merge_sort_union=on, index_merge_intersection=on, index_merge_sort_intersection=off, engine_condition_pushdown=off, index_condition_pushdown=on, derived_merge=on, derived_with_keys=on, firstmatch=on, loosescan=on, materialization=on, in_to_exists=on, semijoin=on, partial_match_rowid_merge=on, partial_match_table_scan=on, subquery_cache=on, mrr=off, mrr_cost_based=off, mrr_sort_keys=off, outer_join_with_cache=on, semijoin_with_cache=on, join_cache_incremental=on, join_cache_hashed=on, join_cache_bka=on, optimize_join_buffer_size=off, table_elimination=on, extended_keys=on, exists_to_in=on, orderby_uses_equalities=on, condition_pushdown_for_derived=on, split_materialized=on

MariaDB 10.2.2	index_merge=on, index_merge_union=on, index_merge_sort_union=on, index_merge_intersection=on, index_merge_sort_intersection=off, engine_condition_pushdown=off, index_condition_pushdown=on, derived_merge=on, derived_with_keys=on, firstmatch=on, loosescan=on, materialization=on, in_to_exists=on, semijoin=on, partial_match_rowid_merge=on, partial_match_table_scan=on, subquery_cache=on, mrr=off, mrr_cost_based=off, mrr_sort_keys=off, outer_join_with_cache=on, semijoin_with_cache=on, join_cache_incremental=on, join_cache_hashed=on, join_cache_bka=on, optimize_join_buffer_size=off, table_elimination=on, extended_keys=on, exists_to_in=on, orderby_uses_equalities=on, condition_pushdown_for_derived=on
MariaDB 10.1.15	index_merge=on, index_merge_union=on, index_merge_sort_union=on, index_merge_intersection=on, index_merge_sort_intersection=off, engine_condition_pushdown=off, index_condition_pushdown=on, derived_merge=on, derived_with_keys=on, firstmatch=on, loosescan=on, materialization=on, in_to_exists=on, semijoin=on, partial_match_rowid_merge=on, partial_match_table_scan=on, subquery_cache=on, mrr=off, mrr_cost_based=off, mrr_sort_keys=off, outer_join_with_cache=on, semijoin_with_cache=on, join_cache_incremental=on, join_cache_hashed=on, join_cache_bka=on, optimize_join_buffer_size=off, table_elimination=on, extended_keys=on, exists_to_in=on, orderby_uses_equalities=off
MariaDB 10.0.12	index_merge=on, index_merge_union=on, index_merge_sort_union=on, index_merge_intersection=on, index_merge_sort_intersection=off, engine_condition_pushdown=off, index_condition_pushdown=on, derived_merge=on, derived_with_keys=on, firstmatch=on, loosescan=on, materialization=on, in_to_exists=on, semijoin=on, partial_match_rowid_merge=on, partial_match_table_scan=on, subquery_cache=on, mrr=off, mrr_cost_based=off, mrr_sort_keys=off, outer_join_with_cache=on, semijoin_with_cache=on, join_cache_incremental=on, join_cache_hashed=on, join_cache_bka=on, optimize_join_buffer_size=off, table_elimination=on, extended_keys=on, exists_to_in=on
MariaDB 10.0.9	index_merge=on, index_merge_union=on, index_merge_sort_union=on, index_merge_intersection=on, index_merge_sort_intersection=off, engine_condition_pushdown=off, index_condition_pushdown=on, derived_merge=on, derived_with_keys=on, firstmatch=on, loosescan=on, materialization=on, in_to_exists=on, semijoin=on, partial_match_rowid_merge=on, partial_match_table_scan=on, subquery_cache=on, mrr=off, mrr_cost_based=off, mrr_sort_keys=off, outer_join_with_cache=on, semijoin_with_cache=on, join_cache_incremental=on, join_cache_hashed=on, join_cache_bka=on, optimize_join_buffer_size=off, table_elimination=on, extended_keys=on, exists_to_in=off
MariaDB 10.0.2	index_merge=on, index_merge_union=on, index_merge_sort_union=on, index_merge_intersection=on, index_merge_sort_intersection=off, engine_condition_pushdown=off, index_condition_pushdown=on, derived_merge=on, derived_with_keys=on, firstmatch=on, loosescan=on, materialization=on, in_to_exists=on, semijoin=on, partial_match_rowid_merge=on, partial_match_table_scan=on, subquery_cache=on, mrr=off, mrr_cost_based=off, mrr_sort_keys=off, outer_join_with_cache=on, semijoin_with_cache=on, join_cache_incremental=on, join_cache_hashed=on, join_cache_bka=on, optimize_join_buffer_size=off, table_elimination=on, extended_keys=off, exists_to_in=off
MariaDB 5.5.21	index_merge=on, index_merge_union=on, index_merge_sort_union=on, index_merge_intersection=on, index_merge_sort_intersection=off, engine_condition_pushdown=off, index_condition_pushdown=on, derived_merge=on, derived_with_keys=on, firstmatch=on, loosescan=on, materialization=on, in_to_exists=on, semijoin=on, partial_match_rowid_merge=on, partial_match_table_scan=on, subquery_cache=on, mrr=off, mrr_cost_based=off, mrr_sort_keys=off, outer_join_with_cache=on, semijoin_with_cache=on, join_cache_incremental=on, join_cache_hashed=on, join_cache_bka=on, optimize_join_buffer_size=off, table_elimination=on, extended_keys=off
MariaDB 5.5.20	index_merge=on, index_merge_union=on, index_merge_sort_union=on, index_merge_intersection=on, index_merge_sort_intersection=off, engine_condition_pushdown=off, index_condition_pushdown=on, derived_merge=on, derived_with_keys=on, firstmatch=on, loosescan=on, materialization=on, in_to_exists=on, semijoin=on, partial_match_rowid_merge=on, partial_match_table_scan=on, subquery_cache=on, mrr=off, mrr_cost_based=off, mrr_sort_keys=off, outer_join_with_cache=on, semijoin_with_cache=on, join_cache_incremental=on, join_cache_hashed=on, join_cache_bka=on, optimize_join_buffer_size=off, table_elimination=on
MariaDB 5.3.3	index_merge=on, index_merge_union=on, index_merge_sort_union=on, index_merge_intersection=on, index_merge_sort_intersection=off, index_condition_pushdown=on, derived_merge=on, derived_with_keys=on, firstmatch=on, loosescan=on, materialization=on, in_to_exists=on, semijoin=on, partial_match_rowid_merge=on, partial_match_table_scan=on, subquery_cache=on, mrr=off, mrr_cost_based=off, mrr_sort_keys=off, outer_join_with_cache=on, semijoin_with_cache=on, join_cache_incremental=on, join_cache_hashed=on, join_cache_bka=on, optimize_join_buffer_size=off, table_elimination=on

MariaDB 5.3.0	index_merge=on, index_merge_union=on, index_merge_sort_union=on, index_merge_intersection=on, index_merge_sort_intersection=off, index_condition_pushdown=off, derived_merge=off, derived_with_keys=off, firstmatch=off, loosescan=off, materialization=off, in_to_exists=on, semijoin=off, partial_match_rowid_merge=on, partial_match_table_scan=on, subquery_cache=off, mrr=off, mrr_cost_based=off, mrr_sort_keys=off, outer_join_with_cache=off, semijoin_with_cache=off, join_cache_incremental=on, join_cache_hashed=on, join_cache_bka=on, optimize_join_buffer_size=off, table_elimination=on
< MariaDB 5.3.0	index_merge=on, index_merge_union=on, index_merge_sort_union=on, index_merge_intersection=on

3.3.4.13 Extended Keys

Syntax

Enable:

```
set optimizer_switch='extended_keys=on';
```

Disable:

```
set optimizer_switch='extended_keys=off';
```

Description

Extended Keys is an optimization set with the optimizer_switch system variable, which makes use of existing components of InnoDB keys to generate more efficient execution plans. Using these components in many cases allows the server to generate execution plans which employ index-only look-ups. It is set by default.

Extended keys can be used with:

- ref and eq-ref accesses
- range scans
- index-merge scans
- loose scans
- min/max optimizations

Examples

An example of how extended keys could be employed for a query built over a DBT-3/TPC-H database with one added index defined on `p_retailprice`:

```
select o_orderkey
from part, lineitem, orders
where p_retailprice > 2095 and o_orderdate='1992-07-01'
      and o_orderkey=l_orderkey and p_partkey=l_partkey;
```

The above query asks for the `orderkeys` of the orders placed on 1992-07-01 which contain parts with a retail price greater than $2095.

Using Extended Keys, the query could be executed by the following execution plan:

1. Scan the entries of the index `i_p_retailprice` where `p_retailprice>2095` and read `p_partkey` values from the extended keys.
2. For each value `p_partkey` make an index look-up into the table lineitem employing index `i_l_partkey` and fetch the values of `l_orderkey` from the extended index.
3. For each fetched value of `l_orderkey`, append it to the date `'1992-07-01'` and use the resulting key for an index look-up by index `i_o_orderdate` to fetch the values of `o_orderkey` from the found index entries.

All access methods of this plan do not touch table rows, which results in much better performance.

Here is the explain output for the above query:

```
MariaDB [dbt3sf10]> explain
   -> select o_orderkey
   ->   from part, lineitem, orders
   ->   where p_retailprice > 2095 and o_orderdate='1992-07-01'
   ->         and o_orderkey=l_orderkey and p_partkey=l_partkey\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: part
         type: range
possible_keys: PRIMARY,i_p_retailprice
          key: i_p_retailprice
      key_len: 9
          ref: NULL
         rows: 100
        Extra: Using where; Using index
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: lineitem
         type: ref
possible_keys: PRIMARY,i_l_suppkey_partkey,i_l_partkey,i_l_orderkey,i_l_orderkey_quantity
          key: i_l_partkey
      key_len: 5
          ref: dbt3sf10.part.p_partkey
         rows: 15
        Extra: Using index
*************************** 3. row ***************************
           id: 1
  select_type: SIMPLE
        table: orders
         type: ref
possible_keys: PRIMARY,i_o_orderdate
          key: i_o_orderdate
      key_len: 8
          ref: const,dbt3sf10.lineitem.l_orderkey
         rows: 1
        Extra: Using index
3 rows in set (0.00 sec)
```

3.3.4.14 How to Quickly Insert Data Into MariaDB

Contents

This article describes different techniques for inserting data quickly into MariaDB.

Background

When inserting new data into MariaDB, the things that take time are: (in order of importance):

- Syncing data to disk (as part of the end of transactions)
- Adding new keys. The larger the index, the more time it takes to keep keys updated.
- Checking against foreign keys (if they exist).
- Adding rows to the storage engine.
- Sending data to the server.

The following describes the different techniques (again, in order of importance) you can use to quickly insert data into a table.

Disabling Keys

You can temporarily disable updating of non unique indexes. This is mostly useful when there are zero (or very few) rows in the table into which you are inserting data.

```
ALTER TABLE table_name DISABLE KEYS;
BEGIN;
... inserting data with INSERT or LOAD DATA ....
COMMIT;
ALTER TABLE table_name ENABLE KEYS;
```

In many storage engines (at least MyISAM and Aria), `ENABLE KEYS` works by scanning through the row data and collecting keys, sorting them, and then creating the index blocks. This is an order of magnitude faster than creating the index one row at a time and it also uses less key buffer memory.

> **Note:** When you insert into an **empty table** with INSERT or LOAD DATA, MariaDB **automatically** does an DISABLE KEYS before and an ENABLE KEYS afterwards.

When inserting big amounts of data, integrity checks are sensibly time-consuming. It is possible to disable the `UNIQUE` indexes and the foreign keys checks using the unique_checks and the foreign_key_checks system variables:

```
SET @@session.unique_checks = 0;
SET @@session.foreign_key_checks = 0;
```

For InnoDB tables, the AUTO_INCREMENT lock mode can be temporarily set to 2, which is the fastest setting:

```
SET @@global.innodb_autoinc_lock_mode = 2;
```

Also, if the table has INSERT triggers or PERSISTENT columns, you may want to drop them, insert all data, and recreate them.

Loading Text Files

The **fastest way** to insert data into MariaDB is through the LOAD DATA INFILE command.

The simplest form of the command is:

```
LOAD DATA INFILE 'file_name' INTO TABLE table_name;
```

You can also read a file locally on the machine where the client is running by using:

```
LOAD DATA LOCAL INFILE 'file_name' INTO TABLE table_name;
```

This is not as fast as reading the file on the server side, but the difference is not that big.

`LOAD DATA INFILE` is very fast because:

1. there is no parsing of SQL.
2. data is read in big blocks.
3. if the table is empty at the beginning of the operation, all non unique indexes are disabled during the operation.
4. the engine is told to cache rows first and then insert them in big blocks (At least MyISAM and Aria support this).
5. for empty tables, some transactional engines (like Aria) do not log the inserted data in the transaction log because one can rollback the operation by just doing a TRUNCATE on the table.

Because of the above speed advantages there are many cases, when you need to insert **many** rows at a time, where it may be faster to create a file locally, add the rows there, and then use `LOAD DATA INFILE` to load them; compared to using `INSERT` to insert the rows.

You will also get progress reporting for `LOAD DATA INFILE` .

mariadb-import

You can import many files in parallel with mariadb-import (`mysqlimport` before MariaDB 10.5). For example:

```
mariadb-import --use-threads=10 database text-file-name [text-file-name...]
```

Internally mariadb-import uses LOAD DATA INFILE to read in the data.

Inserting Data with INSERT Statements

Using Big Transactions

When doing many inserts in a row, you should wrap them with `BEGIN / END` to avoid doing a full transaction (which includes a disk sync) for every row. For example, doing a begin/end every 1000 inserts will speed up your inserts by almost 1000 times.

```
BEGIN;
INSERT ...
INSERT ...
END;
BEGIN;
INSERT ...
INSERT ...
END;
...
```

The reason why you may want to have many `BEGIN/END` statements instead of just one is that the former will use up less transaction log space.

Multi-Value Inserts

You can insert many rows at once with multi-value row inserts:

```
INSERT INTO table_name values(1,"row 1"),(2, "row 2"),...;
```

The limit for how much data you can have in one statement is controlled by the max_allowed_packet server variable.

Inserting Data Into Several Tables at Once

If you need to insert data into several tables at once, the best way to do so is to enable multi-row statements and send many inserts to the server at once:

```
INSERT INTO table_name_1 (auto_increment_key, data) VALUES (NULL,"row 1");
INSERT INTO table_name_2 (auto_increment, reference, data) values (NULL, LAST_INSERT_ID(), "row 2
```

LAST_INSERT_ID() is a function that returns the last `auto_increment` value inserted.

By default, the command line `mariadb` client will send the above as multiple statements.

To test this in the `mariadb` client you have to do:

```
delimiter ;;
select 1; select 2;;
delimiter ;
```

Note: for multi-query statements to work, your client must specify the `CLIENT_MULTI_STATEMENTS` flag to `mysql_real_connect()`.

Server Variables That Can be Used to Tune Insert Speed

Option	Description
innodb_buffer_pool_size	Increase this if you have many indexes in InnoDB/XtraDB tables

key_buffer_size	Increase this if you have many indexes in MyISAM tables
max_allowed_packet	Increase this to allow bigger multi-insert statements
read_buffer_size	Read block size when reading a file with `LOAD DATA`

See Server System Variables for the full list of server variables.

3.3.4.15 Index Condition Pushdown

Contents

Index Condition Pushdown is an optimization that is applied for access methods that access table data through indexes: `range`, `ref`, `eq_ref`, `ref_or_null`, and Batched Key Access.

The idea is to check part of the WHERE condition that refers to index fields (we call it *Pushed Index Condition*) as soon as we've accessed the index. If the *Pushed Index Condition* is not satisfied, we won't need to read the whole table record.

Index Condition Pushdown is **on** by default. To disable it, set its optimizer_switch flag like so:

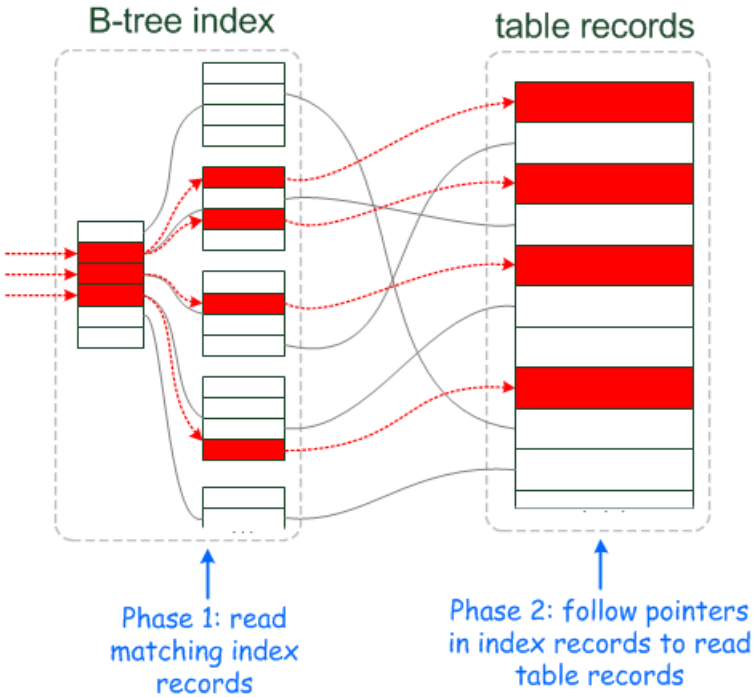
```
SET optimizer_switch='index_condition_pushdown=off'
```

When Index Condition Pushdown is used, EXPLAIN will show "Using index condition":

```
MariaDB [test]> explain select * from tbl where key_col1 between 10 and 11 and key_col2 like
'%foo%';
+----+-------------+-------+-------+---------------+----------+---------+------+------+------
-----------------+
| id | select_type | table | type  | possible_keys | key      | key_len | ref  | rows | Extra
|
+----+-------------+-------+-------+---------------+----------+---------+------+------+------
-----------------+
|  1 | SIMPLE      | tbl   | range | key_col1      | key_col1 | 5       | NULL |    2 | Using
index condition |
+----+-------------+-------+-------+---------------+----------+---------+------+------+------
-----------------+
```

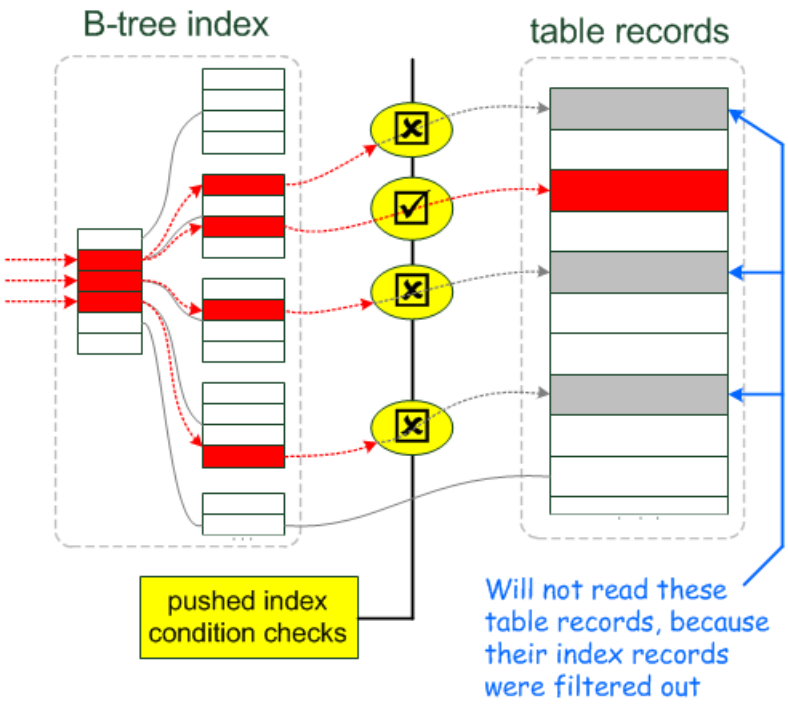
The Idea Behind Index Condition Pushdown

In disk-based storage engines, making an index lookup is done in two steps, like shown on the picture:



Index Condition Pushdown optimization tries to cut down the number of full record reads by checking whether index records

satisfy part of the WHERE condition that can be checked for them:



How much speed will be gained depends on - How many records will be filtered out - How expensive it was to read them

The former depends on the query and the dataset. The latter is generally bigger when table records are on disk and/or are big, especially when they have blobs.

Example Speedup

I used DBT-3 benchmark data, with scale factor=1. Since the benchmark defines very few indexes, we've added a multi-column index (index condition pushdown is usually useful with multi-column indexes: the first component(s) is what index access is done for, the subsequent have columns that we read and check conditions on).

```
alter table lineitem add index s_r (l_shipdate, l_receiptdate);
```

The query was to find big (l_quantity > 40) orders that were made in January 1993 that took more than 25 days to ship:

```
select count(*) from lineitem
where
  l_shipdate between '1993-01-01' and '1993-02-01' and
  datediff(l_receiptdate,l_shipdate) > 25 and
  l_quantity > 40;
```

EXPLAIN without Index Condition Pushdown:

```
-+----------+-------+----------------------+-----+---------+------+--------+-------------+
 | table    | type | possible_keys         | key | key_len | ref | rows    | Extra       |
-+----------+-------+----------------------+-----+---------+------+--------+-------------+
 | lineitem | range | s_r                   | s_r | 4       | NULL | 152064 | Using where |
-+----------+-------+----------------------+-----+---------+------+--------+-------------+
```

with Index Condition Pushdown:

```
-+----------+-------+---------------+-----+---------+------+--------+-----------------------
-------------+
 | table     | type | possible_keys | key | key_len | ref | rows     | Extra
|
-+----------+-------+---------------+-----+---------+------+--------+-----------------------
-------------+
 | lineitem | range | s_r           | s_r | 4       | NULL | 152064 | Using index condition;
Using where |
-+----------+-------+---------------+-----+---------+------+--------+-----------------------
-------------+
```

The speedup was:

- Cold buffer pool: from 5 min down to 1 min
- Hot buffer pool: from 0.19 sec down to 0.07 sec

Status Variables

There are two server status variables:

Variable name	Meaning
Handler_icp_attempts	Number of times pushed index condition was checked.
Handler_icp_match	Number of times the condition was matched.

That way, the value `Handler_icp_attempts - Handler_icp_match` shows the number records that the server did not have to read because of Index Condition Pushdown.

3.3.4.16 Query Limits and Timeouts

This article describes the different methods MariaDB provides to limit/timeout a query:

Contents

LIMIT

```
SELECT ... LIMIT row_count
or
SELECT ... LIMIT offset, row_count
or
SELECT ... LIMIT row_count OFFSET offset
```

The LIMIT clause restricts the number of returned rows.

LIMIT ROWS EXAMINED

```
SELECT ... LIMIT ROWS EXAMINED rows_limit;
```

Stops the query after 'rows_limit' number of rows have been examined.

sql_safe_updates

If the sql_safe_updates variable is set, one can't execute an UPDATE or DELETE statement unless one specifies a key constraint in the `WHERE` clause or provide a `LIMIT` clause (or both).

```
SET @@SQL_SAFE_UPDATES=1
UPDATE tbl_name SET not_key_column=val;
-> ERROR 1175 (HY000): You are using safe update mode
  and you tried to update a table without a WHERE that uses a KEY column
```

sql_select_limit

sql_select_limit acts as an automatic `LIMIT row_count` to any SELECT query.

```
SET @@SQL_SELECT_LIMIT=1000
SELECT * from big_table;
```

The above is the same as:

```
SELECT * from big_table LIMIT 1000;
```

max_join_size

If the max_join_size variable (also called `sql_max_join_size`) is set, then it will limit any SELECT statements that probably need to examine more than `MAX_JOIN_SIZE` rows.

```
SET @@MAX_JOIN_SIZE=1000;
SELECT count(null_column) from big_table;
->ERROR 1104 (42000): The SELECT would examine more than MAX_JOIN_SIZE rows;
  check your WHERE and use SET SQL_BIG_SELECTS=1 or SET MAX_JOIN_SIZE=# if the SELECT is okay
```

max_statement_time

If the max_statement_time variable is set, any query (excluding stored procedures) taking longer than the value of `max_statement_time` (specified in seconds) to execute will be aborted. This can be set globally, by session, as well as per user and per query. See Aborting statements that take longer than a certain time to execute.

3.3.4.17 Aborting Statements that Exceed a Certain Time to Execute

Contents

Overview

MariaDB 10.1.1 introduced the max_statement_time system variable. When set to a non-zero value, any queries taking longer than this time in seconds will be aborted. The default is zero, and no limits are then applied. The aborted query has no effect on any larger transaction or connection contexts. The variable is of type double, thus you can use subsecond timeout. For example you can use value 0.01 for 10 milliseconds timeout.

The value can be set globally or per session, as well as per user or per query (see below). Replicas are not affected by this variable, however from MariaDB 10.10, there is slave_max_statement_time which serves the same purpose on replicas only.

An associated status variable, max_statement_time_exceeded, stores the number of queries that have exceeded the execution time specified by max_statement_time, and a `MAX_STATEMENT_TIME_EXCEEDED` column was added to the CLIENT_STATISTICS and USER STATISTICS Information Schema tables.

The feature was based upon a patch by Davi Arnaut.

User max_statement_time

max_statement_time can be stored per user with the GRANT ... MAX_STATEMENT_TIME syntax.

Per-query max_statement_time

By using max_statement_time in conjunction with SET STATEMENT, it is possible to limit the execution time of individual queries. For example:

```
SET STATEMENT max_statement_time=100 FOR
  SELECT field1 FROM table_name ORDER BY field1;
```

Limitations

- max_statement_time does not work in embedded servers.
- max_statement_time does not work for COMMIT statements in a Galera cluster (see MDEV-18673 for discussion).

Differences Between the MariaDB and MySQL Implementations

MySQL 5.7.4 introduced similar functionality, but the MariaDB implementation differs in a number of ways.

- The MySQL version of max_statement_time (`max_execution_time`) is defined in millseconds, not seconds
- MySQL's implementation can only kill SELECTs, while MariaDB's can kill any queries (excluding stored procedures).
- MariaDB only introduced the max_statement_time_exceeded status variable, while MySQL also introduced a number of other variables which were not seen as necessary in MariaDB.
- The `SELECT MAX_STATEMENT_TIME = N ...` syntax is not valid in MariaDB. In MariaDB one should use `SET STATEMENT MAX_STATEMENT_TIME=N FOR....`

2.5.3 Partition Pruning and Selection

3.3.4.19 Big DELETEs

Contents

The problem

How to DELETE lots of rows from a large table? Here is an example of purging items older than 30 days:

```
DELETE FROM tbl WHERE
  ts < CURRENT_DATE() - INTERVAL 30 DAY
```

If there are millions of rows in the table, this statement may take minutes, maybe hours.

Any suggestions on how to speed this up?

Why it is a problem

- MyISAM will lock the table during the entire operation, thereby nothing else can be done with the table.
- InnoDB won't lock the table, but it will chew up a lot of resources, leading to sluggishness.
- InnoDB has to write the undo information to its transaction logs; this significantly increases the I/O required.
- Replication, being asynchronous, will effectively be delayed (on Slaves) while the DELETE is running.

InnoDB and undo

To be ready for a crash, a transactional engine such as InnoDB will record what it is doing to a log file. To make that somewhat less costly, the log file is sequentially written. If the log files you have (there are usually 2) fill up because the delete is really big, then the undo information spills into the actual data blocks, leading to even more I/O.

Deleting in chunks avoids some of this excess overhead.

Limited benchmarking of total delete elapsed time show two observations:

- Total delete time approximately doubles above some 'chunk' size (as opposed to below that threshold). I do not have a formula relating the log file size with the threshold cutoff.
- Chunk size below several hundred rows is slower. This is probably because the overhead of starting/ending each

chunk dominates the timing.

Solutions

- PARTITION -- Requires some careful setup, but is excellent for purging a time-base series.
- DELETE in chunks -- Carefully walk through the table N rows at a time.

PARTITION

The idea here is to have a sliding window of partitions. Let's say you need to purge news articles after 30 days. The "partition key" would be the datetime (or timestamp) that is to be used for purging, and the PARTITIONs would be "range". Every night, a cron job would come along and build a new partition for the next day, and drop the oldest partition.

Dropping a partition is essentially instantaneous, much faster than deleting that many rows. However, you must design the table so that the entire partition can be dropped. That is, you cannot have some items living longer than others.

PARTITION tables have a lot of restrictions, some are rather weird. You can either have no UNIQUE (or PRIMARY) key on the table, or every UNIQUE key must include the partition key. In this use case, the partition key is the datetime. It should not be the first part of the PRIMARY KEY (if you have a PRIMARY KEY).

You can PARTITION InnoDB or MyISAM tables.

Since two news articles could have the same timestamp, you cannot assume the partition key is sufficient for uniqueness of the PRIMARY KEY, so you need to find something else to help with that.

Reference implementation for Partition maintenance

MariaDB docs on PARTITION

Deleting in chunks

Although the discussion in this section talks about DELETE, it can be used for any other "chunking", such as, say, UPDATE, or SELECT plus some complex processing.

(This discussion applies to both MyISAM and InnoDB.)

When deleting in chunks, be sure to avoid doing a table scan. The code below is good at that; it scans no more than 1001 rows in any one query. (The 1000 is tunable.)

Assuming you have news articles that need to be purged, and you have a schema something like

```
CREATE TABLE tbl
   id INT UNSIGNED NOT NULL AUTO_INCREMENT,
   ts TIMESTAMP,
   ...
   PRIMARY KEY(id)
```

Then, this pseudo-code is a good way to delete the rows older than 30 days:

```
@a = 0
LOOP
   DELETE FROM tbl
      WHERE id BETWEEN @a AND @a+999
        AND ts < DATE_SUB(CURRENT_DATE(), INTERVAL 30 DAY)
   SET @a = @a + 1000
   sleep 1  -- be a nice guy
UNTIL end of table
```

Notes (Most of these caveats will be covered later):

- It uses the PK instead of the secondary key. This gives much better locality of disk hits, especially for InnoDB.
- You could (should?) do something to avoid walking through recent days but doing nothing. Caution -- the code for this could be costly.
- The 1000 should be tweaked so that the DELETE usually takes under, say, one second.
- No INDEX on ts is needed. (This helps INSERTs a little.)
- If your PRIMARY KEY is compound, the code gets messier.
- This code will not work without a numeric PRIMARY or UNIQUE key.
- Read on, we'll develop messier code to deal with most of these caveats.

If there are big gaps in `id` values (and there will after the first purge), then

```
@a = SELECT MIN(id) FROM tbl
LOOP
   SELECT @z := id FROM tbl WHERE id >= @a ORDER BY id LIMIT 1000,1
   If @z is null
      exit LOOP  -- last chunk
   DELETE FROM tbl
      WHERE id >= @a
        AND id <  @z
        AND ts < DATE_SUB(CURRENT_DATE(), INTERVAL 30 DAY)
   SET @a = @z
   sleep 1  -- be a nice guy, especially in replication
ENDLOOP
# Last chunk:
DELETE FROM tbl
   WHERE id >= @a
     AND ts < DATE_SUB(CURRENT_DATE(), INTERVAL 30 DAY)
```

That code works whether id is numeric or character, and it mostly works even if id is not UNIQUE. With a non-unique key, the risk is that you could be caught in a loop whenever @z==@a. That can be detected and fixed thus:

```
...
   SELECT @z := id FROM tbl WHERE id >= @a ORDER BY id LIMIT 1000,1
   If @z == @a
      SELECT @z := id FROM tbl WHERE id > @a ORDER BY id LIMIT 1
...
```

The drawback is that there could be more than 1000 items with a single id. In most practical cases, that is unlikely.

If you do not have a primary (or unique) key defined on the table, and you have an INDEX on ts, then consider

```
LOOP
   DELETE FROM tbl
      WHERE ts < DATE_SUB(CURRENT_DATE(), INTERVAL 30 DAY)
      ORDER BY ts   -- to use the index, and to make it deterministic
      LIMIT 1000
UNTIL no rows deleted
```

This technique is NOT recommended because the LIMIT leads to a warning on replication about it being non-deterministic (discussed below).

InnoDB chunking recommendation

- Have a 'reasonable' size for innodb_log_file_size.
- Use AUTOCOMMIT=1 for the session doing the deletions.
- Pick about 1000 rows for the chunk size.
- Adjust the row count down if asynchronous replication (Statement Based) causes too much delay on the Slaves or hogs the table too much.

Iterating through a compound key

To perform the chunked deletes recommended above, you need a way to walk through the PRIMARY KEY. This can be difficult if the PK has more than one column in it.

To efficiently to do compound 'greater than':

Assume that you left off at ($g, $s) (and have handled that row):

```
INDEX(Genus, species)
SELECT/DELETE ...
   WHERE Genus >= '$g' AND ( species  > '$s' OR Genus > '$g' )
   ORDER BY Genus, species
   LIMIT ...
```

Addenda: The above AND/OR works well in older versions of MySQL; this works better in MariaDB and newer versions of MySQL:

```
      WHERE ( Genus = '$g' AND species  > '$s' ) OR Genus > '$g' )
```

A caution about using @variables for strings. If, instead of '$g', you use @g, you need to be careful to make sure that @g has the same CHARACTER SET and COLLATION as `Genus`, else there could be a charset/collation conversion on the fly that prevents the use of the INDEX. Using the INDEX is vital for performance. It may require a COLLATE clause on SET NAMES and/or the @g in the SELECT.

Reclaiming the disk space

This is costly. (Switch to the PARTITION solution if practical.)

MyISAM leaves gaps in the table (.MYD file); OPTIMIZE TABLE will reclaim the freed space after a big delete. But it may take a long time and lock the table.

InnoDB is block-structured, organized in a BTree on the PRIMARY KEY. An isolated deleted row leaves a block less full. A lot of deleted rows can lead to coalescing of adjacent blocks. (Blocks are normally 16KB - see innodb_page_size.)

In InnoDB, there is no practical way to reclaim the freed space from ibdata1, other than to reuse the freed blocks eventually.

The only option with innodb_file_per_table = 0 is to dump ALL tables, remove ibdata*, restart, and reload. That is rarely worth the effort and time.

InnoDB, even with innodb_file_per_table = 1, won't give space back to the OS, but at least it is only one table to rebuild with. In this case, something like this should work:

```
CREATE TABLE new LIKE main;
INSERT INTO new SELECT * FROM main;  -- This could take a long time
RENAME TABLE main TO old, new TO main;   -- Atomic swap
DROP TABLE old;   -- Space freed up here
```

You do need enough disk space for both copies. You must not write to the table during the process.

Deleting more than half a table

The following technique can be used for any combination of

- Deleting a large portion of the table more efficiently
- Add PARTITIONing
- Converting to innodb_file_per_table = ON
- Defragmenting

This can be done by chunking, or (if practical) all at once:

```
-- Optional:  SET GLOBAL innodb_file_per_table = ON;
CREATE TABLE New LIKE Main;
-- Optional:  ALTER TABLE New ADD PARTITION BY RANGE ...;
-- Do this INSERT..SELECT all at once, or with chunking:
INSERT INTO New
   SELECT * FROM Main
      WHERE ...;  -- just the rows you want to keep
RENAME TABLE main TO Old, New TO Main;
DROP TABLE Old;   -- Space freed up here
```

Notes:

- You do need enough disk space for both copies.
- You must not write to the table during the process. (Changes to Main may not be reflected in New.)

Non-deterministic replication

Any UPDATE, DELETE, etc with LIMIT that is replicated to slaves (via Statement Based Replication) *may* cause inconsistencies between the Master and Slaves. This is because the actual order of the records discovered for updating/deleting may be different on the slave, thereby leading to a different subset being modified. To be safe, add ORDER BY to such statements. Moreover, be sure the ORDER BY is deterministic -- that is, the fields/expressions in the ORDER BY are unique.

An example of an ORDER BY that does not quite work: Assume there are multiple rows for each 'date':

```
DELETE * FROM tbl ORDER BY date LIMIT 111
```

Given that id is the PRIMARY KEY (or UNIQUE), this will be safe:

```
DELETE * FROM tbl ORDER BY date, id LIMIT 111
```

Unfortunately, even with the ORDER BY, MySQL has a deficiency that leads to a bogus warning in mysqld.err. See Spurious "Statement is not safe to log in statement format." warnings

Some of the above code avoids this spurious warning by doing

```
SELECT @z := ... LIMIT 1000,1;   -- not replicated
DELETE ... BETWEEN @a AND @z;    -- deterministic
```

That pair of statements guarantees no more than 1000 rows are touched, not the whole table.

Replication and KILL

If you KILL a DELETE (or any? query) on the master in the middle of its execution, what will be replicated?

If it is InnoDB, the query should be rolled back. (Exceptions??)

In MyISAM, rows are DELETEd as the statement is executed, and there is no provision for ROLLBACK. Some of the rows will be deleted, some won't. You probably have no clue of how much was deleted. In a single server, simply run the delete again. The delete is put into the binlog, but with error 1317. Since replication is supposed to keep the master and slave in sync, and since it has no clue of how to do that, replication stops and waits for manual intervention. In a HA (High Available) system using replication, this is a minor disaster. Meanwhile, you need to go to each slave(s) and verify that it is stuck for this reason, then do

```
SET GLOBAL SQL_SLAVE_SKIP_COUNTER = 1;
START SLAVE;
```

Then (presumably) re-executing the DELETE will finish the aborted task.

(That is yet another reason to move all your tables from MyISAM to InnoDB.)

SBR vs RBR; Galera

TBD -- "Row Based Replication" may impact this discussion.

Postlog

The tips in this document apply to MySQL, MariaDB, and Percona.

3.3.4.20 Data Sampling: Techniques for Efficiently Finding a Random Row

Contents

Fetching random rows from a table (beyond ORDER BY RAND())

The problem

One would like to do "SELECT ... ORDER BY RAND() LIMIT 10" to get 10 rows at random. But this is slow. The optimizer does

- Fetch all the rows -- this is costly

- Append RAND() to the rows
- Sort the rows -- also costly
- Pick the first 10.

All the algorithms given below are "fast", but most introduce flaws:

- Bias -- some rows are more like to be fetched than others.
- Repetitions -- If two random sets contain the same row, they are likely to contain other dups.
- Sometimes failing to fetch the desired number of rows.

"Fast" means avoiding reading all the rows. There are many techniques that require a full table scan, or at least an index scan. They are not acceptable for this list. There is even a technique that averages half a scan; it is relegated to a footnote.

Metrics

Here's a way to measure performance without having a big table.

```
    FLUSH STATUS;
    SELECT ...;
    SHOW SESSION STATUS LIKE 'Handler%';
```

If some of the "Handler" numbers look like the number of rows in the table, then there was a table scan.

None of the queries presented here need a full table (or index) scan. Each has a time proportional to the number of rows returned.

Virtually all published algorithms involve a table scan. The previously published version of this blog had, embarrassingly, several algorithms that had table scans.

Sometimes the scan can be avoided via a subquery. For example, the first of these will do a table scan; the second will not.

```
SELECT *  FROM RandTest AS a
  WHERE id = FLOOR(@min + (@max - @min + 1) * RAND());  -- BAD: table scan
SELECT *
 FROM RandTest AS a
 JOIN (
   SELECT FLOOR(@min + (@max - @min + 1) * RAND()) AS id -- Good; single eval.
      ) b  USING (id);
```

Case: Consecutive AUTO_INCREMENT without gaps, 1 row returned

- Requirement: AUTO_INCREMENT id
- Requirement: No gaps in id

```
  SELECT r.*
      FROM (
          SELECT FLOOR(mm.min_id + (mm.max_id - mm.min_id + 1) * RAND()) AS id
              FROM (
                  SELECT MIN(id) AS min_id,
                         MAX(id) AS max_id
                      FROM RandTest
                   ) AS mm
           ) AS init
      JOIN  RandTest AS r  ON r.id = init.id;
```

(Of course, you might be able to simplify this. For example, min_id is likely to be 1. Or precalculate limits into @min and @max.)

Case: Consecutive AUTO_INCREMENT without gaps, 10 rows

- Requirement: AUTO_INCREMENT id
- Requirement: No gaps in id
- Flaw: Sometimes delivers fewer than 10 rows

```
-- First select is one-time:
SELECT @min := MIN(id),
       @max := MAX(id)
    FROM RandTest;
SELECT DISTINCT *
    FROM RandTest AS a
    JOIN (
        SELECT FLOOR(@min + (@max - @min + 1) * RAND()) AS id
            FROM RandTest
            LIMIT 11     -- more than 10 (to compensate for dups)
         ) b  USING (id)
    LIMIT 10;            -- the desired number of rows
```

The FLOOR expression could lead to duplicates, hence the inflated inner LIMIT. There could (rarely) be so many duplicates that the inflated LIMIT leads to fewer than the desired 10 different rows. One approach to that Flaw is to rerun the query if it delivers too few rows.

A variant:

```
SELECT r.*
    FROM (
        SELECT FLOOR(mm.min_id + (mm.max_id - mm.min_id + 1) * RAND()) AS id
            FROM (
                SELECT MIN(id) AS min_id,
                       MAX(id) AS max_id
                    FROM RandTest
                 ) AS mm
            JOIN ( SELECT id dummy FROM RandTest LIMIT 11 ) z
         ) AS init
    JOIN  RandTest AS r  ON r.id = init.id
    LIMIT 10;
```

Again, ugly but fast, regardless of table size.

Case: AUTO_INCREMENT with gaps, 1 or more rows returned

- Requirement: AUTO_INCREMENT, possibly with gaps due to DELETEs, etc
- Flaw: Only semi-random (rows do not have an equal chance of being picked), but it does partially compensate for the gaps
- Flaw: The first and last few rows of the table are less likely to be delivered.

This gets 50 "consecutive" ids (possibly with gaps), then delivers a random 10 of them.

```
-- First select is one-time:
SELECT @min := MIN(id),
       @max := MAX(id)
    FROM RandTest;
SELECT a.*
    FROM RandTest a
    JOIN ( SELECT id FROM
            ( SELECT id
                FROM ( SELECT @min + (@max - @min + 1 - 50) * RAND()
                  AS start FROM DUAL ) AS init
                JOIN RandTest y
                WHERE    y.id > init.start
                ORDER BY y.id
                LIMIT 50          -- Inflated to deal with gaps
            ) z ORDER BY RAND()
           LIMIT 10               -- number of rows desired (change to 1 if looking for a single r
         ) r ON a.id = r.id;
```

Yes, it is complex, but yes, it is fast, regardless of the table size.

Case: Extra FLOAT column for randomizing

(Unfinished: need to check these.)

Assuming `rnd` is a FLOAT (or DOUBLE) populated with RAND() and INDEXed:

- Requirement: extra, indexed, FLOAT column
- Flaw: Fetches 10 adjacent rows (according to `rnd`), hence not good randomness
- Flaw: Near 'end' of table, can't find 10 rows.

```
SELECT r.*
    FROM ( SELECT RAND() AS start FROM DUAL ) init
    JOIN RandTest r
    WHERE r.rnd >= init.start
    ORDER BY r.rnd
    LIMIT 10;
```

- These two variants attempt to resolve the end-of-table flaw:

```
SELECT r.*
    FROM ( SELECT RAND() * ( SELECT rnd
                    FROM RandTest
                    ORDER BY rnd DESC
                    LIMIT 10,1 ) AS start
         ) AS init
    JOIN RandTest r
    WHERE r.rnd > init.start
    ORDER BY r.rnd
    LIMIT 10;


SELECT @start := RAND(),
       @cutoff := CAST(1.1 * 10 + 5 AS DECIMAL(20,8)) / TABLE_ROWS
    FROM information_schema.TABLES
    WHERE TABLE_SCHEMA = 'dbname'
      AND TABLE_NAME = 'RandTest'; -- 0.0030
SELECT d.*
    FROM (
        SELECT a.id
            FROM RandTest a
            WHERE rnd BETWEEN @start AND @start + @cutoff
         ) sample
    JOIN RandTest d USING (id)
    ORDER BY rand()
    LIMIT 10;
```

Case: UUID or MD5 column

- Requirement: UUID/GUID/MD5/SHA1 column exists and is indexed.
- Similar code/benefits/flaws to AUTO_INCREMENT with gaps.
- Needs 7 random HEX digits:

```
RIGHT( HEX( (1<<24) * (1+RAND()) ), 6)
```

can be used as a `start` for adapting a gapped AUTO_INCREMENT case. If the field is BINARY instead of hex, then

```
UNHEX(RIGHT( HEX( (1<<24) * (1+RAND()) ), 6))
```

3.3.4.21 Data Warehousing High Speed Ingestion

Contents

The problem

You are ingesting lots of data. Performance is bottlenecked in the INSERT area.

This will be couched in terms of Data Warehousing, with a huge `Fact` table and Summary (aggregation) tables.

Overview of solution

- Have a separate staging table.
- Inserts go into `Staging`.
- Normalization and Summarization reads Staging, not Fact.
- After normalizing, the data is copied from Staging to Fact.

`Staging` is one (or more) tables in which the data lives only long enough to be handed off to Normalization, Summary, and the Fact tables.

Since we are probably talking about a billion-row table, shrinking the width of the Fact table by normalizing (as mentioned here). Changing an INT to a MEDIUMINT will save a GB. Replacing a string by an id (normalizing) saves many GB. This helps disk space and cacheability, hence speed.

Injection speed

Some variations:

- Big dump of data once an hour, versus continual stream of records.
- The input stream could be single-threaded or multi-threaded.
- You might have 3rd party software tying your hands.

Generally the fastest injection rate can be achieved by "staging" the INSERTs in some way, then batch processing the staged records. This blog discusses various techniques for staging and batch processing.

Normalization

Let's say your Input has a VARCHAR `host_name` column, but you need to turn that into a smaller MEDIUMINT `host_id` in the Fact table. The "Normalization" table, as I call it, looks something like

```
CREATE TABLE Hosts (
    host_id  MEDIUMINT UNSIGNED  NOT NULL AUTO_INCREMENT,
    host_name VARCHAR(99) NOT NULL,
    PRIMARY KEY (host_id),      -- for mapping one direction
    INDEX(host_name, host_id)   -- for mapping the other direction
) ENGINE=InnoDB;                -- InnoDB works best for Many:Many mapping table
```

Here's how you can use `Staging` as an efficient way achieve the swap from name to id.

Staging has two fields (for this normalization example):

```
    host_name VARCHAR(99) NOT NULL,     -- Comes from the insertion proces
    host_id  MEDIUMINT UNSIGNED  NULL,  -- NULL to start with; see code below
```

Meawhile, the Fact table has:

```
    host_id  MEDIUMINT UNSIGNED NOT NULL,
```

SQL #1 (of 2):

```
# This should not be in the main transaction, and it should be done with autocommit = ON
# In fact, it could lead to strange errors if this were part
#    of the main transaction and it ROLLBACKed.
INSERT IGNORE INTO Hosts (host_name)
    SELECT DISTINCT s.host_name
        FROM Staging AS s
        LEFT JOIN Hosts AS n  ON n.host_name = s.host_name
        WHERE n.host_id IS NULL;
```

By isolating this as its own transaction, we get it finished in a hurry, thereby minimizing blocking. By saying IGNORE, we don't care if other threads are 'simultaneously' inserting the same host_names.

There is a subtle reason for the LEFT JOIN. If, instead, it were INSERT IGNORE..SELECT DISTINCT, then the INSERT would preallocate auto_increment ids for as many rows as the SELECT provides. This is very likely to "burn" a lot of ids, thereby leading to overflowing MEDIUMINT unnecessarily. The LEFT JOIN leads to finding just the new ids that are needed (except for the rare possibility of a 'simultaneous' insert by another thread). More rationale: Mapping table

SQL #2:

```
# Also not in the main transaction, and it should be with autocommit = ON
# This multi-table UPDATE sets the ids in Staging:
UPDATE   Hosts AS n
    JOIN Staging AS s  ON s.host_name = n host_name
    SET s.host_id = n.host_id
```

This gets the IDs, whether already existing, set by another thread, or set by SQL #1.

If the size of `Staging` changes depending on the busy versus idle times of the day, this pair of SQL statements has another comforting feature. The more rows in `Staging`, the more efficient the SQL runs, thereby helping compensate for the "busy" times.

The companion Data Warehouse article folds SQL #2 into the INSERT INTO Fact. But you may need host_id for further normalization steps and/or Summarization steps, so this explicit UPDATE shown here is often better.

Flip-flop staging

The simple way to stage is to ingest for a while, then batch-process what is in `Staging`. But that leads to new records piling up waiting to be staged. To avoid that issue, have 2 processes:

- one process (or set of processes) for INSERTing into `Staging`;
- one process (or set of processes) to do the batch processing (normalization, summarization).

To keep the processes from stepping on each other, we have a pair of staging tables:

- `Staging` is being INSERTed into;
- `StageProcess` is one being processed for normalization, summarization, and moving to the Fact table. A separate process does the processing, then swaps the tables:

```
DROP   TABLE StageProcess;
CREATE TABLE StageProcess LIKE Staging;
RENAME TABLE Staging TO tmp, StageProcess TO Staging, tmp TO StageProcess;
```

This may not seem like the shortest way to do it, but has these features:

- The DROP + CREATE might be faster than TRUNCATE, which is the desired effect.
- The RENAME is atomic, so the INSERT process(es) never find that `Staging` is missing.

A variant on the 2-table flip-flop is to have a separate `Staging` table for each Insertion process. The Processing process would run around to each Staging in turn.

A variant on that would be to have a separate processing process for each Insertion process.

The choice depends on which is faster (insertion or processing). There are tradeoffs; a single processing thread avoids some locks, but lacks some parallelism.

Engine choice

`Fact` table -- InnoDB, if for no other reason than that a system crash would not need a REPAIR TABLE. (REPAIRing a billion-row MyISAM table can take hours or days.)

Normalization tables -- InnoDB, primarily because it can be done efficiently with 2 indexes, whereas, MyISAM would need 4 to achieve the same efficiency.

`Staging` -- Lots of options here.

- If you have multiple Inserters and a single Staging table, InnoDB is desirable due to row-level, not table-level, locking.
- MEMORY may be the fastest and it avoids I/O. This is good for a single staging table.
- For multiple Inserters, a separate Staging table for each Inserter is desired.
- For multiple Inserters into a single Staging table, InnoDB may be faster. (MEMORY does table-level locking.)
- With one non-InnoDB Staging table per Inserter, using an explicit LOCK TABLE avoids repeated implicit locks on each INSERT.
- But, if you are doing LOCK TABLE and the Processing thread is separate, an UNLOCK is necessary periodically to let the RENAME grab the table.
- "Batch INSERTs" (100-1000 rows per SQL) eliminates much of the issues of the above bullet items.

Confused? Lost? There are enough variations in applications that make it impractical to predict what is best. Or, simply good enough. Your ingestion rate may be low enough that you don't hit the brick walls that I am helping you avoid.

Should you do "CREATE TEMPORARY TABLE"? Probably not. Consider `Staging` as part of the data flow, not to be DROPped.

Summarization

This is mostly covered here: Summary Tables
Summarize from the Staging table instead of the Fact table.

Replication Issues

Row Based Replication (RBR) is probably the best option.

The following allows you to keep more of the Ingestion process in the Master, thereby not bogging down the Slave(s) with writes to the Staging table.

- RBR
- `Staging` is in a separate database
- That database is not replicated (binlog-ignore-db on Master)
- In the Processing steps, USE that database, reach into the main db via syntax like "MainDb.Hosts". (Otherwise, the binlog-ignore-db does the wrong thing.)

That way

- Writes to `Staging` are not replicated.
- Normalization sends only the few updates to the normalization tables.
- Summarization sends only the updates to the summary tables.
- Flip-flop does not replicate the DROP, CREATE or RENAME.

Sharding

You could possibly spread the data you are trying ingest across multiple machines in a predictable way (sharding on hash, range, etc). Running "reports" on a sharded Fact table is a challenge unto itself. On the other hand, Summary Tables rarely get too big to manage on a single machine.

For now, Sharding is beyond the scope of this blog.

Push me vs pull me

I have implicitly assumed the data is being pushed into the database. If, instead, you are "pulling" data from some source(s), then there are some different considerations.

Case 1: An hourly upload; run via cron

1. Grab the upload, parse it 2. Put it into the Staging table 3. Normalize -- each SQL in its own transaction (autocommit) 4. BEGIN 5. Summarize 6. Copy from Staging to Fact. 7. COMMIT

If you need parallelism in Summarization, you will have to sacrifice the transactional integrity of steps 4-7.

Caution: If these steps add up to more than an hour, you are in deep dodo.

Case 2: You are polling for the data

It is probably reasonable to have multiple processes doing this, so it will be detailed about locking.

0. Create a Staging table for this polling processor. Loop: 1. With some locked mechanism, decide which 'thing' to poll. 2. Poll for the data, pull it in, parse it. (Potentially polling and parsing are significantly costly) 3. Put it into the process-specific Staging table 4. Normalize -- each SQL in its own transaction (autocommit) 5. BEGIN 6. Summarize 7. Copy from Staging to Fact. 8. COMMIT 9. Declare that you are finished with this 'thing' (see step 1) EndLoop.

iblog_file_size should be larger than the change in the STATUS "Innodb_os_log_written" across the BEGIN...COMMIT transaction (for either Case).

3.3.4.22 Data Warehousing Summary Tables

Contents

Preface

This document discusses the creation and maintenance of "Summary Tables". It is a companion to the document on Data Warehousing Techniques.

The basic terminology ("Fact Table", "Normalization", etc) is covered in that document.

Summary tables for data warehouse "reports"

Summary tables are a performance necessity for large tables. MariaDB and MySQL do not provide any automated way to create such, so I am providing techniques here.

(Other vendors provide something similar with "materialized views".)

When you have millions or billions of rows, it takes a long time to summarize the data to present counts, totals, averages, etc, in a size that is readily digestible by humans. By computing and saving subtotals as the data comes in, one can make "reports" run much faster. (I have seen 10x to 1000x speedups.) The subtotals go into a "summary table". This document guides you on efficiency in both creating and using such tables.

General structure of a summary table

A summary table includes two sets of columns:

- Main KEY: date + some dimension(s)
- Subtotals: COUNT(*), SUM(...), ...; but not AVG()

The "date" might be a DATE (a 3-byte native datatype), or an hour, or some other time interval. A 3-byte MEDIUMINT UNSIGNED 'hour' can be derived from a DATETIME or TIMESTAMP via

```
FLOOR(UNIX_TIMESTAMP(dt) / 3600)
FROM_UNIXTIME(hour * 3600)
```

The "dimensions" (a DW term) are some of the columns of the "Fact" table. Examples: Country, Make, Product, Category, Host Non-dimension examples: Sales, Quantity, TimeSpent

There would be one or more indexes, usually starting with some dimensions and ending with the date field. By ending with the date, one can efficiently get a range of days/weeks/etc. even when each row summarizes only one day.

There will typically be a "few" summary tables. Often one summary table can serve multiple purposes sufficiently efficiently.

As a rule of thumb, a summary table will have one-tenth the number of rows as the Fact table. (This number is very loose.)

Example

Let's talk about a large chain of car dealerships. The Fact table has all the sales with columns such as datetime, salesman_id, city, price, customer_id, make, model, model_year. One Summary table might focus on sales:

```
    PRIMARY KEY(city, datetime),
    Aggregations: ct, sum_price

    # Core of INSERT..SELECT:
    DATE(datetime) AS date, city, COUNT(*) AS ct, SUM(price) AS sum_price

    # Reporting average price for last month, broken down by city:
    SELECT city,
           SUM(sum_price) / SUM(ct) AS 'AveragePrice'
       FROM SalesSummary
       WHERE datetime BETWEEN ...
       GROUP BY city;

    # Monthly sales, nationwide, from same summary table:
    SELECT MONTH(datetime) AS 'Month',
           SUM(ct)         AS 'TotalSalesCount'
           SUM(sum_price)  AS 'TotalDollars'
       FROM SalesSummary
       WHERE datetime BETWEEN ...
       GROUP BY MONTH(datetime);
    # This might benefit from a secondary INDEX(datetime)
```

When to augment the summary table(s)?

"Augment" in this section means to add new rows into the summary table or increment the counts in existing rows.

Plan A: "While inserting" rows into the Fact table, augment the summary table(s). This is simple, and workable for a smaller DW database (under 10 Fact table rows per second). For larger DW databases, Plan A likely to be too costly to be practical.

Plan B: "Periodically", via cron or an EVENT.

Plan C: "As needed". That is, when someone asks for a report, the code first updates the summary tables that will be needed.

Plan D: "Hybrid" of B and C. C, by itself, can led to long delays for the report. By also doing B, those delays can be kept low.

Plan E: (This is not advised.) "Rebuild" the entire summary table from the entire Fact table. The cost of this is prohibitive for large tables. However, Plan E may be needed when you decide to change the columns of a Summary Table, or discover a flaw in the computations. To lessen the impact of an entire build, adapt the chunking techniques in Deleting in chunks .

Plan F: "Staging table". This is primarily for very high speed ingestion. It is mentioned briefly in this blog, and discussed more thoroughly in the companion blog: High Speed Ingestion

Summarizing while Inserting (one row at a time)

```
    INSERT INTO Fact ...;
    INSERT INTO Summary (..., ct, foo, ...) VALUES (..., 1, foo, ...)
        ON DUPLICATE KEY UPDATE ct = ct+1, sum_foo = sum_foo + VALUES(foo), ...;
```

IODKU (Insert On Duplicate Key Update) will update an existing row or create a new row. It knows which to do based on the Summary table's PRIMARY KEY.

Caution: This approach is costly, and will not scale to an ingestion rate of over, say, 10 rows per second (Or maybe 50/second on SSDs). More discussion later.

Summarizing periodically vs as-needed

If your reports need to be up-to-the-second, you need "as needed" or "hybrid". If your reports have less urgency (eg, weekly reports that don't include 'today'), then "periodically" might be best.

For a daily summaries, augmenting the summary tables could be done right after midnight. But, beware of data coming "late".

For both "periodic" and "as needed", you need a definitive way of keeping track of where you "left off".

Case 1: You insert into the Fact table first and it has an AUTO_INCREMENT id: Grab MAX(id) as the upper bound for summarizing and put it either into some other secure place (an extra table), or put it into the row(s) in the Summary table as you insert them. (Caveat: AUTO_INCREMENT ids do not work well in multi-master, including Galera, setups.)

Case 2: If you are using a 'staging' table, there is no issue. (More on staging tables later.)

Summarizing while batch inserting

This applies to multi-row (batch) INSERT and LOAD DATA.

The Fact table needs an AUTO_INCREMENT id, and you need to be able to find the exact range of ids inserted. (This may be impractical in any multi-master setup.)

Then perform bulk summarization using

```
    FROM Fact
    WHERE id BETWEEN min_id and max_id
```

Summarizing when using a staging table

Load the data (via INSERTs or LOAD DATA) en masse into a "staging table". Then perform batch summarization from the Staging table. And batch copy from the Staging table to the Fact table. Note that the Staging table is handy for batching "normalization" during ingestion. See also [[data-warehousing-high-speed-ingestion|High Speed Ingestion

Summary table: PK or not?

Let's say your summary table has a DATE, `dy`, and a dimension, `foo`. The question is: Should (foo, dy) be the PRIMARY KEY? Or a non-UNIQUE index?

Case 1: PRIMARY KEY (foo, dy) and summarization is in lock step with, say, changes in `dy`.

This case is clean and simple -- until you get to endcases. How will you handle the case of data arriving 'late'? Maybe you will need to recalculate some chunks of data? If so, how?

Case 2: (foo, dy) is a non-UNIQUE INDEX.

This case is clean and simple, but it can clutter the summary table because multiple rows can occur for a given (foo, dy) pair. The report will always have to SUM() up values because it cannot assume there is only one row, even when it is reporting on a single `foo` for a single `dy`. This forced-SUM is not really bad -- you should do it anyway; that way all your reports are written with one pattern.

Case 3: PRIMARY KEY (foo, dy) and summarization can happen anytime.

Since you should be using InnoDB, there needs to be an explicit PRIMARY KEY. One approach when you do not have a 'natural' PK is this:

```
    id INT UNSIGNED AUTO_INCREMENT NOT NULL,
    ...
    PRIMARY KEY(foo, dy, id),   -- `id` added to make unique
    INDEX(id)                   -- sufficient to keep AUTO_INCREMENT happy
```

This case pushes the complexity onto the summarization by doing a IODKU.

Advice? Avoid Case 1; too messy. Case 2 is ok if the extra rows are not too common. Case 3 may be the closest to "once size fits all".

Averages, etc.

When summarizing, include `COUNT(*) AS ct and SUM(foo) AS sum_foo`. When reporting, the "average" is computed as SUM(sum_foo) / SUM(ct). That is mathematically correct.

Exception... Let's say you are looking at weather temperatures. And you monitoring station gets the temp periodically, but unreliably. That is, the number of readings for a day varies. Further, you decide that the easiest way to compensate for the inconsistency is to do something like: Compute the avg temp for each day, then average those across the month (or other timeframe).

Formula for Standard Deviation:

```
    SQRT( SUM(sum_foo2)/SUM(ct) - POWER(SUM(sum_foo)/SUM(ct), 2) )
```

Where sum_foo2 is SUM(foo * foo) from the summary table. sum_foo and sum_foo2 should be FLOAT. FLOAT gives you about 7 significant digits, which is more than enough for things like average and standard deviation. FLOAT occupies 4 bytes. DOUBLE would give you more precision, but occupies 8 bytes. INT and BIGINT are not practical because they may lead to complaints about overflow.

Staging table

The idea here is to first load a set of Fact records into a "staging table", with the following characteristics (at least):

- The table is repeatedly populated and truncated
- Inserts could be individual or batched, and from one or many clients
- SELECTs will be table scans, so no indexes needed
- Inserting will be fast (InnoDB may be the fastest)
- Normalization can be done in bulk, hence efficiently
- Copying to the Fact table will be fast
- Summarization can be done in bulk, hence efficiently
- "Bursty" ingestion is smoothed by this process
- Flip-flop a pair of Staging tables

If you have bulk inserts (Batch INSERT or LOAD DATA) then consider doing the normalization and summarization immediately after each bulk insert.

More details: High Speed Ingestion

Extreme design

Here is a more complex way to design the system, with the goal of even more scaling.

- Use master-slave setup: ingest into master; report from slave(s).
- Feed ingestion through a staging table (as described above)
- Single-source of data: ENGINE=MEMORY; multiple sources: InnoDB
- binlog_format = ROW
- Use binlog_ignore_db to avoid replicating staging -- necessitating putting it in a separate database.
- Do the summarization from Staging
- Load Fact via INSERT INTO Fact ... SELECT FROM Staging ...

Explanation and comments:

- ROW + ignore_db avoids replicating Staging, yet replicates the INSERTs based on it. Hence, it lightens the write load on the Slaves
- If using MEMORY, remember that it is volatile -- recover from a crash by starting the ingestion over.
- To aid with debugging, TRUNCATE or re-CREATE Staging at the start of the next cycle.
- Staging needs no indexes -- all operations read all rows from it.

Stats on the system that this 'extreme design' came from: Fact Table: 450GB, 100M rows/day (batch of 4M/hour), 60 day retention (60+24 partitions), 75B/row, 7 summary tables, under 10 minutes to ingest and summarize the hourly batch. The INSERT..SELECT handled over 20K rows/sec going into the Fact table. Spinning drives (not SSD) with RAID-10.

"Left Off"

One technique involves summarizing some of the data, then recording where you "left off", so that next time, you can start there. There are some subtle issues with "left off" that you should be cautious of.

If you use a DATETIME or TIMESTAMP as "left off", beware of multiple rows with the same value.

- Plan A: Use a compound "left off" (eg, TIMESTAMP + ID). This is messy, error prone, etc.
- Plan B: WHERE ts >= $left_off AND ts < $max_ts -- avoids dups, but has other problems (below)
- Separate threads could COMMIT TIMESTAMPs out of order.

If you use an AUTO_INCREMENT as "left off" beware of:

- In InnoDB, separate threads could COMMIT ids in the 'wrong' order.
- Multi-master (including Galera and InnoDB Cluster), could lead to ordering issues.

So, nothing works, at least not in a multi-threaded environment?

If you can live with an occasional hiccup (skipped record), then maybe this is 'not a problem' for you.

The "Flip-Flop Staging" is a safe alternative, optionally combined with the "Extreme Design".

Flip-flop staging

If you have many threads simultaneously INSERTing into one staging table, then here is an efficient way to handle a large load: Have a process that flips that staging table with another, identical, staging table, and performs bulk normalization, Fact insertion, and bulk summarization.

The flipping step uses a fast, atomic, RENAME.

Here is a sketch of the code:

```
    # Prep for flip:
    CREATE TABLE new LIKE Staging;

    # Swap (flip) Staging tables:
    RENAME TABLE Staging TO old, new TO Staging;

    # Normalize new `foo`s:
    # (autocommit = 1)
    INSERT IGNORE INTO Foos SELECT fpp FROM old LEFT JOIN Foos ...

    # Prep for possible deadlocks, etc
    while...
    START TRANSACTION;

    # Add to Fact:
    INSERT INTO Fact ... FROM old JOIN Foos ...

    # Summarize:
    INSERT INTO Summary ... FROM old ... GROUP BY ...

    COMMIT;
    end-while

    # Cleanup:
    DROP TABLE old;
```

Meanwhile, ingestion can continue writing to `Staging`. The ingestion INSERTs will conflict with the RENAME, but will be resolved gracefully and silently and quickly.

How fast should you flip-flop? Probably the best scheme is to

- Have a job that flip-flops in a tight loop (no delay, or a small delay, between iterations), and
- Have a CRON that serves only as a "keep-alive" to restart the job if it dies.

If Staging is 'big', an iteration will take longer, but run more efficiently. Hence, it is self-regulating.

In a Galera (or InnoDB Cluster?) environment, each node could be receiving input. If can afford to loose a few rows, have `Staging` be a non-replicated MEMORY table. Otherwise, have one `Staging` per node and be InnoDB; it will be more secure, but slower and not without problems. In particular, if a node dies completely, you somehow need to process its `Staging` table.

Multiple summary tables

- Look at the reports you will need.
- Design a summary table for each.
- Then look at the summary tables -- you are likely to find some similarities.
- Merge similar ones.

To look at what a report needs, look at the WHERE clause that would provide the data. Some examples, assuming data about service records for automobiles: The GROUP BY to gives a clue of what the report might be about.

1. WHERE make = ? AND model_year = ? GROUP BY service_date, service_type 2. WHERE make = ? AND model = ? GROUP BY service_date, service_type 3. WHERE service_type = ? GROUP BY make, model, service_date 4. WHERE service_date between ? and ? GROUP BY make, model, model_year

You need to allow for 'ad hoc' queries? Well, look at all the ad hoc queries -- they all have a date range, plus nail down one or two other things. (I rarely see something as ugly as '%CL%' for nailing down another dimension.) So, start by thinking of date plus one or two other dimensions as the 'key' into a new summary table. Then comes the question of what data might be desired -- counts, sums, etc. Eventually you have a small set of summary tables. Then build a front end to allow them to pick only from those possibilities. It should encourage use of the existing summary tables, not not be truly 'open ended'.

Later, another 'requirement' may surface. So, build another summary table. Of course, it may take a day to initially populate it.

Games on summary tables

Does one ever need to summarize a summary table? Yes, but only in extreme situations. Usually a 'weekly' report can be derived from a 'daily' summary table; building a separate weekly summary table not being worth the effort.

Would one ever PARTITION a Summary Table? Yes, in extreme situations, such as the table being large, and

- Need to purge old data (unlikely), or
- Recent' data is usually requested, and the index(es) fail to prevent table scans (rare). ("Partition pruning" to the rescue.)

3.3.4.23 Data Warehousing Techniques

Contents

Preface

This document discusses techniques for improving performance for data-warehouse-like tables in MariaDB and MySQL.

- How to load large tables.
- Normalization.
- Developing 'summary tables' to make 'reports' efficient.
- Purging old data.

Details on summary tables is covered in the companion document: Summary Tables.

Terminology

This list mirrors "Data Warehouse" terminology.

- Fact table -- The one huge table with the 'raw' data.
- Summary table -- a redundant table of summarized data that could -- use for efficiency
- Dimension -- columns that identify aspects of the dataset (region, country, user, SKU, zipcode, ...)
- Normalization table (dimension table) -- mapping between strings an ids; used for space and speed.
- Normalization -- The process of building the mapping ('New York City' <-> 123)

Fact table

Techniques that should be applied to the huge Fact table.

- id INT/BIGINT UNSIGNED NOT NULL AUTO_INCREMENT
- PRIMARY KEY (id)
- Probably no other INDEXes
- Accessed only via id
- All VARCHARs are "normalized"; ids are stored instead
- ENGINE = InnoDB
- All "reports" use summary tables, not the Fact table
- Summary tables may be populated from ranges of id (other techniques described below)

There are exceptions where the Fact table must be accessed to retrieve multiple rows. However, you should minimize the number of INDEXes on the table because they are likely to be costly on INSERT.

Why keep the Fact table?

Once you have built the Summary table(s), there is not much need for the Fact table. One option that you should seriously consider is to not have a Fact table. Or, at least, you could purge old data from it sooner than you purge the Summary tables. Maybe even keep the Summary tables forever.

Case 1: You need to find the raw data involved in some event. But how will you find those row(s)? This is where a secondary index may be required.

If a secondary index is bigger than can be cached in RAM, and if the column(s) being indexed is random, then each row inserted may cause a disk hit to update the index. This limits insert speed to something like 100 rows per second (on ordinary disks). Multiple random indexes slow down insertion further. RAID striping and/or SSDs speed up insertion. Write caching helps, but only for bursts.

Case 2: You need some event, but you did not plan ahead with the optimal INDEX. Well, if the data is PARTITIONed on date, so even if you have a clue of when the event occurred, "partition pruning" will keep the query from being too terribly slow.

Case 3: Over time, the application is likely to need new 'reports', which may lead to a new Summary table. At this point, it would be handy to scan through the old data to fill up the new table.

Case 4: You find a flaw in the summarization, and need to rebuild an existing Summary table.

Cases 3 and 4 both need the "raw" data. But they don't necessarily need the data sitting in a database table. It could be in the pre-database format (such as log files). So, consider not building the Fact table, but simply keep the raw data, comressed, on some file system.

Batching the load of the Fact table

When talking about billions of rows in the Fact table, it is essentially mandatory that you "batch" the inserts. There are two main ways:

- INSERT INTO Fact (.,.,.) VALUES (.,.,.), (.,.,.), ...; -- "Batch insert"
- LOAD DATA ...;

A third way is to INSERT or LOAD into a Staging table, then

- INSERT INTO Fact SELECT * FROM Staging; This INSERT..SELECT allows you to do other things, such as normalization. More later.

Batched INSERT Statement

Chunk size should usually be 100-1000 rows.

- 100-1000 an insert will run 10 times as fast as single-row inserts.
- Beyond 100, you may be interfering replication and SELECTs.
- Beyond 1000, you are into diminishing returns -- virtually no further performance gains.
- Don't go past, say, 1MB for the constructed INSERT statement. This deals with packet sizes, etc. (1MB is unlikely to be hit for a Fact table.) Decide whether your application should lean toward the 100 or the 1000.

If your data is coming in continually, and you are adding a batching layer, let's do some math. Compute your ingestion rate -- R rows per second.

- If R < 10 (= 1M/day = 300M/year) -- single-row INSERTs would probably work fine (that is, batching is optional)
- If R < 100 (3B records per year) -- secondary indexes on Fact table may be ok
- If R < 1000 (100M records/day) -- avoid secondary indexes on Fact table.
- If R > 1000 -- Batching may not work. Decide how long (S seconds) you can stall loading the data in order to collect a batch of rows.
- If S < 0.1s -- May not be able to keep up

If batching seems viable, then design the batching layer to gather for S seconds or 100-1000 rows, whichever comes first.

(Note: Similar math applies to rapid UPDATEs of a table.)

Normalization (Dimension) table

Normalization is important in Data Warehouse applications because it significantly cuts down on the disk footprint and improves performance. There are other reasons for normalizing, but space is the important one for DW.

Here is a typical pattern for a Dimension table:

```
CREATE TABLE Emails (
    email_id MEDIUMINT UNSIGNED NOT NULL AUTO_INCREMENT,  -- don't make bigger than needed
    email VARCHAR(...) NOT NULL,
    PRIMARY KEY (email),  -- for looking up one way
    INDEX(email_id)  -- for looking up the other way (UNIQUE is not needed)
) ENGINE = InnoDB;  -- to get clustering
```

Notes:

- MEDIUMINT is 3 bytes with UNSIGNED range of 0..16M; pick SMALLINT, INT, etc, based on a conservative estimate of how many 'foo's you will eventually have.
- datatype sizes
- There may be more than one VARCHAR in the table. Example: For cities, you might have City and Country.
- InnoDB is better than MyISAM because of way the two keys are structured.
- The secondary key is effectively (email_id, email), hence 'covering' for certain queries.
- It is OK to not specify an AUTO_INCREMENT to be UNIQUE.

Batched normalization

I bring this up as a separate topic because of some of the subtle issues that can happen.

You may be tempted to do

```
INSERT IGNORE INTO Foos
    SELECT DISTINCT foo FROM Staging;  -- not wise
```

It has the problem of "burning" AUTO_INCREMENT ids. This is because MariaDB pre-allocates ids before getting to "IGNORE". That could rapidly increase the AUTO_INCREMENT values beyond what you expected.

Better is this...

```
INSERT IGNORE INTO Foos
    SELECT DISTINCT foo
        FROM Staging
        LEFT JOIN Foos ON Foos.foo = Staging.foo
        WHERE Foos.foo_id IS NULL;
```

Notes:

- The LEFT JOIN .. IS NULL finds the `foo`s that are not yet in Foos.
- This INSERT..SELECT must not be done inside the transaction with the rest of the processing. Otherwise, you add to deadlock risks, leading to burned ids.
- IGNORE is used in case you are doing the INSERT from multiple processes simultaneously.

Once that INSERT is done, this will find all the foo_ids it needs:

```
INSERT INTO Fact (..., foo_id, ...)
    SELECT ..., Foos.foo_id, ...
        FROM Staging
        JOIN Foos ON Foos.foo = Staging.foo;
```

An advantage of "Batched Normalization" is that you can summarize directly from the Staging table. Two approaches:

Case 1: PRIMARY KEY (dy, foo) and summarization is in lock step with, say, changes in `dy`.

- This approach can have troubles if new data arrives after you have summarized the day's data.

```
INSERT INTO Summary (dy, foo, ct, blah_total)
    SELECT  DATE(dt) as dy, foo,
            COUNT(*) as ct, SUM(blah) as blah_total)
        FROM Staging
        GROUP BY 1, 2;
```

Case 2: (dy, foo) is a non-UNIQUE INDEX.

- Same code as Case 1.
- By having the index be non-UNIQUE, delayed data simply shows up as extra rows.
- You need to take care to avoid summarizing the data twice. (The id on the Fact table may be a good tool for that.)

Case 3: PRIMARY KEY (dy, foo) and summarization can happen anytime.

```
    INSERT INTO Summary (dy, foo, ct, blah_total)
        ON DUPLICATE KEY UPDATE
            ct = ct + VALUE(ct),
            blah_total = blah_total + VALUE(bt)
        SELECT  DATE(dt) as dy, foo,
                COUNT(*) as ct, SUM(blah) as bt)
            FROM Staging
            GROUP BY 1, 2;
```

Too many choices?

This document lists a number of ways to do things. Your situation may lead to one approach being more/less acceptable. But, if you are thinking "Just tell me what to do!", then here:

- Batch load the raw data into a temporary table (`Staging`).
- Normalize from `Staging` -- use code in Case 3.
- INSERT .. SELECT to move the data from `Staging` into the Fact table
- Summarize from `Staging` to Summary table(s) via IODKU (Insert ... On Duplicate Key Update).
- Drop the Staging

Those techniques should perform well and scale well in most cases. As you develop your situation, you may discover why I described alternative solutions.

Purging old data

Typically the Fact table is PARTITION BY RANGE (10-60 ranges of days/weeks/etc) and needs purging (DROP PARTITION) periodically. This discusses a safe/clean way to design the partitioning and do the DROPs: Purging PARTITIONs

Master / slave

For "read scaling", backup, and failover, use master-slave replication or something fancier. Do ingestion only on a single active master; it replicate to the slave(s). Generate reports on the slave(s).

Sharding

"Sharding" is the splitting of data across multiple servers. (In contrast, replication and Galera have the same data on all servers, requiring all data to be written to all servers.)

With the non-sharding techniques described here, terabyte(s) of data can be handled by a single machine. Tens of terabytes probably requires sharding.

Sharding is beyond the scope of this document.

How fast? How big?

With the techniques described here, you may be able to achieve the following performance numbers. I say "may" because every data warehouse situation is different, and you may require performance-hurting deviations from what I describe here. I give multiple options for some aspects; these may cover some of your deviations.

One big performance killer is UUID/GUID keys. Since they are very 'random', updates of them (at scale) are limited to 1 row = 1 disk hit. Plain disks can handle only 100 hits/second. RAID and/or SSD can increase that to something like 1000 hits/sec. Huge amounts of RAM (for caching the random index) are a costly solution. It is possible to turn type-1 UUIDs into roughly-chronological keys, thereby mittigating the performance problems if the UUIDs are written/read with some chronological clustering. UUID discussion

Hardware, etc:

- Single SATA drive: 100 IOPs (Input/Output operations per second)
- RAID with N physical drives -- 100*N IOPs (roughly)
- SSD -- 5 times as fast as rotating media (in this context)
- Batch INSERT -- 100-1000 rows is 10 times as fast as INSERTing 1 row at a time (see above)
- Purge "old" data -- Do not use DELETE or TRUNCATE, design so you can use DROP PARTITION (see above)
- Think of each INDEX (except the PRIMARY KEY on InnoDB) as a separate table
- Consider access patterns of each table/index: random vs at-the-end vs something in between

"Count the disk hits" -- back-of-envelope performance analysis

- Random accesses to a table/index -- count each as a disk hit.
- At-the-end accesses (INSERT chronologically or with AUTO_INCREMENT; range SELECT) -- count as zero hits.
- In between (hot/popular ids, etc) -- count as something in between
- For INSERTs, do the analysis on each index; add them up.
- For SELECTs, do the analysis on the one index used, plus the table. (Use of 2 indexes is rare.) Insert cost, based on datatype of first column in an index:
- AUTO_INCREMENT -- essentially 0 IOPs
- DATETIME, TIMESTAMP -- essentially 0 for 'current' times
- UUID/GUID -- 1 per insert (terrible)
- Others -- depends on their patterns SELECT cost gets a little tricky:
- Range on PRIMARY KEY -- think of it as getting 100 rows per disk hit.
- IN on PRIMARY KEY -- 1 disk hit per item in IN
- "=" -- 1 hit (for 1 row)
- Secondary key -- First compute the hits for the index, then...
- Think of each row as needing 1 disk hit.
- However, if the rows are likely to be 'near' each other (based on the PRIMARY KEY), then it could be < 1 disk hit/row.

More on Count the Disk Hits

How fast?

Look at your data; compute raw rows per second (or hour or day or year). There are about 30M seconds in a year; 86,400 seconds per day. Inserting 30 rows per second becomes a billion rows per year.

10 rows per second is about all you can expect from an ordinary machine (after allowing for various overheads). If you have less than that, you don't have many worries, but still you should probably create Summary tables. If more than 10/sec, then batching, etc, becomes vital. Even on spiffy hardware, 100/sec is about all you can expect without utilizing the techniques here.

Not so fast?

Let's say your insert rate is only one-tenth of your disk IOPs (eg, 10 rows/sec vs 100 IOPs). Also, let's say your data is not "bursty"; that is, the data comes in somewhat soothly throughout the day.

Note that 10 rows/sec (300M/year) implies maybe 30GB for data + indexes + normalization tables + summary tables for 1 year. I would call this "not so big".

Still, the normalization and summarization are important. Normalization keeps the data from being, say, twice as big. Summarization speeds up the reports by orders of magnitude.

Let's design and analyse a "simple ingestion scheme" for 10 rows/second, without 'batching'.

```
    # Normalize:
    $foo_id = SELECT foo_id FROM Foos WHERE foo = $foo;
    if no $foo_id, then
        INSERT IGNORE INTO Foos ...

    # Inserts:
    BEGIN;
        INSERT INTO Fact ...;
        INSERT INTO Summary ... ON DUPLICATE KEY UPDATE ...;
    COMMIT;
    # (plus code to deal with errors on INSERTs or COMMIT)
```

Depending on the number and randomness of your indexes, etc, 10 Fact rows may (or may not) take less than 100 IOPs.

Also, note that as the data grows over time, random indexes will become less and less likely to be cached. That is, even if runs fine with 1 year's worth of data, it may be in trouble with 2 year's worth.

For those reasons, I started this discussion with a wide margin (10 rows versus 100 IOPs).

References

- sec. 3.3.2: Dimensional Model and "Star schema"
- Summary Tables

3.3.4.24 Equality propagation optimization

Contents

Basic idea

Consider a query with a WHERE clause:

```
WHERE col1=col2 AND ...
```

the WHERE clause will compute to true only if `col1=col2`. This means that in the rest of the WHERE clause occurrences of `col1` can be substituted with `col2` (with some limitations which are discussed in the next section). This allows the optimizer to infer additional restrictions.

For example:

```
WHERE col1=col2 AND col1=123
```

allows to infer a new equality: `col2=123`

```
WHERE col1=col2 AND col1 < 10
```

allows to infer that `col2<10`.

Identity and comparison substitution

There are some limitations to where one can do the substitution, though.

The first and obvious example is the string datatype and collations. Most commonly-used collations in SQL are "case-insensitive", that is `'A'='a'`. Also, most collations have a "PAD SPACE" attribute, which means that comparison ignores the spaces at the end of the value, `'a'='a  '`.

Now, consider a query:

```
INSERT INTO t1 (col1, col2) VALUES ('ab', 'ab   ');
SELECT * FROM t1 WHERE col1=col2 AND LENGTH(col1)=2
```

Here, `col1=col2`, the values are "equal". At the same time `LENGTH(col1)=2`, while `LENGTH(col2)=4`, which means one can't perform the substiution for the argument of LENGTH(...).

It's not only collations. There are similar phenomena when equality compares columns of different datatypes. The exact criteria of when thy happen are rather convoluted.

The take-away is: **sometimes, X=Y does not mean that one can replace any reference to X with Y**. What one CAN do is still replace the occurrence in the comparisons `<`, `>`, `>=`, `<=`, etc.

This is how we get two kinds of substitution:

- **Identity substitution**: X=Y, and any occurrence of X can be replaced with Y.
- **Comparison substitution**: X=Y, and an occurrence of X in a comparison (X<Z) can be replaced with Y (Y<Z).

Place in query optimization

(A draft description): Let's look at how Equality Propagation is integrated with the rest of the query optimization process.

- First, multiple-equalities are built (TODO example from optimizer trace)
  - If multiple-equality includes a constant, fields are substituted with a constant if possible.
- From this point, all optimizations like range optimization, ref access, etc make use of multiple equalities: when they see a reference to `tableX.columnY` somewhere, they also look at all the columns that tableX.columnY is equal to.
- After the join order is picked, the optimizer walks through the WHERE clause and substitutes each field reference with the "best" one - the one that can be checked as soon as possible.
  - Then, the parts of the WHERE condition are attached to the tables where they can be checked.

Interplay with ORDER BY optimization

Consider a query:

```
SELECT ... FROM ... WHERE col1=col2 ORDER BY col2
```

Suppose, there is an `INDEX(col1)`. MariaDB optimizer is able to figure out that it can use an index on `col1` (or sort by the value of `col1`) in order to resolve `ORDER BY col2`.

Optimizer trace

Look at these elements:

- `condition_processing`
- `attaching_conditions_to_tables`

More details

Equality propagation doesn't just happen at the top of the WHERE clause. It is done "at all levels" where a level is:

- A top level of the WHERE clause.
- If the WHERE clause has an OR clause, each branch of the OR clause.
- The top level of any ON expression
- (the same as above about OR-levels)

3.3.4.25 FORCE INDEX

Contents

Description

Forcing an index to be used is mostly useful when the optimizer decides to do a table scan even if you know that using an index would be better. (The optimizer could decide to do a table scan even if there is an available index when it believes that most or all rows will match and it can avoid the overhead of using the index).

FORCE INDEX works by only considering the given indexes (like with USE_INDEX) but in addition it tells the optimizer to regard a table scan as something very expensive. However if none of the 'forced' indexes can be used, then a table scan will be used anyway.

FORCE INDEX cannot force an ignored index to be used - it will be treated as if it doesn't exist.

Example

```
CREATE INDEX Name ON City (Name);
EXPLAIN SELECT Name,CountryCode FROM City FORCE INDEX (Name)
WHERE name>="A" and CountryCode >="A";
```

This produces:

```
id select_type table type possible_keys key key_len ref rows Extra
1 SIMPLE City range Name Name 35 NULL 4079 Using where
```

Index Prefixes

When using index hints (USE, FORCE or IGNORE INDEX), the index name value can also be an unambiguous prefix of an index name.

3.3.4.26 Groupwise Max in MariaDB

Contents

The problem

You want to find the largest row in each group of rows. An example is looking for the largest city in each state. While it is easy to find the MAX(population) ... GROUP BY state, it is hard to find the name of the `city` associated with that `population`. Alas, MySQL and MariaDB do not have any syntax to provide the solution directly.

This article is under construction, mostly for cleanup. The content is reasonably accurate during construction.

The article presents two "good" solutions. They differ in ways that make neither of them 'perfect'; you should try both and weigh the pros and cons.

Also, a few "bad" solutions will be presented, together with why they were rejected.

MySQL manual gives 3 solutions; only the "Uncorrelated" one is "good", the other two are "bad".

Sample data

To show how the various coding attempts work, I have devised this simple task: Find the largest city in each Canadian province. Here's a sample of the source data (5493 rows):

```
+------------------+----------------+------------+
| province         | city           | population |
+------------------+----------------+------------+
| Saskatchewan     | Rosetown       |       2309 |
| British Columbia | Chilliwack     |      51942 |
| Nova Scotia      | Yarmouth       |       7500 |
| Alberta          | Grande Prairie |      41463 |
| Quebec           | Sorel          |      33591 |
| Ontario          | Moose Factory  |       2060 |
| Ontario          | Bracebridge    |       8238 |
| British Columbia | Nanaimo        |      84906 |
| Manitoba         | Neepawa        |       3151 |
| Alberta          | Grimshaw       |       2560 |
| Saskatchewan     | Carnduff       |        950 |
...
```

Here's the desired output (13 rows):

```
+--------------------------+---------------+------------+
| province                 | city          | population |
+--------------------------+---------------+------------+
| Alberta                  | Calgary       |     968475 |
| British Columbia         | Vancouver     |    1837970 |
| Manitoba                 | Winnipeg      |     632069 |
| New Brunswick            | Saint John    |      87857 |
| Newfoundland and Labrador | Corner Brook  |      18693 |
| Northwest Territories    | Yellowknife   |      15866 |
| Nova Scotia              | Halifax       |     266012 |
| Nunavut                  | Iqaluit       |       6124 |
| Ontario                  | Toronto       |    4612187 |
| Prince Edward Island     | Charlottetown |      42403 |
| Quebec                   | Montreal      |    3268513 |
| Saskatchewan             | Saskatoon     |     198957 |
| Yukon                    | Whitehorse    |      19616 |
+--------------------------+---------------+------------+
```

Duplicate max

One thing to consider is whether you want -- or do not want -- to see multiple rows for tied winners. For the dataset being used here, that would imply that the two largest cities in a province had identical populations. For this case, a duplicate would be unlikely. But there are many groupwise-max use cases where duplictes are likely.

The two best algorithms differ in whether they show duplicates.

Using an uncorrelated subquery

Characteristics:

- Superior performance or medium performance
- It will show duplicates
- Needs an extra index
- Probably requires 5.6
- If all goes well, it will run in O(M) where M is the number of output rows.

An 'uncorrelated subquery':

```
SELECT  c1.province, c1.city, c1.population
    FROM  Canada AS c1
    JOIN
      ( SELECT  province, MAX(population) AS population
            FROM  Canada
            GROUP BY  province
      ) AS c2 USING (province, population)
    ORDER BY c1.province;
```

But this also 'requires' an extra index: INDEX(province, population). In addition, MySQL has not always been able to use that index effectively, hence the "requires 5.6". (I am not sure of the actual version.)

Without that extra index, you would need 5.6, which has the ability to create indexes for subqueries. This is indicated by <auto_key0> in the EXPLAIN. Even so, the performance is worse with the auto-generated index than with the manually generated one.

With neither the extra index, nor 5.6, this 'solution' would belong in 'The Duds' because it would run in O(N*N) time.

Using @variables

Characteristics:

- Good performance
- Does not show duplicates (picks one to show)
- Consistent O(N) run time (N = number of input rows)
- Only one scan of the data

```
SELECT
        province, city, population   -- The desired columns
    FROM
      ( SELECT  @prev := '' ) init
    JOIN
      ( SELECT  province != @prev AS first,  -- `province` is the 'GROUP BY'
                @prev := province,           -- The 'GROUP BY'
                province, city, population   -- Also the desired columns
            FROM  Canada           -- The table
            ORDER BY
                province,          -- The 'GROUP BY'
                population DESC    -- ASC for MIN(population), DESC for MAX
      ) x
    WHERE  first
    ORDER BY  province;     -- Whatever you like
```

For your application, change the lines with comments.

The duds

*** 'correlated subquery' (from MySQL doc):**

```
SELECT  province, city, population
    FROM  Canada AS c1
    WHERE  population =
      ( SELECT  MAX(c2.population)
            FROM  Canada AS c2
            WHERE  c2.province= c1.province
      )
    ORDER BY  province;
```

O(N*N) (that is, terrible) performance

*** LEFT JOIN (from MySQL doc):**

```
SELECT  c1.province, c1.city, c1.population
    FROM  Canada AS c1
    LEFT JOIN  Canada AS c2 ON c2.province = c1.province
      AND  c2.population > c1.population
    WHERE  c2.province IS NULL
    ORDER BY province;
```

Medium performance (2N-3N, depending on join_buffer_size).

For O(N*N) time,... It will take one second to do a groupwise-max on a few thousand rows; a million rows could take hours.

Top-N in each group

This is a variant on "groupwise-max" wherein you desire the largest (or smallest) N items in each group. Do these substitutions for your use case:

- province --> your 'GROUP BY'
- Canada --> your table
- 3 --> how many of each group to show
- population --> your numeric field for determining "Top-N"
- city --> more field(s) you want to show
- Change the SELECT and ORDER BY if you desire
- DESC to get the 'largest'; ASC for the 'smallest'

```
SELECT
        province, n, city, population
    FROM
      ( SELECT  @prev := '', @n := 0 ) init
    JOIN
      ( SELECT  @n := if(province != @prev, 1, @n + 1) AS n,
                @prev := province,
                province, city, population
            FROM  Canada
            ORDER BY
                province   ASC,
                population DESC
      ) x
    WHERE  n <= 3
    ORDER BY  province, n;
```

Output:

```
+--------------------------+------+------------------+------------+
| province                 | n    | city             | population |
+--------------------------+------+------------------+------------+
| Alberta                  |    1 | Calgary          |     968475 |
| Alberta                  |    2 | Edmonton         |     822319 |
| Alberta                  |    3 | Red Deer         |      73595 |
| British Columbia         |    1 | Vancouver        |    1837970 |
| British Columbia         |    2 | Victoria         |     289625 |
| British Columbia         |    3 | Abbotsford       |     151685 |
| Manitoba                 |    1 | ...
```

The performance of this is O(N), actually about 3N, where N is the number of source rows.

EXPLAIN EXTENDED gives

```
+----+-------------+------------+--------+---------------+------+---------+------+------+----
------+----------------+
| id | select_type | table      | type   | possible_keys | key  | key_len | ref  | rows |
filtered | Extra          |
+----+-------------+------------+--------+---------------+------+---------+------+------+----
------+----------------+
|  1 | PRIMARY     | <derived2> | system | NULL          | NULL | NULL    | NULL |    1 |
100.00 | Using filesort |
|  1 | PRIMARY     | <derived3> | ALL    | NULL          | NULL | NULL    | NULL | 5484 |
100.00 | Using where    |
|  3 | DERIVED     | Canada     | ALL    | NULL          | NULL | NULL    | NULL | 5484 |
100.00 | Using filesort |
|  2 | DERIVED     | NULL       | NULL   | NULL          | NULL | NULL    | NULL | NULL |
NULL | No tables used |
+----+-------------+------------+--------+---------------+------+---------+------+------+----
------+----------------+
```

Explanation, shown in the same order as the EXPLAIN, but numbered chronologically: 3. Get the subquery id=2 (init) 4. Scan the the output from subquery id=3 (x) 2. Subquery id=3 -- the table scan of Canada 1. Subquery id=2 -- `init` for simply initializing the two @variables Yes, it took two sorts, though probably in RAM.

Main Handler values:

```
| Handler_read_rnd           | 39    |
| Handler_read_rnd_next      | 10971 |
| Handler_write              | 5485  |  -- #rows in Canada (+1)
```

Top-n in each group, take II

This variant is faster than the previous, but depends on `city` being unique across the dataset. (from openark.org)

```
SELECT  province, city, population
    FROM  Canada
    JOIN
      ( SELECT  GROUP_CONCAT(top_in_province) AS top_cities
            FROM
              ( SELECT  SUBSTRING_INDEX(
                            GROUP_CONCAT(city ORDER BY  population DESC),
                        ',', 3) AS top_in_province
                    FROM  Canada
                    GROUP BY  province
              ) AS x
      ) AS y
    WHERE  FIND_IN_SET(city, top_cities)
    ORDER BY  province, population DESC;
```

Output. Note how there can be more than 3 cities per province:

```
| Alberta                  | Calgary          |     968475 |
| Alberta                  | Edmonton         |     822319 |
| Alberta                  | Red Deer         |      73595 |
| British Columbia         | Vancouver        |    1837970 |
| British Columbia         | Victoria         |     289625 |
| British Columbia         | Abbotsford       |     151685 |
| British Columbia         | Sydney           |          0 | -- Nova Scotia's second largest is
| Manitoba                 | Winnipeg         |     632069 |
```

Main Handler values:

```
| Handler_read_next          | 5484  | -- table size
| Handler_read_rnd_next      | 5500  | -- table size + number of provinces
| Handler_write              | 14    | -- number of provinces (+1)
```

Top-n using MyISAM

(This does not need your table to be MyISAM, but it does need MyISAM tmp table for its 2-column PRIMARY KEY feature.) See previous section for what changes to make for your use case.

```
    -- build tmp table to get numbering
    -- (Assumes auto_increment_increment = 1)
    CREATE TEMPORARY TABLE t (
        nth MEDIUMINT UNSIGNED NOT NULL AUTO_INCREMENT,
        PRIMARY KEY(province, nth)
    ) ENGINE=MyISAM
        SELECT province, NULL AS nth, city, population
            FROM Canada
            ORDER BY population DESC;
    -- Output the biggest 3 cities in each province:
    SELECT province, nth, city, population
        FROM t
        WHERE nth <= 3
        ORDER BY province, nth;

+--------------------------+-----+-----------------+------------+
| province                 | nth | city            | population |
+--------------------------+-----+-----------------+------------+
| Alberta                  |   1 | Calgary         |     968475 |
| Alberta                  |   2 | Edmonton        |     822319 |
| Alberta                  |   3 | Red Deer        |      73595 |
| British Columbia         |   1 | Vancouver       |    1837970 |
| British Columbia         |   2 | Victoria        |     289625 |
| British Columbia         |   3 | Abbotsford      |     151685 |
| Manitoba                 |  ...

SELECT for CREATE:
+----+-------------+--------+------+---------------+------+---------+------+------+--------------
| id | select_type | table  | type | possible_keys | key  | key_len | ref  | rows | Extra
+----+-------------+--------+------+---------------+------+---------+------+------+--------------
|  1 | SIMPLE      | Canada | ALL  | NULL          | NULL | NULL    | NULL | 5484 | Using filesor
+----+-------------+--------+------+---------------+------+---------+------+------+--------------
Other SELECT:
+----+-------------+-------+-------+---------------+---------+---------+------+------+-----------
| id | select_type | table | type  | possible_keys | key     | key_len | ref  | rows | Extra
+----+-------------+-------+-------+---------------+---------+---------+------+------+-----------
|  1 | SIMPLE      | t     | index | NULL          | PRIMARY | 104     | NULL |   22 | Using wher
+----+-------------+-------+-------+---------------+---------+---------+------+------+-----------
```

The main handler values (total of all operations):

```
| Handler_read_rnd_next      | 10970 |
| Handler_write              | 5484  |  -- number of rows in Canada (write tmp table)
```

Both "Top-n" formulations probably take about the same amount of time.

Windowing functions

Hot off the press from Percona Live... MariaDB 10.2 has "windowing functions", which make "groupwise max" much more straightforward.

The code:

TBD

Postlog

Developed an first posted, Feb, 2015; Add MyISAM approach: July, 2015; Openark's method added: Apr, 2016; Windowing: Apr 2016

I did not include the technique(s) using GROUP_CONCAT. They are useful in some situations with small datasets. They can be found in the references below.

3.3.4.27 GUID/UUID Performance

Contents

The problem

GUIDs/UUIDs (Globally/Universally Unique Identifiers) are very random. Therefore, INSERTing into an index means jumping around a lot. Once the index is too big to be cached, most INSERTs involve a disk hit. Even on a beefy system, this limits you to a few hundred INSERTs per second.

MariaDB's UUID function.

This blog is mostly eliminated in MySQL 8.0 with the advent of the following function: UUID_TO_BIN(str, swap_flag).

Why it is a problem

A 'standard' GUID/UUID is composed of the time, machine identification and some other stuff. The combination should be unique, even without coordination between different computers that could be generating UUIDs simultaneously.

The top part of the GUID/UUID is the bottom part of the current time. The top part is the primary part of what would be used for placing the value in an ordered list (INDEX). This cycles in about 7.16 minutes.

Some math... If the index is small enough to be cached in RAM, each insert into the index is CPU only, with the writes being delayed and batched. If the index is 20 times as big as can be cached, then 19 out of 20 inserts will be a cache miss. (This math applies to any "random" index.)

Second problem

36 characters is bulky. If you are using that as a PRIMARY KEY in InnoDB and you have secondary keys, remember that each secondary key has an implicit copy of the PK, thereby making it bulky.

It is tempting to declare the UUID VARCHAR(36). And, since you probably are thinking globally, so you have CHARACTER SET utf8 (or utf8mb4). For utf8:

- 2 - Overhead for VAR
- 36 - chars
- 3 (or 4) bytes per character for utf8 (or utf8mb4) So, max length = 2+3*36 = 110 (or 146) bytes. For temp tables 108 (or 144) is actually used if a MEMORY table is used.

To compress

- utf8 is unnecessary (ascii would do); but this is obviated by the next two steps
- Toss dashes
- UNHEX Now it will fit in 16 bytes: BINARY(16)

Combining the problems and crafting a solution

But first, a caveat. This solution only works for "Time based" / "Version 1" UUIDs They are recognizable by the "1" at the beginning of the third clump.

The manual's sample: 6ccd780c-baba-1026-9564-0040f4311e29 . A more current value (after a few years): 49ea2de3-17a2-11e2-8346-001eecac3efa . Notice how the 3rd part has slowly changed over time? Let's data is rearranged, thus:

```
1026-baba-6ccd780c-9564-0040f4311e29
11e2-17a2-49ea2de3-8346-001eecac3efa
11e2-17ac-106762a5-8346-001eecac3efa -- after a few more minutes
```

Now we have a number that increases nicely over time. Multiple sources won't be quite in time order, but they will be close. The "hot" spot for inserting into an INDEX(uuid) will be rather narrow, thereby making it quite cacheable and efficient.

If your SELECTs tend to be for "recent" uuids, then they, too, will be easily cached. If, on the other hand, your SELECTs often reach for old uuids, they will be random and not well cached. Still, improving the INSERTs will help the system overall.

Code to do it

Let's make Stored Functions to do the messy work of the two actions:

- Rearrange fields
- Convert to/from BINARY(16)

```
DELIMITER //

CREATE FUNCTION UuidToBin(_uuid BINARY(36))
    RETURNS BINARY(16)
    LANGUAGE SQL  DETERMINISTIC  CONTAINS SQL  SQL SECURITY INVOKER
RETURN
    UNHEX(CONCAT(
        SUBSTR(_uuid, 15, 4),
        SUBSTR(_uuid, 10, 4),
        SUBSTR(_uuid,  1, 8),
        SUBSTR(_uuid, 20, 4),
        SUBSTR(_uuid, 25) ));
//
CREATE FUNCTION UuidFromBin(_bin BINARY(16))
    RETURNS BINARY(36)
    LANGUAGE SQL  DETERMINISTIC  CONTAINS SQL  SQL SECURITY INVOKER
RETURN
    LCASE(CONCAT_WS('-',
        HEX(SUBSTR(_bin,  5, 4)),
        HEX(SUBSTR(_bin,  3, 2)),
        HEX(SUBSTR(_bin,  1, 2)),
        HEX(SUBSTR(_bin,  9, 2)),
        HEX(SUBSTR(_bin, 11))
             ));

//
DELIMITER ;
```

Then you would do things like

```
-- Letting MySQL create the UUID:
INSERT INTO t (uuid, ...) VALUES (UuidToBin(UUID()), ...);

-- Creating the UUID elsewhere:
INSERT INTO t (uuid, ...) VALUES (UuidToBin(?), ...);

-- Retrieving (point query using uuid):
SELECT ... FROM t WHERE uuid = UuidToBin(?);

-- Retrieving (other):
SELECT UuidFromBin(uuid), ... FROM t ...;
```

Do not flip the WHERE; this will be inefficent because it won't use INDEX(uuid):

```
WHERE UuidFromBin(uuid) = '1026-baba-6ccd780c-9564-0040f4311e29' -- NO
```

TokuDB

> TokuDB has been deprecated by its upstream maintainer. It is disabled from MariaDB 10.5 and has been been removed in MariaDB 10.6 - MDEV-19780. We recommend MyRocks as a long-term migration path.

TokuDB is a viable engine if you must have UUIDs (even non-type-1) in a huge table. TokuDB is available in MariaDB as a 'standard' engine, making the barrier to entry very low. There are a small number of differences between InnoDB and TokuDB; I will not go into them here.

Tokudb, with its “fractal” indexing strategy builds the indexes in stages. In contrast, InnoDB inserts index entries “immediately” — actually that indexing is buffered by most of the size of the buffer_pool. To elaborate…

When adding a record to an InnoDB table, here are (roughly) the steps performed to write the data (and PK) and secondary indexes to disk. (I leave out logging, provision for rollback, etc.) First the PRIMARY KEY and data:

- Check for UNIQUEness constraints

- Fetch the BTree block (normally 16KB) that should contain the row (based on the PRIMARY KEY).
- Insert the row (overflow typically occurs 1% of the time; this leads to a block split).
- Leave the page “dirty” in the buffer_pool, hoping that more rows are added before it is bumped out of cache (buffer_pool).. Note that for AUTO_INCREMENT and TIMESTAMP-based PKs, the “last” block in the data will be updated repeatedly before splitting; hence, this delayed write adds greatly to the efficiency. OTOH, a UUID will be very random; when the table is big enough, the block will almost always be flushed before a second insert occurs in that block. <– This is the inefficiency in UUIDs. Now for any secondary keys:
- All the steps are the same, since an index is essentially a "table" except that the "data" is a copy of the PRIMARY KEY.
- UNIQUEness must be checked immediately — cannot delay the read.
- There are (I think) some other "delays" that avoid some I/O.

Tokudb, on the other hand, does something like

- Write data/index partially sorted records to disk before finding out exactly where it belongs.
- In the background, combine these partially digested blocks. Repeat as needed.
- Eventually move the info into the real table/indexes.

If you are familiar with how sort-merge works, consider the parallels to Tokudb. Each "sort" does some work of ordering things; each "merge" is quite efficient.

To summarize:

- In the extreme (data/index much larger than buffer_pool), InnoDB must read-modify-write one 16KB disk block for each UUID entry.
- Tokudb makes each I/O "count" by merging several UUIDs for each disk block. (Yeah, Toku rereads blocks, but it comes out ahead in the long run.)
- Tokudb excels when the table is really big, which implies high ingestion rate.

Wrapup

This shows three thing for speeding up usage of GUIDs/UUIDs:

- Shrink footprint (Smaller -> more cacheable -> faster).
- Rearrange uuid to make a "hot spot" to improve cachability.
- Use TokuDB (MyRocks shares some architectural traits which may also be beneficial in handling UUIDs, but this is hypothetical and hasn't been tested)

Note that the benefit of the "hot spot" is only partial:

- Chronologically ordered (or approximately ordered) INSERTs benefit; random ones don't.
- SELECTs/UPDATEs by "recent" uuids benefit; old ones don't benefit.

Postlog

Thanks to Trey for some of the ideas here.

The tips in this document apply to MySQL, MariaDB, and Percona.

Written Oct, 2012. Added TokuDB, Jan, 2015.

3.3.4.28 IGNORE INDEX

Syntax

```
IGNORE INDEX [{FOR {JOIN|ORDER BY|GROUP BY}] ([index_list])
```

Contents

Description

You can tell the optimizer to not consider a particular index with the IGNORE INDEX option.

The benefit of using IGNORE_INDEX instead of USE_INDEX is that it will not disable a new index which you may add later.

Also see Ignored Indexes for an option to specify in the index definition that indexes should be ignored.

Index Prefixes

When using index hints (USE, FORCE or IGNORE INDEX), the index name value can also be an unambiguous prefix of an index name.

Example

This is used after the table name in the FROM clause:

```
CREATE INDEX Name ON City (Name);
CREATE INDEX CountryCode ON City (Countrycode);
EXPLAIN SELECT Name FROM City IGNORE INDEX (Name)
WHERE name="Helsingborg" AND countrycode="SWE";
```

This produces:

```
id select_type table type possible_keys key key_len ref rows Extra
1 SIMPLE City ref CountryCode CountryCode 3 const 14 Using where
```

3.3.4.29 not_null_range_scan Optimization

Contents

The NOT NULL range scan optimization enables the optimizer to construct range scans from NOT NULL conditions that it was able to infer from the WHERE clause.

The optimization appeared in MariaDB 10.5.0. It is not enabled by default; one needs to set an `optimizer_switch` flag to enable it.

Description

A basic (but slightly artificial) example:

```
create table items (
  price  decimal(8,2),
  weight decimal(8,2),
  ...
  index(weight)
);
```

```
-- Find items that cost more than 1000 $currency_units per kg:
set optimizer_switch='not_null_range_scan=on';
explain
select * from items where items.price > items.weight / 1000;
```

The WHERE condition in this form cannot be used for range scans. However, one can infer that it will reject rows that NULL for `weight`. That is, infer an additional condition that

```
weight IS NOT NULL
```

and pass it to the range optimizer. The range optimizer can, in turn, evaluate whether it makes sense to construct range access from the condition:

```
+------+-------------+-------+-------+---------------+--------+---------+------+------+----------
| id   | select_type | table | type  | possible_keys | key    | key_len | ref  | rows | Extra
+------+-------------+-------+-------+---------------+--------+---------+------+------+----------
|    1 | SIMPLE      | items | range | NULL          | weight | 5       | NULL | 1    | Using whe
+------+-------------+-------+-------+---------------+--------+---------+------+------+----------
```

Here's another example that's more complex but is based on a real-world query. Consider a join query

```
-- Find orders that were returned
select * from current_orders as O, order_returns as RET
where
  O.return_id= RET.id;
```

Here, the optimizer can infer the condition "return_id IS NOT NULL". If most of the orders are not returned (and so have NULL for return_id), one can use range access to scan only those orders that had a return.

Controlling the Optimization

The optimization is not enabled by default. One can enable it like so

```
set optimizer_switch='not_null_range_scan=on';
```

Optimizer Trace

TODO.

3.3.4.30 Optimizing for "Latest News"-style Queries

Contents

The problem space

Let's say you have "news articles" (rows in a table) and want a web page showing the latest ten articles about a particular topic.

Variants on "topic":

- Category
- Tag
- Provider (of news article)
- Manufacturer (of item for sale)
- Ticker (financial stock)

Variants on "news article"

- Item for sale
- Blog comment
- Blog thread

Variants on "latest"

- Publication date (unix_timestamp)
- Most popular (keep the count)
- Most emailed (keep the count)
- Manual ranking (1..10 -- 'top ten')

Variants on "10" - there is nothing sacred about "10" in this discussion.

The performance issues

Currently you have a table (or a column) that relates the topic to the article. The SELECT statement to find the latest 10 articles has grown in complexity, and performance is poor. You have focused on what index to add, but nothing seems to work.

- If there are multiple topics for each article, you need a many-to-many table.
- You have a flag "is_deleted" that needs filtering on.
- You want to "paginate" the list (ten articles per page, for as many pages as necessary).

The solution

First, let me give you the solution, then I will elaborate on why it works well.

- One new table called, say, Lists.
- Lists has _exactly_ 3 columns: topic, article_id, sequence
- Lists has _exactly_ 2 indexes: PRIMARY KEY(topic, sequence, article_id), INDEX(article_id)
- Only viewable articles are in Lists. (This avoids the filtering on "is_deleted", etc)
- Lists is InnoDB. (This gets "clustering".)
- "sequence" is typically the date of the article, but could be some other ordering.
- "topic" should probably be normalized, but that is not critical to this discussion.
- "article_id" is a link to the bulky row in another table(s) that provide all the details about the article.

The queries

Find the latest 10 articles for a topic:

```
SELECT  a.*
    FROM  Articles a
    JOIN  Lists s ON s.article_id = a.article_id
    WHERE  s.topic = ?
    ORDER BY  s.sequence DESC
    LIMIT  10;
```

You must *not* have any WHERE condition touching columns in Articles.

When you mark an article for deletion; you *must* remove it from Lists:

```
DELETE  FROM  Lists
    WHERE  article_id = ?;
```

I emphasize "must" because flags and other filtering is often the root of performance issues.

Why it works

By now, you may have discovered why it works.

The big goal is to minimize the disk hits. Let's itemize how few disk hits are needed. When finding the latest articles with 'normal' code, you will probably find that it is doing significant scans of the Articles table, failing to quickly home in on the 10 rows you want. With this design, there is only one extra disk hit:

- 1 disk hit: 10 adjacent, narrow, rows in Lists -- probably in a single "block".
- 10 disk hits: The 10 articles. (These hits are unavoidable, but may be cached.) The PRIMARY KEY, and using InnoDB, makes these quite efficient.

OK, you pay for this by removing things that you should avoid.

- 1 disk hit: INDEX(article_id) - finding a few ids
- A few more disk hits to DELETE rows from Lists. This is a small price to pay -- and you are not paying it while the user is waiting for the page to render.

3.3.4.31 Pagination Optimization

Contents

The Desire

You have a website with news articles, or a blog, or some other thing with a list of things that might be too long for a single page. So, you decide to break it into chunks of, say, 10 items and provide a [Next] button to go the next "page".

You spot OFFSET and LIMIT in MariaDB and decide that is the obvious way to do it.

```
SELECT  *
    FROM  items
    WHERE  messy_filtering
    ORDER BY  date DESC
    OFFSET  $M  LIMIT $N
```

Note that the problem requirement needs a [Next] link on each page so that the user can 'page' through the data. He does not really need "GoTo Page #". Jump to the [First] or [Last] page may be useful.

The Problem

All is well -- until you have 50,000 items in a list. And someone tries to walk through all 5000 pages. That 'someone' could be a search engine crawler.

Where's the problem? Performance. Your web page is doing "SELECT ... OFFSET 49990 LIMIT 10" (or the equivalent "LIMIT 49990,10"). MariaDB has to find all 50,000 rows, step over the first 49,990, then deliver the 10 for that distant page.

If it is a crawler ('spider') that read all the pages, then it actually touched about 125,000,000 items to read all 5,000 pages.

Reading the entire table, just to get a distant page, can be so much I/O that it can cause timeouts on the web page. Or it can interfere with other activity, causing other things to be slow.

Other Bugs

In addition to a performance problem, ...

- If an item is inserted or deleted between the time you look at one page and the next, you could miss an item, or see an item duplicated.
- The pages are not easily bookmarked or sent to someone else because the contents shift over time.
- The WHERE clause and the ORDER BY may even make it so that all 50,000 items have to be read, just to find the 10 items for page 1!

What to Do?

Hardware? No, that's just a bandaid. The data will continue to grow and even the new hardware won't handle it.

Better INDEX? No. You must get away from reading the entire table to get the 5000th page.

Build another table saying where the pages start? Get real! That would be a maintenance nightmare, and expensive.

Bottom line: Don't use OFFSET; instead remember where you "left off".

```
First page (latest 10 items):
    SELECT ... WHERE ... ORDER BY id DESC LIMIT 10
Next page (second 10):
    SELECT ... WHERE ... AND id < $left_off ORDER BY id DESC LIMIT 10
```

With INDEX(id), this suddenly becomes very efficient.

Implementation -- Getting Rid of OFFSET

You are probably doing this now: ORDER BY datetime DESC LIMIT 49990,10 You probably have some unique id on the table. This can probably be used for "left off".

Currently, the [Next] button probably has a url something like ?topic=xyz&page=4999&limit=10 The 'topic' (or 'tag' or 'provider' or 'user' or etc) says which set of items are being displayed. The product of page*limit gives the OFFSET. (The "limit=10" might be in the url, or might be hard-coded; this choice is not relevant to this discussion.)

The new variant would be ?topic=xyz&id=12345&limit=10. (Note: the 12345 is not computable from 4999.) By using INDEX(topic, id) you can efficiently say

```
    WHERE topic = 'xyz'
      AND id >= 1234
    ORDER BY id
    LIMIT 10
```

That will hit only 10 rows. This is a huge improvement for later pages. Now for more details.

Implementation -- "Left Off"

What if there are exactly 10 rows left when you display the current page? It would make the UI nice if you grayed out the [Next] button, wouldn't it. (Or you could suppress the button all together.)

How to do that? Instead of LIMIT 10, use LIMIT 11. That will give you the 10 items needed for the current page, plus an indication of whether there is another page. And the id for that page.

So, take the 11th id for the [Next] button: Next

Implementation -- Links Beyond [Next]

Let's extend the 11 trick to also find the next 5 pages and build links for them.

Plan A is to say LIMIT 51. If you are on page 12, that would give you links for pages 13 (using 11th id) through pages 17 (51st).

Plan B is to do two queries, one to get the 10 items for the current page, the other to get the next 41 ids (LIMIT 10, 41) for the next 5 pages.

Which plan to pick? It depends on many things, so benchmark.

A Reasonable Set of Links

Reaching forward and backward by 5 pages is not too much work. It would take two separate queries to find the ids in both directions. Also, having links that take you to the First and Last pages would be easy to do. No id is needed; they can be something like

```
    <a href=?topic=xyz&id=FIRST&limit=10>First</a>
    <a href=?topic=xyz&id=LAST&limit=10>Last</a>
```

The UI would recognize those, then generate a SELECT with something like

```
    WHERE topic = 'xyz'
    ORDER BY id ASC -- ASC for First; DESC for Last
    LIMIT 10
```

The last items would be delivered in reverse order. Either deal with that in the UI, or make the SELECT more complex:

```
    ( SELECT ...
        WHERE topic = 'xyz'
        ORDER BY id DESC
        LIMIT 10
    ) ORDER BY id ASC
```

Let's say you are on page 12 of lots of pages. It could show these links:

```
    [First] ... [7] [8] [9] [10] [11] 12 [13] [14] [15] [16] [17] ... [Last]
```

where the ellipsis is really used. Some end cases:

```
Page one of three:
    First [2] [3]
Page one of many:
    First [2] [3] [4] [5] ... [Last]
Page two of many:
    [First] 2 [3] [4] [5] ... [Last]
If you jump to the Last page, you don't know what page number it is.
So, the best you can do is perhaps:
    [First] ... [Prev] Last
```

Why it Works

The goal is to touch only the relevant rows, not all the rows leading up to the desired rows. This is nicely achieved, except for building links to the "next 5 pages". That may (or may not) be efficiently resolved by the simple SELECT id, discussed above. The reason that may not be efficient deals with the WHERE clause.

Let's discuss the optimal and suboptimal indexes.

For this discussion, I am assuming

- The datetime field might have duplicates -- this can cause troubles
- The id field is unique
- The id field is close enough to datetime-ordered to be used instead of datetime.

Very efficient -- it does all the work in the index:

```
    INDEX(topic, id)
    WHERE topic = 'xyz'
      AND id >= 876
    ORDER BY id ASC
    LIMIT 10,41
<</code??
That will hit 51 consecutive index entries, 0 data rows.

Inefficient -- it must reach into the data:
<<code>>
    INDEX(topic, id)
    WHERE topic = 'xyz'
      AND id >= 876
      AND is_deleted = 0
    ORDER BY id ASC
    LIMIT 10,41
```

That will hit at least 51 consecutive index entries, plus at least 51 _randomly_ located data rows.

Efficient -- back to the previous degree of efficiency:

```
    INDEX(topic, is_deleted, id)
    WHERE topic = 'xyz'
      AND id >= 876
      AND is_deleted = 0
    ORDER BY id ASC
    LIMIT 10,41
```

Note how all the '=' parts of the WHERE come first; then comes both the '>=' and 'ORDER BY', both on id. This means that the INDEX can be used for all the WHERE, plus the ORDER BY.

"Items 11-20 Out of 12345"

You lose the "out of" except when the count is small. Instead, say something like

```
    Items 11-20 out of Many
```

Alternatively... Only a few searches will have too many items to count. Keep another table with the search criteria and a count. This count can be computed daily (or hourly) by some background script. When discovering that the topic is a busy one, look it up in the table to get

```
    Items 11-20 out of about 49,000
```

The background script would round the count off.

The quick way to get an _estimated_ number of rows for an InnoDB table is

```
    SELECT  table_rows
        FROM  information_schema.TABLES
        WHERE  TABLE_SCHEMA = 'database_name'
          AND  TABLE_NAME = 'table_name'
```

However, it does not allow for the WHERE clause that you probably have.

Complex WHERE, or JOIN

If the search criteria cannot be confined to an INDEX in a single table, this technique is doomed. I have another paper that discusses "Lists", which solves that (which extra development work), and even improves on what is discussed here.

How Much Faster?

This depends on

- How many rows (total)
- Whether the WHERE clause prevented the efficient use of the ORDER BY
- Whether the data is bigger than the cache. This last one kicks in when building one page requires reading more data from disk can be cached. At that point, the problem goes from being CPU-bound to being I/O-bound. This is likely to suddenly slow down the loading of a pages by a factor of 10.

What is Lost

- Cannot "jump to Page N", for an arbitrary N. Why do you want to do that?
- Walking backward from the end does not know the page numbers.
- The code is more complex.

Postlog

Designed about 2007; posted 2012.

3.3.4.32 Pivoting in MariaDB

Contents

The problem

You want to "pivot" the data so that a linear list of values with two keys becomes a spreadsheet-like array. See examples, below.

A solution

The best solution is probably to do it in some form of client code (PHP, etc). MySQL and MariaDB do not have a syntax for SELECT that will do the work for you. The code provided here uses a stored procedure to generate code to pivot the data, and then runs the code.

You can edit the SQL generated by the stored procedure to tweak the output in a variety of ways. Or you can tweak the stored procedure to generate what you would prefer.

Reference code for solution

'Source' this into the mysql commandline tool:

```
DELIMITER //
DROP   PROCEDURE IF EXISTS Pivot //
CREATE PROCEDURE Pivot(
    IN tbl_name VARCHAR(99),       -- table name (or db.tbl)
    IN base_cols VARCHAR(99),      -- column(s) on the left, separated by commas
    IN pivot_col VARCHAR(64),      -- name of column to put across the top
    IN tally_col VARCHAR(64),      -- name of column to SUM up
    IN where_clause VARCHAR(99),   -- empty string or "WHERE ..."
    IN order_by VARCHAR(99)        -- empty string or "ORDER BY ..."; usually the base_cols
    )
    DETERMINISTIC
    SQL SECURITY INVOKER
BEGIN
    -- Find the distinct values
    -- Build the SUM()s
    SET @subq = CONCAT('SELECT DISTINCT ', pivot_col, ' AS val ',
                    ' FROM ', tbl_name, ' ', where_clause, ' ORDER BY 1');
    -- select @subq;

    SET @cc1 = "CONCAT('SUM(IF(&p = ', &v, ', &t, 0)) AS ', &v)";
    SET @cc2 = REPLACE(@cc1, '&p', pivot_col);
    SET @cc3 = REPLACE(@cc2, '&t', tally_col);
    -- select @cc2, @cc3;
    SET @qval = CONCAT("'\"', val, '\"'");
    -- select @qval;
    SET @cc4 = REPLACE(@cc3, '&v', @qval);
    -- select @cc4;

    SET SESSION group_concat_max_len = 10000;   -- just in case
    SET @stmt = CONCAT(
            'SELECT  GROUP_CONCAT(', @cc4, ' SEPARATOR ",\n")  INTO @sums',
            ' FROM ( ', @subq, ' ) AS top');
     select @stmt;
    PREPARE _sql FROM @stmt;
    EXECUTE _sql;                       -- Intermediate step: build SQL for columns
    DEALLOCATE PREPARE _sql;
    -- Construct the query and perform it
    SET @stmt2 = CONCAT(
            'SELECT ',
                base_cols, ',\n',
                @sums,
                ',\n SUM(', tally_col, ') AS Total'
            '\n FROM ', tbl_name, ' ',
            where_clause,
            ' GROUP BY ', base_cols,
            '\n WITH ROLLUP',
            '\n', order_by
        );
    select @stmt2;                      -- The statement that generates the result
    PREPARE _sql FROM @stmt2;
    EXECUTE _sql;                       -- The resulting pivot table ouput
    DEALLOCATE PREPARE _sql;
    -- For debugging / tweaking, SELECT the various @variables after CALLing.
END;
//
DELIMITER ;
```

Then do a CALL, like in the examples, below.

Variants

I thought about having several extra options for variations, but decided that would be too messy. Instead, here are instructions for implementing the variations, either by capturing the SELECT that was output by the Stored Procedure, or by modifying the SP, itself.

- The data is strings (not numeric) -- Remove "SUM" (but keep the expression); remove the SUM...AS TOTAL line.
- If you want blank output instead of 0 -- Currently the code says "SUM(IF(... 0))"; change the 0 to NULL, then wrap the SUM: IFNULL(SUM(...), ''). Note that this will distinguish between a zero total (showing '0') and no data (blank).
- Fancier output -- Use PHP/VB/Java/etc.
- No Totals at the bottom -- Remove the WITH ROLLUP line from the SELECT.
- No Total for each row -- Remove the SUM...AS Total line from the SELECT.
- Change the order of the columns -- Modify the ORDER BY 1 ('1' meaning first column) in the SELECT DISTINCT in the SP.
- Example: ORDER BY FIND_IN_SET(DAYOFWEEK(...), 'Sun,Mon,Tue,Wed,Thu,Fri,Sat')

Notes about "base_cols":

- Multiple columns on the left, such as an ID and its meaning -- This is already handled by allowing base_cols to be a commalist like 'id, meaning'
- You cannot call the SP with "foo AS 'blah'" in hopes of changing the labels, but you could edit the SELECT to achieve that goal.

Notes about the "Totals":

- If "base_cols" is more than one column, WITH ROLLUP will be subtotals as well as a grand total.
- NULL shows up in the Totals row in the "base_cols" column; this can be changed via something like IFNULL(..., 'Totals').

Example 1 - Population vs Latitude in US

```
-- Sample input:
+-------+----------------------+---------+------------+
| state | city                 | lat     | population |
+-------+----------------------+---------+------------+
| AK    | Anchorage            | 61.2181 |     276263 |
| AK    | Juneau               | 58.3019 |      31796 |
| WA    | Monroe               | 47.8556 |      15554 |
| WA    | Spanaway             | 47.1042 |      25045 |
| PR    | Arecibo              | 18.4744 |      49189 |
| MT    | Kalispell            | 48.1958 |      18018 |
| AL    | Anniston             | 33.6597 |      23423 |
| AL    | Scottsboro           | 34.6722 |      14737 |
| HI    | Kaneohe              | 21.4181 |      35424 |
| PR    | Candelaria           | 18.4061 |      17632 |
...

-- Call the Stored Procedure:
CALL Pivot('World.US', 'state', '5*FLOOR(lat/5)', 'population', '', '');

-- SQL generated by the SP:
SELECT state,
SUM(IF(5*FLOOR(lat/5) = "15", population, 0)) AS "15",
SUM(IF(5*FLOOR(lat/5) = "20", population, 0)) AS "20",
SUM(IF(5*FLOOR(lat/5) = "25", population, 0)) AS "25",
SUM(IF(5*FLOOR(lat/5) = "30", population, 0)) AS "30",
SUM(IF(5*FLOOR(lat/5) = "35", population, 0)) AS "35",
SUM(IF(5*FLOOR(lat/5) = "40", population, 0)) AS "40",
SUM(IF(5*FLOOR(lat/5) = "45", population, 0)) AS "45",
SUM(IF(5*FLOOR(lat/5) = "55", population, 0)) AS "55",
SUM(IF(5*FLOOR(lat/5) = "60", population, 0)) AS "60",
SUM(IF(5*FLOOR(lat/5) = "70", population, 0)) AS "70",
 SUM(population) AS Total
 FROM World.US  GROUP BY state
 WITH ROLLUP

-- Output from that SQL (also comes out of the SP):
+-------+---------+--------+----------+----------+----------+----------+---------+-------+---
-----+------+-----------+
| state | 15      | 20     | 25       | 30       | 35       | 40       | 45      | 55    | 60
| 70   | Total     |
+-------+---------+--------+----------+----------+----------+----------+---------+-------+---
-----+------+-----------+
| AK    |       0 |      0 |        0 |        0 |        0 |        0 |       0 | 60607 |
360765 | 4336 |    425708 |
| AL    |       0 |      0 |        0 |  1995225 |        0 |        0 |       0 |     0 |
0 |    0 |   1995225 |
| AR    |       0 |      0 |        0 |   595537 |   617361 |        0 |       0 |     0 |
0 |    0 |   1212898 |
| AZ    |       0 |      0 |        0 |  4708346 |   129989 |        0 |       0 |     0 |
0 |    0 |   4838335 |
...
| FL    |       0 |  34706 |  9096223 |  1440916 |        0 |        0 |       0 |     0 |
0 |    0 |  10571845 |
| GA    |       0 |      0 |        0 |  2823939 |        0 |        0 |       0 |     0 |
0 |    0 |   2823939 |
| HI    |   43050 | 752983 |        0 |        0 |        0 |        0 |       0 |     0 |
0 |    0 |    796033 |
...
| WY    |       0 |      0 |        0 |        0 |        0 |   277480 |       0 |     0 |
0 |    0 |    277480 |
| NULL  | 1792991 | 787689 | 16227033 | 44213344 | 47460670 | 61110822 | 7105143 | 60607 |
360765 | 4336 | 179123400 |
+-------+---------+--------+----------+----------+----------+----------+---------+-------+---
-----+------+-----------+
```

Notice how Alaska (AK) has populations in high latitudes and Hawaii (HI) in low latitudes.

Example 2 - Home Solar Power Generation

This give the power (KWh) generated by hour and month for 2012.

```
-- Sample input:
+---------------------+------+
| ts                  | enwh |
+---------------------+------+
| 2012-06-06 11:00:00 |  523 |
| 2012-06-06 11:05:00 |  526 |
| 2012-06-06 11:10:00 |  529 |
| 2012-06-06 11:15:00 |  533 |
| 2012-06-06 11:20:00 |  537 |
| 2012-06-06 11:25:00 |  540 |
| 2012-06-06 11:30:00 |  542 |
| 2012-06-06 11:35:00 |  543 |
Note that it is a reading in watts for each 5 minutes.
So, summing is needed to get the breakdown by month and hour.

-- Invoke the SP:
CALL Pivot('details',    -- Table
           'MONTH(ts)',  -- `base_cols`, to put on left; SUM up over the month
           'HOUR(ts)',   -- `pivot_col` to put across the top; SUM up entries across the hour
           'enwh/1000',  -- The data -- watts converted to KWh
           "WHERE ts >= '2012-01-01' AND ts < '2012-01-01' + INTERVAL 1 year",  -- Limit to one
year
           '');          -- assumes that the months stay in order

-- The SQL generated:
SELECT MONTH(ts),
SUM(IF(HOUR(ts) = "5", enwh/1000, 0)) AS "5",
SUM(IF(HOUR(ts) = "6", enwh/1000, 0)) AS "6",
SUM(IF(HOUR(ts) = "7", enwh/1000, 0)) AS "7",
SUM(IF(HOUR(ts) = "8", enwh/1000, 0)) AS "8",
SUM(IF(HOUR(ts) = "9", enwh/1000, 0)) AS "9",
SUM(IF(HOUR(ts) = "10", enwh/1000, 0)) AS "10",
SUM(IF(HOUR(ts) = "11", enwh/1000, 0)) AS "11",
SUM(IF(HOUR(ts) = "12", enwh/1000, 0)) AS "12",
SUM(IF(HOUR(ts) = "13", enwh/1000, 0)) AS "13",
SUM(IF(HOUR(ts) = "14", enwh/1000, 0)) AS "14",
SUM(IF(HOUR(ts) = "15", enwh/1000, 0)) AS "15",
SUM(IF(HOUR(ts) = "16", enwh/1000, 0)) AS "16",
SUM(IF(HOUR(ts) = "17", enwh/1000, 0)) AS "17",
SUM(IF(HOUR(ts) = "18", enwh/1000, 0)) AS "18",
SUM(IF(HOUR(ts) = "19", enwh/1000, 0)) AS "19",
SUM(IF(HOUR(ts) = "20", enwh/1000, 0)) AS "20",
 SUM(enwh/1000) AS Total
 FROM details WHERE ts >= '2012-01-01' AND ts < '2012-01-01' + INTERVAL 1 year GROUP BY
MONTH(ts)
 WITH ROLLUP

-- That generated decimal places that I did like:
| MONTH(ts) | 5      | 6       | 7        | 8        | 9         | 10        | 11        | 12
| 13        | 14
     | 15        | 16       | 17       | 18       | 19      | 20     | Total      |
+-----------+--------+---------+----------+----------+-----------+-----------+-----------+---
--------+-----------+------
-----+-----------+----------+----------+----------+---------+--------+------------+
|         1 | 0.0000 |  0.0000 |   1.8510 |  21.1620 |   52.3190 |   73.0420 |   89.3220 |
97.0190 |   88.9720 |   75.
4970 |   50.9270 |  12.5130 |   0.5990 |   0.0000 |  0.0000 | 0.0000 |   563.2230 |
|         2 | 0.0000 |  0.0460 |   5.9560 |  35.6330 |   72.4710 |   96.5130 |  112.7770 |
126.0850 |  117.1540 |   96.
7160 |   72.5900 |  33.6230 |   4.7650 |   0.0040 |  0.0000 | 0.0000 |   774.3330 |
```

Other variations made the math go wrong. (Note that there is no CAST to FLOAT.)

While I was at it, I gave an alias to change "MONTH(ts)" to just "Month".

So, I edited the SQL to this and ran it:

```
SELECT MONTH(ts) AS 'Month',
ROUND(SUM(IF(HOUR(ts) = "5", enwh, 0))/1000) AS "5",
...
ROUND(SUM(IF(HOUR(ts) = "20", enwh, 0))/1000) AS "20",
 ROUND(SUM(enwh)/1000) AS Total
 FROM details WHERE ts >= '2012-01-01' AND ts < '2012-01-01' + INTERVAL 1 year
 GROUP BY MONTH(ts)
 WITH ROLLUP;
```

-- Which gave cleaner output:

```
+-------+------+------+------+------+------+------+------+------+------+------+------+------
+------+------+------+------+--------+
| Month | 5    | 6    | 7    | 8    | 9    | 10   | 11   | 12   | 13   | 14   | 15   | 16   |
17   | 18   | 19   | 20   | Total |
+-------+------+------+------+------+------+------+------+------+------+------+------+------
+------+------+------+------+--------+
|     1 |    0 |    0 |    2 |   21 |   52 |   73 |   89 |   97 |   89 |   75 |   51 |   13 |
1 |    0 |    0 |    0 |   563 |
|     2 |    0 |    0 |    6 |   36 |   72 |   97 |  113 |  126 |  117 |   97 |   73 |   34 |
5 |    0 |    0 |    0 |   774 |
|     3 |    0 |    0 |    9 |   46 |   75 |  105 |  121 |  122 |  128 |  126 |  105 |   71 |
33 |   10 |    0 |    0 |   952 |
|     4 |    0 |    1 |   14 |   63 |  111 |  146 |  171 |  179 |  177 |  158 |  141 |  105 |
65 |   26 |    3 |    0 |  1360 |
|     5 |    0 |    4 |   21 |   78 |  128 |  162 |  185 |  199 |  196 |  187 |  166 |  130 |
81 |   36 |    8 |    0 |  1581 |
|     6 |    0 |    4 |   17 |   71 |  132 |  163 |  182 |  191 |  193 |  182 |  161 |  132 |
89 |   43 |   10 |    1 |  1572 |
|     7 |    0 |    3 |   17 |   57 |  121 |  160 |  185 |  197 |  199 |  189 |  168 |  137 |
92 |   44 |   11 |    1 |  1581 |
|     8 |    0 |    1 |   11 |   48 |  104 |  149 |  171 |  183 |  187 |  179 |  156 |  121 |
76 |   32 |    5 |    0 |  1421 |
|     9 |    0 |    0 |    6 |   32 |   77 |  127 |  151 |  160 |  159 |  148 |  124 |   93 |
47 |   12 |    1 |    0 |  1137 |
|    10 |    0 |    0 |    1 |   16 |   54 |   85 |  107 |  115 |  119 |  106 |   85 |   56 |
17 |    2 |    0 |    0 |   763 |
|    11 |    0 |    0 |    5 |   30 |   57 |   70 |   84 |   83 |   76 |   64 |   35 |    8 |
1 |    0 |    0 |    0 |   512 |
|    12 |    0 |    0 |    2 |   17 |   39 |   54 |   67 |   75 |   64 |   58 |   31 |    4 |
0 |    0 |    0 |    0 |   411 |
|  NULL |    0 |   13 |  112 |  516 | 1023 | 1392 | 1628 | 1728 | 1703 | 1570 | 1294 |  902 |
506 |  203 |   38 |    2 | 12629 |
+-------+------+------+------+------+------+------+------+------+------+------+------+------
+------+------+------+------+--------+
```

Midday in the summer is the best time for solar panels, as you would expect. 1-2pm in July was the best.

Postlog

Posted, Feb. 2015

3.3.4.33 Rollup Unique User Counts

Contents

The Problem

The normal way to count "Unique Users" is to take large log files, sort by userid, dedup, and count. This requires a rather large amount of processing. Furthermore, the count derived cannot be rolled up. That is, daily counts cannot be added to get weekly counts -- some users will be counted multiple times.

So, the problem is to store the counts is such a way as to allow rolling up.

The solution

Let's think about what we can do with a hash of the userid. The hash could map to a bit in a bit string. A BIT_COUNT of the bit string would give the 1-bits, representing the number of users. But that bit string would have to be huge. What if we could use shorter bit strings? Then different userids would be folded into the same bit. Let's assume we can solve that.

Meanwhile, what about the rollup? The daily bit strings can be OR'd together to get a similar bit string for the week.

We have now figured out how to do the rollup, but have created another problem -- the counts are too low.

Inflating the BIT_COUNT

A sufficiently random hash (eg MD5) will fold userids into the same bits with a predictable frequency. We need to figure this out, and work backwards. That is, given that X percent of the bits are set, we need a formula that says approximately how many userids were used to get those bits.

I simulated the problem by generating random hashes and calculated the number of bits that would be set. Then, with the help of Eureqa software, I derived the formula:

Y = 0.5456*X + 0.6543*tan(1.39*X*X*X)

How good is it?

The formula is reasonably precise. It is usually within 1% of the correct value; rarely off by 2%.

Of course, if virtually all the bits are set, the forumla can't be very precise. Hence, you need to plan to have the bit strings big enough to handle the expected number of Uniques. In practice, you can use less than 1 bit per Unique. This would be a huge space savings over trying to save all the userids.

Another suggestion... If you are rolling up over a big span of time (eg hourly -> monthly), the bit strings must all be the same length, and the monthly string must be big enough to handle the expected count. This is likely to lead to very sparse hourly bit strings. Hence, it may be prudent to compress the hourly stings.

Postlog

Invented Nov, 2013; published Apr, 2014

Future: Rick is working on actual code (Sep, 2016) It is complicated by bit-wise operations being limited to BIGINT. However, with MySQL 8.0 (freshly released), the desired bit-wise operations can be applied to BLOB, greatly simplifying my code. I hope to publish the pre-8.0 code soon; 8.0 code later.

3.3.4.34 Rowid Filtering Optimization

MariaDB starting with 10.4

Rowid filtering is an optimization available from MariaDB 10.4.

Contents

The target use case for rowid filtering is as follows:

- a table uses ref access on index IDX1
- but it also has a fairly restrictive range predicate on another index IDX2.

In this case, it is advantageous to:

- Do an index-only scan on index IDX2 and collect rowids of index records into a data structure that allows filtering (let's call it $FILTER).
- When doing ref access on IDX1, check $FILTER before reading the full record.

Example

Consider a query

```
SELECT ...
FROM orders JOIN lineitem ON o_orderkey=l_orderkey
WHERE
  l_shipdate BETWEEN '1997-01-01' AND '1997-01-31' AND
  o_totalprice between 200000 and 230000;
```

Suppose the condition on `l_shipdate` is very restrictive, which means lineitem table should go first in the join order. Then, the optimizer can use `o_orderkey=l_orderkey` equality to do an index lookup to get the order the line item is from. On the other hand `o_totalprice between ...` can also be rather selective.

With filtering, the query plan would be:

```
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: lineitem
         type: range
possible_keys: PRIMARY,i_l_shipdate,i_l_orderkey,i_l_orderkey_quantity
          key: i_l_shipdate
      key_len: 4
          ref: NULL
         rows: 98
        Extra: Using index condition
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: orders
         type: eq_ref|filter
possible_keys: PRIMARY,i_o_totalprice
          key: PRIMARY|i_o_totalprice
      key_len: 4|9
          ref: dbt3_s001.lineitem.l_orderkey
         rows: 1 (5%)
        Extra: Using where; Using rowid filter
```

Note that table `orders` has "Using rowid filter". The `type` column has `"|filter"`, the `key` column shows the index that is used to construct the filter. `rows` column shows the expected filter selectivity, it is 5%.

ANALYZE FORMAT=JSON output for table orders will show

```
    "table": {
      "table_name": "orders",
      "access_type": "eq_ref",
      "possible_keys": ["PRIMARY", "i_o_totalprice"],
      "key": "PRIMARY",
      "key_length": "4",
      "used_key_parts": ["o_orderkey"],
      "ref": ["dbt3_s001.lineitem.l_orderkey"],
      "rowid_filter": {
        "range": {
          "key": "i_o_totalprice",
          "used_key_parts": ["o_totalprice"]
        },
        "rows": 69,
        "selectivity_pct": 4.6,
        "r_rows": 71,
        "r_selectivity_pct": 10.417,
        "r_buffer_size": 53,
        "r_filling_time_ms": 0.0716
      }
```

Note the `rowid_filter` element. It has a `range` element inside it. `selectivity_pct` is the expected selectivity, accompanied by the `r_selectivity_pct` showing the actual observed selectivity.

Details

- The optimizer makes a cost-based decision about when the filter should be used.
- The filter data structure is currently an ordered array of rowids. (a Bloom filter would be better here and will probably be introduced in the future versions).

- The optimization needs to be supported by the storage engine. At the moment, it is supported by InnoDB and MyISAM. It is not supported in partitioned tables.

Control

Rowid filtering can be switched on/off using `rowid_filter` flag in the optimizer_switch variable. By default, the optimization is enabled.

3.3.4.35 Sargable UPPER

Starting from MariaDB 11.3, expressions in the form

```
UPPER(key_col) = expr
UPPER(key_col) IN (constant-list)
```

are sargable if `key_col` uses either the `utf8mb3_general_ci` or `utf8mb4_general_ci` collation.

`UCASE` is a synonym for UPPER so is covered as well.

Sargable means that the optimizer is able to use such conditions to construct access methods, estimate their selectivity, or perform partition pruning.

Example

```
create table t1 (
  key1 varchar(32) collate utf8mb4_general_ci,
  ...
  key(key1)
);
```

```
explain select * from t1 where UPPER(key1)='ABC'
+------+-------------+-------+------+---------------+------+---------+-------+------+-------------
| id   | select_type | table | type | possible_keys | key  | key_len | ref   | rows | Extra
+------+-------------+-------+------+---------------+------+---------+-------+------+-------------
|    1 | SIMPLE      | t1    | ref  | key1          | key1 | 131     | const | 1    | Using where
+------+-------------+-------+------+---------------+------+---------+-------+------+-------------
```

Note that `ref` access is used.

An example with join:

```
explain select * from t0,t1 where upper(t1.key1)=t0.col;
+------+-------------+-------+------+---------------+------+---------+-------------+------+------
| id   | select_type | table | type | possible_keys | key  | key_len | ref         | rows | Extra
+------+-------------+-------+------+---------------+------+---------+-------------+------+------
|    1 | SIMPLE      | t0    | ALL  | NULL          | NULL | NULL    | NULL        | 10   | Using
|    1 | SIMPLE      | t1    | ref  | key1          | key1 | 131     | test.t0.col | 1    | Using
+------+-------------+-------+------+---------------+------+---------+-------------+------+------
```

Here, the optimizer was able to construct `ref` access.

Controlling the Optimization

The optimizer_switch variable has the flag `sargable_casefold` to turn the optimization on and off. The default is ON.

Optimizer Trace

The optimization is implemented as a rewrite for a query's WHERE/ON conditions. It uses the `sargable_casefold_removal` object name in the trace:

```
      "join_optimization": {
        "select_id": 1,
        "steps": [
          {
            "sargable_casefold_removal": {
              "before": "ucase(t1.key1) = t0.col",
              "after": "t1.key1 = t0.col"
            }
          },
```

References

- MDEV-31496: Make optimizer handle UCASE(varchar_col)=...
- An analog for LCASE is not possible. See MDEV-31955: Make optimizer handle LCASE(varchar_col)=... for details.

3.3.4.36 USE INDEX

You can limit which indexes are considered with the `USE INDEX` option.

Syntax

```
USE INDEX [{FOR {JOIN|ORDER BY|GROUP BY}] ([index_list])
```

Contents

Description

The default is 'FOR JOIN', which means that the hint only affects how the WHERE clause is optimized.

USE INDEX is used after the table name in the FROM clause.

USE INDEX cannot use an ignored index - it will be treated as if it doesn't exist.

Index Prefixes

When using index hints (USE, FORCE or IGNORE INDEX), the index name value can also be an unambiguous prefix of an index name.

Example

```
CREATE INDEX Name ON City (Name);
CREATE INDEX CountryCode ON City (Countrycode);
EXPLAIN SELECT Name FROM City USE INDEX (CountryCode)
WHERE name="Helsingborg" AND countrycode="SWE";
```

This produces:

```
id select_type table type possible_keys key key_len ref rows Extra
1 SIMPLE City ref CountryCode CountryCode 3 const 14 Using where
```

If we had not used USE INDEX, the Name index would have been in possible keys.

3.3.5 Optimizing Tables

Different ways to optimize tables and data on disk

OPTIMIZE TABLE
Reclaim unused space and defragment data.

ANALYZE TABLE
Store key distributions for a table.

Choosing the Right Storage Engine
Quickly choose the most suitable storage engine for your needs.

Converting Tables from MyISAM to InnoDB
Issues when converting tables from MyISAM to InnoDB.

Histogram-Based Statistics
Histogram-based statistics can improve the optimizer query plan in certain situations.

Defragmenting InnoDB Tablespaces
MariaDB 10.1.1 introduced a feature to defragment InnoDB tablespaces.

Entity-Attribute-Value Implementation
A common, poorly performing, design pattern (EAV); plus an alternative

IP Range Table Performance
IP Range Table Performance Improvements

There are 1 related questions.

1.1.1.2.1.11 OPTIMIZE TABLE

1.1.1.2.1.2 ANALYZE TABLE

5.3.1 Choosing the Right Storage Engine

5.3.23 Converting Tables from MyISAM to InnoDB

3.3.4.6.2 Histogram-Based Statistics

3.3.5.6 Defragmenting InnoDB Tablespaces

Contents

Overview

When rows are deleted from an InnoDB table, the rows are simply marked as deleted and not physically deleted. The free space is not returned to the operating system for re-use.

The purge thread will physically delete index keys and rows, but the free space introduced is still not returned to operating system. This can lead to gaps in the pages. If you have variable length rows, new rows may be larger than old rows and cannot make use of the available space.

You can run OPTIMIZE TABLE or ALTER TABLE <table> ENGINE=InnoDB to reconstruct the table. Unfortunately running `OPTIMIZE TABLE` against an InnoDB table stored in the shared table-space file `ibdata1` does two things:

- Makes the table's data and indexes contiguous inside `ibdata1` .
- Increases the size of `ibdata1` because the contiguous data and index pages are appended to `ibdata1` .

InnoDB Defragmentation

> The feature described below has been deprecated in MariaDB 11.0 and was removed in MariaDB 11.1.0. See MDEV-30544 and MDEV-30545.

MariaDB 10.1 merged Facebook's defragmentation code prepared for MariaDB by Matt, Seong Uck Lee from Kakao. The only major difference to Facebook's code and Matt's patch is that MariaDB does not introduce new literals to SQL and makes no changes to the server code. Instead, OPTIMIZE TABLE is used and all code changes are inside the InnoDB/XtraDB storage engines.

The behaviour of `OPTIMIZE TABLE` is unchanged by default, and to enable this new feature, you need to set the innodb_defragment system variable to `1`.

```
[mysqld]
...
innodb-defragment=1
```

No new tables are created and there is no need to copy data from old tables to new tables. Instead, this feature loads `n` pages (determined by innodb-defragment-n-pages) and tries to move records so that pages would be full of records and then frees pages that are fully empty after the operation.

Note that tablespace files (including ibdata1) will not shrink as the result of defragmentation, but one will get better memory utilization in the InnoDB buffer pool as there are fewer data pages in use.

A number of new system and status variables for controlling and monitoring the feature are introduced.

System Variables

- innodb_defragment: Enable InnoDB defragmentation.
- innodb_defragment_n_pages: Number of pages considered at once when merging multiple pages to defragment.
- innodb_defragment_stats_accuracy: Number of defragment stats changes there are before the stats are written to persistent storage.
- innodb_defragment_fill_factor_n_recs: Number of records of space that defragmentation should leave on the page.
- innodb_defragment_fill_factor: Indicates how full defragmentation should fill a page.
- innodb_defragment_frequency: Maximum times per second for defragmenting a single index.

Status Variables

- Innodb_defragment_compression_failures: Number of defragment re-compression failures
- Innodb_defragment_failures: Number of defragment failures.
- Innodb_defragment_count: Number of defragment operations.

Example

```
set @@global.innodb_file_per_table = 1;
set @@global.innodb_defragment_n_pages = 32;
set @@global.innodb_defragment_fill_factor = 0.95;
CREATE TABLE tb_defragment (
pk1 bigint(20) NOT NULL,
pk2 bigint(20) NOT NULL,
fd4 text,
fd5 varchar(50) DEFAULT NULL,
PRIMARY KEY (pk1),
KEY ix1 (pk2)
) ENGINE=InnoDB;

delimiter //
create procedure innodb_insert_proc (repeat_count int)
begin
  declare current_num int;
  set current_num = 0;
  while current_num < repeat_count do
    INSERT INTO tb_defragment VALUES (current_num, 1, REPEAT('Abcdefg', 20),
REPEAT('12345',5));
    INSERT INTO tb_defragment VALUES (current_num+1, 2, REPEAT('HIJKLM', 20),
REPEAT('67890',5));
    INSERT INTO tb_defragment VALUES (current_num+2, 3, REPEAT('HIJKLM', 20),
REPEAT('67890',5));
    INSERT INTO tb_defragment VALUES (current_num+3, 4, REPEAT('HIJKLM', 20),
REPEAT('67890',5));
    set current_num = current_num + 4;
  end while;
end//
delimiter ;
commit;

set autocommit=0;
call innodb_insert_proc(50000);
commit;
set autocommit=1;
```

After these CREATE and INSERT operations, the following information can be seen from the INFORMATION SCHEMA:

```
select count(*) as Value from information_schema.innodb_buffer_page
  where table_name like '%tb_defragment%' and index_name = 'PRIMARY';
Value
313

select count(*) as Value from information_schema.innodb_buffer_page
  where table_name like '%tb_defragment%' and index_name = 'ix1';
Value
72

select count(stat_value) from mysql.innodb_index_stats
  where table_name like '%tb_defragment%' and stat_name in ('n_pages_freed');
count(stat_value)
0

select count(stat_value) from mysql.innodb_index_stats
  where table_name like '%tb_defragment%' and stat_name in ('n_page_split');
count(stat_value)
0

select count(stat_value) from mysql.innodb_index_stats
  where table_name like '%tb_defragment%' and stat_name in ('n_leaf_pages_defrag');
count(stat_value)
0

SELECT table_name, data_free/1024/1024 AS data_free_MB, table_rows FROM
information_schema.tables
  WHERE engine LIKE 'InnoDB' and table_name like '%tb_defragment%';
table_name data_free_MB table_rows
tb_defragment 4.00000000 50051

SELECT table_name, index_name, sum(number_records), sum(data_size) FROM
information_schema.innodb_buffer_page
  where table_name like '%tb_defragment%' and index_name like 'PRIMARY';
table_name index_name sum(number_records) sum(data_size)
`test`.`tb_defragment` PRIMARY 25873 4739939

SELECT table_name, index_name, sum(number_records), sum(data_size) FROM
information_schema.innodb_buffer_page
  where table_name like '%tb_defragment%' and index_name like 'ix1';
table_name index_name sum(number_records) sum(data_size)
`test`.`tb_defragment` ix1 50071 1051775
```

Deleting three-quarters of the records, leaving gaps, and then optimizing:

```
delete from tb_defragment where pk2 between 2 and 4;

optimize table tb_defragment;
Table Op Msg_type Msg_text
test.tb_defragment optimize status OK
show status like '%innodb_def%';
Variable_name Value
Innodb_defragment_compression_failures 0
Innodb_defragment_failures 1
Innodb_defragment_count 4
```

Now some pages have been freed, and some merged:

```
select count(*) as Value from information_schema.innodb_buffer_page
  where table_name like '%tb_defragment%' and index_name = 'PRIMARY';
Value
0

select count(*) as Value from information_schema.innodb_buffer_page
  where table_name like '%tb_defragment%' and index_name = 'ix1';
Value
0

select count(stat_value) from mysql.innodb_index_stats
  where table_name like '%tb_defragment%' and stat_name in ('n_pages_freed');
count(stat_value)
2

select count(stat_value) from mysql.innodb_index_stats
  where table_name like '%tb_defragment%' and stat_name in ('n_page_split');
count(stat_value)
2

select count(stat_value) from mysql.innodb_index_stats
  where table_name like '%tb_defragment%' and stat_name in ('n_leaf_pages_defrag');
count(stat_value)
2

SELECT table_name, data_free/1024/1024 AS data_free_MB, table_rows FROM
information_schema.tables
  WHERE engine LIKE 'InnoDB';
table_name data_free_MB table_rows
innodb_index_stats 0.00000000 8
innodb_table_stats 0.00000000 0
tb_defragment 4.00000000 12431

SELECT table_name, index_name, sum(number_records), sum(data_size) FROM
information_schema.innodb_buffer_page
  where table_name like '%tb_defragment%' and index_name like 'PRIMARY';
table_name index_name sum(number_records) sum(data_size)
`test`.`tb_defragment` PRIMARY 690 102145

SELECT table_name, index_name, sum(number_records), sum(data_size) FROM
information_schema.innodb_buffer_page
  where table_name like '%tb_defragment%' and index_name like 'ix1';
table_name index_name sum(number_records) sum(data_size)
`test`.`tb_defragment` ix1 5295 111263
```

See Defragmenting unused space on InnoDB tablespace on the Mariadb.org blog for more details.

3.3.5.7 Entity-Attribute-Value Implementation

Contents

The desires

- Open-ended set of "attributes" (key=value) for each "entity". That is, the list of attributes is not known at development time, and will grow in the future. (This makes one column per attribute impractical.)
- "ad hoc" queries testing attributes.
- Attribute values come in different types (numbers, strings, dates, etc.)
- Scale to lots of entities, yet perform well.

It goes by various names

- EAV -- Entity - Attribute - Value
- key-value
- RDF -- This is a flavor of EAV
- MariaDB has dynamic columns that look something like the solution below, with the added advantage of being able to index the columns otherwise hidden in the blob. (There are caveats.)
- MySQL 5.7 Has JSON datatype, plus functions to access parts
- MongoDB, CouchDB -- and others -- Not SQL-based.

Bad solution

- Table with 3 columns: entity_id, key, value
- The "value" is a string, or maybe multiple columns depending on datatype or other kludges.
- a JOIN b ON a.entity=b.entity AND b.key='x' JOIN c ON ... WHERE a.value=... AND b.value=...

The problems

- The SELECTs get messy -- multiple JOINs
- Datatype issues -- It's clumsy to be putting numbers into strings
- Numbers stored in VARCHAR do not compare 'correctly', especially for range tests.
- Bulky.
- Dedupping the values is clumsy.

A solution

Decide which columns need to be searched/sorted by SQL queries. No, you don't need all the columns to be searchable or sortable. Certain columns are frequently used for selection; identify these. You probably won't use all of them in all queries, but you will use some of them in every query.

The solution uses one table for all the EAV stuff. The columns include the searchable fields plus one BLOB. Searchable fields are declared appropriately (INT, TIMESTAMP, etc). The BLOB contains JSON-encoding of all the extra fields.

The table should be InnoDB, hence it should have a PRIMARY KEY. The entitity_id is the 'natural' PK. Add a small number of other indexes (often 'composite') on the searchable fields. PARTITIONing is unlikely to be of any use, unless the Entities should purged after some time. (Example: News Articles)

But what about the ad hoc queries?

You have included the most important fields to search on -- date, category, etc. These should filter the data down significantly. When you also need to filter on something more obscure, that will be handled differently. The application code will look at the BLOB for that; more on this later.

Why it works

- You are not really going to search on more than a few fields.
- The disk footprint is smaller; Smaller --> More cacheable --> Faster
- It needs no JOINs
- The indexes are useful
- The one table has one row per entity, and can grow as needed. (EAV needs many rows per entity.)
- Performance is as good as the indexes you have on the 'searchable fields'.
- Optionally, you can duplicate the indexed fields in the BLOB.
- Values missing from 'searchable fields' would need to be NULL (or whatever), and the code would need to deal with such.

Details on the BLOB/JSON

- Build the extra (or all) key-value pairs in a hash (associative array) in your application. Encode it. COMPRESS it. Insert that string into the BLOB.
- JSON is recommended, but not mandatory; it is simpler than XML. Other serializations (eg, YAML) could be used.
- COMPRESS the JSON and put it into a BLOB (or MEDIUMBLOB) instead of a TEXT field. Compression gives about 3x shrinkage.
- When SELECTing, UNCOMPRESS the blob. Decode the string into a hash. You are now ready to interrogate/display any of the extra fields.
- If you choose to use the JSON features of MariaDB or 5.7, you will have to forgo the compression feature described.

- MySQL 5.7.8's JSON native JSON datatype uses a binary format for more efficient access.

Conclusions

- Schema is reasonably compact (compression, real datatypes, less redundancy, etc, than EAV)
- Queries are fast (since you have picked 'good' indexes)
- Expandable (JSON is happy to have new fields)
- Compatible (No 3rd party products, just supported products)
- Range tests work (unlike storing INTs in VARCHARs)
- (Drawback) Cannot use the non-indexed attributes in WHERE or ORDER BY clauses, must deal with that in the app. (MySQL 5.7 partially alleviates this.)

Postlog

Posted Jan, 2014; Refreshed Feb, 2016.

- MariaDB's Dynamic Columns
- MySQL 5.7's JSON

This looks very promising; I will need to do more research to see how much of this article is obviated by it: Using MySQL as a Document Store in 5.7, more DocStore discussion

If you insist on EAV, set optimizer_search_depth=1.

3.3.5.8 IP Range Table Performance

Contents

The situation

Your data includes a large set of non-overlapping 'ranges'. These could be IP addresses, datetimes (show times for a single station), zipcodes, etc.

You have pairs of start and end values; one 'item' belongs to each such 'range'. So, instinctively, you create a table with start and end of the range, plus info about the item. Your queries involve a WHERE clause that compares for being between the start and end values.

The problem

Once you get a large set of items, performance degrades. You play with the indexes, but find nothing that works well. The indexes fail to lead to optimal functioning because the database does not understand that the ranges are non-overlapping.

The solution

I will present a solution that enforces the fact that items cannot have overlapping ranges. The solution builds a table to take advantage of that, then uses Stored Routines to get around the clumsiness imposed by it.

Performance

The instinctive solution often leads to scanning half the table to do just about anything, such as finding the item containing an 'address'. In complexity terms, this is Order(N).

The solution here can usually get the desired information by fetching a single row, or a small number of rows. It is Order(1).

In a large table, "counting the disk hits" is the important part of performance. Since InnoDB is used, and the PRIMARY KEY (clustered) is used, most operations hit only 1 block.

Finding the 'block' where a given IP address lives:

- For start of block: One single-row fetch using the PRIMARY KEY
- For end of block: Ditto. The record containing this will be 'adjacent' to the other record.

For allocating or freeing a block:

- 2-7 SQL statements, hitting the clustered PRIMARY KEY for the rows containing and immediately adjacent to the block.
- One SQL statement is a DELETE; if hits as many rows as are needed for the block.
- The other statements hit one row each.

Design decisions

This is crucial to the design and its performance:

- Having just one address in the row. These were alternative designs; they seemed to be no better, and possibly worse:
- That one address could have been the 'end' address.
- The routine parameters for a 'block' could have be start of this block and start of next block.
- The IPv4 parameters could have been dotted quads; I chose to keep the reference implemetation simpler instead.
- The IPv6 parameters are 32-digit hex because it was the simpler that BINARY(16) or IPv5 for a reference implementation.

The interesting work is in the Ips, not the second table, so I focus on it. The inconvenience of JOINing to the second table is small compared to the performance gains.

Details

Two, not one, tables will be used. The first table (`Ips` in the reference implementations) is carefully designed to be optimal for all the basic operations needed. The second table contains other infomation about the 'owner' of each 'item'. In the reference implementations `owner` is an id used to JOIN the two tables. This discussion centers around `Ips` and how to efficiently map IP(s) to/from owner(s). The second table has "PRIMARY KEY(owner)".

In addition to the two-table schema, there are a set of Stored Routines to encapsulate the necessary code.

One row of Ips represents one 'item' by specifying the starting IP address and the 'owner'. The next row gives the starting IP address of the next "address block", thereby indirectly providing the ending address for the current block.

This lack of explicitly stating the "end address" leads to some clumsiness. The stored routines hide it from the user.

A special owner (indicated by '0') is reserved for "free" or "not-owned" blocks. Hence, sparse allocation of address blocks is no problem. Also, the 'free' owner is handled no differently than real owners, so there are no extra Stored Routines for such.

Links below give "reference" implementations for IPv4 and IPv6. You will need to make changes for non-IP situations, and may need to make changes even for IP situations.

These are the main stored routines provided:

- IpIncr, IpDecr -- for adding/subtracting 1
- IpStore -- for allocating/freeing a range
- IpOwner, IpRangeOwners, IpFindRanges, Owner2IpStarts, Owner2IpRanges -- for lookups
- IpNext, IpEnd -- IP of start of next block, or end of current block

None of the provided routines JOIN to the other table; you may wish to develop custom queries based on the given reference Stored Procedures.

The Ips table's size is proportional to the number of blocks. A million 'owned' blocks may be 20-50MB. This varies due to

- number of 'free' gaps (between zero and the number of owned blocks)
- datatypes used for `ip` and `owner`
- InnoDB overhead Even 100M blocks is quite manageable in today's hardware. Once things are cached, most operations would take only a few milliseconds. A trillion blocks would work, but most operations would hit the disk a few times -- only a few times.

Reference implementation of IPv4

This specific to IPv4 (32 bit, a la '196.168.1.255'). It can handle anywhere from 'nothing assigned' (1 row) to 'everything assigned' (4B rows) 'equally' well. That is, to ask the question "who owns '11.22.33.44'" is equally efficient regardless of how many blocks of IP addresses exist in the table. (OK, caching, disk hits, etc may make a slight difference.) The one function that can vary is the one that reassigns a range to a new owner. Its speed is a function of how many existing ranges need to be consumed, since those rows will be DELETEd. (It helps that they are, by schema design, 'clustered'.)

Notes on the Reference implementation for IPv4 :

- Externally, the user may use the dotted quad notation (11.22.33.44), but needs to convert to INT UNSIGNED for calling the Stored Procs.
- The user is responsible for converting to/from the calling datatype (INT UNSIGNED) when accessing the stored routine; suggest INET_ATON/INET_NTOA.
- The internal datatype for addresses is the same as the calling datatype (INT UNSIGNED).
- Adding and subtracting 1 (simple arithmetic).
- The datatype of an 'owner' (MEDIUMINT UNSIGNED: 0..16M) -- adjust if needed.
- The address "Off the end" (255.255.255.255+1 - represented as NULL).
- The table is initialized to one row: (ip=0, owner=0), meaning "all addresses are free See the comments in the code for more details.

(The reference implementation does not handle CDRs. Such should be easy to add on, by first turning it into an IP range.)

Reference implementation of IPv6

The code for handling IP address is more complex, but the overall structure is the same as for IPv4. Launch into it only if you need IPv6.

Notes on the reference implementation for IPv6:

- Externally, IPv6 has a complex string, VARCHAR(39) CHARACTER SET ASCII. The Stored Procedure IpStr2Hex() is provided.
- The user is responsible for converting to/from the calling datatype (BINARY(16)) when accessing the stored routine; suggest INET6_ATON/INET6_NTOA.
- The internal datatype for addresses is the same as the calling datatype (BINARY(16)).
- Communication with the Stored routines is via 32-char hex strings.
- Inside the Procedures, and in the Ips table, an address is stored as BINARY(16) for efficiency. HEX() and UNHEX() are used at the boundaries.
- Adding/subtracting 1 is rather complex (see the code).
- The datatype of an 'owner' (MEDIUMINT UNSIGNED: 0..16M); 'free' is represented by 0. You may need a bigger datatype.
- The address "Off the end" (ffff.ffff.ffff.ffff.ffff.ffff.ffff.ffff+1 is represented by NULL).
- The table is initialized to one row: (UNHEX('00000000000000000000000000000000'), 0), meaning "all addresses are free.
- You may need to decide on a canonical representation of IPv4 in IPv6. See the comments in the code for more details.

The INET6* functions were first available in MySQL 5.6.3 and MariaDB 10.0.3

Adapting to a different non-IP 'address range' data

- The external datatype for an 'address' should be whatever is convenient for the application.
- The datatype for the 'address' in the table must be ordered, and should be as compact as possible.
- You must write the Stored functions (IpIncr, IpDecr) for incrementing/decrementing an 'address'.
- An 'owner' is an id of your choosing, but smaller is better.
- A special value (such as 0 or '') must be provided for 'free'.
- The table must be initialized to one row: (SmallestAddress, Free)

"Owner" needs a special value to represent "not owned". The reference implementations use "=" and "!=" to compare two 'owners'. Numeric values and strings work nicely with those operators; NULL does not. Hence, please do not use NULL for "not owned".

Since the datatypes are pervasive in the stored routines, adapting a reference implementation to a different concept of 'address' would require multiple minor changes.

The code enforces that consecutive blocks never have the same 'owner', so the table is of 'minimal' size. Your application can assume that such is always the case.

Postlog

Original writing -- Oct, 2012; Notes on INET6 functions -- May, 2015.

3.3.6 MariaDB Memory Allocation

Contents

Allocating RAM for MariaDB - The Short Answer

If only using MyISAM, set key_buffer_size to 20% of **available** RAM. (Plus innodb_buffer_pool_size=0)

If only using InnoDB, set innodb_buffer_pool_size to 70% of **available** RAM. (Plus key_buffer_size = 10M, small, but not zero.)

Rule of thumb for tuning:

- Start with released copy of my.cnf / my.ini.
- Change key_buffer_size and innodb_buffer_pool_size according to engine usage and RAM.
- Slow queries can usually be 'fixed' via indexes, schema changes, or SELECT changes, not by tuning.
- Don't get carried away with the query cache until you understand what it can and cannot do.
- Don't change anything else unless you run into trouble (eg, max connections).
- Be sure the changes are under the [mysqld] section, not some other section.

The 20%/70% assumes you have at least 4GB of RAM. If you have a tiny antique, or a tiny VM, then those percentages are too high.

Now for the gory details.

How to troubleshoot out-of-memory issues

If the MariaDB server is crashing because of 'out-of-memory' then it is probably wrongly configured.

There are two kind of buffers in MariaDB:

- Global ones that are only allocated once during the lifetime of the server:
  - Storage engine buffers (innodb_buffer_pool_size, key_buffer_size, aria_pagecache_buffer_size, etc)
  - Query cache query_cache_size.
- Global caches onces that grow and shrink dynamically on demand up to max limit:
  - max_user_connections
  - table_open_cache
  - table_definition_cache
  - thread_cache_size
- Local buffers that are allocated on demand whenever needed
  - Internal ones used during engine index creation (myisam_sort_buffer_size, aria_sort_buffer_size).
  - Internal buffers for storing blobs.
    - Some storage engine will keep a temporary cache to store the largest blob seen so far when scanning a table. This will be freed at end of query. Note that temporary blob storage is **not** included in the memory information in information_schema.processlist but only in the total memory used (`show global status like "memory_used"`).

- Buffers and caches used during query execution:

Variable	Description
join_buffer_size	Used when no keys can be used to find a row in next table
mrr_buffer_size	Size of buffer to use when using multi-range read with range access
net_buffer_length	Max size of network packet
read_buffer_size	Used by some storage engines when doing bulk insert
sort_buffer_size	When doing ORDER BY or GROUP BY
max_heap_table_size	Used to store temporary tables in memory. See Optimizing memory tables

If any variables in the last group is very large and you have a lot of simultaneous users that are executing queries that are using these buffers then you can run into trouble.

In a default MariaDB installation the default of most of the above variables are quite small to ensure that one does not run out of memory.

You can check which variables that have been changed in your setup by executing the following sql statement. If you are running into out-of-memory issues, it is very likely that the problematic variable is in this list!

```
select information_schema.system_variables.variable_name,
information_schema.system_variables.default_value,
global_variables.variable_value from
information_schema.system_variables,information_schema.global_variables where
system_variables.variable_name=global_variables.variable_name and
system_variables.default_value <> global_variables.variable_value and
system_variables.default_value <> 0
```

What is the Key Buffer?

MyISAM does two different things for caching.

- Index blocks (1KB each, BTree structured, from .MYI file) live in the "key buffer".
- Data block caching (from .MYD file) is left to the OS, so be sure to leave a bunch of free space for this. Caveat: Some flavors of OS always claim to be using over 90%, even when there is really lots of free space.

```
SHOW GLOBAL STATUS LIKE 'Key%';
```

then calculate Key_read_requests / Key_reads. If it is high (say, over 10), then the key buffer is big enough, otherwise you should adjust the key_buffer_size value.

What is the Buffer Pool?

InnoDB does all its caching in a the buffer pool, whose size is controlled by innodb_buffer_pool_size. By default it contains 16KB data and index blocks from the open tables (see innodb_page_size), plus some maintenance overhead.

From MariaDB 5.5, multiple buffer pools are permitted; this can help because there is one mutex per pool, thereby relieving some of the mutex bottleneck.

More on InnoDB tuning

Another Algorithm

This will set the main cache settings to the minimum; it could be important to systems with lots of other processes and/or RAM is 2GB or smaller.

Do SHOW TABLE STATUS for all the tables in all the databases.

Add up Index_length for all the MyISAM tables. Set key_buffer_size no larger than that size.

Add up Data_length + Index_length for all the InnoDB tables. Set innodb_buffer_pool_size to no more than 110% of that total.

If that leads to swapping, cut both settings back. Suggest cutting them down proportionately.

Run this to see the values for your system. (If you have a lot of tables, it can take minute(s).)

```
SELECT  ENGINE,
        ROUND(SUM(data_length) /1024/1024, 1) AS "Data MB",
        ROUND(SUM(index_length)/1024/1024, 1) AS "Index MB",
        ROUND(SUM(data_length + index_length)/1024/1024, 1) AS "Total MB",
        COUNT(*) "Num Tables"
    FROM  INFORMATION_SCHEMA.TABLES
    WHERE  table_schema not in ("information_schema", "PERFORMANCE_SCHEMA", "SYS_SCHEMA", "ndbinf
    GROUP BY  ENGINE;
```

Query Memory Allocation

There are two variables that dictates how memory are allocated by MariaDB while parsing and executing a query. query_prealloc_size defines the standard buffer for memory used for query execution and query_alloc_block_size that is size of memory blocks if `query_prealloc_size` was not big enough. Getting these variables right will reduce memory fragmentation in the server.

Mutex Bottleneck

MySQL was designed in the days of single-CPU machines, and designed to be easily ported to many different architectures. Unfortunately, that lead to some sloppiness in how to interlock actions. There are a small number (too small) of "mutexes" to gain access to several critical processes. Of note:

- MyISAM's key_buffer
- The Query Cache
- InnoDB's buffer_pool With multi-core boxes, the mutex problem is causing performance problems. In general, past 4-8 cores, MySQL gets slower, not faster. MySQL 5.5 and Percona's XtraDB made that somewhat better in InnoDB; the practical limit for cores is more like 32, and performance tends plateaus after that rather than declining. 5.6 claims to scale up to about 48 cores.

HyperThreading and Multiple Cores (CPUs)

Short answers (for older versions of MySQL and MariaDB):

- Turn off HyperThreading
- Turn off any cores beyond 8
- HyperThreading is mostly a thing of the past, so this section may not apply.

HyperThreading is great for marketing, lousy for performance. It involves having two processing units sharing a single hardware cache. If both units are doing the same thing, the cache will be reasonably useful. If the units are doing different things, they will be clobbering each other's cache entries.

Furthermore MySQL is not great on using multiple cores. So, if you turn off HT, the remaining cores run a little faster.

- 32-bit OS and MariaDB

First, the OS (and the hardware?) may conspire to not let you use all 4GB, if that is what you have. If you have more than 4GB of RAM, the excess beyond 4GB is _totally_ inaccessable and unusable on a 32-bit OS.

Secondly, the OS probably has a limit on how much RAM it will allow any process to use.

Example: FreeBSD's maxdsiz, which defaults to 512MB.

Example:

```
$ ulimit -a
...
max memory size (kbytes, -m) 524288
```

So, once you have determined how much RAM is available to mysqld, then apply the 20%/70%, but round down some.

If you get an error like `[ERROR] /usr/libexec/mysqld: Out of memory (Needed xxx bytes)` , it probably means that MySQL exceeded what the OS is willing to give it. Decrease the cache settings.

- 64-bit OS with 32-bit MariaDB

The OS is not limited by 4GB, but MariaDB is.

If you have at least 4GB of RAM, then maybe these would be good:

- key_buffer_size = 20% of _all_ of RAM, but not more than 3G
- innodb_buffer_pool_size = 3G

You should probably upgrade MariaDB to 64-bit.

- 64-bit OS and MariaDB

MyISAM only: key_buffer_size: Use about 20% of RAM. Set (in my.cnf / my.ini) innodb_buffer_pool_size=0 = 0.

InnoDB only: innodb_buffer_pool_size=0 = 70% of RAM. If you have lots of RAM and are using 5.5 (or later), then consider having multiple pools. Recommend 1-16 innodb_buffer_pool_instances, such that each one is no smaller than 1GB. (Sorry, no metric on how much this will help; probably not a lot.)

Meanwhile, set key_buffer_size = 20M (tiny, but non-zero)

If you have a mixture of engines, lower both numbers.

max_connections, thread_stack Each "thread" takes some amount of RAM. This used to be about 200KB; 100 threads would be 20MB, not a significant size. If you have max_connections = 1000, then you are talking about 200MB, maybe more. Having that many connections probably implies other issues that should be addressed.

In 5.6 (or MariaDB 5.5), optional thread pooling interacts with max_connections. This is a more advanced topic.

Thread stack overrun rarely happens. If it does, do something like thread_stack=256K

More on max_connections, wait_timeout, connection pooling, etc

table_open_cache

(In older versions this was called table_cache)

The OS has some limit on the number of open files it will let a process have. Each table needs 1 to 3 open files. Each PARTITION is effectively a table. Most operations on a partitioned table open _all_ partitions.

In *nix, ulimit tells you what the file limit is. The maximum value is in the tens of thousands, but sometimes it is set to only 1024. This limits you to about 300 tables. More discussion on ulimit

(This paragraph is in disputed.) On the other side, the table cache is (was) inefficiently implemented -- lookups were done with a linear scan. Hence, setting table_cache in the thousands could actually slow down mysql. (Benchmarks have shown this.)

You can see how well your system is performing via SHOW GLOBAL STATUS; and computing the opens/second via Opened_files / Uptime If this is more than, say, 5, table_open_cache should be increased. If it is less than, say, 1, you might get improvement by decreasing table_open_cache.

From MariaDB 10.1, table_open_cache defaults to 2000.

Query Cache

Short answer: query_cache_type = OFF and query_cache_size = 0

The Query Cache (QC) is effectively a hash mapping SELECT statements to resultsets.

Long answer... There are many aspects of the "Query cache"; many are negative.

- Novice Alert! The QC is totally unrelated to the key_buffer and buffer_pool.
- When it is useful, the QC is blazingly fast. It would not be hard to create a benchmark that runs 1000x faster.
- There is a single mutex controlling the QC.
- The QC, unless it is OFF & 0, is consulted for _every_ SELECT.
- Yes, the mutex is hit even if query_cache_type = DEMAND (2).
- Yes, the mutex is hit even for SQL_NO_CACHE.
- Any change to a query (even adding a space) leads (potentially) to a different entry in the QC.
- If my.cnf says type=ON and you later turn it OFF, it is not fully OFF. Ref: https://bugs.mysql.com/bug.php?id=60696

"Pruning" is costly and frequent:

- When _any_ write happens on a table, _all_ entries in the QC for _that_ table are removed.
- It happens even on a readonly Slave.
- Purges are performed with a linear algorithm, so a large QC (even 200MB) can be noticeably slow.

To see how well your QC is performing, SHOW GLOBAL STATUS LIKE 'Qc%'; then compute the read hit rate: Qcache_hits / Qcache_inserts If it is over, say, 5, the QC might be worth keeping.

If you decide the QC is right for you, then I recommend

- query_cache_size = no more than 50M
- query_cache_type = DEMAND
- SQL_CACHE or SQL_NO_CACHE in all SELECTs, based on which queries are likely to benefit from caching.

- Why to turn off the QC
- Discussion about size

thread_cache_size

It is not necessary to tune thread_cache_size from MariaDB 10.2.0. Previously, it was minor tunable variable. Zero will slow down thread (connection) creation. A small (say, 10), non-zero number is good. The setting has essentially no impact on RAM usage.

It is the number of extra processes to hang onto. It does not restrict the number of threads; max_connections does.

Binary Logs

If you have turned on binary logging (via log_bin) for replication and/or point-in-time recovery, the system will create binary logs forever. That is, they can slowly fill up the disk. Suggest setting expire_logs_days = 14 to keep only 14 days' worth of logs.

Swappiness

RHEL, in its infinite wisdom, decided to let you control how aggressively the OS will pre-emptively swap RAM. This is good in general, but lousy for MariaDB.

MariaDB would love for RAM allocations to be reasonably stable -- the caches are (mostly) pre-allocated; the threads, etc, are (mostly) of limited scope. ANY swapping is likely to severely hurt performance of MariaDB.

With a high value for swappiness, you lose some RAM because the OS is trying to keep a lot of space free for future allocations (that MySQL is not likely to need).

With swappiness = 0, the OS will probably crash rather than swap. I would rather have MariaDB limping than die. The latest recommendation is swappiness = 1. (2015)

More confirmation

Somewhere in between (say, 5?) might be a good value for a MariaDB-only server.

NUMA

OK, it's time to complicate the architecture of how a CPU talks to RAM. NUMA (Non-Uniform Memory Access) enters the picture. Each CPU (or maybe socket with several cores) has a part of the RAM hanging off each. This leads to memory access being faster for local RAM, but slower (tens of cycles slower) for RAM hanging off other CPUs.

Then the OS enters the picture. In at least one case (RHEL?), two things seem to be done:

- OS allocations are pinned to the 'first' CPU's RAM.]
- Other allocations go by default to the first CPU until it is full.

Now for the problem.

- The OS and MariaDB have allocated all the 'first' RAM.
- MariaDB has allocated some of the second RAM.
- The OS needs to allocate something. Ouch -- it is out of room in the one CPU where it is willing to allocate its stuff, so it swaps out some of MariaDB. Bad.

dmesg | grep -i numa # to see if you have numa

Probable solution: Configure the BIOS to "interleave" the RAM allocations. This should prevent the premature swapping, at the cost of off-CPU RAM accesses half the time. Well, you have the costly accesses anyway, since you really want to use all of RAM. Older MySQL versions: numactl --interleave=all. Or: innodb_numa_interleave=1

Another possible solution: Turn numa off (if the OS has a way of doing that)

Overall performance loss/gain: A few percent.

Huge Pages

This is another hardware performance gimmick.

For a CPU to access RAM, especially mapping a 64-bit address to somewhere in, say, 128GB or 'real' RAM, the TLB is used. (TLB = Translation Lookup Buffer.) Think of the TLB as a hardware associative memory lookup table; given a 64-bit virtual address, what is the real address.

Because it is an associative memory of finite size, sometimes there will be "misses" that require reaching into real RAM to resolve the lookup. This is costly, so should be avoided.

Normally, RAM is 'paged' in 4KB pieces; the TLB actually maps the top (64-12) bits into a specific page. Then the bottom 12 bits of the virtual address are carried over intact.

For example, 128GB of RAM broken 4KB pages means 32M page-table entries. This is a lot, and probably far exceeds the capacity of the TLB. So, enter the "Huge page" trick.

With the help of both the hardware and the OS, it is possible to have some of RAM in huge pages, of say 4MB (instead of 4KB). This leads to far fewer TLB entries, but it means the unit of paging is 4MB for such parts of RAM. Hence, huge pages tend to be non-pagable.

Now RAM is broken into pagable and non pagable parts; what parts can reasonably be non pagable? In MariaDB, the Innodb Buffer Pool is a perfect candidate. So, by correctly configuring these, InnoDB can run a little faster:

- Huge pages enabled
- Tell the OS to allocate the right amount (namely to match the buffer_pool)
- Tell MariaDB to use huge pages
- innodb memory usage vs swap

That thread has more details on what to look for and what to set.

Overall performance gain: A few percent. Yawn. Too much hassle for too little benefit.

Jumbo Pages? Turn off.

ENGINE=MEMORY

The Memory Storage Engine is a little-used alternative to MyISAM and InnoDB. The data is not persistent, so it has limited uses. The size of a MEMORY table is limited to max_heap_table_size, which defaults to 16MB. I mention it in case you have changed the value to something huge; this would stealing from other possible uses of RAM.

How to Set Variables

In the text file my.cnf (my.ini on Windows), add or modify a line to say something like

innodb_buffer_pool_size = 5G

That is, VARIABLE name, "=", and a value. Some abbreviations are allowed, such as M for million (1048576), G for billion.

For the server to see it, the settings must be in the "[mysqld]" section of the file.

The settings in my.cnf or my.ini will not take effect until you restart the server.

Most settings can be changed on the live system by connecting as user root (or other user with SUPER privilege) and doing something like

```
SET @@global.key_buffer_size = 77000000;
```

Note: No M or G suffix is allowed here.

To see the setting a global VARIABLE do something like

```
SHOW GLOBAL VARIABLES LIKE "key_buffer_size";
+-----------------+----------+
| Variable_name   | Value    |
+-----------------+----------+
| key_buffer_size | 76996608 |
+-----------------+----------+
```

Note that this particular setting was rounded down to some multiple that MariaDB liked.

You may want to do both (SET, and modify my.cnf) in order to make the change immediately and have it so that the next restart (for whatever reason) will again get the value.

Web Server

A web server like Apache runs multiple threads. If each thread opens a connection to MariaDB, you could run out of connections. Make sure MaxClients (or equivalent) is set to some civilized number (under 50).

Tools

- MySQLTuner
- TUNING-PRIMER

There are several tools that advise on memory. One misleading entry they come up with

Maximum possible memory usage: 31.3G (266% of installed RAM)

Don't let it scare you -- the formulas used are excessively conservative. They assume all of max_connections are in use and active, and doing something memory-intensive.

Total fragmented tables: 23 This implies that OPTIMIZE TABLE _might_ help. I suggest it for tables with either a high percentage of "free space" (see SHOW TABLE STATUS) or where you know you do a lot of DELETEs and/or UPDATEs. Still, don't bother to OPTIMIZE too often. Once a month might suffice.

MySQL 5.7

5.7 stores a lot more information in RAM, leading to the footprint being perhaps half a GB more than 5.6. See Memory increase in 5.7.

Postlog

Created 2010; Refreshed Oct, 2012, Jan, 2014

The tips in this document apply to MySQL, MariaDB, and Percona.

3.3.7 System Variables

System and Status Variables Added By Major Release

Lists of status and system variables added in MariaDB major releases.

Full List of MariaDB Options, System and Status Variables

Complete alphabetical list of all MariaDB options as well as system and status variables.

Server Status Variables

List and description of the Server Status Variables.

Server System Variables

List of system variables.

Aria Status Variables

Aria-related server status variables.

Aria System Variables

Aria-related system variables.

Cassandra Status Variables

Cassandra-related status variables

Cassandra System Variables

Cassandra system variables

CONNECT System Variables

System variables related to the CONNECT storage engine.

Galera Cluster Status Variables

Galera Cluster status variables.

Galera Cluster System Variables

Listing and description of Galera Cluster system variables.

InnoDB Server Status Variables

List and description of InnoDB status variables.

InnoDB System Variables

List and description of InnoDB-related server system variables.

MariaDB Audit Plugin Options and System Variables

Description of Server_Audit plugin options and system variables.

MariaDB Audit Plugin - Status Variables

Server Audit plugin status variables

Mroonga Status Variables

Mroonga-related status variables.

Mroonga System Variables

Mroonga-related system variables.

MyISAM System Variables

MyISAM system variables.

MyRocks Status Variables

MyRocks-related status variables.

MyRocks System Variables

MyRocks server system variables.

OQGRAPH System and Status Variables

List and description of OQGRAPH system and status variables.

Performance Schema Status Variables

Performance Schema status variables.

Performance Schema System Variables

Performance Schema system variables.

Replication and Binary Log Status Variables

Replication and binary log status variables.

Replication and Binary Log System Variables

Replication and binary log system variables.

Semisynchronous Replication Plugin Status Variables

Semisynchronous Replication plugin status variables

Semisynchronous Replication

Semisynchronous replication.

Sphinx Status Variables

Sphinx status variables.

Spider Status Variables

Spider server status variables.

Spider System Variables

System variables for the Spider storage engine.

SQL_ERROR_LOG Plugin System Variables

SQL_ERROR_LOG plugin-related system variables.

SSL/TLS Status Variables

List and description of Transport Layer Security (TLS)-related status variables.

SSL/TLS System Variables

List and description of Transport Layer Security (TLS)-related system variables.

Thread Pool System and Status Variables

System and status variables related to the MariaDB thread pool.

TokuDB Status Variables

TokuDB status variables

TokuDB System Variables

TokuDB System Variables

MariaDB Optimization for MySQL Users

MariaDB contains many new options and optimizations which, for compatibilit...

InnoDB Buffer Pool

The most important memory buffer used by InnoDB.

InnoDB Change Buffering

Buffering INSERT, UPDATE and DELETE statements for greater efficiency.

Optimizing table_open_cache

Adjusting table_open_cache to improve performance.

Optimizing key_buffer_size

Optimizing index buffers with key_buffer_size

Segmented Key Cache

Collection of structures for regular MyISAM key caches

Big Query Settings

Recommended settings for large, IO-bound queries

Sample my.cnf Files

Place holder for sample my.cnf files, customized for different memory size ...

Handling Too Many Connections

Dealing with the 'Too many connections' error

System Variable Differences between MariaDB and MySQL

Comparison of variable differences between major versions of MariaDB and MySQL.

MariaDB Memory Allocation

Basic issues in RAM allocation for MariaDB.

Setting Innodb Buffer Pool Size Dynamically

The InnoDB Buffer Pool size can be set dynamically.

There are 3 related questions.

3.3.7.1 System and Status Variables Added By Major Release

System Variables Added in MariaDB 11.3

List of system variables that were added in the MariaDB 11.3 series.

System Variables Added in MariaDB 11.2
List of system variables that were added in the MariaDB 11.2 series.

System Variables Added in MariaDB 11.1
List of system variables that were added in the MariaDB 11.1 series.

System Variables Added in MariaDB 11.0
List of system variables that were added in the MariaDB 11.0 series.

Status Variables Added in MariaDB 11.0
List of status variables that were added in the MariaDB 11.0 series.

System Variables Added in MariaDB 10.11
List of system variables that were added in the MariaDB 10.11 series.

System Variables Added in MariaDB 10.10
List of system variables that were added in the MariaDB 10.10 series.

System Variables Added in MariaDB 10.6
List of system variables that were added in the MariaDB 10.6 series.

Status Variables Added in MariaDB 10.6
List of status variables that were added in the MariaDB 10.6 series.

System Variables Added in MariaDB 10.5
List of system variables that were added in the MariaDB 10.5 series.

Status Variables Added in MariaDB 10.5
List of status variables that were added in the MariaDB 10.5 series.

System Variables Added in MariaDB 10.4
List of system variables that were added in the MariaDB 10.4 series.

Status Variables Added in MariaDB 10.4
List of status variables that were added in the MariaDB 10.4 series.

System and Status Variables Added By Major Unmaintained Release
Lists of status and system variables added in unmaintained MariaDB major releases.

3.3.7.1.1 System Variables Added in MariaDB 11.3

This is a list of system variables that have been added in the MariaDB 11.3 series.

Variable	Added
innodb_truncate_temporary_tablespace_now	MariaDB 11.3.0
redirect_url	MariaDB 11.3.0

3.3.7.1.2 System Variables Added in MariaDB 11.2

This is a list of system variables that have been added in the MariaDB 11.2 series.

Variable	Added
binlog_do_db	MariaDB 11.2.0
binlog_ignore_db	MariaDB 11.2.0
binlog_row_event_max_size	MariaDB 11.2.0

block_encryption_mode	MariaDB 11.2.0
character_set_collations	MariaDB 11.2.1

3.3.7.1.3 System Variables Added in MariaDB 11.1

This is a list of system variables that have been added in the MariaDB 11.1 series.

Variable	Added
transaction_isolation	MariaDB 11.1.0
transaction_read_only	MariaDB 11.1.0

3.3.7.1.4 System Variables Added in MariaDB 11.0

This is a list of system variables that have been added in the MariaDB 11.0 series.

Variable	Added
innodb_data_file_buffering	MariaDB 11.0.1
innodb_data_file_write_through	MariaDB 11.0.1
innodb_log_file_write_through	MariaDB 11.0.1

3.3.7.1.5 Status Variables Added in MariaDB 11.0

This is a list of status variables that were added in the MariaDB 11.0 series.

Variable	Added
max_used_connections_time	MariaDB 11.0.2

3.3.7.1.6 System Variables Added in MariaDB 10.11

This is a list of system variables that have been added in the MariaDB 10.11 series.

Variable	Added
log_slow_min_examined_row_limit	MariaDB 10.11.0
log_slow_query	MariaDB 10.11.0
log_slow_query_file	MariaDB 10.11.1
log_slow_query_time	MariaDB 10.11.0
replicate_rewrite_db	MariaDB 10.11.0
system_versioning_insert_history	MariaDB 10.11.0

3.3.7.1.7 System Variables Added in MariaDB 10.10

This is a list of system variables that have been added in the MariaDB 10.10 series.

Variable	Added

allow_suspicious_udfs	MariaDB 10.10.1
optimizer_extra_pruning_depth	MariaDB 10.10.1
skip_grant_tables	MariaDB 10.10.1
slave_max_statement_time	MariaDB 10.10.1
wsrep_allowlist	MariaDB 10.10.1

3.3.7.1.8 System Variables Added in MariaDB 10.6

This is a list of system variables that have been added in the MariaDB 10.6 series. The list does not include variables that are not part of the default release.

Variable	Added
binlog_expire_logs_seconds	MariaDB 10.6.0
innodb_deadlock_report	MariaDB 10.6.0
innodb_read_only_compressed	MariaDB 10.6.0
note_verbosity	MariaDB 10.6.16
wsrep_mode	MariaDB 10.6.0

3.3.7.1.9 Status Variables Added in MariaDB 10.6

This is a list of status variables that were added in the MariaDB 10.6 series.

Variable	Added
Innodb_buffer_pool_pages_lru_freed	MariaDB 10.6.0
resultset_metadata_skipped	MariaDB 10.6.0

3.3.7.1.10 System Variables Added in MariaDB 10.5

This is a list of system variables that have been added in the MariaDB 10.5 series. The list does not include variables that are not part of the default release.

Variable	Added
binlog_row_metadata	MariaDB 10.5.0
innodb_instant_alter_column_allowed	MariaDB 10.5.3
innodb_lru_flush_size	MariaDB 10.5.7
innodb_max_purge_lag_wait	MariaDB 10.5.7
optimizer_max_sel_arg_weight	MariaDB 10.5.9
performance_schema_events_transactions_history_long_size	MariaDB 10.5.2
performance_schema_events_transactions_history_size	MariaDB 10.5.2
performance_schema_max_index_stat	MariaDB 10.5.2
performance_schema_max_memory_classes	MariaDB 10.5.2
performance_schema_max_metadata_locks	MariaDB 10.5.2
performance_schema_max_prepared_statement_instances	MariaDB 10.5.2
performance_schema_max_program_instances	MariaDB 10.5.2

performance_schema_max_sql_text_length	MariaDB 10.5.2
performance_schema_max_statement_stack	MariaDB 10.5.2
performance_schema_max_table_lock_stat	MariaDB 10.5.2
require_secure_transport	MariaDB 10.5.2
s3_access_key	MariaDB 10.5
s3_block_size	MariaDB 10.5
s3_bucket	MariaDB 10.5
s3_debug	MariaDB 10.5
s3_host_name	MariaDB 10.5
s3_pagecache_age_threshold	MariaDB 10.5
s3_pagecache_buffer_size	MariaDB 10.5
s3_pagecache_division_limit	MariaDB 10.5
s3_pagecache_file_hash_size	MariaDB 10.5
s3_protocol_version	MariaDB 10.5
s3_region	MariaDB 10.5
s3_secret_key	MariaDB 10.5
sql_if_exists	MariaDB 10.5.2
thread_pool_dedicated_listener	MariaDB 10.5.0
thread_pool_exact_stats	MariaDB 10.5.0

3.3.7.1.11 Status Variables Added in MariaDB 10.5

This is a list of status variables that were added in the MariaDB 10.5 series.

Variable	Added
Innodb_adaptive_hash_hash_searches	MariaDB 10.5.0
Innodb_adaptive_hash_non_hash_searches	MariaDB 10.5.0
Innodb_background_log_sync	MariaDB 10.5.0
Innodb_buffer_pool_pages_made_not_young	MariaDB 10.5.0
Innodb_buffer_pool_pages_made_young	MariaDB 10.5.0
Innodb_buffer_pool_pages_old	MariaDB 10.5.0
Innodb_buffer_pool_pages_LRU_flushed	MariaDB 10.5.0
Innodb_buffered_aio_submitted	MariaDB 10.5.0
Innodb_checkpoint_age	MariaDB 10.5.0
Innodb_checkpoint_max_age	MariaDB 10.5.0
Innodb_deadlocks	MariaDB 10.5.0
Innodb_ibuf_discarded_delete_marks	MariaDB 10.5.0
Innodb_ibuf_discarded_deletes	MariaDB 10.5.0
Innodb_ibuf_discarded_inserts	MariaDB 10.5.0
Innodb_ibuf_free_list	MariaDB 10.5.0
Innodb_ibuf_merged_delete_marks	MariaDB 10.5.0
Innodb_ibuf_merged_deletes	MariaDB 10.5.0
Innodb_ibuf_merged_inserts	MariaDB 10.5.0

Innodb_ibuf_merges	MariaDB 10.5.0
Innodb_ibuf_segment_size	MariaDB 10.5.0
Innodb_ibuf_size	MariaDB 10.5.0
Innodb_lsn_current	MariaDB 10.5.0
Innodb_lsn_flushed	MariaDB 10.5.0
Innodb_lsn_last_checkpoint	MariaDB 10.5.0
Innodb_master_thread_active_loops	MariaDB 10.5.0
Innodb_master_thread_idle_loops	MariaDB 10.5.0
Innodb_max_trx_id	MariaDB 10.5.0
Innodb_mem_adaptive_hash	MariaDB 10.5.0
Innodb_mem_dictionary	MariaDB 10.5.0
performance_schema_index_stat_lost	MariaDB 10.5.2
performance_schema_memory_classes_lost	MariaDB 10.5.2
performance_schema_metadata_lock_lost	MariaDB 10.5.2
performance_schema_nested_statement_lost	MariaDB 10.5.2
performance_schema_prepared_statements_lost	MariaDB 10.5.2
performance_schema_program_lost	MariaDB 10.5.2
performance_schema_table_lock_stat_lost	MariaDB 10.5.2
S3_pagecache_blocks_not_flushed	MariaDB 10.5
S3_pagecache_blocks_unused	MariaDB 10.5
S3_pagecache_blocks_used	MariaDB 10.5
S3_pagecache_reads	MariaDB 10.5

3.3.7.1.12 System Variables Added in MariaDB 10.4

This is a list of system variables that have been added in the MariaDB 10.4 series. The list does not include variables that are not part of the default release.

Variable	Added
analyze_sample_percentage	MariaDB 10.4.3
default_password_lifetime	MariaDB 10.4.3
disconnect_on_expired_password	MariaDB 10.4.3
gtid_cleanup_batch_size	MariaDB 10.4.1
innodb_encrypt_temporary_ables	MariaDB 10.4.7
innodb_instant_alter_column_allowed	MariaDB 10.4.13
max_password_errors	MariaDB 10.4.2
optimizer_trace	MariaDB 10.4.3
optimizer_trace_max_mem_size	MariaDB 10.4.3
tcp_nodelay	MariaDB 10.4.0
tls_version	MariaDB 10.4.6
wsrep_certification_rules	MariaDB 10.4.3
wsrep_trx_fragment_size	MariaDB 10.4.2
wsrep_trx_fragment_unit	MariaDB 10.4.2

3.3.7.1.13 Status Variables Added in MariaDB 10.4

This is a list of status variables that were added in the MariaDB 10.4 series.

Variable	Added
Aborted_connects_preauth	MariaDB 10.4.5
Com_backup	MariaDB 10.4.1
Com_backup_lock	MariaDB 10.4.2
Feature_application_time_periods	MariaDB 10.4.5
wsrep_applier_thread_count	MariaDB 10.4.7
wsrep_rollbacker_thread_count	MariaDB 10.4.7

2.7.1 Full List of MariaDB Options, System and Status Variables

3.3.7.3 Server Status Variables

Contents

40. Cassandra_multiget_rows_read
41. Cassandra_row_inserts
42. Cassandra_row_insert_batches
43. Cassandra_timeout_exceptions
44. Cassandra_unavailable_exceptions
45. Column_compressions
46. Column_decompressions
47. Com_admin_commands
48. Com_alter_db
49. Com_alter_db_upgrade
50. Com_alter_event
51. Com_alter_function
52. Com_alter_procedure
53. Com_alter_sequence
54. Com_alter_server
55. Com_alter_table
56. Com_alter_tablespace
57. Com_alter_user
58. Com_analyze
59. Com_assign_to_keycache
60. Com_backup
61. Com_backup_lock
62. Com_backup_table
63. Com_begin
64. Com_binlog
65. Com_call_procedure
66. Com_change_db
67. Com_change_master
68. Com_check
69. Com_checksum
70. Com_commit
71. Com_compound_sql
72. Com_create_db
73. Com_create_event
74. Com_create_function
75. Com_create_index
76. Com_create_package
77. Com_create_package_body
78. Com_create_procedure
79. Com_create_role
80. Com_create_sequence
81. Com_create_server
82. Com_create_table
83. Com_create_temporary_table
84. Com_create_trigger
85. Com_create_udf
86. Com_create_user
87. Com_create_view
88. Com_dealloc_sql
89. Com_delete
90. Com_delete_multi
91. Com_do
92. Com_drop_db
93. Com_drop_event
94. Com_drop_function
95. Com_drop_index
96. Com_drop_package
97. Com_drop_package_body
98. Com_drop_procedure
99. Com_drop_role
100. Com_drop_sequence
101. Com_drop_server
102. Com_drop_table
103. Com_drop_temporary_table
104. Com_drop_trigger
105. Com_drop_user
106. Com_drop_view
107. Com_empty_query

108. Com_execute_immediate
109. Com_execute_sql
110. Com_flush
111. Com_get_diagnostics
112. Com_grant
113. Com_grant_role
114. Com_ha_close
115. Com_ha_open
116. Com_ha_read
117. Com_help
118. Com_insert
119. Com_insert_select
120. Com_install_plugin
121. Com_kill
122. Com_load
123. Com_load_master_data
124. Com_load_master_table
125. Com_multi
126. Com_lock_tables
127. Com_optimize
128. Com_preload_keys
129. Com_prepare_sql
130. Com_purge
131. Com_purge_before_date
132. Com_release_savepoint
133. Com_rename_table
134. Com_rename_user
135. Com_repair
136. Com_replace
137. Com_replace_select
138. Com_reset
139. Com_resignal
140. Com_restore_table
141. Com_revoke
142. Com_revoke_all
143. Com_revoke_grant
144. Com_rollback
145. Com_rollback_to_savepoint
146. Com_savepoint
147. Com_select
148. Com_set_option
149. Com_signal
150. Com_show_authors
151. Com_show_binlog_events
152. Com_show_binlogs
153. Com_show_charsets
154. Com_show_client_statistics
155. Com_show_collations
156. Com_show_column_types
157. Com_show_contributors
158. Com_show_create_db
159. Com_show_create_event
160. Com_show_create_func
161. Com_show_create_package
162. Com_show_create_package_body
163. Com_show_create_proc
164. Com_show_create_table
165. Com_show_create_trigger
166. Com_show_create_user
167. Com_show_databases
168. Com_show_engine_logs
169. Com_show_engine_mutex
170. Com_show_engine_status
171. Com_show_events
172. Com_show_errors
173. Com_show_explain
174. Com_show_fields

175. Com_show_function_status
176. Com_show_generic
177. Com_show_grants
178. Com_show_keys
179. Com_show_index_statistics
180. com_show_master_status
181. com_show_new_master
182. Com_show_open_tables
183. Com_show_package_status
184. Com_show_package_body_status
185. Com_show_plugins
186. Com_show_privileges
187. Com_show_procedure_status
188. Com_show_processlist
189. Com_show_profile
190. Com_show_profiles
191. Com_show_relaylog_events
192. com_show_slave_hosts
193. com_show_slave_status
194. Com_show_status
195. Com_show_storage_engines
196. Com_show_table_statistics
197. Com_show_table_status
198. Com_show_tables
199. Com_show_triggers
200. Com_show_user_statistics
201. Com_show_variable
202. Com_show_warnings
203. Com_shutdown
204. Com_slave_start
205. Com_slave_stop
206. Com_start_all_slaves
207. Com_start_slave
208. Com_stmt_close
209. Com_stmt_execute
210. Com_stmt_fetch
211. Com_stmt_prepare
212. Com_stmt_reprepare
213. Com_stmt_reset
214. Com_stmt_send_long_data
215. Com_stop_all_slaves
216. Com_stop_slave
217. Com_truncate
218. Com_uninstall_plugin
219. Com_unlock_tables
220. Com_update
221. Com_update_multi
222. Com_xa_commit
223. Com_xa_end
224. Com_xa_prepare
225. Com_xa_recover
226. Com_xa_rollback
227. Com_xa_start
228. Compression
229. Connection_errors_accept
230. Connection_errors_internal
231. Connection_errors_max_connections
232. Connection_errors_peer_address
233. Connection_errors_select
234. Connection_errors_tcpwrap
235. Connections
236. Cpu_time
237. Created_tmp_disk_tables
238. Created_tmp_files
239. Created_tmp_tables
240. Delayed_errors
241. Delayed_insert_threads
242. Delayed_writes

242. Delayed_writes
243. Delete_scan
244. Empty_queries
245. Executed_events
246. Executed_triggers
247. Feature_application_time_periods
248. Feature_check_constraint
249. Feature_custom_aggregate_functions
250. Feature_delay_key_write
251. Feature_dynamic_columns
252. Feature_fulltext
253. Feature_gis
254. Feature_insert_returning
255. Feature_invisible_columns
256. Feature_json
257. Feature_locale
258. Feature_subquery
259. Feature_system_versioning
260. Feature_timezone
261. Feature_trigger
262. Feature_window_functions
263. Feature_xml
264. Flush_commands
265. Handler_commit
266. Handler_delete
267. Handler_discover
268. Handler_external_lock
269. Handler_icp_attempts
270. Handler_icp_match
271. Handler_mrr_init
272. Handler_mrr_key_refills
273. Handler_mrr_rowid_refills
274. Handler_prepare
275. Handler_read_first
276. Handler_read_key
277. Handler_read_last
278. Handler_read_next
279. Handler_read_prev
280. Handler_read_retry
281. Handler_read_rnd
282. Handler_read_rnd_deleted
283. Handler_read_rnd_next
284. Handler_rollback
285. Handler_savepoint
286. Handler_savepoint_rollback
287. Handler_tmp_delete
288. Handler_tmp_update
289. Handler_tmp_write
290. Handler_update
291. Handler_write
292. Innodb_adaptive_hash_cells
293. Innodb_adaptive_hash_hash_searches
294. Innodb_adaptive_hash_heap_buffers
295. Innodb_adaptive_hash_non_hash_searches
296. Innodb_available_undo_logs
297. Innodb_background_log_sync
298. Innodb_buffer_pool_bytes_data
299. Innodb_buffer_pool_bytes_dirty
300. Innodb_buffer_pool_dump_status
301. innodb_buffer_pool_load_incomplete
302. innodb_buffer_pool_load_status
303. Innodb_buffer_pool_pages_data
304. Innodb_buffer_pool_pages_dirty
305. Innodb_buffer_pool_pages_flushed
306. Innodb_buffer_pool_pages_LRU_flushed
307. Innodb_buffer_pool_pages_LRU_freed
308. Innodb_buffer_pool_pages_free
309. Innodb_buffer_pool_pages_made_not_young

310. Innodb_buffer_pool_pages_made_young
311. Innodb_buffer_pool_pages_misc
312. Innodb_buffer_pool_pages_old
313. Innodb_buffer_pool_pages_total
314. Innodb_buffer_pool_read_ahead_rnd
315. Innodb_buffer_pool_read_ahead
316. Innodb_buffer_pool_read_ahead_evicted
317. Innodb_buffer_pool_read_requests
318. Innodb_buffer_pool_reads
319. Innodb_buffer_pool_resize_status
320. Innodb_buffer_pool_wait_free
321. Innodb_buffer_pool_write_requests
322. Innodb_buffered_aio_submitted
323. Innodb_checkpoint_age
324. Innodb_checkpoint_max_age
325. Innodb_checkpoint_target_age
326. Innodb_current_row_locks
327. Innodb_data_fsyncs
328. Innodb_data_pending_fsyncs
329. Innodb_data_pending_reads
330. Innodb_data_pending_writes
331. Innodb_data_read
332. Innodb_data_reads
333. Innodb_data_writes
334. Innodb_data_written
335. Innodb_dblwr_pages_written
336. Innodb_dblwr_writes
337. Innodb_deadlocks
338. Innodb_defragment_compression_failures
339. Innodb_defragment_count
340. Innodb_defragment_failures
341. Innodb_dict_tables
342. Innodb_encryption_n_merge_blocks_decrypted
343. Innodb_encryption_n_merge_blocks_encrypted
344. Innodb_encryption_n_rowlog_blocks_decrypted
345. Innodb_encryption_n_rowlog_blocks_encrypted
346. Innodb_encryption_n_temp_blocks_decrypted
347. Innodb_encryption_n_temp_blocks_encrypted
348. Innodb_encryption_num_key_requests
349. Innodb_encryption_rotation_estimated_iops
350. Innodb_encryption_rotation_pages_flushed
351. Innodb_encryption_rotation_pages_modified
352. Innodb_encryption_rotation_pages_read_from_cache
353. Innodb_encryption_rotation_pages_read_from_disk
354. Innodb_have_atomic_builtins
355. Innodb_have_bzip2
356. Innodb_have_lz4
357. Innodb_have_lzma
358. Innodb_have_lzo
359. Innodb_have_snappy
360. Innodb_history_list_length
361. Innodb_ibuf_discarded_delete_marks
362. Innodb_ibuf_discarded_deletes
363. Innodb_ibuf_discarded_inserts
364. Innodb_ibuf_free_list
365. Innodb_ibuf_merged_delete_marks
366. Innodb_ibuf_merged_deletes
367. Innodb_ibuf_merged_inserts
368. Innodb_ibuf_merges
369. Innodb_ibuf_segment_size
370. Innodb_ibuf_size
371. Innodb_instant_alter_column
372. Innodb_log_waits
373. Innodb_log_write_requests
374. Innodb_log_writes
375. Innodb_lsn_current
376. Innodb_lsn_flushed

377. Innodb_lsn_last_checkpoint
378. Innodb_master_thread_1_second_loops
379. Innodb_master_thread_10_second_loops
380. Innodb_master_thread_background_loops
381. Innodb_master_thread_main_flush_loops
382. Innodb_master_thread_sleeps
383. Innodb_max_trx_id
384. Innodb_mem_adaptive_hash
385. Innodb_mem_dictionary
386. Innodb_mem_total
387. Innodb_mutex_os_waits
388. Innodb_mutex_spin_rounds
389. Innodb_mutex_spin_waits
390. Innodb_num_index_pages_written
391. Innodb_num_non_index_pages_written
392. Innodb_num_open_files
393. Innodb_num_page_compressed_trim_op
394. Innodb_num_page_compressed_trim_op_saved
395. Innodb_num_pages_encrypted
396. Innodb_num_pages_page_compressed
397. Innodb_num_pages_page_compression_error
398. Innodb_num_pages_page_decompressed
399. Innodb_num_pages_page_decrypted
400. Innodb_num_pages_page_encryption_error
401. Innodb_oldest_view_low_limit_trx_id
402. Innodb_onlineddl_pct_progress
403. Innodb_onlineddl_rowlog_pct_used
404. Innodb_onlineddl_rowlog_rows
405. Innodb_os_log_fsyncs
406. Innodb_os_log_pending_fsyncs
407. Innodb_os_log_pending_writes
408. Innodb_os_log_written
409. Innodb_page_compression_saved
410. Innodb_page_compression_trim_sect512
411. Innodb_page_compression_trim_sect1024
412. Innodb_page_compression_trim_sect2048
413. Innodb_page_compression_trim_sect4096
414. Innodb_page_compression_trim_sect8192
415. Innodb_page_compression_trim_sect16384
416. Innodb_page_compression_trim_sect32768
417. Innodb_page_size
418. Innodb_pages_created
419. Innodb_pages_read
420. Innodb_pages_written
421. Innodb_purge_trx_id
422. Innodb_purge_undo_no
423. Innodb_row_lock_current_waits
424. Innodb_row_lock_numbers
425. Innodb_row_lock_time
426. Innodb_row_lock_time_avg
427. Innodb_row_lock_time_max
428. Innodb_row_lock_waits
429. Innodb_rows_deleted
430. Innodb_rows_inserted
431. Innodb_rows_read
432. Innodb_rows_updated
433. Innodb_s_lock_os_waits
434. Innodb_s_lock_spin_rounds
435. Innodb_s_lock_spin_waits
436. Innodb_scrub_background_page_reorganizations
437. Innodb_scrub_background_page_split_failures_missing_index
438. Innodb_scrub_background_page_split_failures_out_of_filespace
439. Innodb_scrub_background_page_split_failures_underflow
440. Innodb_scrub_background_page_split_failures_unknown
441. Innodb_scrub_background_page_splits
442. Innodb_secondary_index_triggered_cluster_reads
443. Innodb_secondary_index_triggered_cluster_reads_avoided
444. Innodb_system_rows_deleted

444. Innodb_system_rows_deleted
445. Innodb_system_rows_inserted
446. Innodb_system_rows_read
447. Innodb_system_rows_updated
448. Innodb_truncated_status_writes
449. Innodb_undo_truncations
450. Innodb_x_lock_os_waits
451. Innodb_x_lock_spin_rounds
452. Innodb_x_lock_spin_waits
453. Key_blocks_not_flushed
454. Key_blocks_unused
455. Key_blocks_used
456. Key_blocks_warm
457. Key_read_requests
458. Key_reads
459. Key_write_requests
460. Key_writes
461. Last_query_cost
462. Maria_*
463. Master_gtid_wait_count
464. Master_gtid_wait_time
465. Master_gtid_wait_timeouts
466. Max_statement_time_exceeded
467. Max_used_connections
468. Max_used_connections_time
469. Memory_used
470. Memory_used_initial
471. Mroonga_count_skip
472. Mroonga_fast_order_limit
473. Not_flushed_delayed_rows
474. Open_files
475. Open_streams
476. Open_table_definitions
477. Open_tables
478. Opened_files
479. Opened_plugin_libraries
480. Opened_table_definitions
481. Opened_tables
482. Opened_views
483. Oqgraph_boost_version
484. Oqgraph_compat_mode
485. Oqgraph_verbose_debug
486. Performance_schema_accounts_lost
487. Performance_cond_classes_accounts_lost
488. Performance_schema_cond_instances_lost
489. Performance_schema_digest_lost
490. Performance_schema_file_classes_lost
491. Performance_schema_file_handles_lost
492. Performance_schema_file_instances_lost
493. Performance_schema_hosts_lost
494. Performance_schema_index_stat_lost
495. Performance_schema_locker_lost
496. Performance_schema_memory_classes_lost
497. Performance_metadata_lock_classes_lost
498. Performance_schema_mutex_classes_lost
499. Performance_schema_mutex_instances_lost
500. Performance_schema_nested_statement_lost
501. Performance_schema_prepared_statements_lost
502. Performance_schema_program_lost
503. Performance_schema_rwlock_classes_lost
504. Performance_schema_rwlock_instances_lost
505. Performance_schema_session_connect_attrs_lost
506. Performance_schema_socket_classes_lost
507. Performance_schema_socket_instances_lost
508. Performance_schema_stage_classes_lost
509. Performance_schema_statement_classes_lost
510. Performance_schema_table_handles_lost
511. Performance_schema_table_instances_lost

512. Performance_schema_table_lock_stat_lost
513. Performance_schema_thread_classes_lost
514. Performance_schema_thread_instances_lost
515. Performance_schema_users_lost
516. Prepared_stmt_count
517. Qcache_free_blocks
518. Qcache_free_memory
519. Qcache_hits
520. Qcache_inserts
521. Qcache_lowmem_prunes
522. Qcache_not_cached
523. Qcache_queries_in_cache
524. Qcache_total_blocks
525. Queries
526. Questions
527. Resultset_metadata_skipped
528. Rocksdb_block_cache_add
529. Rocksdb_block_cache_add_failures
530. Rocksdb_block_cache_bytes_read
531. Rocksdb_block_cache_bytes_write
532. Rocksdb_block_cache_data_add
533. Rocksdb_block_cache_data_bytes_insert
534. Rocksdb_block_cache_data_hit
535. Rocksdb_block_cache_data_miss
536. Rocksdb_block_cache_filter_add
537. Rocksdb_block_cache_filter_bytes_evict
538. Rocksdb_block_cache_filter_bytes_insert
539. Rocksdb_block_cache_filter_hit
540. Rocksdb_block_cache_filter_miss
541. Rocksdb_block_cache_hit
542. Rocksdb_block_cache_index_add
543. Rocksdb_block_cache_index_bytes_evict
544. Rocksdb_block_cache_index_bytes_insert
545. Rocksdb_block_cache_index_hit
546. Rocksdb_block_cache_index_miss
547. Rocksdb_block_cache_miss
548. Rocksdb_block_cachecompressed_hit
549. Rocksdb_block_cachecompressed_miss
550. Rocksdb_bloom_filter_full_positive
551. Rocksdb_bloom_filter_full_true_positive
552. Rocksdb_bloom_filter_prefix_checked
553. Rocksdb_bloom_filter_prefix_useful
554. Rocksdb_bloom_filter_useful
555. Rocksdb_bytes_read
556. Rocksdb_bytes_written
557. Rocksdb_compact_read_bytes
558. Rocksdb_compact_write_bytes
559. Rocksdb_compaction_key_drop_new
560. Rocksdb_compaction_key_drop_obsolete
561. Rocksdb_compaction_key_drop_user
562. Rocksdb_covered_secondary_key_lookups
563. Rocksdb_flush_write_bytes
564. Rocksdb_get_hit_l0
565. Rocksdb_get_hit_l1
566. Rocksdb_get_hit_l2_and_up
567. Rocksdb_getupdatessince_calls
568. Rocksdb_iter_bytes_read
569. Rocksdb_l0_num_files_stall_micros
570. Rocksdb_l0_slowdown_micros
571. Rocksdb_manual_compactions_processed
572. Rocksdb_manual_compactions_running
573. Rocksdb_memtable_compaction_micros
574. Rocksdb_memtable_hit
575. Rocksdb_memtable_miss
576. Rocksdb_memtable_total
577. Rocksdb_memtable_unflushed
578. Rocksdb_no_file_closes

579. Rocksdb_no_file_errors
580. Rocksdb_no_file_opens
581. Rocksdb_num_iterators
582. Rocksdb_number_block_not_compressed
583. Rocksdb_number_db_next
584. Rocksdb_number_db_next_found
585. Rocksdb_number_db_prev
586. Rocksdb_number_db_prev_found
587. Rocksdb_number_db_seek
588. Rocksdb_number_db_seek_found
589. Rocksdb_number_deletes_filtered
590. Rocksdb_number_keys_read
591. Rocksdb_number_keys_updated
592. Rocksdb_number_keys_written
593. Rocksdb_number_merge_failures
594. Rocksdb_number_multiget_bytes_read
595. Rocksdb_number_multiget_get
596. Rocksdb_number_multiget_keys_read
597. Rocksdb_number_reseeks_iteration
598. Rocksdb_number_sst_entry_delete
599. Rocksdb_number_sst_entry_merge
600. Rocksdb_number_sst_entry_other
601. Rocksdb_number_sst_entry_put
602. Rocksdb_number_sst_entry_singledelete
603. Rocksdb_number_superversion_acquires
604. Rocksdb_number_superversion_cleanups
605. Rocksdb_number_superversion_releases
606. Rocksdb_queries_point
607. Rocksdb_queries_range
608. Rocksdb_row_lock_deadlocks
609. Rocksdb_row_lock_wait_timeouts
610. Rocksdb_rows_deleted
611. Rocksdb_rows_deleted_blind
612. Rocksdb_rows_expired
613. Rocksdb_rows_filtered
614. Rocksdb_rows_inserted
615. Rocksdb_rows_read
616. Rocksdb_rows_updated
617. Rocksdb_snapshot_conflict_errors
618. Rocksdb_stall_l0_file_count_limit_slowdowns
619. Rocksdb_stall_l0_file_count_limit_stops
620. Rocksdb_stall_locked_l0_file_count_limit_slowdowns
621. Rocksdb_stall_locked_l0_file_count_limit_stops
622. Rocksdb_stall_memtable_limit_slowdowns
623. Rocksdb_stall_memtable_limit_stops
624. Rocksdb_stall_micros
625. Rocksdb_stall_pending_compaction_limit_slowdowns
626. Rocksdb_stall_pending_compaction_limit_stops
627. Rocksdb_stall_total_slowdowns
628. Rocksdb_stall_total_stops
629. Rocksdb_system_rows_deleted
630. Rocksdb_system_rows_inserted
631. Rocksdb_system_rows_read
632. Rocksdb_system_rows_updated
633. Rocksdb_wal_bytes
634. Rocksdb_wal_group_syncs
635. Rocksdb_wal_synced
636. Rocksdb_write_other
637. Rocksdb_write_self
638. Rocksdb_write_timedout
639. Rocksdb_write_wal
640. Rows_read
641. Rows_sent
642. Rows_tmp_read
643. Rpl_semi_sync_master_clients
644. Rpl_semi_sync_master_net_avg_wait_time
645. Rpl_semi_sync_master_net_wait_time
646. Rpl_semi_sync_master_net_waits

646. Rpl_semi_sync_master_net_waits
647. Rpl_semi_sync_master_no_times
648. Rpl_semi_sync_master_no_tx
649. Rpl_semi_sync_master_status
650. Rpl_semi_sync_master_timefunc_failures
651. Rpl_semi_sync_master_tx_avg_wait_time
652. Rpl_semi_sync_master_tx_wait_time
653. Rpl_semi_sync_master_tx_waits
654. Rpl_semi_sync_master_wait_pos_backtraverse
655. Rpl_semi_sync_master_wait_sessions
656. Rpl_semi_sync_master_yes_tx
657. Rpl_semi_sync_slave_status
658. Rpl_status
659. Rpl_transactions_multi_engine
660. S3_pagecache_blocks_not_flushed
661. S3_pagecache_blocks_unused
662. S3_pagecache_blocks_used
663. S3_pagecache_read_requests
664. S3_pagecache_reads
665. Select_full_join
666. Select_full_range_join
667. Select_range
668. Select_range_check
669. Select_scan
670. Server_audit_active
671. Server_audit_current_log
672. Server_audit_last_error
673. Server_audit_writes_failed
674. Slave_connections
675. Slave_heartbeat_period
676. Slave_open_temp_tables
677. Slave_received_heartbeats
678. Slave_retried_transactions
679. Slave_running
680. Slave_skipped_errors
681. Slaves_connected
682. Slaves_running
683. Slow_launch_threads
684. Slow_queries
685. Sort_merge_passes
686. Sort_priority_queue_sorts
687. Sort_range
688. Sort_rows
689. Sort_scan
690. Sphinx_error
691. Sphinx_time
692. Sphinx_total
693. Sphinx_total_found
694. Sphinx_word_count
695. Sphinx_words
696. Spider_direct_aggregate
697. Spider_direct_delete
698. Spider_direct_order_limit
699. Spider_direct_update
700. Spider_direct_mon_table_cache_version
701. Spider_direct_mon_table_cache_version_req
702. Spider_parallel_search
703. Ssl_accept_renegotiates
704. Ssl_accepts
705. Ssl_callback_cache_hits
706. Ssl_cipher
707. Ssl_cipher_list
708. Ssl_client_connects
709. Ssl_connect_renegotiates
710. Ssl_ctx_verify_depth
711. Ssl_ctx_verify_mode
712. Ssl_default_timeout
713. Ssl_finished_accepts

714. Ssl_finished_connects
715. Ssl_server_not_after
716. Ssl_server_not_before
717. Ssl_session_cache_hits
718. Ssl_session_cache_misses
719. Ssl_session_cache_mode
720. Ssl_session_cache_overflows
721. Ssl_session_cache_size
722. Ssl_session_cache_timeouts
723. Ssl_sessions_reused
724. Ssl_used_session_cache_entries
725. Ssl_verify_depth
726. Ssl_verify_mode
727. Ssl_version
728. Subquery_cache_hit
729. Subquery_cache_miss
730. Syncs
731. Table_locks_immediate
732. Table_locks_waited
733. Table_open_cache_active_instances
734. Table_open_cache_hits
735. Table_open_cache_misses
736. Table_open_cache_overflows
737. Tc_log_max_pages_used
738. Tc_log_page_size
739. Tc_log_page_waits
740. Threadpool_idle_threads
741. Threadpool_threads
742. Threads_cached
743. Threads_connected
744. Threads_created
745. Threads_running
746. Tokudb_basement_deserialization_fixed_key
747. Tokudb_basement_deserialization_variable_key
748. Tokudb_basements_decompressed_for_write
749. Tokudb_basements_decompressed_prefetch
750. Tokudb_basements_decompressed_prelocked_range
751. Tokudb_basements_decompressed_target_query
752. Tokudb_basements_fetched_prefetch
753. Tokudb_basements_fetched_for_write
754. Tokudb_basements_fetched_for_write_bytes
755. Tokudb_basements_fetched_for_write_seconds
756. Tokudb_basements_fetched_prefetch_bytes
757. Tokudb_basements_fetched_prefetch_seconds
758. Tokudb_basements_fetched_prelocked_range
759. Tokudb_basements_fetched_prelocked_range_bytes
760. Tokudb_basements_fetched_prelocked_range_seconds
761. Tokudb_basements_fetched_target_query
762. Tokudb_basements_fetched_target_query_bytes
763. Tokudb_basements_fetched_target_query_seconds
764. Tokudb_broadcase_messages_injected_at_root
765. Tokudb_buffers_decompressed_for_write
766. Tokudb_buffers_decompressed_prefetch
767. Tokudb_buffers_decompressed_prelocked_range
768. Tokudb_buffers_decompressed_target_query
769. Tokudb_buffers_fetched_for_write
770. Tokudb_buffers_fetched_for_write_bytes
771. Tokudb_buffers_fetched_for_write_seconds
772. Tokudb_buffers_fetched_prefetch
773. Tokudb_buffers_fetched_prefetch_bytes
774. Tokudb_buffers_fetched_prefetch_seconds
775. Tokudb_buffers_fetched_prelocked_range
776. Tokudb_buffers_fetched_prelocked_range_bytes
777. Tokudb_buffers_fetched_prelocked_range_seconds
778. Tokudb_buffers_fetched_target_query
779. Tokudb_buffers_fetched_target_query_bytes
780. Tokudb_buffers_fetched_target_query_seconds

781. Tokudb_cachetable_cleaner_executions
782. Tokudb_cachetable_cleaner_iterations
783. Tokudb_cachetable_cleaner_period
784. Tokudb_cachetable_evictions
785. Tokudb_cachetable_long_wait_pressure_count
786. Tokudb_cachetable_long_wait_pressure_time
787. Tokudb_cachetable_miss
788. Tokudb_cachetable_miss_time
789. Tokudb_cachetable_prefetches
790. Tokudb_cachetable_size_cachepressure
791. Tokudb_cachetable_size_cloned
792. Tokudb_cachetable_size_current
793. Tokudb_cachetable_size_leaf
794. Tokudb_cachetable_size_limit
795. Tokudb_cachetable_size_nonleaf
796. Tokudb_cachetable_size_rollback
797. Tokudb_cachetable_size_writing
798. Tokudb_cachetable_wait_pressure_count
799. Tokudb_cachetable_wait_pressure_time
800. Tokudb_checkpoint_begin_time
801. Tokudb_checkpoint_duration
802. Tokudb_checkpoint_duration_last
803. Tokudb_checkpoint_failed
804. Tokudb_checkpoint_last_began
805. Tokudb_checkpoint_last_complete_began
806. Tokudb_checkpoint_last_complete_ended
807. Tokudb_checkpoint_long_begin_count
808. Tokudb_checkpoint_long_begin_time
809. Tokudb_checkpoint_period
810. Tokudb_checkpoint_taken
811. Tokudb_cursor_skip_deleted_leaf_entry
812. Tokudb_db_closes
813. Tokudb_db_open_current
814. Tokudb_db_open_max
815. Tokudb_db_opens
816. Tokudb_descriptor_set
817. Tokudb_dictionary_broadcast_updates
818. Tokudb_dictionary_updates
819. Tokudb_filesystem_fsync_num
820. Tokudb_filesystem_fsync_time
821. Tokudb_filesystem_long_fsync_num
822. Tokudb_filesystem_long_fsync_time
823. Tokudb_filesystem_threads_blocked_by_full_disk
824. Tokudb_leaf_compression_to_memory_seconds
825. Tokudb_leaf_decompression_to_memory_seconds
826. Tokudb_leaf_deserialization_to_memory_seconds
827. Tokudb_leaf_node_compression_ratio
828. Tokudb_leaf_node_full_evictions
829. Tokudb_leaf_node_full_evictions_bytes
830. Tokudb_leaf_node_partial_evictions
831. Tokudb_leaf_node_partial_evictions_bytes
832. Tokudb_leaf_nodes_created
833. Tokudb_leaf_nodes_destroyed
834. Tokudb_leaf_nodes_flushed_to_disk_checkpoint
835. Tokudb_leaf_nodes_flushed_to_disk_checkpoint_bytes
836. Tokudb_leaf_nodes_flushed_to_disk_checkpoint_seconds
837. Tokudb_leaf_nodes_flushed_to_disk_checkpoint_uncompressed_bytes
838. Tokudb_leaf_nodes_flushed_to_disk_not_checkpoint
839. Tokudb_leaf_nodes_flushed_to_disk_not_checkpoint_bytes
840. Tokudb_leaf_nodes_flushed_to_disk_not_checkpoint_seconds
841. Tokudb_leaf_nodes_flushed_to_disk_not_checkpoint_uncompressed_bytes
842. Tokudb_leaf_serialization_to_memory_seconds
843. Tokudb_loader_num_created
844. Tokudb_loader_num_current
845. Tokudb_loader_num_max
846. Tokudb_locktree_escalation_num
847. Tokudb_locktree_escalation_seconds
848. Tokudb_locktree_latest_post_escalation_memory_size

849. Tokudb_locktree_long_wait_count
850. Tokudb_locktree_long_wait_escalation_count
851. Tokudb_locktree_long_wait_escalation_time
852. Tokudb_locktree_long_wait_time
853. Tokudb_locktree_memory_size
854. Tokudb_locktree_memory_size_limit
855. Tokudb_locktree_open_current
856. Tokudb_locktree_pending_lock_requests
857. Tokudb_locktree_sto_eligible_num
858. Tokudb_locktree_sto_ended_num
859. Tokudb_locktree_sto_ended_seconds
860. Tokudb_locktree_timeout_count
861. Tokudb_locktree_wait_count
862. Tokudb_locktree_wait_escalation_count
863. Tokudb_locktree_wait_escalation_time
864. Tokudb_locktree_wait_time
865. Tokudb_logger_wait_long
866. Tokudb_logger_writes
867. Tokudb_logger_writes_bytes
868. Tokudb_logger_writes_seconds
869. Tokudb_logger_writes_uncompressed_bytes
870. Tokudb_mem_estimated_maximum_memory_footprint
871. Tokudb_messages_flushed_from_h1_to_leaves_bytes
872. Tokudb_messages_ignored_by_leaf_due_to_msn
873. Tokudb_messages_in_trees_estimate_bytes
874. Tokudb_messages_injected_at_root
875. Tokudb_messages_injected_at_root_bytes
876. Tokudb_nonleaf_compression_to_memory_seconds
877. Tokudb_nonleaf_decompression_to_memory_seconds
878. Tokudb_nonleaf_deserialization_to_memory_seconds
879. Tokudb_nonleaf_node_compression_ratio
880. Tokudb_nonleaf_node_full_evictions
881. Tokudb_nonleaf_node_full_evictions_bytes
882. Tokudb_nonleaf_node_partial_evictions
883. Tokudb_nonleaf_node_partial_evictions_bytes
884. Tokudb_nonleaf_nodes_created
885. Tokudb_nonleaf_nodes_destroyed
886. Tokudb_nonleaf_nodes_flushed_checkpoint
887. Tokudb_nonleaf_nodes_flushed_checkpoint_bytes
888. Tokudb_nonleaf_nodes_flushed_checkpoint_seconds
889. Tokudb_nonleaf_nodes_flushed_checkpoint_uncompressed_bytes
890. Tokudb_nonleaf_nodes_flushed_not_checkpoint
891. Tokudb_nonleaf_nodes_flushed_not_checkpoint_bytes
892. Tokudb_nonleaf_nodes_flushed_not_checkpoint_seconds
893. Tokudb_nonleaf_nodes_flushed_not_checkpoint_uncompressed_bytes
894. Tokudb_nonleaf_serialization_to_memory_seconds
895. Tokudb_overall_node_compression_ratio
896. Tokudb_pivots_fetched_for_query
897. Tokudb_pivots_fetched_for_query_bytes
898. Tokudb_pivots_fetched_for_query_seconds
899. Tokudb_pivots_fetched_for_prefetch
900. Tokudb_pivots_fetched_for_prefetch_bytes
901. Tokudb_pivots_fetched_for_prefetch_seconds
902. Tokudb_pivots_fetched_for_write
903. Tokudb_pivots_fetched_for_write_bytes
904. Tokudb_pivots_fetched_for_write_seconds
905. Tokudb_promotion_h1_roots_injected_into
906. Tokudb_promotion_injections_at_depth_0
907. Tokudb_promotion_injections_at_depth_1
908. Tokudb_promotion_injections_at_depth_2
909. Tokudb_promotion_injections_at_depth_3
910. Tokudb_promotion_injections_lower_than_depth_3
911. Tokudb_promotion_leaf_roots_injected_into
912. Tokudb_promotion_roots_split
913. Tokudb_promotion_stopped_after_locking_child
914. Tokudb_promotion_stopped_at_height_1
915. Tokudb_promotion_stopped_child_not_fully_in_memory

916. Tokudb_promotion_stopped_child_locked_or_not_in_memory
917. Tokudb_promotion_stopped_nonempty_buffer
918. Tokudb_txn_aborts
919. Tokudb_txn_begin
920. Tokudb_txn_begin_read_only
921. Tokudb_txn_commits
922. Transactions_gtid_foreign_engine
923. Transactions_multi_engine
924. Update_scan
925. Uptime
926. Uptime_since_flush_status
927. wsrep
928. wsrep_applier_thread_count
929. wsrep_apply_oooe
930. wsrep_apply_oool
931. wsrep_cert_deps_distance
932. wsrep_cluster_capabilities
933. wsrep_cluster_conf_id
934. wsrep_cluster_size
935. wsrep_cluster_state_uuid
936. wsrep_cluster_status
937. wsrep_connected
938. wsrep_flow_control_paused
939. wsrep_flow_control_recv
940. wsrep_flow_control_sent
941. wsrep_last_committed
942. wsrep_local_bf_aborts
943. wsrep_local_cert_failures
944. wsrep_local_commits
945. wsrep_local_index
946. wsrep_local_recv_queue
947. wsrep_local_recv_queue_avg
948. wsrep_local_replays
949. wsrep_local_send_queue
950. wsrep_local_send_queue_avg
951. wsrep_local_state
952. wsrep_local_state_comment
953. wsrep_local_state_uuid
954. wsrep_protocol_version
955. wsrep_provider_name
956. wsrep_provider_vendor
957. wsrep_provider_version
958. wsrep_ready
959. wsrep_received
960. wsrep_received_bytes
961. wsrep_replicated
962. wsrep_replicated_bytes
963. wsrep_rollbacker_thread_count
964. wsrep_thread_count

The full list of status variables are listed in the contents on this page; most are described on this page, but some are described elsewhere:

- Aria Status Variables
- Galera Status Variables
- InnoDB Status Variables
- Mroonga Status Variables
- MyRocks Status Variables
- Performance Scheme Status Variables
- Replication and Binary Log Status Variables
- S3 Storage Engine Status Variables
- Server_Audit Status Variables
- Sphinx Status Variables
- Spider Status Variables
- TokuDB Status Variables

See also the Full list of MariaDB options, system and status variables.

Use the SHOW STATUS statement to view status variables. This information also can be obtained using the mariadb-admin

extended-status command, or by querying the Information Schema GLOBAL_STATUS and SESSION_STATUS tables.

Issuing a FLUSH STATUS will reset many status variables to zero.

List of Server Status Variables

`Aborted_clients`

- **Description:** Number of aborted client connections. This can be due to the client not calling mysql_close() before exiting, the client sleeping without issuing a request to the server for more seconds than specified by wait_timeout or interactive_timeout, or by the client program ending in the midst of transferring data. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

`Aborted_connects`

- **Description:** Number of failed server connection attempts. This can be due to a client using an incorrect password, a client not having privileges to connect to a database, a connection packet not containing the correct information, or if it takes more than connect_timeout seconds to get a connect packet. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

`Aborted_connects_preauth`

- **Description:** Number of connection attempts that were aborted prior to authentication (regardless of whether or not an error occured).
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.4.5

`Access_denied_errors`

- **Description:** Number of access denied errors.
- **Scope:** Global
- **Data Type:** `numeric`

`Acl_column_grants`

- **Description:** Number of column permissions granted (rows in the mysql.columns_priv table).
- **Scope:** Global
- **Data Type:** `numeric`

`Acl_database_grants`

- **Description:** Number of database permissions granted (rows in the mysql.db table).
- **Scope:** Global
- **Data Type:** `numeric`

`Acl_function_grants`

- **Description:** Number of function permissions granted (rows in the mysql.procs_priv table with a routine type of `FUNCTION`).
- **Scope:** Global
- **Data Type:** `numeric`

Acl_package_body_grants

- **Description:**
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.3.5

Acl_package_spec_grants

- **Description:**
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.3.5

Acl_procedure_grants

- **Description:** Number of procedure permissions granted (rows in the mysql.procs_priv table with a routine type of `PROCEDURE`).
- **Scope:** Global
- **Data Type:** `numeric`

Acl_proxy_users

- **Description:** Number of proxy permissions granted (rows in the mysql.proxies_priv table).
- **Scope:** Global
- **Data Type:** `numeric`

Acl_role_grants

- **Description:** Number of role permissions granted (rows in the mysql.roles_mapping table).
- **Scope:** Global
- **Data Type:** `numeric`

Acl_roles

- **Description:** Number of roles (rows in the mysql.user table where `is_role='Y'`).
- **Scope:** Global
- **Data Type:** `numeric`

Acl_table_grants

- **Description:** Number of table permissions granted (rows in the mysql.tables_priv table).
- **Scope:** Global
- **Data Type:** `numeric`

Acl_users

- **Description:** Number of users (rows in the mysql.user table where `is_role='N'`).
- **Scope:** Global
- **Data Type:** `numeric`

Busy_time

- **Description:** Cumulative time in seconds of activity on connections.
- **Scope:** Global
- **Data Type:** `numeric`

Bytes_received

- **Description:** Total bytes received from all clients.
- **Scope:** Global
- **Data Type:** `numeric`

Bytes_sent

- **Description:** Total bytes sent to all clients.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_admin_commands

- **Description:** Number of admin commands executed. These include table dumps, change users, binary log dumps, shutdowns, pings and debugs.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_alter_db

- **Description:** Number of ALTER DATABASE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_alter_db_upgrade

- **Description:** Number of ALTER DATABASE ... UPGRADE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_alter_event

- **Description:** Number of ALTER EVENT commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_alter_function

- **Description:** Number of ALTER FUNCTION commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_alter_procedure

- **Description:** Number of ALTER PROCEDURE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_alter_sequence

- **Description:** Number of ALTER SEQUENCE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.3.1

Com_alter_server

- **Description:** Number of ALTER SERVER commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_alter_table

- **Description:** Number of ALTER TABLE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_alter_tablespace

- **Description:** Number of ALTER TABLESPACE commands executed (unsupported by MariaDB).
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Removed:** MariaDB 10.7.0

Com_alter_user

- **Description:** Number of ALTER USER commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.0

Com_analyze

- **Description:** Number of ANALYZE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_assign_to_keycache

- **Description:** Number of assign to keycache commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_backup

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Removed:** MariaDB 10.4.1

Com_backup_lock

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Removed:** MariaDB 10.4.2

Com_backup_table

- **Description:** Removed in MariaDB 5.5. In older versions, Com_backup_table contains the number of BACKUP TABLE commands executed.
- **Scope:** Global, Session

- **Data Type:** `numeric`
- **Removed:** MariaDB 5.5

`Com_begin`

- **Description:** Number of BEGIN or START TRANSACTION statements executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Com_binlog`

- **Description:** Number of BINLOG commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Com_call_procedure`

- **Description:** Number of CALL procedure_name statements executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Com_change_db`

- **Description:** Number of USE database_name commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Com_check`

- **Description:** Number of CHECK TABLE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Com_checksum`

- **Description:** Number of CHECKSUM TABLE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Com_commit`

- **Description:** Number of COMMIT commands executed. Differs from Handler_commit, which counts internal commit statements.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Com_compound_sql`

- **Description:** Number of compund sql statements.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Com_create_db`

- **Description:** Number of CREATE DATABASE commands executed.
- **Scope:** Global, Session

- **Data Type:** `numeric`

Com_create_event

- **Description:** Number of CREATE EVENT commands executed. Differs from Executed_events in that it is incremented when the CREATE EVENT is run, and not when the event executes.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_create_function

- **Description:** Number of CREATE FUNCTION commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_create_index

- **Description:** Number of CREATE INDEX commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_create_package

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.3.5

Com_create_package_body

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.3.5

Com_create_procedure

- **Description:** Number of CREATE PROCEDURE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_create_role

- **Description:** Number of CREATE ROLE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_create_sequence

- **Description:** Number of CREATE SEQUENCE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.3.0

Com_create_server

- **Description:** Number of CREATE SERVER commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_create_table

- **Description:** Number of CREATE TABLE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_create_temporary_table

- **Description:** Number of CREATE TEMPORARY TABLE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_create_trigger

- **Description:** Number of CREATE TRIGGER commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_create_udf

- **Description:** Number of CREATE UDF commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_create_user

- **Description:** Number of CREATE USER commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_create_view

- **Description:** Number of CREATE VIEW commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_dealloc_sql

- **Description:** Number of DEALLOCATE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_delete

- **Description:** Number of DELETE commands executed. Differs from Handler_delete, which counts the number of times rows have been deleted from tables.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_delete_multi

- **Description:** Number of multi-table DELETE commands executed.

- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_do

- **Description:** Number of DO commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_drop_db

- **Description:** Number of DROP DATABASE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_drop_event

- **Description:** Number of DROP EVENT commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_drop_function

- **Description:** Number of DROP FUNCTION commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_drop_index

- **Description:** Number of DROP INDEX commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_drop_package

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.3.5

Com_drop_package_body

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.3.5

Com_drop_procedure

- **Description:** Number of DROP PROCEDURE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_drop_role

- **Description:** Number of DROP ROLE commands executed.

- **Scope:** Global, Session
- **Data Type:** `numeric`

`Com_drop_sequence`

- **Description:** Number of DROP SEQUENCE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.3.0

`Com_drop_server`

- **Description:** Number of DROP SERVER commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Com_drop_table`

- **Description:** Number of DROP TABLE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Com_drop_temporary_table`

- **Description:** Number of DROP TEMPORARY TABLE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Com_drop_trigger`

- **Description:** Number of DROP TRIGGER commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Com_drop_user`

- **Description:** Number of DROP USER commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Com_drop_view`

- **Description:** Number of DROP VIEW commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Com_empty_query`

- **Description:** Number of queries to the server that do not produce SQL queries. An SQL query simply returning no results does not increment `Com_empty_query` - see Empty_queries instead. An example of an empty query sent to the server is `mariadb --comments -e '-- sql comment'`
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Com_execute_immediate`

- **Description:** Number of EXECUTE IMMEDIATE statements executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.3

`Com_execute_sql`

- **Description:** Number of EXECUTE statements executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Com_flush`

- **Description:** Number of FLUSH commands executed. This differs from Flush_commands, which also counts internal server flush requests.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Com_get_diagnostics`

- **Description:** Number of GET DIAGNOSTICS commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Com_grant`

- **Description:** Number of GRANT commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Com_grant_role`

- **Description:** Number of GRANT role commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Com_ha_close`

- **Description:** Number of HANDLER table_name CLOSE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Com_ha_open`

- **Description:** Number of HANDLER table_name OPEN commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Com_ha_read`

- **Description:** Number of HANDLER table_name READ commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Com_help`

- **Description:** Number of HELP commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Com_insert`

- **Description:** Number of INSERT commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Com_insert_select`

- **Description:** Number of INSERT ... SELECT commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Com_install_plugin`

- **Description:** Number of INSTALL PLUGIN commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Com_kill`

- **Description:** Number of KILL commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Com_load`

- **Description:** Number of LOAD commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Com_load_master_data`

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Removed:** MariaDB 5.5

`Com_load_master_table`

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Removed:** MariaDB 5.5

`Com_multi`

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.0

Com_lock_tables

- **Description:** Number of [lock-tables|LOCK TABLES]] commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_optimize

- **Description:** Number of OPTIMIZE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_preload_keys

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_prepare_sql

- **Description:** Number of PREPARE statements executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_purge

- **Description:** Number of PURGE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_purge_before_date

- **Description:** Number of PURGE BEFORE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_release_savepoint

- **Description:** Number of RELEASE SAVEPOINT commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_rename_table

- **Description:** Number of RENAME TABLE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_rename_user

- **Description:** Number of RENAME USER commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_repair

- **Description:** Number of REPAIR TABLE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_replace

- **Description:** Number of REPLACE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_replace_select

- **Description:** Number of REPLACE ... SELECT commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_reset

- **Description:** Number of RESET commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_resignal

- **Description:** Number of RESIGNAL commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_restore_table

- **Description:** Removed in MariaDB 5.5. In older versions, Com_restore_table contains the number of RESTORE TABLE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Removed:** MariaDB 5.5

Com_revoke

- **Description:** Number of REVOKE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_revoke_all

- **Description:** Number of REVOKE ALL commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_revoke_grant

- **Description:** Number of REVOKE role commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_rollback

- **Description:** Number of ROLLBACK commands executed. Differs from Handler_rollback, which is the number of transaction rollback requests given to a storage engine.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_rollback_to_savepoint

- **Description:** Number of ROLLBACK ... TO SAVEPOINT commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_savepoint

- **Description:** Number of SAVEPOINT commands executed. Differs from Handler_savepoint, which is the number of transaction savepoint creation requests.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_select

- **Description:** Number of SELECT commands executed. Also includes queries that make use of the query cache.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_set_option

- **Description:** Number of SET OPTION commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_signal

- **Description:** Number of SIGNAL statements executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_authors

- **Description:** Number of SHOW AUTHORS commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_binlog_events

- **Description:** Number of SHOW BINLOG EVENTS statements executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_binlogs

- **Description:** Number of SHOW BINARY LOGS commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_charsets

- **Description:** Number of SHOW CHARACTER SET commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_client_statistics

- **Description:** Number of SHOW CLIENT STATISTICS commands executed. Removed in MariaDB 10.1.1 when that statement was replaced by the generic SHOW information_schema_table.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Removed:** MariaDB 10.1.1

Com_show_collations

- **Description:** Number of SHOW COLLATION commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_column_types

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Removed:** MariaDB 5.5

Com_show_contributors

- **Description:** Number of SHOW CONTRIBUTORS commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_create_db

- **Description:** Number of SHOW CREATE DATABASE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_create_event

- **Description:** Number of SHOW CREATE EVENT commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_create_func

- **Description:** Number of SHOW CREATE FUNCTION commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_create_package

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.3.5

Com_show_create_package_body

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.3.5

Com_show_create_proc

- **Description:** Number of SHOW CREATE PROCEDURE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_create_table

- **Description:** Number of SHOW CREATE TABLE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_create_trigger

- **Description:** Number of SHOW CREATE TRIGGER commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_create_user

- **Description:** Number of SHOW CREATE USER commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.0

Com_show_databases

- **Description:** Number of SHOW DATABASES commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_engine_logs

- **Description:** Number of SHOW ENGINE LOGS commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_engine_mutex

- **Description:** Number of SHOW ENGINE MUTEX commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_engine_status

- **Description:** Number of SHOW ENGINE STATUS commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_events

- **Description:** Number of SHOW EVENTS commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_errors

- **Description:** Number of SHOW ERRORS commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_explain

- **Description:** Number of SHOW EXPLAIN commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_fields

- **Description:** Number of SHOW COLUMNS or SHOW FIELDS commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_function_status

- **Description:** Number of SHOW FUNCTION STATUS commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_generic

- **Description:** Number of generic SHOW commands executed, such as SHOW INDEX_STATISTICS and SHOW TABLE_STATISTICS
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_grants

- **Description:** Number of SHOW GRANTS commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_keys

- **Description:** Number of SHOW INDEX or SHOW KEYS commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_index_statistics

- **Description:** Number of SHOW INDEX_STATISTICS commands executed. Removed in MariaDB 10.1.1 when that statement was replaced by the generic SHOW information_schema_table.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Removed:** MariaDB 10.1.1

Com_show_open_tables

- **Description:** Number of SHOW OPEN TABLES commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_package_status

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.3.5

Com_show_package_body_status

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.3.5

Com_show_plugins

- **Description:** Number of SHOW PLUGINS commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_privileges

- **Description:** Number of SHOW PRIVILEGES commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_procedure_status

- **Description:** Number of SHOW PROCEDURE STATUS commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_processlist

- **Description:** Number of SHOW PROCESSLIST commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_profile

- **Description:** Number of SHOW PROFILE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_profiles

- **Description:** Number of SHOW PROFILES commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_relaylog_events

- **Description:** Number of SHOW RELAYLOG EVENTS commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_status

- **Description:** Number of SHOW STATUS commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric` st

Com_show_storage_engines

- **Description:** Number of SHOW STORAGE ENGINES - or `SHOW ENGINES` - commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_table_statistics

- **Description:** Number of SHOW TABLE STATISTICS commands executed. Removed in MariaDB 10.1.1 when that statement was replaced by the generic SHOW information_schema_table.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Removed:** MariaDB 10.1.1

Com_show_table_status

- **Description:** Number of SHOW TABLE STATUS commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_tables

- **Description:** Number of SHOW TABLES commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_triggers

- **Description:** Number of SHOW TRIGGERS commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_user_statistics

- **Description:** Number of SHOW USER STATISTICS commands executed. Removed in MariaDB 10.1.1 when that statement was replaced by the generic SHOW information_schema_table.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Removed:** MariaDB 10.1.1

Com_show_variable

- **Description:** Number of SHOW VARIABLES commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_show_warnings

- **Description:** Number of SHOW WARNINGS commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_shutdown

- **Description:** Number of SHUTDOWN commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_stmt_close

- **Description:** Number of prepared statements closed (deallocated or dropped).
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_stmt_execute

- **Description:** Number of prepared statements executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_stmt_fetch

- **Description:** Number of prepared statements fetched.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_stmt_prepare

- **Description:** Number of prepared statements prepared.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_stmt_reprepare

- **Description:** Number of prepared statements reprepared.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_stmt_reset

- **Description:** Number of prepared statements where the data of a prepared statement which was accumulated in chunks by sending long data has been reset.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_stmt_send_long_data

- **Description:** Number of prepared statements where the parameter data has been sent in chunks (long data).
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_truncate

- **Description:** Number of TRUNCATE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_uninstall_plugin

- **Description:** Number of UNINSTALL PLUGIN commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_unlock_tables

- **Description:** Number of UNLOCK TABLES commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_update

- **Description:** Number of UPDATE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_update_multi

- **Description:** Number of multi-table UPDATE commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_xa_commit

- **Description:** Number of XA statements committed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_xa_end

- **Description:** Number of XA statements ended.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_xa_prepare

- **Description:** Number of XA statements prepared.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_xa_recover

- **Description:** Number of XA RECOVER statements executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Com_xa_rollback

- **Description:** Number of XA statements rolled back.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Com_xa_start`

- **Description:** Number of XA statements started.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Compression`

- **Description:** Whether client-server traffic is compressed.
- **Scope:** Session
- **Data Type:** `boolean`

`Connection_errors_accept`

- **Description:** Number of errors that occurred during calls to accept() on the listening port. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

`Connection_errors_internal`

- **Description:** Number of refused connections due to internal server errors, for example out of memory errors, or failed thread starts. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

`Connection_errors_max_connections`

- **Description:** Number of refused connections due to the max_connections limit being reached. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

`Connection_errors_peer_address`

- **Description:** Number of errors while searching for the connecting client IP address. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

`Connection_errors_select`

- **Description:** Number of errors during calls to select() or poll() on the listening port. The client would not necessarily have been rejected in these cases. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

`Connection_errors_tcpwrap`

- **Description:** Number of connections the libwrap library refused. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

Connections

- **Description:** Number of connection attempts (both successful and unsuccessful)
- **Scope:** Global
- **Data Type:** `numeric`

Cpu_time

- **Description:** Total CPU time used.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Created_tmp_disk_tables

- **Description:** Number of on-disk temporary tables created.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Created_tmp_files

- **Description:** Number of temporary files created. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

Created_tmp_tables

- **Description:** Number of in-memory temporary tables created.
- **Scope:** Global
- **Data Type:** `numeric`

Delayed_errors

- **Description:** Number of errors which occurred while doing INSERT DELAYED. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

Delayed_insert_threads

- **Description:** Number of INSERT DELAYED threads.
- **Scope:** Global
- **Data Type:** `numeric`

Delayed_writes

- **Description:** Number of INSERT DELAYED rows written. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

Delete_scan

- **Description:** Number of DELETEs that required a full table scan.
- **Scope:** Global
- **Data Type:** `numeric`

Empty_queries

- **Description:** Number of queries returning no results. Note this is not the same as Com_empty_query.
- **Scope:** Global, Session
- **Data Type:** numeric

Executed_events

- **Description:** Number of times events created with CREATE EVENT have executed. This differs from Com_create_event in that it is only incremented when the event has run, not when it executes.
- **Scope:** Global, Session
- **Data Type:** numeric

Executed_triggers

- **Description:** Number of times triggers created with CREATE TRIGGER have executed. This differs from Com_create_trigger in that it is only incremented when the trigger has run, not when it executes.
- **Scope:** Global, Session
- **Data Type:** numeric

Feature_application_time_periods

- **Description:** Number of times a table created with periods has been opened.
- **Scope:** Global, Session
- **Data Type:** numeric
- **Introduced:** MariaDB 10.4.5

Feature_check_constraint

- **Description:** Number of times constraints were checked. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global, Session
- **Data Type:** numeric
- **Introduced:** MariaDB 10.2.2

Feature_custom_aggregate_functions

- **Description:** Number of queries which make use of custom aggregate functions.
- **Scope:** Global, Session
- **Data Type:** numeric
- **Introduced:** MariaDB 10.3.6

Feature_delay_key_write

- **Description:** Number of tables opened that are using delay_key_write. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global, Session
- **Data Type:** numeric

Feature_dynamic_columns

- **Description:** Number of times the COLUMN_CREATE() function was used.
- **Scope:** Global, Session
- **Data Type:** numeric

Feature_fulltext

- **Description:** Number of times the MATCH … AGAINST() function was used.

- **Scope:** Global, Session
- **Data Type:** `numeric`

`Feature_gis`

- **Description:** Number of times a table with a any of the geometry columns was opened.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Feature_insert_returning`

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.5.0

`Feature_invisible_columns`

- **Description:** Number of invisible columns in all opened tables.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.3.3

`Feature_json`

- **Description:** Number of times JSON functionality has been used, such as one of the JSON functions. Does not include the CONNECT engine JSON type, or EXPLAIN/ANALYZE FORMAT=JSON.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.3.7

`Feature_locale`

- **Description:** Number of times the @@lc_messages variable was assigned into.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Feature_subquery`

- **Description:** Number of subqueries (excluding subqueries in the FROM clause) used.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Feature_system_versioning`

- **Description:** Number of times system versioning functionality has been used (opening a table WITH SYSTEM VERSIONING).
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.3.7

`Feature_timezone`

- **Description:** Number of times an explicit timezone (excluding UTC and SYSTEM) was specified.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Feature_trigger

- **Description:** Number of triggers loaded.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Feature_window_functions

- **Description:** Number of times window functions were used.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.2

Feature_xml

- **Description:** Number of times XML functions (EXTRACTVALUE() and UPDATEXML()) were used.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Flush_commands

- **Description:** Number of FLUSH statements executed, as well as due to internal server flush requests. This differs from Com_flush, which simply counts FLUSH statements, not internal server flush operations.
- **Scope:** Global
- **Data Type:** `numeric`
- **Removed:** MariaDB 10.5.1

Handler_commit

- **Description:** Number of internal COMMIT requests. Differs from Com_commit, which counts the number of COMMIT statements executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Handler_delete

- **Description:** Number of times rows have been deleted from tables. Differs from Com_delete, which counts DELETE statements.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Handler_discover

- **Description:** Discovery is when the server asks the NDBCLUSTER storage engine if it knows about a table with a given name. Handler_discover indicates the number of times that tables have been discovered in this way.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Handler_external_lock

- **Description:** Incremented for each call to the external_lock() function, which generally occurs at the beginning and end of access to a table instance.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Handler_icp_attempts

- **Description:** Number of times pushed index condition was checked. The smaller the ratio of Handler_icp_attempts to Handler_icp_match the better the filtering. See Index Condition Pushdown.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Handler_icp_match`

- **Description:** Number of times pushed index condition was matched. The smaller the ratio of Handler_icp_attempts to Handler_icp_match the better the filtering. See See Index Condition Pushdown.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Handler_mrr_init`

- **Description:** Counts how many MRR (multi-range read) scans were performed. See Multi Range Read optimization.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Handler_mrr_key_refills`

- **Description:** Number of times key buffer was refilled (not counting the initial fill). A non-zero value indicates there wasn't enough memory to do key sort-and-sweep passes in one go. See Multi Range Read optimization.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Handler_mrr_rowid_refills`

- **Description:** Number of times rowid buffer was refilled (not counting the initial fill). A non-zero value indicates there wasn't enough memory to do rowid sort-and-sweep passes in one go. See Multi Range Read optimization.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Handler_prepare`

- **Description:** Number of two-phase commit prepares.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Handler_read_first`

- **Description:** Number of requests to read the first row from an index. A high value indicates many full index scans, e.g. `SELECT a FROM table_name` where `a` is an indexed column.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Handler_read_key`

- **Description:** Number of row read requests based on an index value. A high value indicates indexes are regularly being used, which is usually positive.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Handler_read_last`

- **Description:** Number of requests to read the last row from an index. ORDER BY DESC results in a last-key request followed by several previous-key requests.
- **Scope:** Global, Session

- **Data Type:** `numeric`

`Handler_read_next`

- **Description:** Number of requests to read the next row from an index (in order). Increments when doing an index scan or querying an index column with a range constraint.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Handler_read_prev`

- **Description:** Number of requests to read the previous row from an index (in order). Mostly used with ORDER BY DESC.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Handler_read_retry`

- **Description:** Number of read retrys triggered by semi_consistent_read (InnoDB feature).
- **Scope:** Global
- **Data Type:** `numeric`

`Handler_read_rnd`

- **Description:** Number of requests to read a row based on its position. If this value is high, you may not be using joins that don't use indexes properly, or be doing many full table scans.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Handler_read_rnd_deleted`

- **Description:** Number of requests to delete a row based on its position.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Handler_read_rnd_next`

- **Description:** Number of requests to read the next row. A large number of these may indicate many table scans and improperly used indexes.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Handler_rollback`

- **Description:** Number of transaction rollback requests given to a storage engine. Differs from Com_rollback, which is the number of ROLLBACK commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Handler_savepoint`

- **Description:** Number of transaction savepoint creation requests. Differs from Com_savepoint which is the number of SAVEPOINT commands executed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Handler_savepoint_rollback

- **Description:** Number of requests to rollback to a transaction savepoint.
- **Scope:** Global, Session
- **Data Type:** numeric

Handler_tmp_delete

- **Description:** Number of requests to delete a row in a temporary table.
- **Scope:** Global, Session
- **Data Type:** numeric
- **Introduced:** MariaDB 10.3.0

Handler_tmp_update

- **Description:** Number of requests to update a row to a temporary table.
- **Scope:** Global, Session
- **Data Type:** numeric

Handler_tmp_write

- **Description:** Number of requests to write a row to a temporary table.
- **Scope:** Global, Session
- **Data Type:** numeric

Handler_update

- **Description:** Number of requests to update a row in a table. Since MariaDB 5.3, this no longer counts temporary tables - see Handler_tmp_update.
- **Scope:** Global, Session
- **Data Type:** numeric

Handler_write

- **Description:** Number of requests to write a row to a table. Since MariaDB 5.3, this no longer counts temporary tables - see Handler_tmp_write.
- **Scope:** Global, Session
- **Data Type:** numeric

Key_blocks_not_flushed

- **Description:** Number of key cache blocks which have been modified but not flushed to disk.
- **Scope:** Global
- **Data Type:** numeric

Key_blocks_unused

- **Description:** Number of unused key cache blocks.
- **Scope:** Global
- **Data Type:** numeric

Key_blocks_used

- **Description:** Max number of key cache blocks which have been used simultaneously.
- **Scope:** Global
- **Data Type:** numeric

Key_blocks_warm

- **Description:** Number of key cache blocks in the warm list.
- **Scope:** Global
- **Data Type:** `numeric`

Key_read_requests

- **Description:** Number of key cache block read requests. See Optimizing key_buffer_size.
- **Scope:** Global
- **Data Type:** `numeric`

Key_reads

- **Description:** Number of physical index block reads. See Optimizing key_buffer_size.
- **Scope:** Global
- **Data Type:** `numeric`

Key_write_requests

- **Description:** Number of requests to write a block to the key cache.
- **Scope:** Global
- **Data Type:** `numeric`

Key_writes

- **Description:** Number of key cache block write requests
- **Scope:** Global
- **Data Type:** `numeric`

Last_query_cost

- **Description:** The most recent query optimizer query cost calculation. Can not be calculated for complex queries, such as subqueries or UNION. It will be set to 0 for complex queries.
- **Scope:** Session
- **Data Type:** `numeric`

Maria_*

- **Description:** When the Maria storage engine was renamed Aria, the Maria variables existing at the time were renamed at the same time. See Aria Server Status Variables.

Max_statement_time_exceeded

- **Description:** Number of queries that exceeded the execution time specified by max_statement_time. See Aborting statements that take longer than a certain time to execute.
- **Data Type:** `numeric`

Max_used_connections

- **Description:** Max number of connections ever open at the same time. The global value can be flushed by FLUSH STATUS.
- **Scope:** Global
- **Data Type:** `numeric`

Max_used_connections_time

- **Description:** The time at which the last change of max_used_connections occured. The global value can be flushed by FLUSH STATUS.
- **Scope:** Global
- **Data Type:** `datetime`
- **Introduced:** MariaDB 11.0.2, MariaDB 11.1.1

Memory_used

- **Description:** Global or per-connection memory usage, in bytes. This includes all per-connection memory allocations, but excludes global allocations such as the key_buffer, innodb_buffer_pool etc.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Memory_used_initial

- **Description:** Amount of memory that was used when the server started to service the user connections.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.3.3

Not_flushed_delayed_rows

- **Description:** Number of INSERT DELAYED rows waiting to be written.
- **Scope:** Global
- **Data Type:** `numeric`

Open_files

- **Description:** Number of regular files currently opened by the server. Does not include sockets or pipes, or storage engines using internal functions.
- **Scope:** Global
- **Data Type:** `numeric`

Open_streams

- **Description:** Number of currently opened streams, usually log files.
- **Scope:** Global
- **Data Type:** `numeric`

Open_table_definitions

- **Description:** Number of currently cached .frm files.
- **Scope:** Global
- **Data Type:** `numeric`

Open_tables

- **Description:** Number of currently opened tables, excluding temporary tables.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Opened_files

- **Description:** Number of files the server has opened.
- **Scope:** Global

- **Data Type:** `numeric`

`Opened_plugin_libraries`

- **Description:** Number of shared libraries that the server has opened to load plugins.
- **Scope:** Global
- **Data Type:** `numeric`

`Opened_table_definitions`

- **Description:** Number of .frm files that have been cached.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Opened_tables`

- **Description:** Number of tables the server has opened.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Opened_views`

- **Description:** Number of views the server has opened.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Prepared_stmt_count`

- **Description:** Current number of prepared statements.
- **Scope:** Global
- **Data Type:** `numeric`

`Qcache_free_blocks`

- **Description:** Number of free query cache memory blocks.
- **Scope:** Global
- **Data Type:** `numeric`

`Qcache_free_memory`

- **Description:** Amount of free query cache memory.
- **Scope:** Global
- **Data Type:** `numeric`

`Qcache_hits`

- **Description:** Number of requests served by the query cache. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

`Qcache_inserts`

- **Description:** Number of queries ever cached in the query cache. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global

- **Data Type:** `numeric`

`Qcache_lowmem_prunes`

- **Description:** Number of pruning operations performed to remove old results to make space for new results in the query cache. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

`Qcache_not_cached`

- **Description:** Number of queries that are uncacheable by the query cache, or use SQL_NO_CACHE. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

`Qcache_queries_in_cache`

- **Description:** Number of queries currently cached by the query cache.
- **Scope:** Global
- **Data Type:** `numeric`

`Qcache_total_blocks`

- **Description:** Number of blocks used by the query cache.
- **Scope:** Global
- **Data Type:** `numeric`

`Queries`

- **Description:** Number of statements executed by the server, excluding COM_PING and COM_STATISTICS. Differs from Questions in that it also counts statements executed within stored programs.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Questions`

- **Description:** Number of statements executed by the server, excluding COM_PING, COM_STATISTICS, COM_STMT_PREPARE, COM_STMT_CLOSE, and COM_STMT_RESET statements. Differs from Queries in that it doesn't count statements executed within stored programs.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Resultset_metadata_skipped`

- **Description:** Number of times sending the metadata has been skipped. Metadata is not resent if metadata does not change between prepare and execute of prepared statement, or between executes.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.6.0

`Rows_read`

- **Description:** Number of requests to read a row (excluding temporary tables).
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rows_sent

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rows_tmp_read

- **Description:** Number of requests to read a row in a temporary table.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Select_full_join

- **Description:** Number of joins which did not use an index. If not zero, you may need to check table indexes.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Select_full_range_join

- **Description:** Number of joins which used a range search of the first table.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Select_range

- **Description:** Number of joins which used a range on the first table.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Select_range_check

- **Description:** Number of joins without keys that check for key usage after each row. If not zero, you may need to check table indexes.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Select_scan

- **Description:** Number of joins which used a full scan of the first table.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Slow_launch_threads

- **Description:** Number of threads which took longer than slow_launch_time to create. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Slow_queries

- **Description:** Number of queries which took longer than long_query_time to run. The slow query log does not need to be active for this to be recorded.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Sort_merge_passes

- **Description:** Number of merge passes performed by the sort algorithm. If too high, you may need to look at improving your query indexes, or increasing the sort_buffer_size.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Sort_priority_queue_sorts

- **Description:** The number of times that sorting was done through a priority queue. (The total number of times sorting was done is a sum Sort_range and Sort_scan). See filesort with small LIMIT optimization.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Sort_range

- **Description:** Number of sorts which used a range.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Sort_rows

- **Description:** Number of rows sorted.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Sort_scan

- **Description:** Number of sorts which used a full table scan.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Subquery_cache_hit

- **Description:** Counter for all subquery cache hits. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Subquery_cache_miss

- **Description:** Counter for all subquery cache misses. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Syncs

- **Description:** Number of times my_sync() has been called, or the number of times the server has had to force data to disk. Covers the binary log, .frm creation (if these operations are configured to sync) and some storage engines (Archive, CSV, Aria), but not XtraDB/InnoDB).
- **Scope:** Global, Session
- **Data Type:** `numeric`

Table_locks_immediate

- **Description:** Number of table locks which were completed immediately. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global

- **Data Type:** `numeric`

`Table_locks_waited`

- **Description:** Number of table locks which had to wait. Indicates table lock contention. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

`Table_open_cache_active_instances`

- **Description:** Number of active instances for open tables cache lookups.
- **Scope:**
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.3.3

`Table_open_cache_hits`

- **Description:** Number of hits for open tables cache lookups.
- **Scope:**
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.3.3

`Table_open_cache_misses`

- **Description:** Number of misses for open tables cache lookups.
- **Scope:**
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.3.3

`Table_open_cache_overflows`

- **Description:** Number of overflows for open tables cache lookups.
- **Scope:**
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.3.3

`Tc_log_max_pages_used`

- **Description:** Max number of pages used by the memory-mapped file-based transaction coordinator log. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

`Tc_log_page_size`

- **Description:** Page size of the memory-mapped file-based transaction coordinator log.
- **Scope:** Global
- **Data Type:** `numeric`

`Tc_log_page_waits`

- **Description:** Number of times a two-phase commit was forced to wait for a free memory-mapped file-based transaction coordinator log page. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

Threads_cached

- **Description:** Number of threads cached in the thread cache. This value will be zero if the thread pool is in use.
- **Scope:** Global
- **Data Type:** `numeric`

Threads_connected

- **Description:** Number of clients connected to the server. See Handling Too Many Connections. The `Threads_connected` name is inaccurate when the thread pool is in use, since each client connection does not correspond to a dedicated thread in that case.
- **Scope:** Global
- **Data Type:** `numeric`

Threads_created

- **Description:** Number of threads created to respond to client connections. If too large, look at increasing thread_cache_size.
- **Scope:** Global
- **Data Type:** `numeric`

Threads_running

- **Description:** Number of client connections that are actively running a command, and not just sleeping while waiting to receive the next command to execute. Some internal system threads also count towards this status variable if they would show up in the output of the `SHOW PROCESSLIST` statement.
  - In MariaDB 10.3.2 and before, a global counter was updated each time a client connection dispatched a command. In these versions, the global and session status variable are always the same value.
  - In MariaDB 10.3.3 and later, the global counter has been removed as a performance improvement. Instead, when the global status variable is queried, it is calculated dynamically by essentially adding up all the running client connections as they would appear in `SHOW PROCESSLIST` output. A client connection is only considered to be running if its thread `COMMAND` value is not equal to `Sleep`. When the session status variable is queried, it always returns `1`.
- **Scope:** Global
- **Data Type:** `numeric`

Update_scan

- **Description:** Number of updates that required a full table scan.
- **Scope:** Global
- **Data Type:** `numeric`

Uptime

- **Description:** Number of seconds the server has been running.
- **Scope:** Global
- **Data Type:** `numeric`

Uptime_since_flush_status

- **Description:** Number of seconds since the last FLUSH STATUS.
- **Scope:** Global
- **Data Type:** `numeric`

2.7.2 Server System Variables

5.3.4.5 Aria Status Variables

5.3.4.6 Aria System Variables

5.3.7.13 CONNECT System Variables

3.2.9 Galera Cluster Status Variables

3.2.10 Galera Cluster System Variables

5.3.2.5 InnoDB Server Status Variables

5.3.2.4 InnoDB System Variables

5.4.5.7 MariaDB Audit Plugin Status Variables

5.3.12.3 Mroonga Status Variables

5.3.12.4 Mroonga System Variables

5.3.13.2 MyISAM System Variables

5.3.14.5 MyRocks System Variables

5.3.14.6 MyRocks Status Variables

5.3.15.6 OQGRAPH System and Status Variables

1.1.1.2.9.2.3 Performance Schema Status Variables

1.1.1.2.9.2.4 Performance Schema System Variables

3.1.12 Replication and Binary Log Status Variables

3.1.11 Replication and Binary Log System Variables

3.3.7.23 Semisynchronous Replication Plugin Status Variables

Contents

This page documents status variables related to the Semisynchronous Replication Plugin (which has been merged into the main server from MariaDB 10.3.3). See Server Status Variables for a complete list of status variables that can be viewed with SHOW STATUS.

See also the Full list of MariaDB options, system and status variables.

`Rpl_semi_sync_master_clients`

- **Description:** Number of semisynchronous slaves.
- **Data Type:** `numeric`

`Rpl_semi_sync_master_net_avg_wait_time`

- **Description:** Average time the master waited for a slave to reply, in microseconds.
- **Data Type:** `numeric`

`Rpl_semi_sync_master_net_wait_time`

- **Description:** Total time the master waited for slave replies, in microseconds.
- **Data Type:** `numeric`

`Rpl_semi_sync_master_net_waits`

- **Description:** Total number of times the master waited for slave replies.
- **Data Type:** `numeric`

`Rpl_semi_sync_master_no_times`

- **Description:** Number of times the master turned off semisynchronous replication. The global value can be flushed by `FLUSH STATUS`.
- **Data Type:** `numeric`

`Rpl_semi_sync_master_no_tx`

- **Description:** Number of commits that have not been successfully acknowledged by a slave. The global value can be flushed by `FLUSH STATUS`.
- **Data Type:** `numeric`

`Rpl_semi_sync_master_status`

- **Description:** Whether or not semisynchronous replication is currently operating on the master. The value will be `ON` if both the plugin has been enabled and a commit acknowledgment has occurred. It will be `OFF` if either the plugin has not been enabled, or the master is replicating asynchronously due to a commit acknowledgment timeout.

- **Data Type:** `boolean`

`Rpl_semi_sync_master_timefunc_failures`

- **Description:** Number of times the master failed when calling a time function, such as gettimeofday(). The global value can be flushed by `FLUSH STATUS` .
- **Data Type:** `numeric`

`Rpl_semi_sync_master_tx_avg_wait_time`

- **Description:** Average time the master waited for each transaction, in microseconds.
- **Data Type:** `numeric`

`Rpl_semi_sync_master_tx_wait_time`

- **Description:** Total time the master waited for transactions, in microseconds.
- **Data Type:** `numeric`

`Rpl_semi_sync_master_tx_waits`

- **Description:** Total number of times the master waited for transactions.
- **Data Type:** `numeric`

`Rpl_semi_sync_master_wait_pos_backtraverse`

- **Description:** Total number of times the master waited for an event that had binary coordinates lower than previous events waited for. Occurs if the order in which transactions start waiting for a reply is different from the order in which their binary log events were written. The global value can be flushed by `FLUSH STATUS` .
- **Data Type:** `numeric`

`Rpl_semi_sync_master_wait_sessions`

- **Description:** Number of sessions that are currently waiting for slave replies.
- **Data Type:** `numeric`

`Rpl_semi_sync_master_yes_tx`

- **Description:** Number of commits that have been successfully acknowledged by a slave. The global value can be flushed by `FLUSH STATUS` .
- **Data Type:** `numeric`

`Rpl_semi_sync_slave_status`

- **Description:** Whether or not semisynchronous replication is currently operating on the slave. `ON` if both the plugin has been enabled and the slave I/O thread is running.
- **Data Type:** `boolean`

3.1.24 Semisynchronous Replication

3.3.7.25 Sphinx Status Variables

Contents

This page documents status variables related to the Sphinx storage engine. See Server Status Variables for a complete list of status variables that can be viewed with SHOW STATUS.

See also the Full list of MariaDB options, system and status variables.

`Sphinx_error`

- **Description:** See SHOW ENGINE SPHINX STATUS.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Sphinx_time`

- **Description:** See SHOW ENGINE SPHINX STATUS.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Sphinx_total`

- **Description:** See SHOW ENGINE SPHINX STATUS.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Sphinx_total_found`

- **Description:** See SHOW ENGINE SPHINX STATUS.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Sphinx_word_count`

- **Description:** See SHOW ENGINE SPHINX STATUS.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Sphinx_words`

- **Description:** See SHOW ENGINE SPHINX STATUS.
- **Scope:** Global, Session
- **Data Type:** `numeric`

5.3.19.9 Spider Status Variables

3.3.7.27 Spider Server System Variables

3.3.7.28 SQL_ERROR_LOG Plugin System Variables

Contents

This page documents system variables related to the SQL_Error_Log Plugin. See Server System Variables for a complete list of system variables and instructions on setting them.

See also the Full list of MariaDB options, system and status variables.

sql_error_log_filename

- **Description:** The name (and optionally path) of the logfile containing the errors. Rotation will use a naming convention such as `sql_error_log_filename.001`. If no path is specified, the log file will be written to the data directory.
- **Commandline:** `--sql-error-log-filename=value`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** `sql_errors.log`

sql_error_log_rate

- **Description:** The logging sampling rate. Setting to `10`, for example, means that one in ten errors will be logged. If set to zero, logging is disabled. The default, `1`, logs every error.
- **Commandline:** `--sql-error-log-rate=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `1`

sql_error_log_rotate

- **Description:** Setting to #1 `forces log rotation.`
- **Commandline:** `--sql-error-log-rate[={0|1}]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

sql_error_log_rotations

- **Description:** Number of rotations before the log is removed. When rotated, the current log file is stored and a new, empty, log is created. Any rotations older than this setting are removed.
- **Commandline:** `--sql-error-log-rotations=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `9`
- **Range:** `1` to `999`

sql_error_log_size_limit

- **Description:** The log file size limit in bytes. After reaching this size, the log file is rotated.
- **Commandline:** `--sql-error-log-size-limit=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `1000000`

- **Range:** `100` to `9223372036854775807`

2.2.1.1.1.7 SSL/TLS Status Variables

2.2.1.1.1.6 SSL/TLS System Variables

3.3.7.31 Thread Pool System and Status Variables

Contents

This article describes the system and status variables used by the MariaDB thread pool. For a full description, see Thread Pool in MariaDB.

System variables

`extra_max_connections`

- **Description:** The number of connections on the `extra_port`.
  - See Thread Pool in MariaDB: Configuring the Extra Port for more information.
- **Commandline:** `--extra-max-connections=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1`
- **Range:** `1` to `100000`

`extra_port`

- **Description:** Extra port number to use for TCP connections in a `one-thread-per-connection` manner. If set to `0`, then no extra port is used.
  - See Thread Pool in MariaDB: Configuring the Extra Port for more information.
- **Commandline:** `--extra-port=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `0`

`thread_handling`

- **Description:** Determines how the server handles threads for client connections. In addition to threads for client

connections, this also applies to certain internal server threads, such as Galera slave threads. On Windows, if you would like to use the thread pool, then you do not need to do anything, because the default for the thread_handling system variable is already preset to `pool-of-threads`.
    - When the default `one-thread-per-connection` mode is enabled, the server uses one thread to handle each client connection.
    - When the `pool-of-threads` mode is enabled, the server uses the thread pool for client connections.
    - When the `no-threads` mode is enabled, the server uses a single thread for all client connections, which is really only usable for debugging.
- **Commandline:** `--thread-handling=name`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `enumeration`
- **Default Value:** `one-thread-per-connection` (non-Windows), `pool-of-threads` (Windows)
- **Valid Values:** `no-threads`, `one-thread-per-connection`, `pool-of-threads`.
- **Documentation:** Using the thread pool.
- **Notes:** In MySQL, the thread pool is only available in MySQL Enterprise. In MariaDB it's available in all versions.

---

thread_pool_dedicated_listener

- **Description:** If set to 1, then each group will have its own dedicated listener, and the listener thread will not pick up work items. As a result, the queueing time in the Information Schema Threadpool_Queues and the actual queue size in the Information Schema Threadpool_Groups table will be more exact, since IO requests are immediately dequeued from poll, without delay.
    - This system variable is only meaningful on **Unix**.
- **Commandline:** `thread-pool-dedicated-listener={0|1}`
- **Scope:**
- **Dynamic:**
- **Data Type:** `boolean`
- **Default Value:** `0`
- **Introduced:** MariaDB 10.5.0

---

thread_pool_exact_stats

- **Description:** If set to 1, provides better queueing time statistics by using a high precision timestamp, at a small performance cost, for the time when the connection was added to the queue. This timestamp helps calculate the queuing time shown in the Information Schema Threadpool_Queues table.
    - This system variable is only meaningful on **Unix**.
- **Commandline:** `thread-pool-exact-stats={0|1}`
- **Scope:**
- **Dynamic:**
- **Data Type:** `boolean`
- **Default Value:** `0`
- **Introduced:** MariaDB 10.5.0

---

thread_pool_idle_timeout

- **Description:** The number of seconds before an idle worker thread exits. The default value is `60`. If there is currently no work to do, how long should an idle thread wait before exiting?
    - This system variable is only meaningful on **Unix**.
    - The `thread_pool_min_threads` system variable is comparable for Windows.
- **Commandline:** `thread-pool-idle-timeout=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `60`
- **Documentation:** Using the thread pool.

---

thread_pool_max_threads

- **Description:** The maximum number of threads in the thread pool. Once this limit is reached, no new threads will be created in most cases.

    - On Unix, in rare cases, the actual number of threads can slightly exceed this, because each thread group needs at least two threads (i.e. at least one worker thread and at least one listener thread) to prevent deadlocks.
- **Scope:**
- **Commandline:** `thread-pool-max-threads=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:**
    - `65536` (>= MariaDB 10.2.4)
    - `1000` (<= MariaDB 10.2.3, >= MariaDB 10.1)
    - `500` (<= MariaDB 10.0)
- **Range:** `1` to `65536`
- **Documentation:** Using the thread pool.

---

thread_pool_min_threads

- **Description:** Minimum number of threads in the thread pool. In bursty environments, after a period of inactivity, threads would normally be retired. When the next burst arrives, it would take time to reach the optimal level. Setting this value higher than the default would prevent thread retirement even if inactive.
    - This system variable is only meaningful on **Windows**.
    - The `thread_pool_idle_timeout` system variable is comparable for Unix.
- **Commandline:** `thread-pool-min-threads=#`
- **Data Type:** `numeric`
- **Default Value:** `1`
- **Documentation:** Using the thread pool.

---

thread_pool_oversubscribe

- **Description:** Determines how many worker threads in a thread group can remain active at the same time once a thread group is oversubscribed due to stalls. The default value is `3`. Usually, a thread group only has one active worker thread at a time. However, the timer thread can add more active worker threads to a thread group if it detects a stall. There are trade-offs to consider when deciding whether to allow **only one** thread per CPU to run at a time, or whether to allow **more than one** thread per CPU to run at a time. Allowing only one thread per CPU means that the thread can have unrestricted access to the CPU while its running, but it also means that there is additional overhead from putting threads to sleep or waking them up more frequently. Allowing more than one thread per CPU means that the threads have to share the CPU, but it also means that there is less overhead from putting threads to sleep or waking them up.
    - See Thread Groups in the Unix Implementation of the Thread Pool: Thread Group Oversubscription for more information.
    - This is primarily for **internal** use, and it is **not** meant to be changed for most users.
    - This system variable is only meaningful on **Unix**.
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `3`
- **Range:** `1` to `65536`
- **Documentation:** Using the thread pool.

---

thread_pool_prio_kickup_timer

- **Description:** Time in milliseconds before a dequeued low-priority statement is moved to the high-priority queue.
    - This system variable is only meaningful on **Unix**.
- **Commandline:** `thread-pool-kickup-timer=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1000`
- **Range:** `0` to `4294967295`
- **Introduced:** MariaDB 10.2.2
- **Documentation:** Using the thread pool.

---

thread_pool_priority

- **Description:** Thread pool priority. High-priority connections usually start executing earlier than low-priority. If set to 'auto' (the default), the actual priority (low or high) is determined by whether or not the connection is inside a transaction.
- **Commandline:** `--thread-pool-priority=#`
- **Scope:** Global,Connection
- **Data Type:** `enum`
- **Default Value:** `auto`
- **Valid Values:** `high`, `low`, `auto`.
- **Introduced:** MariaDB 10.2.2
- **Documentation:** Using the thread pool.

---

thread_pool_size

- **Description:** The number of thread groups in the thread pool, which determines how many statements can execute simultaneously. The default value is the number of CPUs on the system. When setting this system variable's value at system startup, the max value is 100000. However, it is not a good idea to set it that high. When setting this system variable's value dynamically, the max value is either 128 or the value that was set at system startup--whichever value is higher.
  - See Thread Groups in the Unix Implementation of the Thread Pool for more information.
  - This system variable is only meaningful on **Unix**.
- **Commandline:** `--thread-pool-size=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** Based on the number of processors (but see MDEV-7806).
- **Range:** `1` to `128` (< MariaDB 5.5.37, MariaDB 10.0.11), `1` to `100000` (>= MariaDB 5.5.37, MariaDB 10.0.11)
- **Documentation:** Using the thread pool.

---

thread_pool_stall_limit

- **Description:** The number of milliseconds between each stall check performed by the timer thread. The default value is `500`. Stall detection is used to prevent a single client connection from monopolizing a thread group. When the timer thread detects that a thread group is stalled, it wakes up a sleeping worker thread in the thread group, if one is available. If there isn't one, then it creates a new worker thread in the thread group. This temporarily allows several client connections in the thread group to run in parallel. However, note that the timer thread will not create a new worker thread if the number of threads in the thread pool is already greater than or equal to the maximum defined by the `thread_pool_max_threads` variable, unless the thread group does not already have a listener thread.
  - See Thread Groups in the Unix Implementation of the Thread Pool: Thread Group Stalls for more information.
  - This system variable is only meaningful on **Unix**.
  - Note that if you are migrating from the MySQL Enterprise thread pool plugin, then the unit used in their implementation is 10ms, not 1ms.
- **Commandline:** `--thread-pool-stall-limit=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `500`
- **Range:** `10` to `4294967295` (< MariaDB 10.5), `1` to `4294967295` (>= MariaDB 10.5)
- **Documentation:** Using the thread pool.

Status variables

Threadpool_idle_threads

- **Description:** Number of inactive threads in the thread pool. Threads become inactive for various reasons, such as by waiting for new work. However, an inactive thread is not necessarily one that has not been assigned work. Threads are also considered inactive if they are being blocked while waiting on disk I/O, or while waiting on a lock, etc.
  - This status variable is only meaningful on **Unix**.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Threadpool_threads`

- **Description:** Number of threads in the thread pool. In rare cases, this can be slightly higher than `thread_pool_max_threads`, because each thread group needs at least two threads (i.e. at least one worker thread and at least one listener thread) to prevent deadlocks.
- **Scope:** Global, Session
- **Data Type:** `numeric`

3.3.7.32 MariaDB Optimization for MySQL Users

MariaDB contains many new options and optimizations which, for compatibility or other reasons, are not enabled in the default install. Enabling them helps you gain extra performance from the same hardware when upgrading from MySQL to MariaDB. This article contains information on options and settings which you should enable, or at least look in to, when making the switch.

```
aria-pagecache-buffer-size=##
```

If you are using a log of on-disk temporary tables, increase the above to as much as you can afford. See Aria Storage Engine for more details.

```
key-cache-segments=8
```

If you use/have a lot of MyISAM files, increase the above to 4 or 8. See Segmented Key Cache and Segmented Key Cache Performance for more information.

```
thread-handling=pool-of-threads
```

Threadpool is a great way to increase performance in situations where queries are relatively short and the load is CPU bound (e.g. OLTP workloads). To enable it, add the above to your my.cnf file. See Threadpool in 5.5 for more information.

5.3.2.7 InnoDB Buffer Pool

5.3.2.8 InnoDB Change Buffering

3.3.7.35 Optimizing table_open_cache

table_open_cache can be a useful variable to adjust to improve performance.

Each concurrent session accessing the same table does so independently. This improves performance, although it comes at a cost of extra memory usage.

table_open_cache indicates the maximum number of tables the server can keep open in any one table cache instance. Ideally, you'd like this set so as to re-open a table as infrequently as possible.

However, note that this is not a hard limit. When the server needs to open a table, it evicts the least recently used closed table from the cache, and adds the new table. If all tables are used, the server adds the new table and does not evict any table. As soon as a table is not used anymore, it will be evicted from the list even if no table needs to be open, until the number of open tables will be equal to *table_open_cache*

table_open_cache has defaulted to 2000 since MariaDB 10.1.7. Before that, the default was 400.

You can view the current setting in the my.cnf file, or by running:

```
select @@table_open_cache;
+--------------------+
| @@table_open_cache |
+--------------------+
|                400 |
+--------------------+
```

To evaluate whether you could do with a higher table_open_cache, look at the number of opened tables, in conjunction with the server uptime (Opened_tables and Uptime status variables):

```
show global status like 'opened_tables';
+---------------+--------+
| Variable_name | Value  |
+---------------+--------+
| Opened_tables | 354858 |
+---------------+--------+
```

If the number of opened tables is increasing rapidly, you should look at increasing the table_open_cache value. Try to find a value that sees a slow, or possibly even no, increase in the number of opened tables.

> Make sure that your operating system can cope with the number of open file descriptors required by the table_open_cache setting. If table_open_cache is set too high, MariaDB may start to refuse connections as the operating system runs out of file descriptors. Also note that the MyISAM (and Aria?) storage engines need two file descriptors per open table.

It's possible that the open_table_cache can even be reduced.

If your number of open_tables has not yet reached the table_open_cache_size, and the server has been up a while, you can look at decreasing the value.

```
show global status like 'open_tables';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Open_tables   | 354   |
+---------------+-------+
```

The open table cache can be emptied with FLUSH TABLES or with the `flush-tables` or `refresh` mariadb-admin commands.

Automatic Creation of New Table Open Cache Instances

MariaDB Server can create multiple instances of the table open cache. It initially creates just a single instance. However, whenever it detects contention on the existing instances, it will automatically create a new instance. When the number of instances has been increased due to contention, it does not decrease again.

When MariaDB Server creates a new instance, it prints a message like the following to the error log:

```
[Note] Detected table cache mutex contention at instance 1: 25% waits. Additional
  table cache instance activated. Number of instances after activation: 2.
```

The maximum number of instances is defined by the table_open_cache_instances system variable. The default value of the table_open_cache_instances system variable is `8`, which is expected to handle up to 100 CPU cores. If your system is larger than this, then you may benefit from increasing the value of this system variable.

Depending on the ratio of actual available file handles, and table_open_cache size, the max. instance count may be auto adjusted to a lower value on server startup.

The implementation and behavior of this feature is different than the same feature in MySQL 5.6.

3.3.7.36 Optimizing key_buffer_size

key_buffer_size is a MyISAM variable which determines the size of the index buffers held in memory, which affects the speed of index reads. Note that Aria tables by default make use of an alternative setting, aria-pagecache-buffer-size.

A good rule of thumb for servers consisting particularly of MyISAM tables is for about 25% or more of the available server memory to be dedicated to the key buffer.

A good way to determine whether to adjust the value is to compare the key_read_requests value, which is the total value of requests to read an index, and the key_reads values, the total number of requests that had to be read from disk.

The ratio of key_reads to key_read_requests should be as low as possible, 1:100 is the highest acceptable, 1:1000 is better, and 1:10 is terrible.

The effective maximum size might be lower than what is set, depending on the server's available physical RAM and the per-

process limit determined by the operating system.

If you don't make use of MyISAM tables at all, you can set this to a very low value, such as 64K.

5.3.13.8 Segmented Key Cache

3.3.7.38 Big Query Settings

MariaDB 5.3 and beyond have a number of features that are targeted at big queries and so are disabled by default.

This page describes recommended settings for IO-bound queries that shovel through lots of records.

First, turn on Batched Key Access:

```
# Turn on disk-ordered reads
optimizer_switch='mrr=on'
optimizer_switch='mrr_cost_based=off'

# Turn on Batched Key Access (BKA)
join_cache_level = 6
```

Give BKA buffer space to operate on. Ideally, it should have enough space to fit all the data examined by the query.

```
# Size limit for the whole join
join_buffer_space_limit = 300M

# Limit for each individual table
join_buffer_size = 100M
```

Turn on index_merge/sort-intersection:

```
optimizer_switch='index_merge_sort_intersection=on'
```

If your queries examine big fraction of the tables (somewhere more than ~ 30%), turn on hash join:

```
# Turn on both Hash Join and Batched Key Access
join_cache_level = 8
```

3.3.7.39 Sample my.cnf Files

Place holder for sample my.cnf files, customized for different memory size and storage engines. In addition, we'd like to hear from you what works for you, so the knowledge can be crowd-sourced and shared.

3.3.7.40 Handling Too Many Connections

Systems that get too busy can return the `too_many_connections` error.

When the number of threads_connected exceeds the `max_connections` server variable, it's time to make a change. Viewing the threads_connected status variable shows only the current number of connections, but it's more useful to see what the value has peaked at, and this is shown by the max_used_connections status variable.

This error may be a symptom of slow queries and other bottlenecks, but if the system is running smoothly this can be addressed by increasing the value of max_connections.

2.1.14.1.12 System Variable Differences between MariaDB and MySQL

3.3.6 MariaDB Memory Allocation

3.3.7.43 Setting Innodb Buffer Pool Size

Dynamically

Resizing the buffer pool is performed in chunks determined by the size of the innodb_buffer_pool_chunk_size variable.

The resize operation waits until all active transactions and operations are completed, and new transactions and operations that need to access the buffer pool must wait until the resize is complete (although when decreasing the size, access is permitted when defragmenting and withdrawing pages).

Nested transactions may fail if started after the buffer pool resize has begun.

The new buffer pool size must be a multiple of innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances (note that `innodb_buffer_pool_instances` is ignored from MariaDB 10.5, and removed in MariaDB 10.6, as the buffer pool is no longer split into multiple instances). If you attempt to set a different figure, the value is automatically adjusted to a multiple of at least the attempted size. Note that adjusting the innodb_buffer_pool_chunk_size setting can result in a change in the buffer pool size.

The number of chunks as calculated by innodb_buffer_pool_size / innodb_buffer_pool_chunk_size should not exceed 1000 in order to avoid performance issues.

A background thread performs the resizing operation. The Innodb_buffer_pool_resize_status status variable shows the progress of the resizing operation, for example:

```
SHOW STATUS LIKE 'Innodb_buffer_pool_resize_status';
+----------------------------------+----------------------------------+
| Variable_name                    | Value                            |
+----------------------------------+----------------------------------+
| Innodb_buffer_pool_resize_status | Resizing also other hash tables. |
+----------------------------------+----------------------------------+
```

or

```
SHOW STATUS LIKE 'Innodb_buffer_pool_resize_status';
+----------------------------------+-------------------------------------------------------+
| Variable_name                    | Value                                                 |
+----------------------------------+-------------------------------------------------------+
| Innodb_buffer_pool_resize_status | Completed resizing buffer pool at 161103 16:26:54. |
+----------------------------------+-------------------------------------------------------+
```

Progress is also logged in the error log.

3.3.8 Buffers, Caches and Threads

MariaDB makes use of a number of buffering, caching and threading techniques to improve performance.

Thread Pool

MariaDB thread pool

Thread States

Descriptions of the various thread states

InnoDB Buffer Pool

The most important memory buffer used by InnoDB.

InnoDB Change Buffering

Buffering INSERT, UPDATE and DELETE statements for greater efficiency.

Query Cache

Caching SELECT queries for better performance.

Segmented Key Cache

Collection of structures for regular MyISAM key caches

Subquery Cache

Subquery cache for optimizing the evaluation of correlated subqueries.

Thread Command Values

Thread command values from SHOW PROCESSLIST or Information Schema PROCESSLIST Table

There are 1 related questions.

3.3.8.1 Thread Pool

MariaDB 5.1 introduced a thread pool, while MariaDB 5.5 introduced an improved version.

Thread Pool in MariaDB
Thread pool introduced in MariaDB 5.5.

Thread Groups in the Unix Implementation of the Thread Pool
On Unix, the thread pool divides up client connections into independent sets of threads.

Thread Pool System and Status Variables
System and status variables related to the MariaDB thread pool.

Thread Pool in MariaDB 5.1 - 5.3
The old thread pool introduced in MariaDB 5.1

There are 1 related questions.

3.3.8.2 Thread Pool in MariaDB

Contents

Problem That Thread Pools Solve

The task of scalable server software (and a DBMS like MariaDB is an example of such software) is to maintain top performance with an increasing number of clients. MySQL traditionally assigned a thread for every client connection, and as the number of concurrent users grows this model shows performance drops. Many active threads are a performance killer, because increasing the number of threads leads to extensive context switching, bad locality for CPU caches, and increased contention for hot locks. An ideal solution that would help to reduce context switching is to maintain a lower number of threads than the number of clients. But this number should not be too low either, since we also want to utilize CPUs to their fullest, so ideally, there should be a single active thread for each CPU on the machine.

MariaDB Thread Pool Features

The current MariaDB thread pool was implemented in MariaDB 5.5. It replaced the legacy thread pool that was introduced in MariaDB 5.1. The main drawback of the previous solution was that this pool was static–it had a fixed number of threads. Static thread pools can have their merits, for some limited use cases, such as cases where callbacks executed by the threads never block and do not depend on each other. For example, iimagine something like an echo server.

However, DBMS clients are more complicated. For example, a thread may depend on another thread's completion, and they may block each other via locks and/or I/O. Thus it is very hard, and sometimes impossible, to predict how many threads would be ideal or even sufficient to prevent deadlocks in every situation. MariaDB 5.5 implements a dynamic/adaptive pool that itself takes care of creating new threads in times of high demand and retiring threads if they have nothing to do. This is a complete reimplementation of the legacy `pool-of-threads` scheduler, with the following goals:

- Make the pool dynamic, so that it will grow and shrink whenever required.
- Minimize the amount of overhead that is required to maintain the thread pool itself.
- Make the best use of underlying OS capabilities. For example, if a native thread pool implementation is available, then it should be used, and if not, then the best I/O multiplexing method should be used.
- Limit the resources used by threads.

There are currently two different low-level implementations – depending on OS. One implementation is designed specifically for Windows which utilizes a native `CreateThreadpool` API. The second implementation is primarily intended to be used in Unix-like systems. Because the implementations are different, some system variables differ between Windows and Unix.

When to Use the Thread Pool

Thread pools are most efficient in situations where queries are relatively short and the load is CPU-bound, such as in OLTP workloads. If the workload is not CPU-bound, then you might still benefit from limiting the number of threads to save memory for the database memory buffers.

When the Thread Pool is Less Efficient

There are special, rare cases where the thread pool is likely to be less efficient.

- If you have a **very bursty workload**, then the thread pool may not work well for you. These tend to be workloads in which there are long periods of inactivity followed by short periods of very high activity by many users. These also tend to be workloads in which delays cannot be tolerated, so the throttling of thread creation that the thread pool uses is not ideal. Even in this situation, performance can be improved by tweaking how often threads are retired. For example, with `thread_pool_idle_timeout` on Unix, or with `thread_pool_min_threads` on Windows.
- If you have **many concurrent, long, non-yielding** queries, then the thread pool may not work well for you. In this context, a "non-yielding" query is one that never waits or which does not indicate waits to the thread pool. These kinds of workloads are mostly used in data warehouse scenarios. Long-running, non-yielding queries will delay execution of other queries. However, the thread pool has stall detection to prevent them from totally monopolizing the thread pool. See Thread Groups in the Unix Implementation of the Thread Pool: Thread Group Stalls for more information. Even when the whole thread pool is blocked by non-yielding queries, you can still connect to the server through the `extra-port` TCP/IP port.
- If you rely on the fact that **simple queries always finish quickly**, no matter how loaded your database server is, then the thread pool may not work well for you. When the thread pool is enabled on a busy server, even simple queries might be queued to be executed later. This means that even if the statement itself doesn't take much time to execute, even a simple `SELECT 1`, might take a bit longer when the thread pool is enabled than with `one-thread-per-connection` if it gets queued.

Configuring the Thread Pool

The `thread_handling` system variable is the primary system variable that is used to configure the thread pool.

There are several other system variables as well, which are described in the sections below. Many of the system variables documented below are dynamic, meaning that they can be changed with `SET GLOBAL` on a running server.

Generally, there is no need to tweak many of these system variables. The goal of the thread pool was to provide good performance out-of-the box. However, the system variable values can be changed, and we intended to expose as many knobs from the underlying implementation as we could. Feel free to tweak them as you see fit.

If you find any issues with any of the default behavior, then we encourage you to submit a bug report.

See Thread Pool System and Status Variables for the full list of the thread pool's system variables.

Configuring the Thread Pool on Unix

On Unix, if you would like to use the thread pool, then you can use the thread pool by setting the `thread_handling` system variable to `pool-of-threads` in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
thread_handling=pool-of-threads
```

The following system variables can also be configured on Unix:

- `thread_pool_size` – The number of thread groups in the thread pool, which determines how many statements can execute simultaneously. The default value is the number of CPUs on the system. When setting this system variable's value at system startup, the max value is 100000. However, it is not a good idea to set it that high. When setting this system variable's value dynamically, the max value is either 128 or the value that was set at system startup--whichever value is higher. See Thread Groups in the Unix Implementation of the Thread Pool for more information.
- `thread_pool_max_threads` – The maximum number of threads in the thread pool. Once this limit is reached, no new threads will be created in most cases. In rare cases, the actual number of threads can slightly exceed this, because each thread group needs at least two threads (i.e. at least one worker thread and at least one listener thread) to prevent deadlocks. The default value in MariaDB 5.5 and MariaDB 10.0 is `500`. The default value in MariaDB 10.1 is `1000` in MariaDB 10.1. The default value in MariaDB 10.2 and later is `65536`.
- `thread_pool_stall_limit` – The number of milliseconds between each stall check performed by the timer thread. The default value is `500`. Stall detection is used to prevent a single client connection from monopolizing a thread group. When the timer thread detects that a thread group is stalled, it wakes up a sleeping worker thread in the thread group, if one is available. If there isn't one, then it creates a new worker thread in the thread group. This temporarily allows several client connections in the thread group to run in parallel. However, note that the timer thread will not create a new worker thread if the number of threads in the thread pool is already greater than or equal to the maximum defined by the `thread_pool_max_threads` variable, unless the thread group does not already have a listener thread. See Thread Groups in the Unix Implementation of the Thread Pool: Thread Group Stalls for more information.
- `thread_pool_oversubscribe` – Determines how many worker threads in a thread group can remain active at the same time once a thread group is oversubscribed due to stalls. The default value is `3`. Usually, a thread group only has one active worker thread at a time. However, the timer thread can add more active worker threads to a thread group if it detects a stall. There are trade-offs to consider when deciding whether to allow **only one** thread per CPU to run at a time, or whether to allow **more than one** thread per CPU to run at a time. Allowing only one thread per CPU means that the thread can have unrestricted access to the CPU while its running, but it also means that there is additional overhead from putting threads to sleep or waking them up more frequently. Allowing more than one thread per CPU means that the threads have to share the CPU, but it also means that there is less overhead from putting threads to sleep or waking them up. This is primarily for **internal** use, and it is **not** meant to be changed for most users. See Thread Groups in the Unix Implementation of the Thread Pool: Thread Group Oversubscription for more information.
- `thread_pool_idle_timeout` – The number of seconds before an idle worker thread exits. The default value is `60`. If there is currently no work to do, how long should an idle thread wait before exiting?

Configuring the Thread Pool on Windows

The Windows implementation of the thread pool uses a native thread pool created with the `CreateThreadpool` API.

On Windows, if you would like to use the thread pool, then you do not need to do anything, because the default for the `thread_handling` system variable is already preset to `pool-of-threads`.

However, if you would like to use the old one thread per-connection behavior on Windows, then you can use use that by setting the `thread_handling` system variable to `one-thread-per-connection` in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
thread_handling=one-thread-per-connection
```

> On older versions of Windows, such as XP and 2003, `pool-of-threads` is not implemented, and the server will silently switch to using the legacy `one-thread-per-connection` method.

The native `CreateThreadpool` API allows applications to set the minimum and maximum number of threads in the pool. The following system variables can be used to configure those values on Windows:

- `thread_pool_min_threads` – The minimum number of threads in the pool. Default is 1. This applicable in a special case of very “bursty” workloads. Imagine having longer periods of inactivity after periods of high activity. While the thread pool is idle, Windows would decide to retire pool threads (based on experimentation, this seems to happen

after thread had been idle for 1 minute). Next time high load comes, it could take some milliseconds or seconds until the thread pool size stabilizes again to optimal value. To avoid thread retirement, one could set the parameter to a higher value.

- `thread_pool_max_threads` – The maximum number of threads in the pool. Threads are not created when this value is reached. The default from MariaDB 5.5 to MariaDB 10.0 is 500 (this has been increased to 1000 in MariaDB 10.1). This parameter can be used to prevent the creation of new threads if the pool can have short periods where many or all clients are blocked (for example, with “FLUSH TABLES WITH READ LOCK”, high contention on row locks, or similar). New threads are created if a blocking situation occurs (such as after a throttling interval), but sometimes you want to cap the number of threads, if you’re familiar with the application and need to, for example, save memory. If your application constantly pegs at 500 threads, it might be a strong indicator for high contention in the application, and the thread pool does not help much.

Configuring Priority Scheduling

Starting with MariaDB 10.2.2, it is possible to configure connection prioritization. The priority behavior is configured by the `thread_pool_priority` system variable.

By default, if `thread_pool_priority` is set to `auto`, then queries would be given a higher priority, in case the current connection is inside a transaction. This allows the running transaction to finish faster, and has the effect of lowering the number of transactions running in parallel. The default setting will generally improve throughput for transactional workloads. But it is also possible to explicitly set the priority for the current connection to either 'high' or 'low'.

There is also a mechanism in place to ensure that higher priority connections are not monopolizing the worker threads in the pool (which would cause indefinite delays for low priority connections). On Unix, low priority connections are put into the high priority queue after the timeout specified by the `thread_pool_prio_kickup_timer` system variable.

Configuring the Extra Port

MariaDB allows you to configure an extra port for administrative connections. This is primarily intended to be used in situations where all threads in the thread pool are blocked, and you still need a way to access the server. However, it can also be used to ensure that monitoring systems (including MaxScale's monitors) always have access to the system, even when all connections on the main port are used. This extra port uses the old `one-thread-per-connection` thread handling.

You can enable this and configure a specific port by setting the `extra_port` system variable.

You can configure a specific number of connections for this port by setting the `extra_max_connections` system variable.

These system variables can be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
extra_port = 8385
extra_max_connections = 10
```

Once you have the extra port configured, you can use the mariadb client with the `-P` option to connect to the port.

```
$ mariadb -u root -P 8385 -p
```

Monitoring Thread Pool Activity

Currently there are two status variables exposed to monitor pool activity.

Variable	Description
`Threadpool_threads`	Number of threads in the thread pool. In rare cases, this can be slightly higher than `thread_pool_max_threads`, because each thread group needs at least two threads (i.e. at least one worker thread and at least one listener thread) to prevent deadlocks.
`Threadpool_idle_threads`	Number of inactive threads in the thread pool. Threads become inactive for various reasons, such as by waiting for new work. However, an inactive thread is not necessarily one that has not been assigned work. Threads are also considered inactive if they are being blocked while waiting on disk I/O, or while waiting on a lock, etc. This status variable is only meaningful on **Unix**.

Thread Groups in the Unix Implementation of the Thread Pool

On Unix, the thread pool implementation uses objects called thread groups to divide up client connections into many independent sets of threads. See Thread Groups in the Unix Implementation of the Thread Pool for more information.

Fixing a Blocked Thread Pool

When using global locks, even with a high value on the thread_pool_max_threads system variable, it is still possible to block the entire pool.

Imagine the case where a client performs FLUSH TABLES WITH READ LOCK then pauses. If then the number of other clients connecting to the server to start write operations exceeds the maximum number of threads allowed in the pool, it can block the Server. This makes it impossible to issue the UNLOCK TABLES statement. It can also block MaxScale from monitoring the Server.

To mitigate the issue, MariaDB allows you to configure an extra port for administrative connections. See Configuring the Extra Port for information on how to configure this.

Once you have the extra port configured, you can use the mariadb client with the `-P` option to connect to the port.

```
$ mariadb -u root -P 8385 -p
```

This ensures that your administrators can access the server in cases where the number of threads is already equal to the configured value of the thread_pool_max_threads system variable, and all threads are blocked. It also ensures that MaxScale can still access the server in such situations for monitoring information.

Once you are connected to the extra port, you can solve the issue by increasing the value on the thread_pool_max_threads system variable, or by killing the offending connection, (that is, the connection that holds the global lock, which would be in the `sleep` state).

MariaDB Thread Pool vs Oracle MySQL Enterprise Thread Pool

Commercial editions of MySQL since 5.5 include an Oracle MySQL Enterprise thread pool implemented as a plugin, which delivers similar functionality. A detailed discussion about the design of the feature is at Mikael Ronstrom's blog. Here is the summary of similarities and differences, based on the above materials.

Similarities

- On Unix, both MariaDB and Oracle MySQL Enterprise Threadpool will partition client connections into groups. The thread_pool_size parameter thus has the same meaning for both MySQL and MariaDB.
- Both implementations use similar schema checking for thread stalls, and both have the same parameter name for thread_pool_stall_limit (though in MariaDB it is measured using millisecond units, not 10ms units like in Oracle MySQL).

Differences

- The Windows implementation is completely different – MariaDB's uses native Windows threadpooling, while Oracle's implementation includes a convenience function *WSAPoll()* (provided for convenience to port Unix applications). As a consequence of relying on *WSAPoll()*, Oracle's implementation will not work with named pipes and shared memory connections.
- MariaDB uses the most efficient I/O multiplexing facilities for each operating system: Windows (the I/O completion port is used internally by the native threadpool), Linux (epoll), Solaris (event ports), FreeBSD and OSX (kevent). Oracle uses optimized I/O multiplexing only on Linux, with epoll, and uses poll() otherwise.
- Unlike Oracle MySQL Enterprise Threadpool, MariaDB's one is builtin, not a plugin.

MariaDB Thread Pool vs Percona Thread Pool

Percona's implementation is a port of the MariaDB's threadpool with some added features. In particular, Percona added priority scheduling to its 5.5-5.7 releases. MariaDB 10.2 and Percona priority scheduling work in a similar fashion, but there are some differences in details.

- MariaDB's 10.2 thread_pool_priority=auto,high, low correspond to Percona's thread_pool_high_prio_mode=transactions,statements,none
- Percona has a thread_pool_high_prio_tickets connection variable to allow every nth low priority query to be put into the high priority queue. MariaDB does not have corresponding settings.
- MariaDB has a thread_pool_prio_kickup_timer setting, which Percona does not have.

Thread Pool Internals

Low-level implementation details are documented in the WL#246

Running Benchmarks

When running sysbench and maybe other benchmarks, that create many threads on the same machine as the server, it is advisable to run benchmark driver and server on different CPUs to get the realistic results. Running lots of driver threads and only several server threads on the same CPUs will have the effect that OS scheduler will schedule benchmark driver threads to run with much higher probability than the server threads, that is driver will pre-empt the server. Use "taskset –c" on Linuxes, and "set /affinity" on Windows to separate benchmark driver and server CPUs, as the preferred method to fix this situation.

A possible alternative on Unix (if taskset or a separate machine running the benchmark is not desired for some reason) would be to increase thread_pool_size to make the server threads more "competitive" against the client threads.

When running sysbench, a good rule of thumb could be to give 1/4 of all CPUs to the sysbench, and 3/4 of CPUs to mysqld. It is also good idea to run sysbench and mysqld on different NUMA nodes, if possible.

Notes

The thread_cache_size system variable is not used when the thread pool is used and the Threads_cached status variable will have a value of 0.

3.3.8.3 Thread Groups in the Unix Implementation of the Thread Pool

This article does not apply to the thread pool implementation on Windows. On Windows, MariaDB uses a native thread pool created with the `CreateThreadpool` API, which has its own methods to distribute threads between CPUs.

Contents

On Unix, the thread pool implementation uses objects called thread groups to divide up client connections into many independent sets of threads. The `thread_pool_size` system variable defines the number of thread groups on a system. Generally speaking, the goal of the thread group implementation is to have one running thread on each CPU on the system at a time. Therefore, the default value of the `thread_pool_size` system variable is auto-sized to the number of CPUs on the system.

When setting the `thread_pool_size` system variable's value at system startup, the max value is `100000`. However, it is not a good idea to set it that high. When setting its value dynamically, the max value is either `128` or the value that was set at system startup--whichever value is higher. It can be changed dynamically with `SET GLOBAL`. For example:

```
SET GLOBAL thread_pool_size=32;
```

It can also be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
..
thread_handling=pool-of-threads
thread_pool_size=32
```

If you do not want MariaDB to use all CPUs on the system for some reason, then you can set it to a lower value than the number of CPUs. For example, this would make sense if the MariaDB Server process is limited to certain CPUs with the `taskset` utility on Linux.

If you set the value to the number of CPUs and if you find that the CPUs are still underutilized, then try increasing the value.

The `thread_pool_size` system variable tends to have the most visible performance effect. It is roughly equivalent to the number of threads that can run at the same time. In this case, run means use CPU, rather than sleep or wait. If a client connection needs to sleep or wait for some reason, then it wakes up another client connection in the thread group before it does so.

> One reason that CPU underutilization may occur in rare cases is that the thread pool is not always informed when a thread is going to wait. For example, some waits, such as a page fault or a miss in the OS buffer cache, cannot be detected by MariaDB. Prior to MariaDB 10.0, network I/O related waits could also be missed.

Distributing Client Connections Between Thread Groups

When a new client connection is created, its thread group is determined using the following calculation:

```
thread_group_id = connection_id %  thread_pool_size
```

The `connection_id` value in the above calculation is the same monotonically increasing number that you can use to identify connections in `SHOW PROCESSLIST` output or the `information_schema.PROCESSLIST` table.

This calculation should assign client connections to each thread group in a round-robin manner. In general, this should result in an even distribution of client connections among thread groups.

Types of Threads

Thread Group Threads

Thread groups have two different kinds of threads: a **listener thread** and **worker threads**.

- A thread group's **worker threads** actually perform work on behalf of client connections. A thread group can have many **worker threads**, but usually, only one will be actively running at a time. This is not always the case. For example, the thread group can become *oversubscribed* if the thread pool's **timer thread** detects that the thread group is *stalled*. This is explained more in the sections below.
- A thread group's **listener thread** listens for I/O events and distributes work to the **worker threads**. If it detects that there is a request that needs to be worked on, then it can wake up a sleeping **worker thread** in the thread group, if any exist. If the **listener thread** is the only thread in the thread group, then it can also create a new **worker thread**. If there is only one request to handle, and if the `thread_pool_dedicated_listener` system variable is not enabled, then the **listener thread** can also become a **worker thread** and handle the request itself. This helps decrease the overhead that may be introduced by excessively waking up sleeping **worker threads** and excessively creating new **worker threads**.

Global Threads

The thread pool has one global thread: a **timer thread**. The **timer thread** performs tasks, such as:

- Checks each thread group for stalls.
- Ensures that each thread group has a **listener thread**.

Thread Creation

A new thread is created in a thread group in the scenarios listed below.

In all of the scenarios below, the thread pool implementation prefers to wake up a sleeping **worker thread** that already

exists in the thread group, rather than to create a new thread.

Worker Thread Creation by Listener Thread

A thread group's **listener thread** can create a new **worker thread** when it has more client connection requests to distribute, but no pre-existing **worker threads** are available to work on the requests. This can help to ensure that the thread group always has enough threads to keep one **worker thread** active at a time.

A thread group's **listener thread** creates a new **worker thread** if all of the following conditions are met:

- The **listener thread** receives a client connection request that needs to be worked on.
- There are more client connection requests in the thread group's work queue that the **listener thread** still needs to distribute to **worker threads**, so the **listener thread** should not become a **worker thread**.
- There are no active **worker threads** in the thread group.
- There are no sleeping **worker threads** in the thread group that the **listener thread** can wake up.
- And one of the following conditions is also met:
  - The entire thread pool has fewer than `thread_pool_max_threads`.
  - There are fewer than two threads in the thread group. This is to guarantee that each thread group can have at least two threads, even if `thread_pool_max_threads` has already been reached or exceeded.

Thread Creation by Worker Threads during Waits

A thread group's **worker thread** can create a new **worker thread** when the thread has to wait on something, and the thread group has more client connection requests queued, but no pre-existing **worker threads** are available to work on them. This can help to ensure that the thread group always has enough threads to keep one **worker thread** active at a time. For most workloads, this tends to be the primary mechanism that creates new **worker threads**.

A thread group's **worker thread** creates a new thread if all of the following conditions are met:

- The **worker thread** has to wait on some request. For example, it might be waiting on disk I/O, or it might be waiting on a lock, or it might just be waiting for a query that called the `SLEEP()` function to finish.
- There are no active **worker threads** in the thread group.
- There are no sleeping **worker threads** in the thread group that the **worker thread** can wake up.
- And one of the following conditions is also met:
  - The entire thread pool has fewer than `thread_pool_max_threads`.
  - There are fewer than two threads in the thread group. This is to guarantee that each thread group can have at least two threads, even if `thread_pool_max_threads` has already been reached or exceeded.
- And one of the following conditions is also met:
  - There are more client connection requests in the thread group's work queue that the **listener thread** still needs to distribute to **worker threads**. In this case, the new thread is intended to be a **worker thread**.
  - There is currently no **listener thread** in the thread group. For example, if the `thread_pool_dedicated_listener` system variable is not enabled, then the thread group's **listener thread** can became a **worker thread**, so that it could handle some client connection request. In this case, the new thread can become the thread group's **listener thread**.

Listener Thread Creation by Timer Thread

The thread pool's **timer thread** can create a new **listener thread** for a thread group when the thread group has more client connection requests that need to be distributed, but the thread group does not currently have a **listener thread** to distribute them. This can help to ensure that the thread group does not miss client connection requests because it has no **listener thread**.

The thread pool's **timer thread** creates a new **listener thread** for a thread group if all of the following conditions are met:

- The thread group has not handled any I/O events since the last check by the timer thread.
- There is currently no **listener thread** in the thread group. For example, if the `thread_pool_dedicated_listener` system variable is not enabled, then the thread group's **listener thread** can became a **worker thread**, so that it could handle some client connection request. In this case, the new thread can become the thread group's **listener thread**.
- There are no sleeping **worker threads** in the thread group that the **timer thread** can wake up.
- And one of the following conditions is also met:
  - The entire thread pool has fewer than `thread_pool_max_threads`.
  - There are fewer than two threads in the thread group. This is to guarantee that each thread group can have at least two threads, even if `thread_pool_max_threads` has already been reached or exceeded.
- If the thread group already has active **worker threads**, then the following condition also needs to be met:
  - A **worker thread** has not been created for the thread group within the *throttling interval*.

Worker Thread Creation by Timer Thread during Stalls

The thread pool's **timer thread** can create a new **worker thread** for a thread group when the thread group is stalled. This can help to ensure that a long query can't monopole its thread group.

The thread pool's **timer thread** creates a new **worker thread** for a thread group if all of the following conditions are met:

- The **timer thread** thinks that the thread group is stalled. This means that the following conditions have been met:
  - There are more client connection requests in the thread group's work queue that the **listener thread** still needs to distribute to **worker threads**.
  - No client connection requests have been allowed to be dequeued to run since the last stall check by the **timer thread**.
- There are no sleeping **worker threads** in the thread group that the **timer thread** can wake up.
- And one of the following conditions is also met:
  - The entire thread pool has fewer than `thread_pool_max_threads`.
  - There are fewer than two threads in the thread group. This is to guarantee that each thread group can have at least two threads, even if `thread_pool_max_threads` has already been reached or exceeded.
- A **worker thread** has not been created for the thread group within the *throttling interval*.

Thread Creation Throttling

In some of the scenarios listed above, a thread is only created within a thread group if no new threads have been created for the thread group within the *throttling interval*. The throttling interval depends on the number of threads that are already in the thread group.

MariaDB starting with 10.5

In MariaDB 10.5 and later, thread creation is not throttled until a thread group has more than 1 + `thread_pool_oversubscribe` threads:

Number of Threads in Thread Group	Throttling Interval (milliseconds)
0-(1 + `thread_pool_oversubscribe`)	0
4-7	50 * `THROTTLING_FACTOR`
8-15	100 * `THROTTLING_FACTOR`
16-65536	20 * `THROTTLING_FACTOR`

```
THROTTLING_FACTOR = (thread_pool_stall_limit / MAX (500,thread_pool_stall_limit))
```

MariaDB until 10.4

In MariaDB 10.4 and before, thread creation is throttled when a thread group has more than 3 threads:

Number of Threads in Thread Group	Throttling Interval (milliseconds)
0-3	0
4-7	50
8-15	100
16-65536	200

Thread Group Stalls

The thread pool has a feature that allows it to detect if a client connection is executing a long-running query that may be monopolizing its thread group. If a client connection were to monopolize its thread group, then that could prevent other client connections in the thread group from running their queries. In other words, the thread group would appear to be *stalled*.

This stall detection feature is implemented by creating a **timer thread** that periodically checks if any of the thread groups are stalled. There is only a single **timer thread** for the entire thread pool. The `thread_pool_stall_limit` system variable defines the number of milliseconds between each stall check performed by the timer thread. The default value is `500`. It can be changed dynamically with `SET GLOBAL`. For example:

```
SET GLOBAL thread_pool_stall_limit=300;
```

It can also be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
..
thread_handling=pool-of-threads
thread_pool_size=32
thread_pool_stall_limit=300
```

The **timer thread** considers a thread group to be stalled if the following is true:

- There are more client connection requests in the thread group's work queue that the **listener thread** still needs to distribute to **worker threads**.
- No client connection requests have been allowed to be dequeued to run since the last stall check by the **timer thread**.

This indicates that the one or more client connections currently using the active **worker threads** may be monopolizing the thread group, and preventing the queued client connections from performing work. When the **timer thread** detects that a thread group is stalled, it wakes up a sleeping **worker thread** in the thread group, if one is available. If there isn't one, then it creates a new **worker thread** in the thread group. This temporarily allows several client connections in the thread group to run in parallel.

The `thread_pool_stall_limit` system variable essentially defines the limit for what a "fast query" is. If a query takes longer than `thread_pool_stall_limit`, then the thread pool is likely to think that it is too slow, and it will either wake up a sleeping worker thread or create a new worker thread to let another client connection in the thread group run a query in parallel.

In general, changing the value of the `thread_pool_stall_limit` system variable has the following effect:

- Setting it to **higher** values can help avoid starting too many parallel threads if you expect a lot of client connections to execute long-running queries.
- Setting it to **lower** values can help prevent deadlocks.

Thread Group Oversubscription

If the **timer thread** were to detect a stall in a thread group, then it would either wake up a sleeping **worker thread** or create a new **worker thread** in that thread group. At that point, the thread group would have multiple active **worker threads**. In other words, the thread group would be *oversubscribed*.

You might expect that the thread pool would shutdown one of the **worker threads** when the stalled client connection finished what it was doing, so that the thread group would only have one active **worker thread** again. However, this does not always happen. Once a thread group is oversubscribed, the `thread_pool_oversubscribe` system variable defines the upper limit for when **worker threads** start shutting down after they finish work for client connections. The default value is `3`. It can be changed dynamically with `SET GLOBAL`. For example:

```
SET GLOBAL thread_pool_oversubscribe=10;
```

It can also be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
..
thread_handling=pool-of-threads
thread_pool_size=32
thread_pool_stall_limit=300
thread_pool_oversubscribe=10
```

To clarify, the `thread_pool_oversubscribe` system variable does not play any part in the creation of new **worker threads**. The `thread_pool_oversubscribe` system variable is only used to determine how many **worker threads** should remain active in a thread group, once a thread group is already oversubscribed due to stalls.

In general, the default value of `3` should be adequate for most users. Most users should not need to change the value of the `thread_pool_oversubscribe` system variable.

3.3.7.31 Thread Pool System and Status Variables

3.3.8.5 Thread Pool in MariaDB 5.1 - 5.3

Contents

This article describes the old thread pool in MariaDB 5.1 - 5.3.

MariaDB 5.5 and later use an improved thread pool - see Thread pool in MariaDB.

About pool of threads

This is an extended version of the pool-of-threads code from MySQL 6.0. This allows you to use a limited set of threads to handle all queries, instead of the old 'one-thread-per-connection' style. In recent times, its also been referred to as "thread pool" or "thread pooling" as this feature (in a different implementation) is available in Enterprise editions of MySQL (not in the Community edition).

This can be a very big win if most of your queries are short running queries and there are few table/row locks in your system.

Instructions

To enable pool-of-threads you must first run configure with the `--with-libevent` option. (This is automatically done if you use any 'max' scripts in the BUILD directory):

```
./configure --with-libevent
```

When starting mysqld with the pool of threads code you should use

```
mysqld --thread-handling=pool-of-threads --thread-pool-size=20
```

Default values are:

```
thread-handling=  one-thread-per-connection
thread-pool-size= 20
```

One issue with pool-of-threads is that if all worker threads are doing work (like running long queries) or are locked by a row/table lock no new connections can be established and you can't login and find out what's wrong or login and kill queries.

To help this, we have introduced two new options for mysqld; extra_port and extra_max_connections:

```
--extra-port=#              (Default 0)
--extra-max-connections=#   (Default 1)
```

If extra-port is <> 0 then you can connect max_connections number of normal threads + 1 extra SUPER user through the 'extra-port' TCP/IP port. These connections use the old one-thread-per-connection method.

To connect with through the extra port, use:

```
mysql --port='number-of-extra-port' --protocol=tcp
```

This allows you to freely use, on connection bases, the optimal connection/thread model.

3.3.9 Thread States

Thread states can be viewed with the `STATE` values listed by the SHOW PROCESSLIST statement or in the Information Schema PROCESSLIST Table as well as the `PROCESSLIST_STATE` value listed in the Performance Schema threads Table. `Slave_IO_State` shown by SHOW SLAVE STATUS also shows slave-related thread states.

Delayed Insert Connection Thread States

Thread states related to the connection thread that processes INSERT DELAYED statements

Delayed Insert Handler Thread States

Thread states related to the handler thread that inserts the results of INSERT DELAYED statements

Event Scheduler Thread States

Thread states related to the Event Scheduler

General Thread States

Thread states

Master Thread States

Thread states related to replication master threads

Query Cache Thread States

Thread states related to the query cache

Slave Connection Thread States

Thread states related to slave connection threads

Slave I/O Thread States

Thread states related to replication slave I/O threads

Slave SQL Thread States

Thread states related to replication slave SQL threads.

3.3.9.1 Delayed Insert Connection Thread States

This article documents thread states that are related to the connection thread that processes INSERT DELAYED statements.

These correspond to the `STATE` values listed by the SHOW PROCESSLIST statement or in the Information Schema PROCESSLIST Table as well as the `PROCESSLIST_STATE` value listed in the Performance Schema threads Table.

Value	Description
allocating local table	Preparing to allocate rows to the delayed-insert handler thread. Follows from the `got handler lock` state.
Creating delayed handler	Creating a handler for the delayed-inserts.
got handler lock	Lock to access the delayed-insert handler thread has been received. Follows from the `waiting for handler lock` state and before the `allocating local table` state.
got old table	The initialization phase is over. Follows from the `waiting for handler open` state.
storing row into queue	Adding new row to the list of rows to be inserted by the delayed-insert handler thread.
waiting for delay_list	Initializing (trying to find the delayed-insert handler thread).
waiting for handler insert	Waiting for new inserts, as all inserts have been processed.
waiting for handler lock	Waiting for delayed insert-handler lock to access the delayed-insert handler thread.
waiting for handler open	Waiting for the delayed-insert handler thread to initialize. Follows from the `Creating delayed handler` state and before the `got old table` state.

3.3.9.2 Delayed Insert Handler Thread States

This article documents thread states that are related to the handler thread that inserts the results of INSERT DELAYED statements.

These correspond to the `STATE` values listed by the SHOW PROCESSLIST statement or in the Information Schema PROCESSLIST Table as well as the `PROCESSLIST_STATE` value listed in the Performance Schema threads Table.

Value	Description
insert	About to insert rows into the table.
reschedule	Sleeping in order to let other threads function, after inserting a number of rows into the table.
upgrading lock	Attempting to get lock on the table in order to insert rows.
Waiting for INSERT	Waiting for the delayed-insert connection thread to add rows to the queue.

3.3.9.3 Event Scheduler Thread States

This article documents thread states that are related to event scheduling and execution. These include the Event Scheduler thread, threads that terminate the Event Scheduler, and threads for executing events.

These correspond to the `STATE` values listed by the SHOW PROCESSLIST statement or in the Information Schema PROCESSLIST Table as well as the `PROCESSLIST_STATE` value listed in the Performance Schema threads Table

Value	Description
Clearing	Thread is terminating.
Initialized	Thread has be initialized.
Waiting for next activation	The event queue contains items, but the next activation is at some time in the future.
Waiting for scheduler to stop	Waiting for the event scheduler to stop after issuing `SET GLOBAL event_scheduler=OFF`.
Waiting on empty queue	Sleeping, as the event scheduler's queue is empty.

3.3.9.4 General Thread States

This article documents the major general thread states. More specific lists related to delayed inserts, replication, the query cache and the event scheduler are listed in:

- Event Scheduler Thread States
- Query Cache Thread States
- Master Thread States
- Slave Connection Thread States
- Slave I/O Thread States
- Slave SQL Thread States

These correspond to the `STATE` values listed by the SHOW PROCESSLIST statement or in the Information Schema PROCESSLIST Table as well as the `PROCESSLIST_STATE` value listed in the Performance Schema threads Table

Value	Description
After create	The function that created (or tried to create) a table (temporary or non-temporary) has just ended.
Analyzing	Calculating table key distributions, such as when running an ANALYZE TABLE statement.
checking permissions	Checking to see whether the permissions are adequate to perform the statement.
Checking table	Checking the table.
cleaning up	Preparing to reset state variables and free memory after executing a command.
closing tables	Flushing the changes to disk and closing the table. This state will only persist if the disk is full or under extremely high load.
converting HEAP to Aria	Converting an internal MEMORY temporary table into an on-disk Aria temporary table.
converting HEAP to MyISAM	Converting an internal MEMORY temporary table into an on-disk MyISAM temporary table.
copy to tmp table	A new table has been created as part of an ALTER TABLE statement, and rows are about to be copied into it.

Copying to group table	Sorting the rows by group and copying to a temporary table, which occurs when a statement has different GROUP BY and ORDER BY criteria.
Copying to tmp table	Copying to a temporary table in memory.
Copying to tmp table on disk	Copying to a temporary table on disk, as the resultset is too large to fit into memory.
Creating index	Processing an ALTER TABLE ... ENABLE KEYS for an Aria or MyISAM table.
Creating sort index	Processing a SELECT statement resolved using an internal temporary table.
creating table	Creating a table (temporary or non-temporary).
Creating tmp table	Creating a temporary table (in memory or on-disk).
deleting from main table	Deleting from the first table in a multi-table delete, saving columns and offsets for use in deleting from the other tables.
deleting from reference tables	Deleting matched rows from secondary reference tables as part of a multi-table delete.
discard_or_import_tablespace	Processing an ALTER TABLE ... IMPORT TABLESPACE or ALTER TABLE ... DISCARD TABLESPACE statement.
end	State before the final cleanup of an ALTER TABLE, CREATE VIEW, DELETE, INSERT, SELECT, or UPDATE statement.
executing	Executing a statement.
Execution of init_command	Executing statements specified by the `--init_command` mariadb client option.
filling schema table	A table in the `information_schema` database is being built.
freeing items	Freeing items from the query cache after executing a command. Usually followed by the `cleaning up` state.
Flushing tables	Executing a FLUSH TABLES statement and waiting for other threads to close their tables.
FULLTEXT initialization	Preparing to run a full-text search
init	About to initialize an ALTER TABLE, DELETE, INSERT, SELECT, or UPDATE statement. Could be performaing query cache cleanup, or flushing the binary log or InnoDB log.
Killed	Thread will abort next time it checks the kill flag. Requires waiting for any locks to be released.
Locked	Query has been locked by another query.
logging slow query	Writing statement to the slow query log.
NULL	State used for SHOW PROCESSLIST.
login	Connection thread has not yet been authenticated.
manage keys	Enabling or disabling a table index.
Opening table[s]	Trying to open a table. Usually very quick unless the limit set by table_open_cache has been reached, or an ALTER TABLE or LOCK TABLE is in progress.
optimizing	Server is performing initial optimizations in for a query.
preparing	State occurring during query optimization.
Purging old relay logs	Relay logs that are no longer needed are being removed.
query end	Query has finished being processed, but items have not yet been freed (the `freeing items` state.
Reading file	Server is reading the file (for example during LOAD DATA INFILE).
Reading from net	Server is reading a network packet.
Removing duplicates	Duplicated rows being removed before sending to the client. This happens when `SELECT DISTINCT` is used in a way that the distinct operation could not be optimized at an earlier point.
removing tmp table	Removing an internal temporary table after processing a SELECT statement.
rename	Renaming a table.

rename result table	Renaming a table that results from an ALTER TABLE statement having created a new table.
Reopen tables	Table is being re-opened after thread obtained a lock but the underlying table structure had changed, so the lock was released.
Repair by sorting	Indexes are being created with the use of a sort. Much faster than the related `Repair with keycache`.
Repair done	Multi-threaded repair has been completed.
Repair with keycache	Indexes are being created through the key cache, one-by-one. Much slower than the related `Repair by sorting`.
Rolling back	A transaction is being rolled back.
Saving state	New table state is being saved. For example, after, analyzing a MyISAM table, the key distributions, rowcount etc. are saved to the .MYI file.
Searching rows for update	Finding matching rows before performing an UPDATE, which is needed when the UPDATE would change the index used for the UPDATE
Sending data	Sending data to the client as part of processing a SELECT statement or other statements that returns data like INSERT ... RETURNING . Often the longest-occurring state as it also include all reading from tables and disk read activities. Where an aggregation or un-indexed filtering occurs there is significantly more rows read than what is sent to the client.
setup	Setting up an ALTER TABLE operation.
Sorting for group	Sorting as part of a GROUP BY
Sorting for order	Sorting as part of an ORDER BY
Sorting index	Sorting index pages as part of a table optimization operation.
Sorting result	Processing a SELECT statement using a non-temporary table.
statistics	Calculating statistics as part of deciding on a query execution plan. Usually a brief state unless the server is disk-bound.
System lock	Requesting or waiting for an *external* lock for a specific table. The storage engine determines what kind of *external* lock to use. For example, the MyISAM storage engine uses file-based locks. However, MyISAM's *external* locks are disabled by default, due to the default value of the `skip_external_locking` system variable. Transactional storage engines such as InnoDB also register the transaction or statement with MariaDB's transaction coordinator while in this thread state. See MDEV-19391 for more information about that.
Table lock	About to request a table's *internal* lock after acquiring the table's *external* lock. This thread state occurs after the *System lock* thread state.
update	About to start updating table.
Updating	Searching for and updating rows in a table.
updating main table	Updating the first table in a multi-table update, and saving columns and offsets for use in the other tables.
updating reference tables	Updating the secondary (reference) tables in a multi-table update
updating status	This state occurs after a query's execution is complete. If the query's execution time exceeds `long_query_time`, then `Slow_queries` is incremented, and if the slow query log is enabled, then the query is logged. If the `SERVER_AUDIT` plugin is enabled, then the query is also logged into the audit log at this stage. If the `userstats` plugin is enabled, then CPU statistics are also updated at this stage.
User lock	About to request or waiting for an advisory lock from a GET LOCK() call. For SHOW PROFILE, means requesting a lock only.
User sleep	A SLEEP() call has been invoked.
Waiting for commit lock	FLUSH TABLES WITH READ LOCK is waiting for a commit lock, or a statement resulting in an explicit or implicit commit is waiting for a read lock to be released. This state was called `Waiting for all running commits to finish` in earlier versions.
Waiting for global read lock	Waiting for a global read lock.

Waiting for table level lock	External lock acquired,and internal lock about to be requested. Occurs after the `System lock` state. In earlier versions, this was called `Table lock`.
Waiting for *xx* lock	Waiting to obtain a lock of type `xx`.
Waiting on cond	Waiting for an unspecified condition to occur.
Writing to net	Writing a packet to the network.

3.3.9.5 Master Thread States

This article documents thread states that are related to replication master threads. These correspond to the `STATE` values listed by the SHOW PROCESSLIST statement or in the Information Schema PROCESSLIST Table as well as the `PROCESSLIST_STATE` value listed in the Performance Schema threads Table.

Value	Description
Finished reading one binlog; switching to next binlog	After completing one binary log, the next is being opened for sending to the slave.
Master has sent all binlog to slave; waiting for binlog to be updated	All events have been read from the binary logs and sent to the slave. Now waiting for the binary log to be updated with new events.
Sending binlog event to slave	An event has been read from the binary log, and is now being sent to the slave.
Waiting to finalize termination	State that only occurs very briefly while the thread is terminating.

3.3.9.6 Query Cache Thread States

This article documents thread states that are related to the Query Cache. These correspond to the `STATE` values listed by the SHOW PROCESSLIST statement or in the Information Schema PROCESSLIST Table as well as the `PROCESSLIST_STATE` value listed in the Performance Schema threads Table.

Value	Description
checking privileges on cached query	Checking whether the user has permission to access a result in the query cache.
checking query cache for query	Checking whether the current query exists in the query cache.
invalidating query cache entries	Marking query cache entries as invalid as the underlying tables have changed.
sending cached result to client	A result found in the query cache is being sent to the client.
storing result in query cache	Saving the the result of a query into the query cache.
Waiting for query cache lock	Waiting to take a query cache lock.

3.3.9.7 Slave Connection Thread States

This article documents thread states that are related to connection threads that occur on a replication slave. These correspond to the `STATE` values listed by the SHOW PROCESSLIST statement or in the Information Schema PROCESSLIST Table as well as the `PROCESSLIST_STATE` value listed in the Performance Schema threads Table.

Value	Description
Changing master	Processing a CHANGE MASTER TO statement.
Killing slave	Processing a STOP SLAVE statement.
Opening master dump table	A table has been created from a master dump and is now being opened.
Reading master dump table data	After the table created by a master dump (the `Opening master dump table` state) the table is now being read.
Rebuilding the index on master dump table	After the table created by a master dump has been opened and read (the `Reading master dump table data` state), the index is built.

3.3.9.8 Slave I/O Thread States

This article documents thread states that are related to replication slave I/O threads. These correspond to the `Slave_IO_State` shown by SHOW SLAVE STATUS and the `STATE` values listed by the SHOW PROCESSLIST statement or in the Information Schema PROCESSLIST Table as well as the `PROCESSLIST_STATE` value listed in the Performance Schema threads Table.

Value	Description
Checking master version	Checking the master's version, which only occurs very briefly after establishing a connection with the master.
Connecting to master	Attempting to connect to master.
Queueing master event to the relay log	Event is being copied to the relay log after being read, where it can be processed by the SQL thread.
Reconnecting after a failed binlog dump request	Attempting to reconnect to the master after a previously failed binary log dump request.
Reconnecting after a failed master event read	Attempting to reconnect to the master after a previously failed request. After successfully connecting, the state will change to `Waiting for master to send event`.
Registering slave on master	Registering the slave on the master, which only occurs very briefly after establishing a connection with the master.
Requesting binlog dump	Requesting the contents of the binary logs from the given log file name and position. Only occurs very briefly after establishing a connection with the master.
Waiting for master to send event	Waiting for binary log events to arrive after successfully connecting. If there are no new events on the master, this state can persist for as many seconds as specified by the slave_net_timeout system variable, after which the thread will reconnect.
Waiting for slave mutex on exit	Waiting for slave mutex while the thread is stopping. Only occurs very briefly.
Waiting for the slave SQL thread to free enough relay log space.	Relay log has reached its maximum size, determined by relay_log_space_limit (no limit by default), so waiting for the SQL thread to free up space by processing enough relay log events.
Waiting for master update	State before connecting to master.
Waiting to reconnect after a failed binlog dump request	Waiting to reconnect after a binary log dump request has failed due to disconnection. The length of time in this state is determined by the `MASTER_CONNECT_RETRY` clause of the CHANGE MASTER TO statement.
Waiting to reconnect after a failed master event read	Sleeping while waiting to reconnect after a disconnection error. The time in seconds is determined by the `MASTER_CONNECT_RETRY` clause of the CHANGE MASTER TO statement.

3.3.9.9 Slave SQL Thread States

This article documents thread states that are related to replication slave SQL threads. These correspond to the `Slave_SQL_State` shown by SHOW SLAVE STATUS as well as the `STATE` values listed by the SHOW PROCESSLIST statement and the Information Schema PROCESSLIST as well as the `PROCESSLIST_STATE` value listed in the Performance Schema threads Table.

Value	Description
Apply log event	Log event is being applied.
Making temp file	Creating a temporary file containing the row data as part of a LOAD DATA INFILE statement.
Reading event from the relay log	Reading an event from the relay log in order to process the event.
Slave has read all relay log; waiting for the slave I/O thread to update it	All relay log events have been processed, now waiting for the I/O thread to write new events to the relay log.
Waiting for work from SQL thread	In parallel replication the worker thread is waiting for more things from the SQL thread.
Waiting for prior transaction to start commit before starting next transaction	In parallel replication the worker thread is waiting for conflicting things to end before starting executing.

Waiting for worker threads to be idle	Happens in parallel replication when moving to a new binary log after a master restart. All slave temporary files are deleted and worker threads are restarted.
Waiting due to global read lock	In parallel replication when worker threads are waiting for a global read lock to be released.
Waiting for worker threads to pause for global read lock	FLUSH TABLES WITH READ LOCK is waiting for worker threads to finish what they are doing.
Waiting while replication worker thread pool is busy	Happens in parallel replication during a FLUSH TABLES WITH READ LOCK or when changing number of parallel workers.
Waiting for other master connection to process GTID received on multiple master connections	A worker thread noticed that there is already another thread executing the same GTID from another connection and it's waiting for the other to complete.
Waiting for slave mutex on exit	Thread is stopping. Only occurs very briefly.
Waiting for the next event in relay log	State before reading next event from the relay log.

5.3.2.7 InnoDB Buffer Pool

5.3.2.8 InnoDB Change Buffering

3.3.9.12 Query Cache

Contents

The query cache stores results of SELECT queries so that if the identical query is received in future, the results can be quickly returned.

This is extremely useful in high-read, low-write environments (such as most websites). It does not scale well in environments with high throughput on multi-core machines, so it is disabled by default.

Note that the query cache cannot be enabled in certain environments. See Limitations.

Setting Up the Query Cache

Unless MariaDB has been specifically built without the query cache, the query cache will always be available, although inactive. The have_query_cache server variable will show whether the query cache is available.

```
SHOW VARIABLES LIKE 'have_query_cache';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| have_query_cache | YES   |
+------------------+-------+
```

If this is set to `NO`, you cannot enable the query cache unless you rebuild or reinstall a version of MariaDB with the cache available.

To see if the cache is enabled, view the query_cache_type server variable. It is enabled by default in MariaDB versions up to 10.1.6, but disabled starting with MariaDB 10.1.7 - if needed enable it by setting `query_cache_type` to `1`.

Although enabled in versions prior to MariaDB 10.1.7, the query_cache_size is by default 0KB there, which effectively disables the query cache. From 10.1.7 on the cache size defaults to 1MB. If needed set the cache to a size large enough amount, for example:

```
SET GLOBAL query_cache_size = 1000000;
```

Starting from MariaDB 10.1.7, `query_cache_type` is automatically set to ON if the server is started with the `query_cache_size` set to a non-zero (and non-default) value.

See Limiting the size of the Query Cache below for details.

How the Query Cache Works

When the query cache is enabled and a new SELECT query is processed, the query cache is examined to see if the query appears in the cache.

Queries are considered identical if they use the same database, same protocol version and same default character set. Prepared statements are always considered as different to non-prepared statements, see Query cache internal structure for more info.

If the identical query is not found in the cache, the query will be processed normally and then stored, along with its result set, in the query cache. If the query is found in the cache, the results will be pulled from the cache, which is much quicker than processing it normally.

Queries are examined in a case-sensitive manner, so :

```
SELECT * FROM t
```

Is different from :

```
select * from t
```

Comments are also considered and can make the queries differ, so :

```
/* retry */SELECT * FROM t
```

Is different from :

```
/* retry2 */SELECT * FROM t
```

See the query_cache_strip_comments server variable for an option to strip comments before searching.

Each time changes are made to the data in a table, all affected results in the query cache are cleared. It is not possible to retrieve stale data from the query cache.

When the space allocated to query cache is exhausted, the oldest results will be dropped from the cache.

When using `query_cache_type=ON`, and the query specifies `SQL_NO_CACHE` (case-insensitive), the server will not cache the query and will not fetch results from the query cache.

When using `query_cache_type=DEMAND` (after MDEV-6631 feature request) and the query specifies `SQL_CACHE`, the server will cache the query.

One important point of MDEV-6631 is : switching between `query_cache_type=ON` and `query_cache_type=DEMAND` can "turn off" query cache of old queries without the `SQL_CACHE` string, that's not yet defined if we should include another `query_cache_type` (DEMAND_NO_PRUNE) value or not to allow use of old queries

Queries Stored in the Query Cache

If the query_cache_type system variable is set to `1`, or `ON`, all queries fitting the size constraints will be stored in the cache unless they contain a `SQL_NO_CACHE` clause, or are of a nature that caching makes no sense, for example making use of a function that returns the current time. Check that `SQL_NO_CACHE` will force server to don't use query cache locks.

If any of the following functions are present in a query, it will not be cached. Queries with these functions are sometimes called 'non-deterministic' - don't get confused with the use of this term in other contexts.

BENCHMARK()	CONNECTION_ID()
CONVERT_TZ()	CURDATE()
CURRENT_DATE()	CURRENT_TIME()
CURRENT_TIMESTAMP()	CURTIME()
DATABASE()	ENCRYPT() (one parameter)
FOUND_ROWS()	GET_LOCK()
LAST_INSERT_ID()	LOAD_FILE()
MASTER_POS_WAIT()	NOW()
RAND()	RELEASE_LOCK()
SLEEP()	SYSDATE()
UNIX_TIMESTAMP() (no parameters)	USER()
UUID()	UUID_SHORT()

A query will also not be added to the cache if:

- It is of the form:
  - SELECT SQL_NO_CACHE ...
  - SELECT ... INTO OUTFILE ...
  - SELECT ... INTO DUMPFILE ...
  - SELECT ... FOR UPDATE
  - SELECT * FROM ... WHERE autoincrement_column IS NULL
  - SELECT ... LOCK IN SHARE MODE
- It uses TEMPORARY table
- It uses no tables at all
- It generates a warning
- The user has a column-level privilege on any table in the query
- It accesses a table from INFORMATION_SCHEMA, mysql or the performance_schema database
- It makes use of user or local variables
- It makes use of stored functions
- It makes use of user-defined functions
- It is inside a transaction with the SERIALIZABLE isolation level
- It is quering a table inside a transaction after the same table executed a query cache invalidation using INSERT, UPDATE or DELETE

The query itself can also specify that it is not to be stored in the cache by using the `SQL_NO_CACHE` attribute. Query-level control is an effective way to use the cache more optimally.

It is also possible to specify that *no* queries must be stored in the cache unless the query requires it. To do this, the query_cache_type server variable must be set to `2`, or `DEMAND`. Then, only queries with the `SQL_CACHE` attribute are cached.

Limiting the Size of the Query Cache

There are two main ways to limit the size of the query cache. First, the overall size in bytes is determined by the query_cache_size server variable. About 40KB is needed for various query cache structures.

The query cache size is allocated in 1024 byte-blocks, thus it should be set to a multiple of 1024.

The query result is stored using a minimum block size of query_cache_min_res_unit. Check two conditions to use a good value of this variable: Query cache insert result blocks with locks, each new block insert lock query cache, a small value will increase locks and fragmentation and waste less memory for small results, a big value will increase memory use wasting more memory for small results but it reduce locks. Test with your workload for fine tune this variable.

If the strict mode is enabled, setting the query cache size to an invalid value will cause an error. Otherwise, it will be set to the nearest permitted value, and a warning will be triggered.

```
SHOW VARIABLES LIKE 'query_cache_size';
+------------------+----------+
| Variable_name    | Value    |
+------------------+----------+
| query_cache_size | 67108864 |
+------------------+----------+

SET GLOBAL query_cache_size = 8000000;
Query OK, 0 rows affected, 1 warning (0.03 sec)

SHOW VARIABLES LIKE 'query_cache_size';
+------------------+---------+
| Variable_name    | Value   |
+------------------+---------+
| query_cache_size | 7999488 |
+------------------+---------+
```

The ideal size of the query cache is very dependent on the specific needs of each system. Setting a value too small will result in query results being dropped from the cache when they could potentially be re-used later. Setting a value too high could result in reduced performance due to lock contention, as the query cache is locked during updates.

The second way to limit the cache is to have a maximum size for each set of query results. This prevents a single query with a huge result set taking up most of the available memory and knocking a large number of smaller queries out of the cache. This is determined by the query_cache_limit server variable.

If you attempt to set a query cache that is too small (the amount depends on the architecture), the resizing will fail and the query cache will be set to zero, for example :

```
SET GLOBAL query_cache_size=40000;
Query OK, 0 rows affected, 2 warnings (0.03 sec)

SHOW WARNINGS;
+---------+------+-----------------------------------------------------------------------+
| Level   | Code | Message                                                               |
+---------+------+-----------------------------------------------------------------------+
| Warning | 1292 | Truncated incorrect query_cache_size value: '40000'                   |
| Warning | 1282 | Query cache failed to set size 39936; new query cache size is 0 |
+---------+------+-----------------------------------------------------------------------+
```

Examining the Query Cache

A number of status variables provide information about the query cache.

```
SHOW STATUS LIKE 'Qcache%';
+-------------------------+----------+
| Variable_name           | Value    |
+-------------------------+----------+
| Qcache_free_blocks      | 1158     |
| Qcache_free_memory      | 3760784  |
| Qcache_hits             | 31943398 |
| Qcache_inserts          | 42998029 |
| Qcache_lowmem_prunes    | 34695322 |
| Qcache_not_cached       | 652482   |
| Qcache_queries_in_cache | 4628     |
| Qcache_total_blocks     | 11123    |
+-------------------------+----------+
```

`Qcache_inserts` contains the number of queries added to the query cache, `Qcache_hits` contains the number of queries that have made use of the query cache, while `Qcache_lowmem_prunes` contains the number of queries that were dropped from the cache due to lack of memory.

The above example could indicate a poorly performing cache. More queries have been added, and more queries have been dropped, than have actually been used.

Note that before MariaDB 5.5, queries returned from the query cache did not increment the Com_select status variable, so to find the total number of valid queries run on the server, add Com_select to Qcache_hits. Starting from MariaDB 5.5, results returned by the query cache count towards `Com_select` (see MDEV-4981).

The QUERY_CACHE_INFO plugin creates the QUERY_CACHE_INFO table in the INFORMATION_SCHEMA, allowing you to examine the contents of the query cache.

Query Cache Fragmentation

The Query Cache uses blocks of variable length, and over time may become fragmented. A high `Qcache_free_blocks` relative to `Qcache_total_blocks` may indicate fragmentation. FLUSH QUERY CACHE will defragment the query cache without dropping any queries :

```
FLUSH QUERY CACHE;
```

After this, there will only be one free block :

```
SHOW STATUS LIKE 'Qcache%';
+-------------------------+----------+
| Variable_name           | Value    |
+-------------------------+----------+
| Qcache_free_blocks      | 1        |
| Qcache_free_memory      | 6101576  |
| Qcache_hits             | 31981126 |
| Qcache_inserts          | 43002404 |
| Qcache_lowmem_prunes    | 34696486 |
| Qcache_not_cached       | 655607   |
| Qcache_queries_in_cache | 4197     |
| Qcache_total_blocks     | 8833     |
+-------------------------+----------+
```

Emptying and disabling the Query Cache

To empty or clear all results from the query cache, use RESET QUERY CACHE. FLUSH TABLES will have the same effect.

Setting either query_cache_type or query_cache_size to `0` will disable the query cache, but to free up the most resources, set both to `0` when you wish to disable caching.

Limitations

- The query cache needs to be disabled in order to use OQGRAPH.
- The query cache is not used by the Spider storage engine (amongst others).
- The query cache also needs to be disabled for MariaDB Galera cluster versions prior to "5.5.40-galera", "10.0.14-galera" and "10.1.2".

LOCK TABLES and the Query Cache

The query cache can be used when tables have a write lock (which may seem confusing since write locks should avoid table reads). This behaviour can be changed by setting the query_cache_wlock_invalidate system variable to `ON`, in which case each write lock will invalidate the table query cache. Setting to `OFF`, the default, means that cached queries can be returned even when a table lock is being held. For example:

```
1> SELECT * FROM T1
+---+
| a |
+---+
| 1 |
+---+
-- Here the query is cached

-- From another connection execute:
2> LOCK TABLES T1 WRITE;

-- Expected result with: query_cache_wlock_invalidate = OFF
1> SELECT * FROM T1
+---+
| a |
+---+
| 1 |
+---+
-- read from query cache


-- Expected result with: query_cache_wlock_invalidate = ON
1> SELECT * FROM T1
-- Waiting Table Write Lock
```

Transactions and the Query Cache

The query cache handles transactions. Internally a flag (FLAGS_IN_TRANS) is set to 0 when a query was executed outside a transaction, and to 1 when the query was inside a transaction (BEGIN / COMMIT / ROLLBACK). This flag is part of the "query cache hash", in others words one query inside a transaction is different from a query outside a transaction.

Queries that change rows (INSERT / UPDATE / DELETE / TRUNCATE) inside a transaction will invalidate all queries from the table, and turn off the query cache to the changed table. Transactions that don't end with COMMIT / ROLLBACK check that even without COMMIT / ROLLBACK, the query cache is turned off to allow row level locking and consistency level.

Examples:

```
SELECT * FROM T1 <first insert to query cache, using FLAGS_IN_TRANS=0>
+---+
| a |
+---+
| 1 |
+---+
```

```
BEGIN;
SELECT * FROM T1 <first insert to query cache, using FLAGS_IN_TRANS=1>
+---+
| a |
+---+
| 1 |
+---+
```

```
SELECT * FROM T1 <result from query cache, using FLAGS_IN_TRANS=1>
+---+
| a |
+---+
| 1 |
+---+
```

```
INSERT INTO T1 VALUES(2);  <invalidate queries from table T1 and disable query cache to table
T1>
```

```
SELECT * FROM T1 <don't use query cache, a normal query from innodb table>
+---+
| a |
+---+
| 1 |
| 2 |
+---+
```

```
SELECT * FROM T1 <don't use query cache, a normal query from innodb table>
+---+
| a |
+---+
| 1 |
| 2 |
+---+
```

```
COMMIT;  <query cache is now turned on to T1 table>
```

```
SELECT * FROM T1 <first insert to query cache, using FLAGS_IN_TRANS=0>
+---+
| a |
+---+
| 1 |
+---+
```

```
SELECT * FROM T1 <result from query cache, using FLAGS_IN_TRANS=0>
+---+
| a |
+---+
| 1 |
+---+
```

Query Cache Internal Structure

Internally, each flag that can change a result using the same query is a different query. For example, using the latin1 charset and using the utf8 charset with the same query are treated as different queries by the query cache.

Some fields that differentiate queries are (from "Query_cache_query_flags" internal structure) :

- query (string)
- current database schema name (string)
- client long flag (0/1)
- client protocol 4.1 (0/1)
- protocol type (internal value)
- more results exists (protocol flag)
- in trans (inside transaction or not)
- autocommit (autocommit session variable)
- pkt_nr (protocol flag)
- character set client (character_set_client session variable)
- character set results (character_set_results session variable)
- collation connection (collation_connection session variable)
- limit (sql_select_limit session variable)
- time zone (time_zone session variable)
- sql_mode (sql_mode session variable)
- max_sort_length (max_sort_length session variable)
- group_concat_max_len (group_concat_max_len session variable)
- default_week_format (default_week_format session variable)
- div_precision_increment (div_precision_increment session variable)
- lc_time_names (lc_time_names session variable)

More information can be found by viewing the source code (MariaDB 10.1) :

- https://github.com/MariaDB/server/blob/10.1/sql/sql_cache.cc
- https://github.com/MariaDB/server/blob/10.1/sql/sql_cache.h

Timeout and Mutex Contention

When searching for a query inside the query cache, a try_lock function waits with a timeout of 50ms. If the lock fails, the query isn't executed via the query cache. This timeout is hard coded (MDEV-6766 include two variables to tune this timeout).

From the sql_cache.cc, function "try_lock" using TIMEOUT :

```
        struct timespec waittime;
        set_timespec_nsec(waittime,(ulong)(50000000L));  /* Wait for 50 msec */
        int res= mysql_cond_timedwait(&COND_cache_status_changed,
                                      &structure_guard_mutex, &waittime);
        if (res == ETIMEDOUT)
          break;
```

When inserting a query inside the query cache or aborting a query cache insert (using the KILL command for example), a try_lock function waits until the query cache returns; no timeout is used in this case.

When two processes execute the same query, only the last process stores the query result. All other processes increase the Qcache_not_cached status variable.

SQL_NO_CACHE and SQL_CACHE

There are two aspects to the query cache: placing a query in the cache, and retrieving it from the cache.

1. Adding a query to the query cache. This is done automatically for cacheable queries (see (Queries Stored in the Query Cache) when the query_cache_type system variable is set to `1`, or `ON` and the query contains no SQL_NO_CACHE clause, or when the query_cache_type system variable is set to `2`, or `DEMAND`, and the query contains the SQL_CACHE clause.
2. Retrieving a query from the cache. This is done after the server receives the query and before the query parser. In this case one point should be considered:

When using SQL_NO_CACHE, it should be after the first SELECT hint, for example :

```
SELECT SQL_NO_CACHE .... FROM (SELECT SQL_CACHE ...) AS temp_table
```

instead of

```
SELECT SQL_CACHE .... FROM (SELECT SQL_NO_CACHE ...) AS temp_table
```

The second query will be checked. The query cache only checks if SQL_NO_CACHE/SQL_CACHE exists after the first SELECT. (More info at MDEV-6631)

5.3.13.8 Segmented Key Cache

3.3.4.2.5 Subquery Cache

3.3.9.15 Thread Command Values

A thread can have any of the following `COMMAND` values (displayed by the `COMMAND` field listed by the SHOW PROCESSLIST statement or in the Information Schema PROCESSLIST Table, as well as the `PROCESSLIST_COMMAND` value listed in the Performance Schema threads Table). These indicate the nature of the thread's activity.

Value	Description
Binlog Dump	Master thread for sending binary log contents to a slave.
Change user	Executing a change user operation.
Close stmt	Closing a prepared statement.
Connect	Replication slave is connected to its master.
Connect Out	Replication slave is in the process of connecting to its master.
Create DB	Executing an operation to create a database.

Daemon	Internal server thread rather than for servicing a client connection.
Debug	Generating debug information.
Delayed insert	A delayed-insert handler.
Drop DB	Executing an operation to drop a database.
Error	Error.
Execute	Executing a prepared statement.
Fetch	Fetching the results of an executed prepared statement.
Field List	Retrieving table column information.
Init DB	Selecting default database.
Kill	Killing another thread.
Long Data	Retrieving long data from the result of executing a prepared statement.
Ping	Handling a server ping request.
Prepare	Preparing a prepared statement.
Processlist	Preparing processlist information about server threads.
Query	Executing a statement.
Quit	In the process of terminating the thread.
Refresh	Flushing a table, logs or caches, or refreshing replication server or status variable information.
Register Slave	Registering a slave server.
Reset stmt	Resetting a prepared statement.
Set option	Setting or resetting a client statement execution option.
Sleep	Waiting for the client to send a new statement.
Shutdown	Shutting down the server.
Statistics	Preparing status information about the server.
Table Dump	Sending the contents of a table to a slave.
Time	Not used.

3.3.10 Optimizing Data Structure

Good database design is an important part of a well-run system. This section looks at some of the elements to consider.

Numeric vs String Fields

Choosing numeric over string fields

Optimizing MEMORY Tables

MEMORY tables are a good choice for data that needs to be accessed often, a...

Optimizing String and Character Fields

Optimizations when comparing string columns and VARCHAR vs BLOB

3.3.10.1 Numeric vs String Fields

A large numeric value is stored in far fewer bytes than the equivalent string value. It is therefore faster to move and compare numeric data, so it's best to choose numeric columns for unique id's and other similar fields.

3.3.10.2 Optimizing MEMORY Tables

MEMORY tables are a good choice for data that needs to be accessed often, and is rarely updated. Being in memory, it's not suitable for critical data or for storage, but if data can be moved to memory for reading without needing to be regenerated often, if at all, it can provide a significant performance boost.

The MEMORY Storage Engine has a key feature in that it permits its indexes to be either B-tree or Hash. Choosing the best index type can lead to better performance. See Storage Engine index types for more on the characteristics of each index type.

3.3.10.3 Optimizing String and Character Fields

Comparing String Columns

When values from different columns are compared, the comparison runs more quickly when the columns are of the same character set and collation. If they are different, the strings need to be converted while the query runs. So, where possible, declare string columns using the same character set and collation when you may need to compare them.

VARCHAR vs BLOB

ORDER BY and GROUP BY clauses can generate temporary tables in memory (see MEMORY Storage Engine) if the original table doesn't contain any BLOB fields. If a column is less than 8KB, you can make use of a Binary VARCHAR rather than a BLOB.

3.3.11 MariaDB Internal Optimizations

Different optimizations strategies done internally in MariaDB

Binary Log Group Commit and InnoDB Flushing Performance

Improvement for group commit for InnoDB transactions with the binary log enabled.

Fair Choice Between Range and Index_merge Optimizations

index_merge is a method used by the optimizer to retrieve rows from a singl...

Improvements to ORDER BY Optimization

Several Improvements to the ORDER BY Optimizer in Version 10.1 of MariaDB.

Multi Range Read Optimization

An optimization for improving performance of IO-bound queries which scan many rows.

5.3.2.20 Binary Log Group Commit and InnoDB Flushing Performance

3.3.11.2 Fair Choice Between Range and Index_merge Optimizations

`index_merge` is a method used by the optimizer to retrieve rows from a single table using several index scans. The results of the scans are then merged.

When using EXPLAIN, if `index_merge` is the plan chosen by the optimizer, it will show up in the "type" column. For example:

```
MariaDB [ontime]> SELECT COUNT(*) FROM ontime;
+--------+
|count(*)|
+--------+
| 1578171|
+--------+

MySQL [ontime]> EXPLAIN SELECT * FROM ontime WHERE (Origin='SEA' OR Dest='SEA');
+--+-----------+------+-----------+-------------+-----------+-------+----+-----+-------------
-------------------------+
|id|select_type|table |type       |possible_keys|key        |key_len|ref |rows |Extra
|
+--+-----------+------+-----------+-------------+-----------+-------+----+-----+-------------
-------------------------+
| 1|SIMPLE     |ontime|index_merge|Origin,Dest  |Origin,Dest|6,6    |NULL|92800|Using union
(Origin,Dest); Using where|
+--+-----------+------+-----------+-------------+-----------+-------+----+-----+-------------
-------------------------+
```

The "rows" column gives us a way to compare efficiency between `index_merge` and other plans.

It is sometimes necessary to discard index_merge in favor of a different plan to avoid a combinatorial explosion of possible range and/or index_merge strategies. But, the old logic in MySQL for when index_merge was rejected caused some good index_merge plans to not even be considered. Specifically, additional `AND` predicates in `WHERE` clauses could cause an index_merge plan to be rejected in favor of a less efficient plan. The slowdown could be anywhere from 10x to over 100x. Here are two examples (based on the previous query) using MySQL:

```
MySQL [ontime]> EXPLAIN SELECT * FROM ontime WHERE (Origin='SEA' OR Dest='SEA') AND
securitydelay=0;
+--+-----------+------+----+-----------------------+-------------+-------+-----+------+----
-------+
|id|select_type|table |type|possible_keys          |key          |key_len|ref  |rows  |Extra
|
+--+-----------+------+----+-----------------------+-------------+-------+-----+------+----
-------+
| 1|SIMPLE     |ontime|ref |Origin,Dest,SecurityDelay|SecurityDelay|5      |const|791546|Using
where|
+--+-----------+------+----+-----------------------+-------------+-------+-----+------+----
-------+

MySQL [ontime]> EXPLAIN SELECT * FROM ontime WHERE (Origin='SEA' OR Dest='SEA') AND depdelay <
12*60;
+--+-----------+------+----+-------------------+----+-------+----+-------+-----------+
|id|select_type|table |type|possible_keys      |key |key_len|ref |rows   |Extra      |
+--+-----------+------+----+-------------------+----+-------+----+-------+-----------+
| 1|SIMPLE     |ontime|ALL |Origin,DepDelay,Dest|NULL|NULL   |NULL|1583093|Using where|
+--+-----------+------+----+-------------------+----+-------+----+-------+-----------
```

In the above output, the "rows" column shows that the first is almost 10x less efficient and the second is over 15x less efficient than `index_merge`.

Starting in MariaDB 5.3, the optimizer will delay discarding potential `index_merge` plans until the point where it is really necessary. See MWL#24 for more information.

By not discarding potential `index_merge` plans until absolutely necessary, the two queries stay just as efficient as the original:

```
MariaDB [ontime]> EXPLAIN SELECT * FROM ontime WHERE (Origin='SEA' or Dest='SEA');
+--+-----------+------+-----------+-------------+-----------+-------+----+-----+-------------
-----------------------+
|id|select_type|table |type       |possible_keys|key        |key_len|ref |rows |Extra
|
+--+-----------+------+-----------+-------------+-----------+-------+----+-----+-------------
-----------------------+
| 1|SIMPLE     |ontime|index_merge|Origin,Dest  |Origin,Dest|6,6    |NULL|92800|Using
union(Origin,Dest); Using where|
+--+-----------+------+-----------+-------------+-----------+-------+----+-----+-------------
-----------------------+

MariaDB [ontime]> EXPLAIN SELECT * FROM ontime WHERE (Origin='SEA' or Dest='SEA') AND
securitydelay=0;
+--+-----------+------+-----------+----------------------+-----------+-------+----+-----+-
-----------------------------------+
|id|select_type|table |type       |possible_keys         |key        |key_len|ref |rows
|Extra                               |
+--+-----------+------+-----------+----------------------+-----------+-------+----+-----+-
-----------------------------------+
| 1|SIMPLE     |ontime|index_merge|Origin,Dest,SecurityDelay|Origin,Dest|6,6
|NULL|92800|Using union(Origin,Dest); Using where|
+--+-----------+------+-----------+----------------------+-----------+-------+----+-----+-
-----------------------------------+

MariaDB [ontime]> EXPLAIN SELECT * FROM ontime WHERE (Origin='SEA' or Dest='SEA') AND depdelay
< 12*60;
+--+-----------+------+-----------+-------------------+-----------+-------+----+-----+------
------------------------------+
|id|select_type|table |type       |possible_keys      |key        |key_len|ref |rows |Extra
|
+--+-----------+------+-----------+-------------------+-----------+-------+----+-----+------
------------------------------+
| 1|SIMPLE     |ontime|index_merge|Origin,DepDelay,Dest|Origin,Dest|6,6    |NULL|92800|Using
union(Origin,Dest); Using where|
+--+-----------+------+-----------+-------------------+-----------+-------+----+-----+------
------------------------------+
```

This new behavior is always on and there is no need to enable it. There are no known issues or gotchas with this new optimization.

3.3.4.3.5 Improvements to ORDER BY Optimization

3.3.11.4 Multi Range Read Optimization

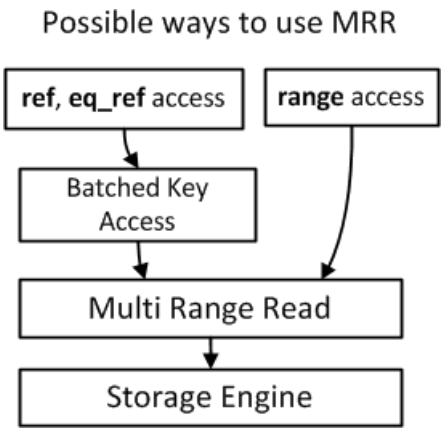
Contents

Multi Range Read is an optimization aimed at improving performance for IO-bound queries that need to scan lots of rows.

Multi Range Read can be used with

- `range` access
- `ref` and `eq_ref` access, when they are using Batched Key Access

as shown in this diagram:



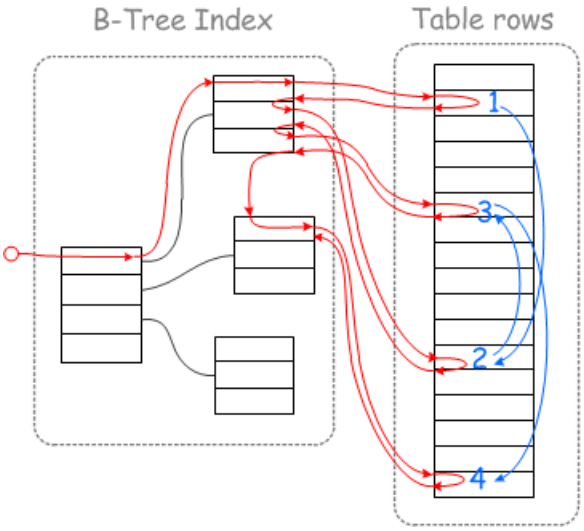
The Idea

Case 1: Rowid Sorting for Range Access

Consider a range query:

```
explain select * from tbl where tbl.key1 between 1000 and 2000;
+----+-------------+-------+-------+---------------+------+---------+------+------+----------
-------------+
| id | select_type | table | type  | possible_keys | key  | key_len | ref  | rows | Extra
|
+----+-------------+-------+-------+---------------+------+---------+------+------+----------
-------------+
|  1 | SIMPLE      | tbl   | range | key1          | key1 | 5       | NULL |  960 | Using
index condition |
+----+-------------+-------+-------+---------------+------+---------+------+------+----------
-------------+
```

When this query is executed, disk IO access pattern will follow the red line in this figure:



Execution will hit the table rows in random places, as marked with the blue line/numbers in the figure.

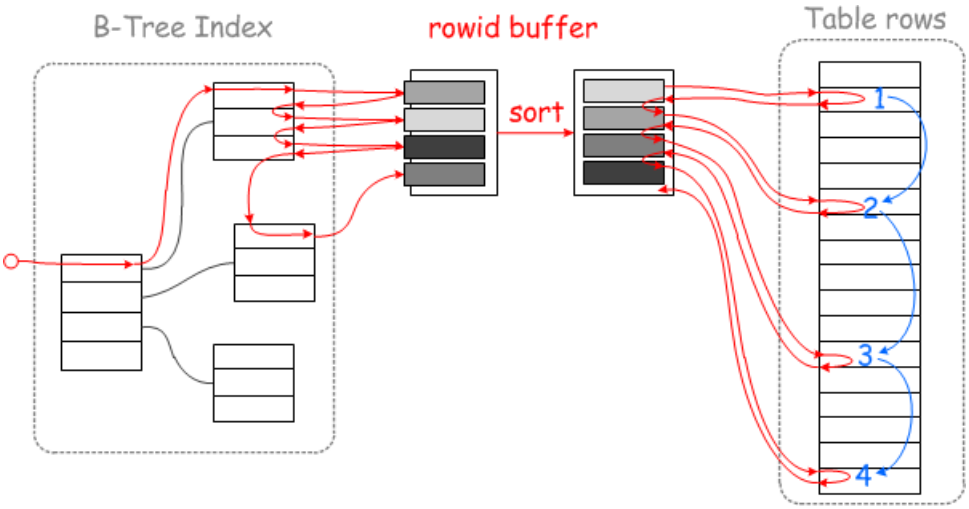
When the table is sufficiently big, each table record read will need to actually go to disk (and be served from buffer pool or OS cache), and query execution will be too slow to be practical. For example, a 10,000 RPM disk drive is able to make 167 seeks per second, so in the worst case, query execution will be capped at reading about 167 records per second.

SSD drives do not need to do disk seeks, so they will not be hurt as badly, however the performance will still be poor in many cases.

Multi-Range-Read optimization aims to make disk access faster by sorting record read requests and then doing one ordered disk sweep. If one enables Multi Range Read, `EXPLAIN` will show that a " `Rowid-ordered scan` " is used:

```
set optimizer_switch='mrr=on';
Query OK, 0 rows affected (0.06 sec)

explain select * from tbl where tbl.key1 between 1000 and 2000;
+----+-------------+-------+-------+---------------+------+---------+------+------+----------
----------------------------------+
| id | select_type | table | type  | possible_keys | key  | key_len | ref  | rows | Extra
|
+----+-------------+-------+-------+---------------+------+---------+------+------+----------
----------------------------------+
|  1 | SIMPLE      | tbl   | range | key1          | key1 | 5       | NULL |  960 | Using
index condition; Rowid-ordered scan |
+----+-------------+-------+-------+---------------+------+---------+------+------+----------
----------------------------------+
1 row in set (0.03 sec)
```

and the execution will proceed as follows:



Reading disk data sequentially is generally faster, because

- Rotating drives do not have to move the head back and forth
- One can take advantage of IO-prefetching done at various levels
- Each disk page will be read exactly once, which means we won't rely on disk cache (or buffer pool) to save us from reading the same page multiple times.

The above can make a huge difference on performance. There is also a catch, though:

- If you're scanning small data ranges in a table that is sufficiently small so that it completely fits into the OS disk cache, then you may observe that the only effect of MRR is that extra buffering/sorting adds some CPU overhead.
- `LIMIT n` and `ORDER BY ... LIMIT n` queries with small values of `n` may become slower. The reason is that MRR reads data *in disk order*, while `ORDER BY ... LIMIT n` wants first `n` records *in index order*.

Case 2: Rowid Sorting for Batched Key Access

Batched Key Access can benefit from rowid sorting in the same way as range access does. If one has a join that uses index lookups:

```
explain select * from t1,t2 where t2.key1=t1.col1;
+----+-------------+-------+------+---------------+------+---------+-------------+------+---
----------+
| id | select_type | table | type | possible_keys | key  | key_len | ref         | rows |
Extra       |
+----+-------------+-------+------+---------------+------+---------+-------------+------+---
----------+
|  1 | SIMPLE      | t1    | ALL  | NULL          | NULL | NULL    | NULL        | 1000 |
Using where |
|  1 | SIMPLE      | t2    | ref  | key1          | key1 | 5       | test.t1.col1 |    1 |
|
+----+-------------+-------+------+---------------+------+---------+-------------+------+---
----------+
2 rows in set (0.00 sec)
```

Execution of this query will cause table `t2` to be hit in random locations by lookups made through `t2.key1=t1.col`. If

you enable Multi Range and and Batched Key Access, you will get table `t2` to be accessed using a `Rowid-ordered scan`:

```
set optimizer_switch='mrr=on';
Query OK, 0 rows affected (0.06 sec)

set join_cache_level=6;
Query OK, 0 rows affected (0.00 sec)

explain select * from t1,t2 where t2.key1=t1.col1;
+----+-------------+-------+------+---------------+------+---------+-------------+------+---
-----------------------------------------------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref         | rows |
Extra                                                |
+----+-------------+-------+------+---------------+------+---------+-------------+------+---
-----------------------------------------------------+
|  1 | SIMPLE      | t1    | ALL  | NULL          | NULL | NULL    | NULL        | 1000 |
Using where                                          |
|  1 | SIMPLE      | t2    | ref  | key1          | key1 | 5       | test.t1.col1 |    1 |
Using join buffer (flat, BKA join); Rowid-ordered scan |
+----+-------------+-------+------+---------------+------+---------+-------------+------+---
-----------------------------------------------------+
2 rows in set (0.00 sec)
```

The benefits will be similar to those listed for `range` access.

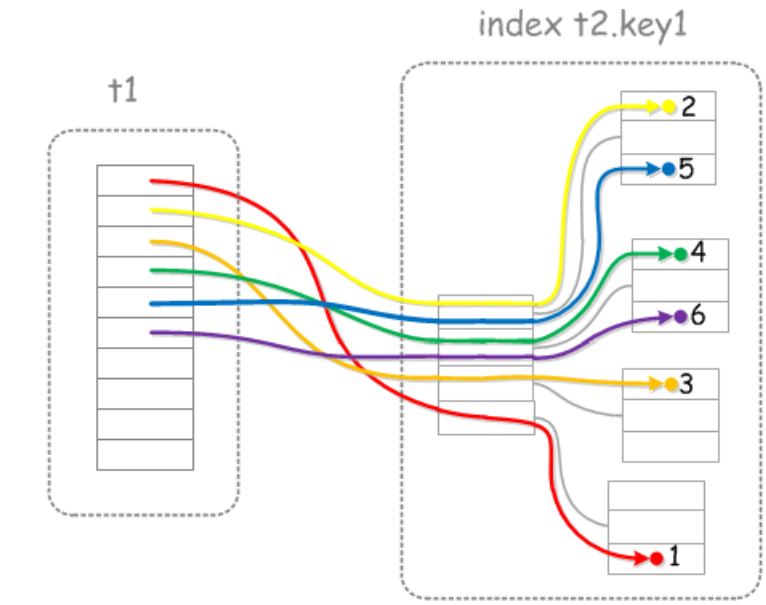
An additional source of speedup is this property: if there are multiple records in `t1` that have the same value of `t1.col1`, then regular Nested-Loops join will make multiple index lookups for the same value of `t2.key1=t1.col1`. The lookups may or may not hit the cache, depending on how big the join is. With Batched Key Access and Multi-Range Read, no duplicate index lookups will be made.

Case 3: Key Sorting for Batched Key Access

Let us consider again the nested loop join example, with `ref` access on the second table:

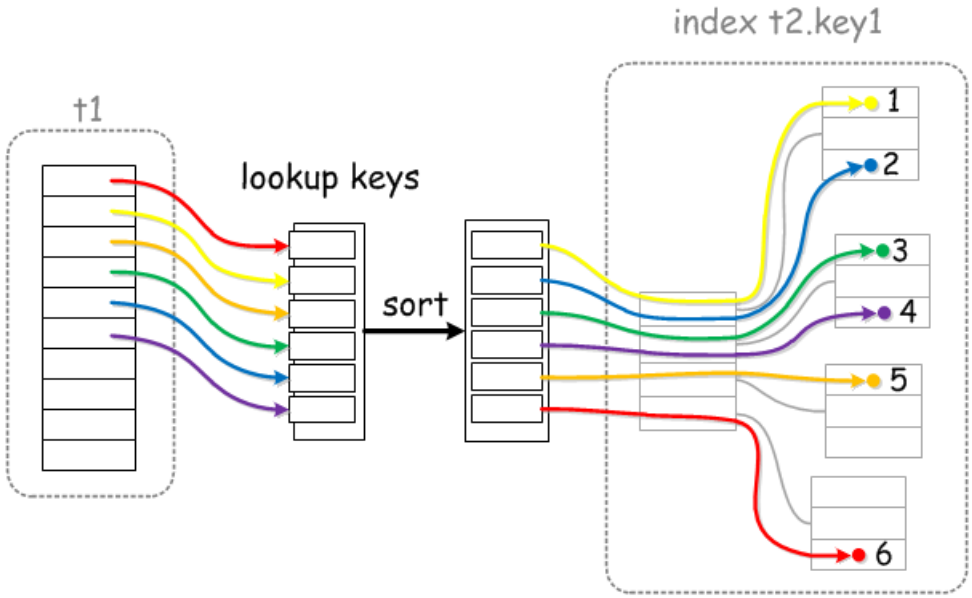
```
explain select * from t1,t2 where t2.key1=t1.col1;
+----+-------------+-------+------+---------------+------+---------+-------------+------+---
----------+
| id | select_type | table | type | possible_keys | key  | key_len | ref         | rows |
Extra       |
+----+-------------+-------+------+---------------+------+---------+-------------+------+---
----------+
|  1 | SIMPLE      | t1    | ALL  | NULL          | NULL | NULL    | NULL        | 1000 |
Using where |
|  1 | SIMPLE      | t2    | ref  | key1          | key1 | 5       | test.t1.col1 |    1 |
|
+----+-------------+-------+------+---------------+------+---------+-------------+------+---
----------+
```

Execution of this query plan will cause random hits to be made into the index `t2.key1`, as shown in this picture:



Regular Nested Loops Join will hit the index at random

In particular, on step #5 we'll read the same index page that we've read on step #2, and the page we've read on step #4 will be re-read on step#6. If all pages you're accessing are in the cache (in the buffer pool, if you're using InnoDB, and in the key cache, if you're using MyISAM), this is not a problem. However, if your hit ratio is poor and you're going to hit the disk, it makes sense to sort the lookup keys, like shown in this figure:



With *Batched Nested Loops Join* and *Key-Ordered retrieval*, index lookups are done in one "sweep"

This is roughly what `Key-ordered scan` optimization does. In EXPLAIN, it looks as follows:

```
set optimizer_switch='mrr=on,mrr_sort_keys=on';
Query OK, 0 rows affected (0.00 sec)

set join_cache_level=6;
Query OK, 0 rows affected (0.02 sec)
explain select * from t1,t2 where t2.key1=t1.col1\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t1
         type: ALL
possible_keys: a
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1000
        Extra: Using where
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: t2
         type: ref
possible_keys: key1
          key: key1
      key_len: 5
          ref: test.t1.col1
         rows: 1
        Extra: Using join buffer (flat, BKA join); Key-ordered Rowid-ordered scan
2 rows in set (0.00 sec)
```

((TODO: a note about why sweep-read over InnoDB's clustered primary index scan (which is, actually the whole InnoDB table itself) will use `Key-ordered scan` algorithm, but not `Rowid-ordered scan` algorithm, even though conceptually they are the same thing in this case))

Buffer Space Management

As was shown above, Multi Range Read requires sort buffers to operate. The size of the buffers is limited by system variables. If MRR has to process more data than it can fit into its buffer, it will break the scan into multiple passes. The more passes are made, the less is the speedup though, so one needs to balance between having too big buffers (which consume lots of memory) and too small buffers (which limit the possible speedup).

Range Access

When MRR is used for `range` access, the size of its buffer is controlled by the mrr_buffer_size system variable. Its value specifies how much space can be used for each table. For example, if there is a query which is a 10-way join and MRR is used for each table, `10*@@mrr_buffer_size` bytes may be used.

Batched Key Access

When Multi Range Read is used by Batched Key Access, then buffer space is managed by BKA code, which will automatically provide a part of its buffer space to MRR. You can control the amount of space used by BKA by setting

- join_buffer_size to limit how much memory BKA uses for each table, and
- join_buffer_space_limit to limit the total amount of memory used by BKA in the join.

Status Variables

There are three status variables related to Multi Range Read:

Variable name	Meaning
Handler_mrr_init	Counts how many Multi Range Read scans were performed
Handler_mrr_key_refills	Number of times key buffer was refilled (not counting the initial fill)
Handler_mrr_rowid_refills	Number of times rowid buffer was refilled (not counting the initial fill)

Non-zero values of `Handler_mrr_key_refills` and/or `Handler_mrr_rowid_refills` mean that Multi Range Read

scan did not have enough memory and had to do multiple key/rowid sort-and-sweep passes. The greatest speedup is achieved when Multi Range Read runs everything in one pass, if you see lots of refills it may be beneficial to increase sizes of relevant buffers mrr_buffer_size join_buffer_size and join_buffer_space_limit

Effect on Other Status Variables

When a Multi Range Read scan makes an index lookup (or some other "basic" operation), the counter of the "basic" operation, e.g. Handler_read_key, will also be incremented. This way, you can still see total number of index accesses, including those made by MRR. Per-user/table/index statistics counters also include the row reads made by Multi Range Read scans.

Why Using Multi Range Read Can Cause Higher Values in Status Variables

Multi Range Read is used for scans that do full record reads (i.e., they are not "Index only" scans). A regular non-index-only scan will read

1. an index record, to get a rowid of the table record
2. a table record Both actions will be done by making one call to the storage engine, so the effect of the call will be that the relevan `Handler_read_XXX` counter will be incremented BY ONE, and Innodb_rows_read will be incremented BY ONE.

Multi Range Read will make separate calls for steps #1 and #2, causing TWO increments to `Handler_read_XXX` counters and TWO increments to `Innodb_rows_read` counter. To the uninformed, this looks as if Multi Range Read was making things worse. Actually, it doesn't - the query will still read the same index/table records, and actually Multi Range Read may give speedups because it reads data in disk order.

Multi Range Read Factsheet

- Multi Range Read is used by
  - `range` access method for range scans.
  - Batched Key Access for joins
- Multi Range Read can cause slowdowns for small queries over small tables, so it is disabled by default.
- There are two strategies:
  - Rowid-ordered scan
  - Key-ordered scan
- : and you can tell if either of them is used by checking the `Extra` column in `EXPLAIN` output.
- There are three optimizer_switch flags you can switch ON:
  - `mrr=on` - enable MRR and rowid ordered scans
  - `mrr_sort_keys=on` - enable Key-ordered scans (you must also set `mrr=on` for this to have any effect)
  - `mrr_cost_based=on` - enable cost-based choice whether to use MRR. Currently not recommended, because cost model is not sufficiently tuned yet.

Differences from MySQL

- MySQL supports only `Rowid ordered scan` strategy, which it shows in `EXPLAIN` as `Using MRR`.
- EXPLAIN in MySQL shows `Using MRR`, while in MariaDB it may show
  - `Rowid-ordered scan`
  - `Key-ordered scan`
  - `Key-ordered Rowid-ordered scan`
- MariaDB uses mrr_buffer_size as a limit of MRR buffer size for `range` access, while MySQL uses read_rnd_buffer_size.
- MariaDB has three MRR counters: Handler_mrr_init, `Handler_mrr_extra_rowid_sorts`, `Handler_mrr_extra_key_sorts`, while MySQL has only `Handler_mrr_init`, and it will only count MRR scans that were used by BKA. MRR scans used by range access are not counted.

3.3.12 Compression

There are a number of different kinds of compression in MariaDB

Encryption, Hashing and Compression Functions

Functions used for encryption, hashing and compression.

Storage-Engine Independent Column Compression
Storage-engine independent support for column compression.

InnoDB Page Compression
InnoDB page compression, which is more sophisticated than the COMPRESSED row format.

Compression Plugins
Five MariaDB compression libraries are available as plugins.

Compressing Events to Reduce Size of the Binary Log
Binlog events can be compressed to save space on disk and in network transfers.

InnoDB COMPRESSED Row Format
Similar to the COMPACT row format, but can store even more data on overflow pages.

ColumnStore Compression Mode
ColumnStore has the ability to compress data

1.2.8.2 Encryption, Hashing and Compression Functions

3.3.12.2 Storage-Engine Independent Column Compression

Contents

Storage-engine independent column compression enables TINYBLOB, BLOB, MEDIUMBLOB, LONGBLOB, TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT, VARCHAR and VARBINARY columns to be compressed.

This is performed by means of a new COMPRESSED column attribute: `COMPRESSED[=<compression_method>]`

Currently the only supported compression method is `zlib`.

Field Length Compatibility

When using the `COMPRESSED` attribute, note that FIELD LENGTH is reduced by 1; for example, a BLOB has a length of 65535, while BLOB COMPRESSED has 65535-1. See MDEV-15592.

New System Variables

`column_compression_threshold`

- **Description:** Minimum column data length eligible for compression.
- **Commandline:** `--column-compression-threshold=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `100`
- **Range:** `0` to `4294967295`

column_compression_zlib_level

- **Description:** zlib compression level (1 gives best speed, 9 gives best compression).
- **Commandline:** `--column-compression-zlib-level=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `6`
- **Range:** `1` to `9`

---

column_compression_zlib_strategy

- **Description:** The strategy parameter is used to tune the compression algorithm. Use the value `DEFAULT_STRATEGY` for normal data, `FILTERED` for data produced by a filter (or predictor), `HUFFMAN_ONLY` to force Huffman encoding only (no string match), or `RLE` to limit match distances to one (run-length encoding). Filtered data consists mostly of small values with a somewhat random distribution. In this case, the compression algorithm is tuned to compress them better. The effect of `FILTERED` is to force more Huffman coding and less string matching; it is somewhat intermediate between `DEFAULT_STRATEGY` and `HUFFMAN_ONLY`. `RLE` is designed to be almost as fast as `HUFFMAN_ONLY`, but give better compression for PNG image data. The strategy parameter only affects the compression ratio but not the correctness of the compressed output even if it is not set appropriately. `FIXED` prevents the use of dynamic Huffman codes, allowing for a simpler decoder for special applications.
- **Commandline:** `--column-compression-zlib-strategy=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `enum`
- **Default Value:** `DEFAULT_STRATEGY`
- **Valid Values:** `DEFAULT_STRATEGY`, `FILTERED`, `HUFFMAN_ONLY`, `RLE`, `FIXED`

---

column_compression_zlib_wrap

- **Description:** If set to `1` (`0` is default), generate zlib header and trailer and compute adler32 check value. It can be used with storage engines that don't provide data integrity verification to detect data corruption.
- **Commandline:** `--column-compression-zlib-wrap{=0|1}`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

New Status Variables

Column_compressions

- **Description:** Incremented each time field data is compressed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Column_decompressions

- **Description:** Incremented each time field data is decompressed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Limitations

- The only supported method currently is zlib.
- The CSV storage engine stores data uncompressed on-disk even if the COMPRESSED attribute is present.
- It is not possible to create indexes over compressed columns.

Comparison with InnoDB Page Compression

Storage-independent column compression is different to InnoDB Page Compression in a number of ways.

- It is storage engine independent, while InnoDB page compression applies to InnoDB only.
- By being specific to a column, one can access non-compressed fields without the decompression overhead.
- Only zlib is available, while InnoDB page compression can offer alternative compression algorithms.
- It is not recommended to use multiple forms of compression over the same data.
- It is intended for compressing large blobs, while InnoDB page compression is suitable for a more general case.
- Columns cannot be indexed, while with InnoDB page compression indexes are possible as usual.

Examples

```
CREATE TABLE cmp (i TEXT COMPRESSED);

CREATE TABLE cmp2 (i TEXT COMPRESSED=zlib);
```

5.3.2.21 InnoDB Page Compression

5.4.11.11 Compression Plugins

3.1.13.15 Compressing Events to Reduce Size of the Binary Log

5.3.2.12.5 InnoDB COMPRESSED Row Format

3.3.12.7 ColumnStore Compression Mode

MariaDB ColumnStore has the ability to compress data and this is controlled through a compression mode. This compression mode may be set as a default for the instance or set at the session level.

To set the compression mode at the session level, the following command is used. Once the session has ended, any subsequent session will return to the default for the instance.

```
set infinidb_compression_type = n
```

where n is:

- 0) compression is turned off. Any subsequent table create statements run will have compression turned off for that table unless any statement overrides have been performed. Any alter statements run to add a column will have compression turned off for that column unless any statement override has been performed.
- 2) compression is turned on. Any subsequent table create statements run will have compression turned on for that table unless any statement overrides have been performed. Any alter statements run to add a column will have compression turned on for that column unless any statement override has been performed. ColumnStore uses snappy compression in this mode.

3.4 Connection Redirection Mechanism in the MariaDB Client/Server Protocol

MariaDB starting with 11.3

A connection redirection mechanism was added in MariaDB 11.3.0 (MDEV-15935)

Redirection mechanisms are widely used in proxy-based scenarios.

Previously, when multiple servers shared one proxy, the proxy forwarded all packets between servers and clients. Thus, the proxy added latency, consuming computing resources and impacting overall performance. For scenarios with many short connections, such as WordPress, latency can be a critical issue.

With a redirection mechanism, much like HTTP redirects or Oracle redirected connections, clients get the servers' address from proxies and connect to servers transparently, without latency and without wasting resources.

Usage

Redirection is handled through a new system variable, redirect_url. The value defaults to an empty string, but can also contain a connection string in the conventional format (in the style of a Connector/C etc. connection url).

This variable is appended to the default value of the session_track_system_variables system variable. If not empty, clients will be redirected to the specified server.

Possible Use Cases

- Always redirect all clients to a new location:
  - set @@global.redirect_url, start the server with --redurect-url=, or put it in my.cnf
- redirect to a group of servers randomly
  - create a table with connection urls, one per row.
  - use an sql script that selects a random row from it and sets @@redirect_url to this value
  - specify this script in the --init-connect server parameter
- dynamically redirect from the primary to one of the replicas
  - same as above, but use INFORMATION_SCHEMA.PROCESSLIST to get the list of active replicas.

Example

```
set global redirect_url="mysql://mariadb.org:12345";
```

Invalid formats are not permitted:

```
set global redirect_url="mysql://mariadb.org:";
ERROR 1231 (42000): Variable 'redirect_url' can't be set to the value of 'mysql://mariadb.org:'
```

4 Programming & Customizing MariaDB

Ways to add simple code to SQL statements, or create your own functions or stored procedures.

Programmatic & Compound Statements

Compound SQL statements for stored routines and in general.

Stored Routines

Stored procedures and functions.

Triggers & Events

Creating triggers and scheduled events within MariaDB.

Views

Stored queries for generating a virtual table.

User-Defined Functions

Extending MariaDB with custom functions.

There are 1 related questions.

1.1.1.6 Programmatic & Compound Statements

4.2 Stored Routines

Stored procedures and stored functions.

Stored Procedures

Routine invoked with a CALL statement.

Stored Functions

Defined functions for use with SQL statements.

Stored Routine Statements

SQL statements related to creating and using stored routines.

Binary Logging of Stored Routines

Stored routines require extra consideration when binary logging.

Stored Routine Limitations

SQL statements not permitted in stored programs.

Stored Routine Privileges

Privileges associated with stored functions and stored procedures.

4.2.1 Stored Procedures

A stored procedure is a routine invoked with a CALL statement. It may have input parameters, output parameters and parameters that are both input parameters and output parameters.

Stored Procedure Overview

A Stored Procedure is a routine invoked with a CALL statement.

Stored Routine Privileges

Privileges associated with stored functions and stored procedures.

CREATE PROCEDURE

Creates a stored procedure.

ALTER PROCEDURE

Change stored procedure characteristics.

DROP PROCEDURE

Drop stored procedure.

SHOW CREATE PROCEDURE

Returns the string used for creating a stored procedure.

SHOW PROCEDURE CODE

Display internal implementation of a stored procedure.

SHOW PROCEDURE STATUS

Stored procedure characteristics.

Binary Logging of Stored Routines

Stored routines require extra consideration when binary logging.

Information Schema ROUTINES Table

Stored procedures and stored functions information

SQL_MODE=ORACLE

MariaDB understands a subset of Oracle's PL/SQL language.

Stored Procedure Internals

Internal implementation of MariaDB stored procedures.

There are 3 related questions.

4.2.1.1 Stored Procedure Overview

Contents

A Stored Procedure is a routine invoked with a CALL statement. It may have input parameters, output parameters and parameters that are both input parameters and output parameters.

Creating a Stored Procedure

Here's a skeleton example to see a stored procedure in action:

```
DELIMITER //

CREATE PROCEDURE Reset_animal_count()
 MODIFIES SQL DATA
 UPDATE animal_count SET animals = 0;
//

DELIMITER ;
```

First, the delimiter is changed, since the function definition will contain the regular semicolon delimiter. The procedure is named `Reset_animal_count`. `MODIFIES SQL DATA` indicates that the procedure will perform a write action of sorts, and modify data. It's for advisory purposes only. Finally, there's the actual SQL statement - an UPDATE.

```
SELECT * FROM animal_count;
+---------+
| animals |
+---------+
|     101 |
+---------+

CALL Reset_animal_count();

SELECT * FROM animal_count;
+---------+
| animals |
+---------+
|       0 |
+---------+
```

A more complex example, with input parameters, from an actual procedure used by banks:

```
CREATE PROCEDURE
  Withdraw                              /* Routine name */
  (parameter_amount DECIMAL(6,2),     /* Parameter list */
  parameter_teller_id INTEGER,
  parameter_customer_id INTEGER)
  MODIFIES SQL DATA                     /* Data access clause */
  BEGIN                         /* Routine body */
    UPDATE Customers
        SET balance = balance - parameter_amount
        WHERE customer_id = parameter_customer_id;
    UPDATE Tellers
        SET cash_on_hand = cash_on_hand + parameter_amount
        WHERE teller_id = parameter_teller_id;
    INSERT INTO Transactions VALUES (
        parameter_customer_id,
        parameter_teller_id,
        parameter_amount);
  END;
```

See CREATE PROCEDURE for full syntax details.

Why use Stored Procedures?

Security is a key reason. Banks commonly use stored procedures so that applications and users don't have direct access to the tables. Stored procedures are also useful in an environment where multiple languages and clients are all used to perform the same operations.

Stored Procedure listings and definitions

To find which stored functions are running on the server, use SHOW PROCEDURE STATUS.

```
SHOW PROCEDURE STATUS\G
*************************** 1. row ***************************
                  Db: test
                Name: Reset_animal_count
                Type: PROCEDURE
             Definer: root@localhost
            Modified: 2013-06-03 08:55:03
             Created: 2013-06-03 08:55:03
       Security_type: DEFINER
             Comment:
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
```

or query the routines table in the INFORMATION_SCHEMA database directly:

```
SELECT ROUTINE_NAME FROM INFORMATION_SCHEMA.ROUTINES
  WHERE ROUTINE_TYPE='PROCEDURE';
+--------------------+
| ROUTINE_NAME       |
+--------------------+
| Reset_animal_count |
+--------------------+
```

To find out what the stored procedure does, use SHOW CREATE PROCEDURE.

```
SHOW CREATE PROCEDURE Reset_animal_count\G
*************************** 1. row ***************************
           Procedure: Reset_animal_count
            sql_mode:
    Create Procedure: CREATE DEFINER=`root`@`localhost` PROCEDURE `Reset_animal_count`()
    MODIFIES SQL DATA
UPDATE animal_count SET animals = 0
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
```

Dropping and Updating a Stored Procedure

To drop a stored procedure, use the DROP PROCEDURE statement.

```
DROP PROCEDURE Reset_animal_count();
```

To change the characteristics of a stored procedure, use ALTER PROCEDURE. However, you cannot change the parameters or body of a stored procedure using this statement; to make such changes, you must drop and re-create the procedure using CREATE OR REPLACE PROCEDURE (which retains existing privileges), or DROP PROCEDURE followed CREATE PROCEDURE .

Permissions in Stored Procedures

See the article Stored Routine Privileges.

4.2.1.2 Stored Routine Privileges

Contents

It's important to give careful thought to the privileges associated with stored functions and stored procedures. The following is an explanation of how they work.

Creating Stored Routines

- To create a stored routine, the `CREATE ROUTINE` privilege is needed. The `SUPER` privilege is required if a `DEFINER` is declared that's not the creator's account (see DEFINER clause below). The `SUPER` privilege is also required if statement-based binary logging is used. See Binary Logging of Stored Routines for more details.

Altering Stored Routines

- To make changes to, or drop, a stored routine, the `ALTER ROUTINE` privilege is needed. The creator of a routine is temporarily granted this privilege if they attempt to change or drop a routine they created, unless the automatic_sp_privileges variable is set to `0` (it defaults to 1).
- The `SUPER` privilege is also required if statement-based binary logging is used. See Binary Logging of Stored Routines for more details.

Running Stored Routines

- To run a stored routine, the `EXECUTE` privilege is needed. This is also temporarily granted to the creator if they attempt to run their routine unless the automatic_sp_privileges variable is set to `0` .
- The `SQL SECURITY clause` (by default `DEFINER`) specifies what privileges are used when a routine is called. If `SQL SECURITY` is `INVOKER` , the function body will be evaluated using the privileges of the user calling the function. If `SQL SECURITY` is `DEFINER` , the function body is always evaluated using the privileges of the definer account. `DEFINER` is the default. Thus, by default, users who can access the database associated with the stored routine can also run the routine, and potentially perform operations they wouldn't normally have permissions for.
- The creator of a routine is the account that ran the `CREATE FUNCTION` or `CREATE PROCEDURE` statement, regardless of whether a `DEFINER` is provided. The definer is by default the creator unless otherwise specified.
- The server automatically changes the privileges in the mysql.proc table as required, but will not look out for manual changes.

DEFINER Clause

If left out, the `DEFINER` is treated as the account that created the stored routine or view. If the account creating the routine has the `SUPER` privilege, another account can be specified as the `DEFINER` .

SQL SECURITY Clause

This clause specifies the context the stored routine or view will run as. It can take two values - `DEFINER` or `INVOKER` . `DEFINER` is the account specified as the `DEFINER` when the stored routine or view was created (see the section above). `INVOKER` is the account invoking the routine or view.

As an example, let's assume a routine, created by a superuser who's specified as the `DEFINER` , deletes all records from a table. If `SQL SECURITY=DEFINER` , anyone running the routine, regardless of whether they have delete privileges, will be able to delete the records. If `SQL SECURITY = INVOKER` , the routine will only delete the records if the account invoking the routine has permission to do so.

`INVOKER` is usually less risky, as a user cannot perform any operations they're normally unable to. However, it's not uncommon for accounts to have relatively limited permissions, but be specifically granted access to routines, which are then invoked in the `DEFINER` context.

Dropping Stored Routines

All privileges that are specific to a stored routine will be dropped when a DROP FUNCTION or DROP ROUTINE is run.

However, if a CREATE OR REPLACE FUNCTION or CREATE OR REPLACE PROCEDURE is used to drop and replace and the routine, any privileges specific to that routine will not be dropped.

1.1.1.3.1.9 CREATE PROCEDURE

1.1.1.2.1.1.6 ALTER PROCEDURE

1.1.1.3.3.9 DROP PROCEDURE

1.1.1.2.8.16 SHOW CREATE PROCEDURE

1.1.1.2.8.41 SHOW PROCEDURE CODE

1.1.1.2.8.42 SHOW PROCEDURE STATUS

3.1.13.5 Binary Logging of Stored Routines

1.1.1.2.9.1.1.40 Information Schema ROUTINES Table

2.1.14.3.1 SQL_MODE=ORACLE

4.2.1.12 Stored Procedure Internals

Contents

Implementation Specification for Stored Procedures

How Parsing and Execution of Queries Work

In order to execute a query, the function `sql_parse.cc:mysql_parse()` is called, which in turn calls the parser (`yyparse()`) with an updated Lex structure as the result. `mysql_parse()` then calls `mysql_execute_command()` which dispatches on the command code (in Lex) to the corresponding code for executing that particular query.

There are three structures involved in the execution of a query which are of interest to the stored procedure implementation:

- Lex (mentioned above) is the "compiled" query, that is the output from the parser and what is then interpreted to do the actual work. It constains an enum value (sql_command) which is the query type, and all the data collected by the parser needed for the execution (table names, fields, values, etc).
- THD is the "run-time" state of a connection, containing all that is needed for a particular client connection, and, among other things, the Lex structure currently being executed.
- `Item_*` : During parsing, all data is translated into "items", objects of the subclasses of "Item", such as `Item_int` , `Item_real` , `Item_string` , etc., for basic datatypes, and also various more specialized Item types for expressions

to be evaluated (`Item_func` objects).

How to Fit Stored Procedures into this Scheme

Overview of the Classes and Files for Stored Procedures

(More detailed APIs at the end of this page)

class sp_head (sp_head.{cc,h})

This contains, among other things, an array of "instructions" and the method for executing the procedure.

class sp_pcontext (sp_pcontext.{cc,h}

This is the parse context for the procedure. It's primarily used during parsing to keep track of local parameters, variables and labels, but it's also used at CALL time to find the parameters mode (IN, OUT or INOUT) and type when setting up the runtime context.

class sp_instr (sp_head.{cc,h})

This is the base class for "instructions", that is, what is generated by the parser. It turns out that we only need a minimum of 5 different sub classes:

- sp_instr_stmt Execute a statement. This is the "call-out" any normal SQL statement, like a SELECT, INSERT etc. It contains the Lex structure for the statement in question.
- sp_instr_set Set the value of a local variable (or parameter)
- sp_instr_jump An unconditional jump.
- sp_instr_jump_if_not Jump if condition is not true. It turns out that the negative test is most convenient when generating the code for the flow control constructs.
- sp_instr_freturn Return a value from a FUNCTION and exit. For condition HANDLERs some special instructions are also needed, see that section below.

class sp_rcontext (sp_rcontext.h)

This is the runtime context in the THD structure. It contains an array of items, the parameters and local variables for the currently executing stored procedure. This means that variable value lookup is in runtime is constant time, a simple index operation.

class Item_splocal (Item.{cc,h})

This is a subclass of Item. Its sole purpose is to hide the fact that the real Item is actually in the current frame (runtime context). It contains the frame offset and defers all methods to the real Item in the frame. This is what the parser generates for local variables.

Utility Functions (sp.{cc,h})

This contains functions for creating, dropping and finding a stored procedure in the mysql.proc table (or the internal cache).

Parsing CREATE PROCEDURE

When parsing a CREATE PROCEDURE the parser first initializes the `sphead` and `spcont` (runtime context) fields in the Lex. The sql_command code for the result of parsing a is `SQLCOM_CREATE_PROCEDURE` .

The parsing of the parameter list and body is relatively straightforward:

- Parameters: name, type and mode (IN/OUT/INOUT) is pushed to `spcont`
- Declared local variables: Same as parameters (mode is then IN)
- Local Variable references: If an identifier is found in in `spcont` , an `Item_splocal` is created with the variable's frame index, otherwise an `Item_field` or `Item_ref` is created (as before).
- Statements: The Lex in THD is replaced by a new Lex structure and the statement, is parsed as usual. A `sp_instr_stmt` is created, containing the new Lex, and added to the instructions in `sphead` . Afterwards, the procedure's Lex is restored in THD.
- SET var: Setting a local variable generates a `sp_instr_set` instruction, containing the variable's frame offset, the expression (an Item), and the type.
- Flow control: Flow control constructs such as IF, WHILE, etc, generate a conditional and unconditional jumps in the "obvious" way, but a few notes may be required:
- Forward jumps: When jumping forward, the exact destination is not known at the time of the creation of the jump instruction. The

1. sphead `therefore contains a list of instruction-label pairs for each forward reference. When the position later is known, the instructions in the list are updated with the correct location.`

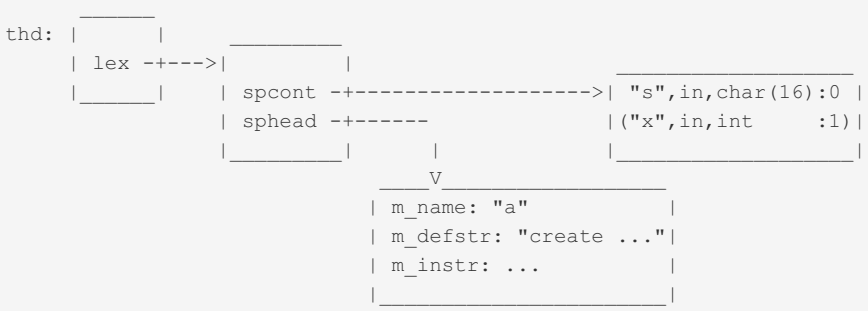
- Loop constructs have optional labels. If a loop doesn't have a label, an anonymous label is generated to simplify the parsing.
- There are two types of CASE. The "simple" case is implemented with an anonymous variable bound to the value to be tested.

A Simple Example

Parsing the procedure:

```
create procedure a(s char(16))
begin
  declare x int;
  set x = 3;
  while x > 0 do
    set x = x-1;
    insert into db.tab values (x, s);
  end while;
end
```

would generate the following structures:



Note that the contents of the `spcont` is changing during the parsing, at all times reflecting the state of the would-be runtime frame. The `m_instr` is an array of instructions:

```
Pos.  Instruction
 0    sp_instr_set(1, '3')
 1    sp_instr_jump_if_not(5, 'x>0')
 2    sp_instr_set(1, 'x-1')
 3    sp_instr_stmt('insert into ...')
 4    sp_instr_jump(1)
 5    <end>
```

Here, '3', 'x>0', etc, represent the Items or Lex for the respective expressions or statements.

Parsing CREATE FUNCTION

Creating a function is essentially the same thing as for a PROCEDURE, with the addition that a FUNCTION has a return type and a RETURN statement, but no OUT or INOUT parameters.

The main difference during parsing is that we store the result type in the sp_head. However, there are big differences when it comes to invoking a FUNCTION. (See below.)

Storing, Caching, Dropping

As seen above, the entired definition string, including the "CREATE PROCEDURE" (or "FUNCTION") is kept. The procedure definition string is stored in the table mysql.proc with the name and type as the key, the type being one of the enum ("procedure","function").

A PROCEDURE is just stored in the mysql.proc table. A FUNCTION has an additional requirement. They will be called in expressions with the same syntax as UDFs, so UDFs and stored FUNCTIONs share the namespace. Thus, we must make sure that we do not have UDFs and FUNCTIONs with the same name (even if they are stored in different places).

This means that we can reparse the procedure as many time as we want. The first time, the resulting Lex is used to store the procedure in the database (using the function sp.c:sp_create_procedure()).

The simplest way would be to just leave it at that, and re-read the procedure from the database each time it is called. (And in fact, that's the way the earliest implementation will work.) However, this is not very efficient, and we can do better. The full implementation should work like this:

1. Upon creation time, parse and store the procedure. Note that we still need to parse it to catch syntax errors, but we can't check if called procedures exists for instance.
2. Upon first CALL, read from the database, parse it, and cache the resulting Lex in memory. This time we can do more error checking.
3. Upon subsequent CALLs, use the cached Lex.

Note that this implies that the Lex structure with its sphead must be reentrant, that is, reusable and shareable between different threads and calls. The runtime state for a procedure is kept in the sp_rcontext in THD.

The mechanisms of storing, finding, and dropping procedures are encapsulated in the files sp.{cc,h}.

CALLing a Procedure

A CALL is parsed just like any statement. The resulting Lex has the sql_command SQLCOM_CALL, the procedure's name and the parameters are pushed to the Lex' value_list.

sql_parse.cc:mysql_execute_command() then uses sp.cc:sp_find() to get the sp_head for the procedure (which may have been read from the database or fetched from the in-memory cache) and calls the sp_head's method execute(). Note: It's important that substatements called by the procedure do not do send_ok(). Fortunately, there is a flag in THD->net to disable this during CALLs. If a substatement fails, it will however send an error back to the client, so the CALL mechanism must return immediately and without sending an error.

The sp_head::execute() method works as follows:

1. Keep a pointer to the old runtime context in THD (if any)
2. Create a new runtime context. The information about the required size is in sp_head's parse time context.
3. Push each parameter (from the CALL's Lex->value_list) to the new context. If it's an OUT or INOUT parameter, the parameter's offset in the caller's frame is set in the new context as well.
4. For each instruction, call its execute() method. The result is a pointer to the next instruction to execute (or NULL) if an error occurred.
5. On success, set the new values of the OUT and INOUT parameters in the caller's frame.

USE database

Before executing the instruction we also keeps the current default database (if any). If this was changed during execution (i.e. a USE statement has been executed), we restore the current database to the original.

This is the most useful way to handle USE in procedures. If we didn't, the caller would find himself in a different database after calling a function, which can be confusing. Restoring the database also gives full freedom to the procedure writer: - It's possible to write "general" procedures that are independent of the actual database name. - It's possible to write procedures that work on a particular database by calling USE, without having to use fully qualified table names everywhere (which doesn't help if you want to call other, "general", procedures anyway).

Evaluating Items

There are three occasions where we need to evaluate an expression:

- When SETing a variable
- When CALLing a procedure
- When testing an expression for a branch (in IF, WHILE, etc)

The semantics in stored procedures is "call-by-value", so we have to evaluate any "func" Items at the point of the CALL or SET, otherwise we would get a kind of "lazy" evaluation with unexpected results with respect to OUT parameters for instance. For this the support function, `sp_head.cc:eval_func_item()` is needed.

Calling a FUNCTION

Functions don't have an explicit call keyword like procedures. Instead, they appear in expressions with the conventional syntax "fun(arg, ...)". The problem is that we already have User Defined Functions (UDFs) which are called the same way. A UDF is detected by the lexical analyzer (not the parser!), in the `find_keyword()` function, and returns a `UDF_*_FUNC` or `UDA_*_SUM` token with the `udf_func` object as the yylval.

So, stored functions must be handled in a similar way, and as a consequence, UDFs and functions must not have the same name.

Detecting and Parsing a FUNCTION Invocation

The existence of UDFs are checked during the lexical analysis (in sql_lex.cc:find_keyword()). This has the drawback that they must exist before they are referred to, which was ok before SPs existed, but then it becomes a problem. The first implementation of SP FUNCTIONs will work the same way, but this should be fixed a.s.a.p. (This will required some reworking of the way UDFs are handled, which is why it's not done from the start.) For the time being, a FUNCTION is detected the same way, and returns the token SP_FUNC. During the parsing we only check for the *existence* of the function, we don't parse it, since wa can't call the parser recursively.

When encountering a SP_FUNC with parameters in the expression parser, an instance of the new Item_func_sp class is created. Unlike UDFs, we don't have different classes for different return types, since we at this point don't know the type.

Collecting FUNCTIONs to invoke

A FUNCTION differs from a PROCEDURE in one important aspect: Whereas a PROCEDURE is CALLed as statement by itself, a FUNCTION is invoked "on-the-fly" during the execution of *another* statement. This makes things a lot more complicated compared to CALL: - We can't read and parse the FUNCTION from the mysql.proc table at the point of invocation; the server requires that all tables used are opened and locked at the beginning of the query execution. One "obvious" solution would be to simply push "mysql.proc" to the list of tables used by the query, but this implies a "join" with this table if the query is a select, so it doesn't work (and we can't exclude this table easily; since a privileged used might in fact want to search the proc table). Another solution would of course be to allow the opening and closing of the mysql.proc table during a query execution, but this it not possible at the present.

So, the solution is to collect the names of the referred FUNCTIONs during parsing in the lex. Then, before doing anything else in `mysql_execute_command()`, read all functions from the database an keep them in the THD, where the function `sp_find_function()` can find them during the execution. Note: Even with an in-memory cache, we must still make sure that the functions are indeed read and cached at this point. The code that read and cache functions from the database must also be invoked recursively for each read FUNCTION to make sure we have *all* the functions we need.

Parsing DROP PROCEDURE/FUNCTION

The procedure name is pushed to Lex->value_list. The sql_command code for the result of parsing a is `SQLCOM_DROP_PROCEDURE` / `SQLCOM_DROP_FUNCTION`.

Dropping is done by simply getting the procedure with the sp_find() function and calling `sp_drop()` (both in `sp.{cc,h}`).

DROP PROCEDURE/DROP FUNCTION also supports the non-standard "IF EXISTS", analogous to other DROP statements in MariaDB.

Condition and Handlers

Condition names are lexical entities and are kept in the parser context just like variables. But, condition are just "aliases" for SQLSTATE strings, or mysqld error codes (which is a non-standard extension in MySQL), and are only used during parsing.

Handlers comes in three types, CONTINUE, EXIT and UNDO. The latter is like an EXIT handler with an implicit rollback, and is currently not implemented. The EXIT handler jumps to the end of its BEGIN-END block when finished. The CONTINUE handler returns to the statement following that which invoked the handler.

The handlers in effect at any point is part of each thread's runtime state, so we need to push and pop handlers in the sp_rcontext during execution. We use special instructions for this: - sp_instr_hpush_jump Push a handler. The instruction contains the necessary information, like which conditions we handle and the location of the handler. The jump takes us to the location after the handler code. - sp_instr_hpop Pop the handlers of the current frame (which we are just leaving).

It might seems strange to jump past the handlers like that, but there's no extra cost in doing this, and for technical reasons it's easiest for the parser to generate the handler instructions when they occur in the source.

When an error occurs, one of the error routines is called and an error message is normally sent back to the client immediately. Catching a condition must be done in these error routines (there are quite a few) to prevent them from doing this. We do this by calling a method in the THD's sp_rcontext (if there is one). If a handler is found, this is recorded in the context and the routine returns without sending the error message. The execution loop (sp_head::execute()) checks for this after each statement and invokes the handler that has been found. If several errors or warnings occurs during one statement, only the first is caught, the rest are ignored.

Invoking and returning from a handler is trivial in the EXIT case. We simply jump to it, and it will have an sp_instr_jump as its last instruction.

Calling and returning from a CONTINUE handler poses some special problems. Since we need to return to the point after its invocation, we push the return location on a stack in the sp_rcontext (this is done by the execution loop). The handler then ends with a special instruction, sp_instr_hreturn, which returns to this location.

CONTINUE handlers have one additional problem: They are parsed at the lexical level where they occur, so variable offsets will assume that it's actually called at that level. However, a handler might be invoked from a sub-block where additional local variables have been declared, which will then share the location of any local variables in the handler itself. So, when calling a CONTINUE handler, we need to save any local variables above the handler's frame offset, and restore them upon

return. (This is not a problem for EXIT handlers, since they will leave the block anyway.) This is taken care of by the execution loop and the sp_instr_hreturn instruction.

Examples

EXIT handler:

```
begin
  declare x int default 0;

  begin
    declare exit handler for 'XXXXX' set x = 1;

    (statement1);
    (statement2);
  end;
  (statement3);
end
```

```
Pos.  Instruction
 0    sp_instr_set(0, '0')
 1    sp_instr_hpush_jump(4, 1)           # location and frame size
 2    sp_instr_set(0, '1')
 3    sp_instr_jump(6)
 4    sp_instr_stmt('statement1')
 5    sp_instr_stmt('statement2')
 6    sp_instr_hpop(1)
 7    sp_instr_stmt('statement3')
```

CONTINUE handler:

```
create procedure hndlr1(val int)
begin
  declare x int default 0;
  declare foo condition for 1146;
  declare continue handler for foo set x = 1;

  insert into t3 values ("hndlr1", val);     # Non-existing table?
  if x>0 then
    insert into t1 values ("hndlr1", val);   # This instead then
  end if;
end|
```

```
Pos.  Instruction
 0    sp_instr_set(1, '0')
 1    sp_instr_hpush_jump(4, 2)
 2    sp_instr_set(1, '1')
 3    sp_instr_hreturn(2)                 # frame size
 4    sp_instr_stmt('insert ... t3 ...')
 5    sp_instr_jump_if_not(7, 'x>0')
 6    sp_instr_stmt('insert ... t1 ...')
 7    sp_instr_hpop(2)
```

Cursors

For stored procedures to be really useful, you want to have cursors. MySQL doesn't yet have "real" cursor support (with API and ODBC support, allowing updating, arbitrary scrolling, etc), but a simple asensitive, non-scrolling, read-only cursor can be implemented in SPs using the class Protocol_cursor. This class intecepts the creation and sending of results sets and instead stores it in-memory, as MYSQL_FIELDS and MYSQL_ROWS (as in the client API).

To support this, we need the usual name binding support in sp_pcontext (similar to variables and conditions) to keep track on declared cursor names, and a corresponding run-time mechanism in sp_rcontext. Cursors are lexically scoped like everything with a body or BEGIN/END block, so they are pushed and poped as usual (see conditions and variables above). The basic operations on a cursor are OPEN, FETCH and CLOSE, which will each have a corresponding instruction. In addition, we need instructions to push a new cursor (this will encapsulate the LEX of the SELECT statement of the cursor), and a pop instruction: - sp_instr_cpush Push a cursor to the sp_rcontext. This instruction contains the LEX for the select statement - sp_instr_cpop Pop a number of cursors from the sp_rcontext. - sp_instr_copen Open a cursor: This will execute the select and get the result set in a sepeate memroot. - sp_instr_cfetch Fetch the next row from the in-memory result set. The instruction contains a list of the variables (frame offsets) to set. - sp_instr_cclose Free the result set.

A cursor is a separate class, sp_cursor (defined in sp_rcontex.h) which encapsulates the basic operations used by the above instructions. This class contains the LEX, Protocol_cursor object, and its memroot, as well as the cursor's current state. Compiling and executing is fairly straight-forward. sp_instr_copen is a subclass of sp_instr_stmt and uses its mechanism to execute a substatement.

Example

```
begin
  declare x int;
  declare c cursor for select a from t1;

  open c;
  fetch c into x;
  close c;
end
```

```
Pos.  Instruction
 0    sp_instr_cpush('select a from ...')
 1    sp_instr_copen(0)                  # The 0'th cursor
 2    sp_instr_cfetch(0)                 # Contains the variable list
 3    sp_instr_cclose(0)
 4    sp_instr_cpop(1)
```

The SP cache

There are two ways to cache SPs:

1. one global cache, share by all threads/connections,
2. one cache per thread.

There are pros and cons with both methods:

1.
   - Pros: Save memory, each SP only read from table once,
   - Cons: Needs locking (= serialization at access), requires thread-safe data structures,
2.
   - Pros: Fast, no locking required (almost), limited thread-safe requirement,
   - Cons: Uses more memory, each SP read from table once per thread.

Unfortunately, we cannot use alternative 1 for the time being, as most of the data structures to be cached (lex and items) are not reentrant and thread-safe. (Things are modified at execution, we have THD pointers stored everywhere, etc.) This leaves us with alternative 2, one cache per thread; or actually two, since we keep FUNCTIONs and PROCEDUREs in separate caches. This is not that terrible; the only case when it will perform significantly worse than a global cache is when we have an application where new threads are connecting, calling a procedure, and disconnecting, over and over again.

The cache implementation itself is simple and straightforward, a hashtable wrapped in a class and a C API (see APIs below).

There is however one issue with multiple caches: dropping and altering procedures. Normally, this should be a very rare event in a running system; it's typically something you do during development and testing, so it's not unthinkable that we would simply ignore the issue and let any threads running with a cached version of an SP keep doing so until its disconnected. But assuming we want to keep the caches consistent with respect to drop and alter, it can be done:

1. A global counter is needed, initialized to 0 at start.
2. At each DROP or ALTER, increase the counter by one.
3. Each cache has its own copy of the counter, copied at the last read.
4. When looking up a name in the cache, first check if the global counter is larger than the local copy. If so, clear the cache and return "not found", and update the local counter; otherwise, lookup as usual.

This minimizes the cost to a single brief lock for the access of an integer when operating normally. Only in the event of an actual drop or alter, is the cache cleared. This may seem to be drastic, but since we assume that this is a rare event, it's not a problem. It would of course be possible to have a much more fine-grained solution, keeping track of each SP, but the overhead of doing so is not worth the effort.

Class and Function APIs

This is an outline of the key types. Some types and other details in the actual files have been omitted for readability.

The parser context: sp_pcontext.h

```
typedef enum
{
  sp_param_in,
```

```
  sp_param_out,
  sp_param_inout
} sp_param_mode_t;

typedef struct
{
  LEX_STRING name;
  enum enum_field_types type;
  sp_param_mode_t mode;
  uint offset;                    // Offset in current frame
  my_bool isset;
} sp_pvar_t;

typedef struct sp_cond_type
{
  enum { number, state, warning, notfound, exception } type;
  char sqlstate[6];
  uint mysqlerr;
} sp_cond_type_t;

class sp_pcontext
{
  sp_pcontext();

  // Return the maximum frame size
  uint max_framesize();

  // Return the current frame size
  uint current_framesize();

  // Return the number of parameters
  uint params();

  // Set the number of parameters to the current frame size
  void set_params();

  // Set type of the variable at offset 'i' in the frame
  void set_type(uint i, enum enum_field_types type);

  // Mark the i:th variable to "set" (i.e. having a value) with
  // 'val' true.
  void set_isset(uint i, my_bool val);

  // Push the variable 'name' to the frame.
  void push_var(LEX_STRING *name,
                enum enum_field_types type, sp_param_mode_t mode);

  // Pop 'num' variables from the frame.
  void pop_var(uint num = 1);

  // Find variable by name
  sp_pvar_t *find_pvar(LEX_STRING *name);

  // Find variable by index
  sp_pvar_t *find_pvar(uint i);

  // Push label 'name' of instruction index 'ip' to the label context
  sp_label_t *push_label(char *name, uint ip);

  // Find label 'name' in the context
  sp_label_t *find_label(char *name);

  // Return the last pushed label
  sp_label_t *last_label();

  // Return and remove the last pushed label.
  sp_label_t *pop_label();

  // Push a condition to the context
  void push_cond(LEX_STRING *name, sp_cond_type_t *val);

  // Pop a 'num' condition from the context
  void pop_cond(uint num);

  // Find a condition in the context
```

```
      // Find a condition in the context
      sp_cond_type_t *find_cond(LEX_STRING *name);

      // Increase the handler count
      void add_handler();

      // Returns the handler count
      uint handlers();

// Push a cursor
      void push_cursor(LEX_STRING *name);

// Find a cursor
my_bool find_cursor(LEX_STRING *name, uint *poff);

// Pop 'num' cursors
void pop_cursor(uint num);

// Return the number of cursors
uint cursors();
    }
```

Run-time context (call frame): sp_rcontext.h:

```
  #define SP_HANDLER_NONE      0
  #define SP_HANDLER_EXIT      1
  #define SP_HANDLER_CONTINUE  2
  #define SP_HANDLER_UNDO      3

  typedef struct
  {
    struct sp_cond_type *cond;
    uint handler;             // Location of handler
    int type;
    uint foffset;             // Frame offset for the handlers declare level
  } sp_handler_t;

  class sp_rcontext
  {
    // 'fsize' is the max size of the context, 'hmax' the number of handlers,
    // 'cmax' the number of cursors
    sp_rcontext(uint fsize, uint hmax, , uint cmax);

    // Push value (parameter) 'i' to the frame
    void push_item(Item *i);

    // Set slot 'idx' to value 'i'
    void set_item(uint idx, Item *i);

    // Return the item in slot 'idx'
    Item *get_item(uint idx);

    // Set the "out" index 'oidx' for slot 'idx. If it's an IN slot,
    // use 'oidx' -1.
    void set_oindex(uint idx, int oidx);

    // Return the "out" index for slot 'idx'
    int get_oindex(uint idx);

    // Set the FUNCTION result
    void set_result(Item *i);

    // Get the FUNCTION result
    Item *get_result();

    // Push handler at location 'h' for condition 'cond'. 'f' is the
    // current variable frame size.
    void push_handler(sp_cond_type_t *cond, uint h, int type, uint f);

    // Pop 'count' handlers
    void pop_handlers(uint count);

    // Find a handler for this error. This sets the state for a found
    // handler in the context. If called repeatedly without clearing,
```

```
    // only the first call's state is kept.
    int find_handler(uint sql_errno);

    // Returns 1 if a handler has been found, with '*ip' and '*fp' set
    // to the handler location and frame size respectively.
    int found_handler(uint *ip, uint *fp);

    // Clear the found handler state.
    void clear_handler();

    // Push a return address for a CONTINUE handler
    void push_hstack(uint ip);

    // Pop the CONTINUE handler return stack
    uint pop_hstack();

    // Save variables from frame index 'fp' and up.
    void save_variables(uint fp);

    // Restore saved variables from to frame index 'fp' and up.
    void restore_variables(uint fp);

    // Push a cursor for the statement (lex)
    void push_cursor(LEX *lex);

    // Pop 'count' cursors
    void pop_cursors(uint count);

    // Pop all cursors
    void pop_all_cursors();

    // Get the 'i'th cursor
    sp_cursor *get_cursor(uint i);

  }
```

The procedure: sp_head.h:

```
#define TYPE_ENUM_FUNCTION  1
#define TYPE_ENUM_PROCEDURE 2

class sp_head
{
  int m_type;             // TYPE_ENUM_FUNCTION or TYPE_ENUM_PROCEDURE

  sp_head();

  void init(LEX_STRING *name, LEX *lex, LEX_STRING *comment, char suid);

  // Store this procedure in the database. This is a wrapper around
  // the function sp_create_procedure().
  int create(THD *);

  // Invoke a FUNCTION
  int
  execute_function(THD *thd, Item **args, uint argcount, Item **resp);

  // CALL a PROCEDURE
  int
  execute_procedure(THD *thd, List<Item> *args);

  // Add the instruction to this procedure.
  void add_instr(sp_instr *);

  // Returns the number of instructions.
  uint instructions();

  // Returns the last instruction
  sp_instr *last_instruction();

  // Resets lex in 'thd' and keeps a copy of the old one.
  void reset_lex(THD *);

  // Restores lex in 'thd' from our copy, but keeps some status from the
  // one in 'thd', like ptr, tables, fields, etc.
  void restore_lex(THD *);

  // Put the instruction on the backpatch list, associated with
  // the label.
  void push_backpatch(sp_instr *, struct sp_label *);

  // Update all instruction with this label in the backpatch list to
  // the current position.
  void backpatch(struct sp_label *);

  // Returns the SP name (with optional length in '*lenp').
  char *name(uint *lenp = 0);

  // Returns the result type for a function
  Item_result result();

  // Sets various attributes
  void sp_set_info(char *creator, uint creatorlen,
                   longlong created, longlong modified,
                   bool suid, char *comment, uint commentlen);
}
```

Instructions

The base class

```
        class sp_instr
        {
          // 'ip' is the index of this instruction
          sp_instr(uint ip);

          // Execute this instrution.
          // '*nextp' will be set to the index of the next instruction
          // to execute. (For most instruction this will be the
          // instruction following this one.)
          // Returns 0 on success, non-zero if some error occurred.
          virtual int execute(THD *, uint *nextp)
        }
<<code>>


===== Statement instruction
<<code>>
        class sp_instr_stmt : public sp_instr
        {
          sp_instr_stmt(uint ip);

          int execute(THD *, uint *nextp);

          // Set the statement's Lex
          void set_lex(LEX *);

          // Return the statement's Lex
          LEX *get_lex();
        }
```

SET instruction

```
        class sp_instr_set : public sp_instr
        {
          // 'offset' is the variable's frame offset, 'val' the value,
          // and 'type' the variable type.
          sp_instr_set(uint ip,
                       uint offset, Item *val, enum enum_field_types type);

          int execute(THD *, uint *nextp);
        }
```

Unconditional jump

```
        class sp_instr_jump : public sp_instr
        {
          // No destination, must be set.
          sp_instr_jump(uint ip);

          // 'dest' is the destination instruction index.
          sp_instr_jump(uint ip, uint dest);

          int execute(THD *, uint *nextp);

          // Set the destination instruction 'dest'.
          void set_destination(uint dest);
        }
```

Conditional jump

```
        class sp_instr_jump_if_not : public sp_instr_jump
        {
          // Jump if 'i' evaluates to false. Destination not set yet.
          sp_instr_jump_if_not(uint ip, Item *i);

          // Jump to 'dest' if 'i' evaluates to false.
          sp_instr_jump_if_not(uint ip, Item *i, uint dest)

          int execute(THD *, uint *nextp);
        }
```

Return a function value

```
        class sp_instr_freturn : public sp_instr
        {
          // Return the value 'val'
          sp_instr_freturn(uint ip, Item *val, enum enum_field_types type);

          int execute(THD *thd, uint *nextp);
        }
```

Push a handler and jump

```
        class sp_instr_hpush_jump : public sp_instr_jump
        {
          // Push handler of type 'htype', with current frame size 'fp'
          sp_instr_hpush_jump(uint ip, int htype, uint fp);

          int execute(THD *thd, uint *nextp);

          // Add condition for this handler
          void add_condition(struct sp_cond_type *cond);
        }
```

Pops handlers

```
        class sp_instr_hpop : public sp_instr
        {
          // Pop 'count' handlers
          sp_instr_hpop(uint ip, uint count);

          int execute(THD *thd, uint *nextp);
        }
```

Return from a CONTINUE handler

```
        class sp_instr_hreturn : public sp_instr
        {
          // Return from handler, and restore variables to 'fp'.
          sp_instr_hreturn(uint ip, uint fp);

          int execute(THD *thd, uint *nextp);
        }
```

Push a CURSOR

```
        class sp_instr_cpush : public sp_instr_stmt
{
          // Push a cursor for statement 'lex'
  sp_instr_cpush(uint ip, LEX *lex)

  int execute(THD *thd, uint *nextp);
        }
```

Pop CURSORs

```
       class sp_instr_cpop : public sp_instr_stmt
{
         // Pop 'count' cursors
  sp_instr_cpop(uint ip, uint count)

  int execute(THD *thd, uint *nextp);
       }
```

Open a CURSOR

```
       class sp_instr_copen : public sp_instr_stmt
{
         // Open the 'c'th cursor
  sp_instr_copen(uint ip, uint c);

  int execute(THD *thd, uint *nextp);
       }
```

Close a CURSOR

```
       class sp_instr_cclose : public sp_instr
{
         // Close the 'c'th cursor
  sp_instr_cclose(uint ip, uint c);

  int execute(THD *thd, uint *nextp);
       }
```

Fetch a row with CURSOR

```
       class sp_instr_cfetch : public sp_instr
{
         // Fetch next with the 'c'th cursor
  sp_instr_cfetch(uint ip, uint c);

  int execute(THD *thd, uint *nextp);

  // Add a target variable for the fetch
  void add_to_varlist(struct sp_pvar *var);
       }
```

Utility functions: sp.h

```
#define SP_OK                 0
#define SP_KEY_NOT_FOUND     -1
#define SP_OPEN_TABLE_FAILED -2
#define SP_WRITE_ROW_FAILED  -3
#define SP_DELETE_ROW_FAILED -4
#define SP_GET_FIELD_FAILED  -5
#define SP_PARSE_ERROR       -6

// Finds a stored procedure given its name. Returns NULL if not found.
sp_head *sp_find_procedure(THD *, LEX_STRING *name);

// Store the procedure 'name' in the database. 'def' is the complete
// definition string ("create procedure ...").
int sp_create_procedure(THD *,
                        char *name, uint namelen,
                        char *def, uint deflen,
                        char *comment, uint commentlen, bool suid);

// Drop the procedure 'name' from the database.
int sp_drop_procedure(THD *, char *name, uint namelen);

// Finds a stored function given its name. Returns NULL if not found.
sp_head *sp_find_function(THD *, LEX_STRING *name);

// Store the function 'name' in the database. 'def' is the complete
// definition string ("create function ...").
int sp_create_function(THD *,
                       char *name, uint namelen,
                       char *def, uint deflen,
                       char *comment, uint commentlen, bool suid);

// Drop the function 'name' from the database.
int sp_drop_function(THD *, char *name, uint namelen);
```

The cache: sp_cache.h

```
/* Initialize the SP caching once at startup */
void sp_cache_init();

/* Clear the cache *cp and set *cp to NULL */
void sp_cache_clear(sp_cache **cp);

/* Insert an SP to cache. If **cp points to NULL, it's set to a
   new cache */
void sp_cache_insert(sp_cache **cp, sp_head *sp);

/* Lookup an SP in cache */
sp_head *sp_cache_lookup(sp_cache **cp, char *name, uint namelen);

/* Remove an SP from cache */
void sp_cache_remove(sp_cache **cp, sp_head *sp);
```

The mysql.proc schema

This is the mysql.proc table used in MariaDB 10.4:

```
CREATE TABLE `proc` (
  `db` char(64) CHARACTER SET utf8 COLLATE utf8_bin NOT NULL DEFAULT '',
  `name` char(64) NOT NULL DEFAULT '',
  `type` enum('FUNCTION','PROCEDURE','PACKAGE','PACKAGE BODY') NOT NULL,
  `specific_name` char(64) NOT NULL DEFAULT '',
  `language` enum('SQL') NOT NULL DEFAULT 'SQL',
  `sql_data_access` enum('CONTAINS_SQL','NO_SQL','READS_SQL_DATA','MODIFIES_SQL_DATA') NOT NULL
DEFAULT 'CONTAINS_SQL',
  `is_deterministic` enum('YES','NO') NOT NULL DEFAULT 'NO',
  `security_type` enum('INVOKER','DEFINER') NOT NULL DEFAULT 'DEFINER',
  `param_list` blob NOT NULL,
  `returns` longblob NOT NULL,
  `body` longblob NOT NULL,
  `definer` char(141) CHARACTER SET utf8 COLLATE utf8_bin NOT NULL DEFAULT '',
  `created` timestamp NOT NULL DEFAULT current_timestamp() ON UPDATE current_timestamp(),
  `modified` timestamp NOT NULL DEFAULT '0000-00-00 00:00:00',
  `sql_mode`
set('REAL_AS_FLOAT','PIPES_AS_CONCAT','ANSI_QUOTES','IGNORE_SPACE','IGNORE_BAD_TABLE_OPTIONS','
ONLY_FULL_GROUP_BY','NO_UNSIGNED_SUBTRACTION','NO_DIR_IN_CREATE','POSTGRESQL','ORACLE','MSSQL',
'DB2','MAXDB','NO_KEY_OPTIONS','NO_TABLE_OPTIONS','NO_FIELD_OPTIONS','MYSQL323','MYSQL40','ANSI
','NO_AUTO_VALUE_ON_ZERO','NO_BACKSLASH_ESCAPES','STRICT_TRANS_TABLES','STRICT_ALL_TABLES','NO_
ZERO_IN_DATE','NO_ZERO_DATE','INVALID_DATES','ERROR_FOR_DIVISION_BY_ZERO','TRADITIONAL','NO_AUT
O_CREATE_USER','HIGH_NOT_PRECEDENCE','NO_ENGINE_SUBSTITUTION','PAD_CHAR_TO_FULL_LENGTH','EMPTY_
STRING_IS_NULL','SIMULTANEOUS_ASSIGNMENT') NOT NULL DEFAULT '',
  `comment` text CHARACTER SET utf8 COLLATE utf8_bin NOT NULL,
  `character_set_client` char(32) CHARACTER SET utf8 COLLATE utf8_bin DEFAULT NULL,
  `collation_connection` char(32) CHARACTER SET utf8 COLLATE utf8_bin DEFAULT NULL,
  `db_collation` char(32) CHARACTER SET utf8 COLLATE utf8_bin DEFAULT NULL,
  `body_utf8` longblob DEFAULT NULL,
  `aggregate` enum('NONE','GROUP') NOT NULL DEFAULT 'NONE',
  PRIMARY KEY (`db`,`name`,`type`)
) ENGINE=Aria DEFAULT CHARSET=utf8 PAGE_CHECKSUM=1 TRANSACTIONAL=1 COMMENT='Stored Procedures'
```

4.2.2 Stored Functions

A stored function is a defined function that is called from within an SQL statement like a regular function, and returns a single value.

Stored Function Overview

Function called from within an SQL statement, returning a single value

Stored Routine Privileges

Privileges associated with stored functions and stored procedures.

CREATE FUNCTION

Creates a stored function.

ALTER FUNCTION

Change the characteristics of a stored function.

DROP FUNCTION

Drop a stored function.

SHOW CREATE FUNCTION

Statement that created the function.

SHOW FUNCTION STATUS

Stored function characteristics

SHOW FUNCTION CODE

Representation of the internal implementation of the stored function

Stored Aggregate Functions

Custom aggregate functions.

Binary Logging of Stored Routines

Stored routines require extra consideration when binary logging.

Stored Function Limitations

Restrictions applying to stored functions

Information Schema ROUTINES Table

Stored procedures and stored functions information

4.2.2.1 Stored Function Overview

Contents

A Stored Function is a defined function that is called from within an SQL statement like a regular function, and returns a single value.

Creating Stored Functions

Here's a skeleton example to see a stored function in action:

```
DELIMITER //

CREATE FUNCTION FortyTwo() RETURNS TINYINT DETERMINISTIC
BEGIN
 DECLARE x TINYINT;
 SET x = 42;
 RETURN x;
END

//

DELIMITER ;
```

First, the delimiter is changed, since the function definition will contain the regular semicolon delimiter. See Delimiters in the mariadb client for more. Then the function is named `FortyTwo` and defined to return a `tinyin` . The `DETERMINISTIC` keyword is not necessary in all cases (although if binary logging is on, leaving it out will throw an error), and is to help the query optimizer choose a query plan. A deterministic function is one that, given the same arguments, will always return the same result.

Next, the function body is placed between BEGIN and END statements. It declares a tinyint, `x` , which is simply set to 42, and this is the result returned.

```
SELECT FortyTwo();
+------------+
| FortyTwo() |
+------------+
|         42 |
+------------+
```

Of course, a function that doesn't take any arguments is of little use. Here's a more complex example:

```
DELIMITER //
CREATE FUNCTION VatCents(price DECIMAL(10,2)) RETURNS INT DETERMINISTIC
BEGIN
 DECLARE x INT;
 SET x = price * 114;
 RETURN x;
END //
Query OK, 0 rows affected (0.04 sec)
DELIMITER ;
```

This function takes an argument, `price` which is defined as a DECIMAL, and returns an INT.

Take a look at the CREATE FUNCTION page for more details.

From MariaDB 10.3.3, it is also possible to create stored aggregate functions.

Stored Function listings and definitions

To find which stored functions are running on the server, use SHOW FUNCTION STATUS.

```
SHOW FUNCTION STATUS\G
*************************** 1. row ***************************
                  Db: test
                Name: VatCents
                Type: FUNCTION
             Definer: root@localhost
            Modified: 2013-06-01 12:40:31
             Created: 2013-06-01 12:40:31
       Security_type: DEFINER
             Comment:
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
1 row in set (0.00 sec)
```

or query the routines table in the INFORMATION_SCHEMA database directly:

```
SELECT ROUTINE_NAME FROM INFORMATION_SCHEMA.ROUTINES WHERE
  ROUTINE_TYPE='FUNCTION';
+--------------+
| ROUTINE_NAME |
+--------------+
| VatCents     |
+--------------+
```

To find out what the stored function does, use SHOW CREATE FUNCTION.

```
SHOW CREATE FUNCTION VatCents\G
*************************** 1. row ***************************
            Function: VatCents
            sql_mode:
     Create Function: CREATE DEFINER=`root`@`localhost` FUNCTION `VatCents`(price
DECIMAL(10,2)) RETURNS int(11)
    DETERMINISTIC
BEGIN
 DECLARE x INT;
 SET x = price * 114;
 RETURN x;
END
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
```

Dropping and Updating Stored Functions

To drop a stored function, use the DROP FUNCTION statement.

```
DROP FUNCTION FortyTwo;
```

To change the characteristics of a stored function, use ALTER FUNCTION. Note that you cannot change the parameters or body of a stored function using this statement; to make such changes, you must drop and re-create the function using DROP FUNCTION and CREATE FUNCTION.

Permissions in Stored Functions

See the article Stored Routine Privileges.

4.2.1.2 Stored Routine Privileges

1.1.1.3.1.3 CREATE FUNCTION

1.1.1.2.1.1.4 ALTER FUNCTION

1.1.1.3.3.3 DROP FUNCTION

1.1.1.2.8.13 SHOW CREATE FUNCTION

1.1.1.2.8.29 SHOW FUNCTION STATUS

4.2.2.8 SHOW FUNCTION CODE

Syntax

```
SHOW FUNCTION CODE func_name
```

Description

`SHOW FUNCTION CODE` shows a representation of the internal implementation of the stored function.

It is similar to `SHOW PROCEDURE CODE` but for stored functions.

1.2.4.1 Stored Aggregate Functions

3.1.13.5 Binary Logging of Stored Routines

4.2.2.11 Stored Function Limitations

The following restrictions apply to stored functions.

- All of the restrictions listed in Stored Routine Limitations.
- Any statements that return a result set are not permitted. For example, a regular SELECTs is not permitted, but a SELECT INTO is. A cursor and FETCH statement is permitted.
- FLUSH statements are not permitted.
- Statements that perform explicit or implicit commits or rollbacks are not permitted
- Cannot be used recursively.
- Cannot make changes to a table that is already in use (reading or writing) by the statement invoking the stored function.
- Cannot refer to a temporary table multiple times under different aliases, even in different statements.
- ROLLBACK TO SAVEPOINT and RELEASE SAVEPOINT statement which are in a stored function cannot refer to a savepoint which has been defined out of the current function.
- Prepared statements (PREPARE, EXECUTE, DEALLOCATE PREPARE) cannot be used, and therefore nor can statements be constructed as strings and then executed.

1.1.1.2.9.1.1.40 Information Schema ROUTINES Table

1.1.1.7 Stored Routine Statements

3.1.13.5 Binary Logging of Stored Routines

4.2.5 Stored Routine Limitations

The following SQL statements are not permitted inside any stored routines (stored functions, stored procedures, events or triggers).

- ALTER VIEW; you can use CREATE OR REPLACE VIEW instead.
- LOAD DATA and LOAD TABLE.
- CHANGE MASTER TO
- INSERT DELAYED is permitted, but the statement is handled as a regular INSERT.
- LOCK TABLES and UNLOCK TABLES.
- References to local variables within prepared statements inside a stored routine (use user-defined variables instead).
- BEGIN (WORK) is treated as the beginning of a BEGIN END block, not a transaction, so START TRANSACTION needs to be used instead.
- The number of permitted recursive calls is limited to max_sp_recursion_depth. If this variable is 0 (default), recursivity is disabled. The limit does not apply to stored functions.
- Most statements that are not permitted in prepared statements are not permitted in stored programs. See Prepare Statement:Permitted statements for a list of statements that can be used. SIGNAL, RESIGNAL and GET DIAGNOSTICS are exceptions, and may be used in stored routines.

There are also further limitations specific to the kind of stored routine.

Note that, if a stored program calls another stored program, the latter will inherit the caller's limitations. So, for example, if a stored procedure is called by a stored function, that stored procedure will not be able to produce a result set, because stored functions can't do this.

4.2.1.2 Stored Routine Privileges

4.3 Triggers & Events

Triggers
Set of statements that run when an event occurs on a table.

Event Scheduler
Named database objects containing SQL statements to be executed at a later stage.

4.3.1 Triggers

A trigger is a set of statements that run when an event occurs on a table.

Trigger Overview
Statements run when an event occurs on a table.

Binary Logging of Stored Routines
Stored routines require extra consideration when binary logging.

CREATE TRIGGER
Create a new trigger.

DROP TRIGGER
Drops a trigger.

Information Schema TRIGGERS Table
Information about triggers

Running Triggers on the Replica for Row-based Events
Running triggers on the replica for row-based events.

SHOW CREATE TRIGGER
Shows the CREATE TRIGGER statement used to create the trigger

SHOW TRIGGERS
Shows currently-defined triggers

Trigger Limitations
Restrictions applying to triggers.

Triggers and Implicit Locks

Implicit locks due to triggers and LOCK TABLE

There are 4 related questions.

4.3.1.1 Trigger Overview

Contents

A trigger, as its name suggests, is a set of statements that run, or are triggered, when an event occurs on a table.

Events

The event can be an INSERT, an UPDATE or a DELETE. The trigger can be executed BEFORE or AFTER the event. Until MariaDB 10.2.3, a table could have only one trigger defined for each event/timing combination: for example, a table could only have one BEFORE INSERT trigger.

The LOAD DATA INFILE and LOAD XML statements invoke INSERT triggers for each row that is being inserted.

The REPLACE statement is executed with the following workflow:

- BEFORE INSERT;
- BEFORE DELETE (only if a row is being deleted);
- AFTER DELETE (only if a row is being deleted);
- AFTER INSERT.

The INSERT ... ON DUPLICATE KEY UPDATE statement, when a row already exists, follows the following workflow:

- BEFORE INSERT;
- BEFORE UPDATE;
- AFTER UPDATE.

Otherwise, it works like a normal INSERT statement.

Note that TRUNCATE TABLE does not activate any triggers.

Triggers and Errors

With non-transactional storage engines, if a BEFORE statement produces an error, the statement will not be executed. Statements that affect multiple rows will fail before inserting the current row.

With transactional engines, triggers are executed in the same transaction as the statement that invoked them.

If a warning is issued with the SIGNAL or RESIGNAL statement (that is, an error with an SQLSTATE starting with '01'), it will be treated like an error.

Creating a Trigger

Here's a simple example to demonstrate a trigger in action. Using these two tables as an example:

```
CREATE TABLE animals (id mediumint(9)
NOT NULL AUTO_INCREMENT,
name char(30) NOT NULL,
PRIMARY KEY (`id`));

CREATE TABLE animal_count (animals int);

INSERT INTO animal_count (animals) VALUES(0);
```

We want to increment a counter each time a new animal is added. Here's what the trigger will look like:

```
CREATE TRIGGER increment_animal
AFTER INSERT ON animals
FOR EACH ROW
UPDATE animal_count SET animal_count.animals = animal_count.animals+1;
```

The trigger has:

- a *name* (in this case `increment_animal`)
- a trigger time (in this case *after* the specified trigger event)
- a trigger event (an `INSERT`)
- a table with which it is associated (`animals`)
- a set of statements to run (here, just the one UPDATE statement)

`AFTER INSERT` specifies that the trigger will run *after* an `INSERT`. The trigger could also be set to run *before*, and the statement causing the trigger could be a `DELETE` or an `UPDATE` as well.

The set of statements to run are the statements on the table of the trigger, therefore columns/values that change are always just a column name or an expression like `NEW.column_name`. Table references of other tables must come from explicit table references.

Now, if we insert a record into the `animals` table, the trigger will run, incrementing the animal_count table;

```
SELECT * FROM animal_count;
+---------+
| animals |
+---------+
|       0 |
+---------+

INSERT INTO animals (name) VALUES('aardvark');
INSERT INTO animals (name) VALUES('baboon');

SELECT * FROM animal_count;
+---------+
| animals |
+---------+
|       2 |
+---------+
```

For more details on the syntax, see CREATE TRIGGER.

Dropping Triggers

To drop a trigger, use the DROP TRIGGER statement. Triggers are also dropped if the table with which they are associated is also dropped.

```
DROP TRIGGER increment_animal;
```

Triggers Metadata

The Information Schema TRIGGERS Table stores information about triggers.

The SHOW TRIGGERS statement returns similar information.

The SHOW CREATE TRIGGER statement returns a CREATE TRIGGER statement that creates the given trigger.

More Complex Triggers

Triggers can consist of multiple statements enclosed by a BEGIN and END. If you're entering multiple statements on the command line, you'll want to temporarily set a new delimiter so that you can use a semicolon to delimit the statements inside your trigger. See Delimiters in the mariadb client for more.

```
DROP TABLE animals;

UPDATE animal_count SET animals=0;

CREATE TABLE animals (id mediumint(9) NOT NULL AUTO_INCREMENT,
name char(30) NOT NULL,
PRIMARY KEY (`id`))
ENGINE=InnoDB;

DELIMITER //
CREATE TRIGGER the_mooses_are_loose
AFTER INSERT ON animals
FOR EACH ROW
BEGIN
 IF NEW.name = 'Moose' THEN
  UPDATE animal_count SET animal_count.animals = animal_count.animals+100;
 ELSE
  UPDATE animal_count SET animal_count.animals = animal_count.animals+1;
 END IF;
END; //

DELIMITER ;

INSERT INTO animals (name) VALUES('Aardvark');

SELECT * FROM animal_count;
+---------+
| animals |
+---------+
|       1 |
+---------+

INSERT INTO animals (name) VALUES('Moose');

SELECT * FROM animal_count;
+---------+
| animals |
+---------+
|     101 |
+---------+
```

Trigger Errors

If a trigger contains an error and the engine is transactional, or it is a BEFORE trigger, the trigger will not run, and will prevent the original statement from running as well. If the engine is non-transactional, and it is an AFTER trigger, the trigger will not run, but the original statement will.

Here, we'll drop the above examples, and then recreate the trigger with an error, a field that doesn't exist, first using the default InnoDB, a transactional engine, and then again using MyISAM, a non-transactional engine.

```
DROP TABLE animals;

CREATE TABLE animals (id mediumint(9) NOT NULL AUTO_INCREMENT,
name char(30) NOT NULL,
PRIMARY KEY (`id`))
ENGINE=InnoDB;

CREATE TRIGGER increment_animal
AFTER INSERT ON animals
FOR EACH ROW
UPDATE animal_count SET animal_count.id = animal_count_id+1;

INSERT INTO animals (name) VALUES('aardvark');
ERROR 1054 (42S22): Unknown column 'animal_count.id' in 'field list'

SELECT * FROM animals;
Empty set (0.00 sec)
```

And now the identical procedure, but with a MyISAM table.

```
DROP TABLE animals;

CREATE TABLE animals (id mediumint(9) NOT NULL AUTO_INCREMENT,
name char(30) NOT NULL,
PRIMARY KEY (`id`))
ENGINE=MyISAM;

CREATE TRIGGER increment_animal
AFTER INSERT ON animals
FOR EACH ROW
UPDATE animal_count SET animal_count.id = animal_count_id+1;

INSERT INTO animals (name) VALUES('aardvark');
ERROR 1054 (42S22): Unknown column 'animal_count.id' in 'field list'

SELECT * FROM animals;
+----+----------+
| id | name     |
+----+----------+
|  1 | aardvark |
+----+----------+
```

The following example shows how to use a trigger to validate data. The SIGNAL statement is used to intentionally produce an error if the email field is not a valid email. As the example shows, in that case the new row is not inserted (because it is a BEFORE trigger).

```
CREATE TABLE user (
 id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
 first_name CHAR(20),
 last_name CHAR(20),
 email CHAR(100)
)
 ENGINE = MyISAM;

DELIMITER //
CREATE TRIGGER bi_user
  BEFORE INSERT ON user
  FOR EACH ROW
BEGIN
  IF NEW.email NOT LIKE '_%@_%.__%' THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Email field is not valid';
  END IF;
END; //
DELIMITER ;

INSERT INTO user (first_name, last_name, email) VALUES ('John', 'Doe', 'john_doe.example.net');
ERROR 1644 (45000): Email field is not valid

SELECT * FROM user;
Empty set (0.00 sec)
```

3.1.13.5 Binary Logging of Stored Routines

1.1.1.3.1.15 CREATE TRIGGER

1.1.1.3.3.15 DROP TRIGGER

1.1.1.2.9.1.1.58 Information Schema TRIGGERS Table

3.1.23 Running Triggers on the Replica for Row-based Events

1.1.1.2.8.19 SHOW CREATE TRIGGER

1.1.1.2.8.54 SHOW TRIGGERS

4.3.1.9 Trigger Limitations

Contents

The following restrictions apply to triggers.

- All of the restrictions listed in Stored Routine Limitations.
- All of the restrictions listed in Stored Function Limitations.
- Until MariaDB 10.2.3, each table can have only one trigger for each timing/event combination (ie: you can't define two BEFORE INSERT triggers for the same table).
- Triggers are always executed for each row. The standard `FOR EACH STATEMENT` option is not supported in MariaDB,
- Triggers cannot operate on any tables in the mysql, information_schema or performance_schema database.
- Cannot return a resultset.
- The RETURN statement is not permitted, since triggers don't return any values. Use LEAVE to immediately exit a trigger.
- Triggers are not activated by foreign key actions.
- If a trigger is loaded into cache, it is not automatically reloaded when the table metadata changes. In this case a trigger can operate using the outdated metadata.
- By default, with row-based replication, triggers run on the master, and the effects of their executions are replicated to the slaves. However, starting from MariaDB 10.1.1, it is possible to run triggers on the slaves. See Running triggers on the slave for Row-based events.

4.3.1.10 Triggers and Implicit Locks

A trigger may reference multiple tables, and if a `LOCK TABLES` statement is used on one of the tables, other tables may at the same time also implicitly be locked due to the trigger.

If the trigger only reads from the other table, that table will be read locked. If the trigger writes to the other table, it will be write locked. If a table is read-locked for reading via `LOCK TABLES`, but needs to be write-locked because it could be modified by a trigger, a write lock is taken.

All locks are acquired together when the `LOCK TABLES` statement is issued and they are released together on `UNLOCK TABLES`.

Example

```
LOCK TABLE table1 WRITE
```

Assume `table1` contains the following trigger:

```
CREATE TRIGGER trigger1 AFTER INSERT ON table1 FOR EACH ROW
BEGIN
  INSERT INTO table2 VALUES (1);
  UPDATE table3 SET writes = writes+1
    WHERE id = NEW.id AND EXISTS (SELECT id FROM table4);
END;
```

Not only is `table1` write locked, `table2` and `table3` are also write locked, due to the possible `INSERT` and `UPDATE`, while `table4` is read locked due to the `SELECT`.

4.3.2 Event Scheduler

Events are named database objects containing SQL statements that are to be executed at a later stage, either once off, or at regular intervals.

Events Overview
Scheduled events.

Event Limitations
Restrictions applying to events

CREATE EVENT

Create and schedule a new event.

ALTER EVENT

Change an existing event.

DROP EVENT

Removes an existing event.

Information Schema EVENTS Table

Server event information

SHOW EVENTS

Shows information about events

SHOW CREATE EVENT

Displays the CREATE EVENT statement needed to re-create a given event

Automating MariaDB Tasks with Events

Using MariaDB events for automating tasks.

mysql.event Table

Information about MariaDB events.

There are 2 related questions.

4.3.2.1 Events Overview

Contents

Events are named database objects containing SQL statements that are to be executed at a later stage, either once off, or at regular intervals.

They function very similarly to the Windows Task Scheduler or Unix cron jobs.

Creating, modifying or deleting events requires the EVENT privilege.

Creating Events

Events are created with the CREATE EVENT statement.

Example

```
CREATE EVENT test_event
  ON SCHEDULE EVERY 1 MINUTE DO
   UPDATE test.t1 SET a = a + 1;
```

Executing Events

Events are only executed if the event scheduler is running. This is determined by the value of the event_scheduler system variable, which needs to be set to `On` for the event scheduler to be running.

You can check if the Event scheduler is running with:

```
SHOW PROCESSLIST;
+----+-----------------+-----------+------+---------+------+------------------------------+---
---------------+----------+
| Id | User            | Host      | db   | Command | Time | State                        |
Info             | Progress |
+----+-----------------+-----------+------+---------+------+------------------------------+---
---------------+----------+
| 40 | root            | localhost | test | Sleep   | 4687 |                              |
NULL             |    0.000 |
| 41 | root            | localhost | test | Query   |    0 | init                         |
SHOW PROCESSLIST |    0.000 |
| 42 | event_scheduler | localhost | NULL | Daemon  |   30 | Waiting for next activation | NULL
|    0.000 |
+----+-----------------+-----------+------+---------+------+------------------------------+---
---------------+----------+
```

If the event scheduler is not running and `event_scheduler` has been set to `OFF`, use:

```
SET GLOBAL event_scheduler = ON;
```

to activate it. If `event_scheduler` has been set to `Disabled`, you cannot change the value at runtime. Changing the value of the `event_scheduler` variable requires the SUPER privilege.

Since MariaDB 10.0.22, setting the event_scheduler system variable will also try to reload the mysql.event table if it was not properly loaded at startup.

Viewing Current Events

A list of current events can be obtained with the SHOW EVENTS statement. This only shows the event name and interval - the full event details, including the SQL, can be seen by querying the Information Schema EVENTS table, or with SHOW CREATE EVENT.

If an event is currently being executed, it can be seen by querying the Information Schema PROCESSLIST table, or with the SHOW PROCESSLIST statement.

Example

```
SHOW EVENTS\G;
*************************** 1. row ***************************
                  Db: test
                Name: test_event
             Definer: root@localhost
           Time zone: SYSTEM
                Type: RECURRING
          Execute at: NULL
      Interval value: 1
      Interval field: MINUTE
              Starts: 2013-05-20 13:46:56
                Ends: NULL
              Status: ENABLED
          Originator: 1
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
```

```
SHOW CREATE EVENT test_event\G
*************************** 1. row ***************************
               Event: test_event
            sql_mode:
           time_zone: SYSTEM
        Create Event: CREATE DEFINER=`root`@`localhost` EVENT `test_event` ON SCHEDULE EVERY 1
MINUTE STARTS '2013-05-20 13:46:56' ON COMPLETION NOT PRESERVE ENABLE DO UPDATE test.t1 SET a =
a + 1
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
```

Altering Events

An event can be changed with the ALTER EVENT statement.

Example

```
ALTER EVENT test_event ON SCHEDULE EVERY '2:3' DAY_HOUR;
```

Dropping Events

Events are dropped with the DROP EVENT statement. Events are also also automatically dropped once they have run for the final time according to their schedule, unless the ON COMPLETION PRESERVE clause has been specified.

Example

```
DROP EVENT test_event;
Query OK, 0 rows affected (0.00 sec)
```

4.3.2.2 Event Limitations

The following restrictions apply to Events.

- All of the restrictions listed in Stored Routine Limitations.
- Events cannot return a resultset.
- Event names are case insensitive, so it's not possible to define two events in the same database if their case insensitive names will match. This restriction has applied since MariaDB/MySQL 5.1.8. If you are upgrading from an older version of MySQL, and have events that could clash, these events need to be renamed before the upgrade.
- Events do not support dates beyond the maximum that can be represented in the Unix epoch (2038-01-19).
- Events cannot be created, dropped or altered by another stored program, trigger or event.
- Events cannot create, drop or alter stored programs or triggers
- Event timings cannot be strictly predicted. The intervals MONTH, YEAR_MONTH, QUARTER and YEAR are all resolved in months. All others are resolved in seconds. A delay of up to two seconds is possible in extreme cases, and events scheduled to run at the same second cannot be executed in a given order. The `LAST_EXECUTED` column in the `INFORMATION_SCHEMA.EVENTS` table will however always be accurate to within a second.
- A new connection is used for each execution of statements within the body of an event, so the session counts for server status variables such as Com_delete and Com_select will not reflect these.
- Because the Event Scheduler depends on grant tables for its functionality, it is automatically disabled when the server is running with `--skip-grant-tables` .

1.1.1.3.1.2 CREATE EVENT

1.1.1.2.1.1.3 ALTER EVENT

1.1.1.3.3.2 DROP EVENT

1.1.1.2.9.1.1.16 Information Schema EVENTS Table

1.1.1.2.8.27 SHOW EVENTS

1.1.1.2.8.12 SHOW CREATE EVENT

2.1.2.14.14 Automating MariaDB Tasks with Events

1.1.1.2.9.3.4 mysql.event Table

4.4 Views

Views are stored queries that act as a virtual table.

Creating & Using Views

A tutorial on creating and using views.

CREATE VIEW

Create or replace a view.

ALTER VIEW

Change a view definition.

DROP VIEW

Removes one or more views.

SHOW CREATE VIEW

Show the CREATE VIEW statement that created a view.

Inserting and Updating with Views

Views can be used for inserting or updating with certain limitations.

RENAME TABLE

Change a table's name.

View Algorithms

Optional ALGORITHM clause when creating views

Information Schema VIEWS Table

Information about views.

SHOW TABLES

List of non-temporary tables, views or sequences.

6.2.3 Creating & Using Views

1.1.1.3.1.17 CREATE VIEW

1.1.1.2.1.1.11 ALTER VIEW

1.1.1.3.3.17 DROP VIEW

1.1.1.2.8.21 SHOW CREATE VIEW

4.4.6 Inserting and Updating with Views

Contents

A view can be used for inserting or updating. However, there are certain limitations.

Updating with views

A view cannot be used for updating if it uses any of the following:

- ALGORITHM=TEMPTABLE (see View Algorithms)

- HAVING
- GROUP BY
- DISTINCT
- UNION
- UNION ALL
- An aggregate function, such as MAX(), MIN(), SUM() or COUNT()
- subquery in the SELECT list
- subquery in the WHERE clause referring to a table in the FROM clause
- if it has no underlying table because it refers only to literal values
- the FROM clause contains a non-updatdable view.
- multiple references to any base table column
- an outer join
- an inner join where more than one table in the view definition is being updated
- if there's a LIMIT clause, the view does not contain all primary or not null unique key columns from the underlying table and the updatable_views_with_limit system variable is set to `0`.

Inserting with views

A view cannot be used for inserting if it fails any of the criteria for updating, and must also meet the following conditions:

- the view contains all base table columns that don't have default values
- no base table columns are present in view select list more than once
- the view columns are all simple columns, and not derived in any way. The following are examples of derived columns
  - column_name + 25
  - LOWER(column_name)
  - (subquery)
  - 9.5
  - column1 / column2

Checking whether a view is updatable

MariaDB stores an IS_UPDATABLE flag with each view, so it is always possible to see if MariaDB considers a view updatable (although not necessarily insertable) by querying the IS_UPDATABLE column in the INFORMATION_SCHEMA.VIEWS table.

WITH CHECK OPTION

The WITH CHECK OPTION clause is used to prevent updates or inserts to views unless the WHERE clause in the SELECT statement is true.

There are two keywords that can be applied. WITH LOCAL CHECK OPTION restricts the CHECK OPTION to only the view being defined, while WITH CASCADED CHECK OPTION checks all underlying views as well. CASCADED is treated as default if neither keyword is given.

If a row is rejected because of the CHECK OPTION, an error similar to the following is produced:

```
ERROR 1369 (HY000): CHECK OPTION failed 'db_name.view_name'
```

A view with a WHERE which is always false (like `WHERE 0`) and WITH CHECK OPTION is similar to a BLACKHOLE table: no row is ever inserted and no row is ever returned. An insertable view with a WHERE which is always false but no CHECK OPTION is a view that accepts data but does not show them.

Examples

```
CREATE TABLE table1 (x INT);

CREATE VIEW view1 AS SELECT x, 99 AS y FROM table1;
```

Checking whether the view is updateable:

```
SELECT TABLE_NAME,IS_UPDATABLE FROM INFORMATION_SCHEMA.VIEWS;
+------------+--------------+
| TABLE_NAME | IS_UPDATABLE |
+------------+--------------+
| view1      | YES          |
+------------+--------------+
```

This query works, as the view is updateable:

```
UPDATE view1 SET x = 5;
```

This query fails, since column `y` is a literal.

```
UPDATE view1 SET y = 5;
ERROR 1348 (HY000): Column 'y' is not updatable
```

Here are three views to demonstrate the WITH CHECK OPTION clause.

```
CREATE VIEW view_check1 AS SELECT * FROM table1 WHERE x < 100 WITH CHECK OPTION;

CREATE VIEW view_check2 AS SELECT * FROM view_check1 WHERE x > 10 WITH LOCAL CHECK OPTION;

CREATE VIEW view_check3 AS SELECT * FROM view_check1 WHERE x > 10 WITH CASCADED CHECK OPTION;
```

This insert succeeds, as `view_check2` only checks the insert against `view_check2`, and the WHERE clause evaluates to true (`150` is `>10`).

```
INSERT INTO view_check2 VALUES (150);
```

This insert fails, as `view_check3` checks the insert against both `view_check3` and the underlying views. The WHERE clause for `view_check1` evaluates as false (`150` is `>10`, but `150` is not `<100`), so the insert fails.

```
INSERT INTO view_check3 VALUES (150);
ERROR 1369 (HY000): CHECK OPTION failed 'test.view_check3'
```

1.1.1.2.1.12 RENAME TABLE

4.4.8 View Algorithms

Contents

Description

The CREATE VIEW statement accepts an optional ALGORITHM clause, an extension to standard SQL for Views.

It can contain one of three values: MERGE, TEMPTABLE or UNDEFINED, and affects how MariaDB will process the view.

With MERGE, the view definition and the related portion of the statement referring to the view are merged. If TEMPTABLE is selected, the view results are stored in a temporary table.

MERGE is usually more efficient, and a view can only be updated with this algorithm. TEMPTABLE can be useful in certain situations, as locks on the underlying tables can be released before the statement is finished processing.

If it's UNDEFINED (or the ALGORITHM clause is not used), MariaDB will choose what it thinks is the best algorithm. An algorithm can also be UNDEFINED if its defined as MERGE, but the view requires a temporary table.

Views with definition ALGORITHM=MERGE or ALGORITHM=TEMPTABLE got accidentally swapped between MariaDB and MySQL. When upgrading, you have to re-create views created with either of these definitions (see

MDEV-6916).

MERGE Limitations

A view cannot be of type ALGORITHM=MERGE if it uses any of the following:

- HAVING
- LIMIT
- GROUP BY
- DISTINCT
- UNION
- UNION ALL
- An aggregate function, such as MAX(), MIN(), SUM() or COUNT()
- subquery in the SELECT list
- if it has no underlying table because it refers only to literal values

MERGE Examples

Example 1

Here's an example of how MariaDB handles a view with a MERGE algorithm. Take a view defined as follows:

```
CREATE ALGORITHM = MERGE VIEW view_name (view_field1, view_field2) AS
SELECT field1, field2 FROM table_name WHERE field3 > '2013-06-01';
```

Now, if we run a query on this view, as follows:

```
SELECT * FROM view_name;
```

to execute the view `view_name` becomes the underlying table, `table_name` , the `*` becomes the fields `view_field1` and `view_field2` , corresponding to `field1` and `field2` and the WHERE clause, `WHERE field3 > 100` is added, so the actual query executed is:

```
SELECT field1, field2 FROM table_name WHERE field3 > '2013-06-01'
```

Example 2

Given the same view as above, if we run the query:

```
SELECT * FROM view_name WHERE view_field < 8000;
```

everything occurs as it does in the previous example, but `view_field < 8000` takes the corresponding field name and becomes `field1 < 8000` , connected with `AND` to the `field3 > '2013-06-01'` part of the query.

So the resulting query is:

```
SELECT field1, field2 FROM table_name WHERE (field3 > '2013-06-01') AND (field1 < 8000);
```

When connecting with `AND` , parentheses are added to make sure the correct precedence is used.

1.1.1.2.9.1.1.62 Information Schema VIEWS Table

1.1.1.2.8.52 SHOW TABLES

4.5 User-Defined Functions

A user-defined function (UDF) is a way to extend MariaDB with a new function that works like a native (built-in) MariaDB function such as ABS( ) or CONCAT( ).

Statements making use of user-defined functions are not safe for replication.

For an example, see `sql/udf_example.cc` in the source tree. For a collection of existing UDFs go to the UDF Repository on GitHub.

There are alternative ways to add a new function: writing a native function, which requires modifying and compiling the server source code; or writing a stored function.

Creating User-Defined Functions

How to create user-defined functions in C/C++.

User-Defined Functions Calling Sequences

Declaring the functions required in a user-defined function.

User-Defined Functions Security

MariaDB imposes a number of limitations on user-defined functions for security purposes.

CREATE FUNCTION UDF

Create a user-defined function.

DROP FUNCTION UDF

Drop a user-defined function.

mysql.func Table

User-defined function information

There are 2 related questions.

4.5.1 Creating User-Defined Functions

Contents

User-defined functions allow MariaDB to be extended with a new function that works like a native (built-in) MariaDB function such as ABS() or CONCAT(). There are alternative ways to add a new function: writing a native function (which requires modifying and compiling the server source code), or writing a stored function.

Statements making use of user-defined functions are not safe for replication.

Functions are written in C or C++, and to make use of them, the operating system must support dynamic loading.

Each new SQL function requires corresponding functions written in C/C++. In the list below, at least the main function - x() - and one other, are required. *x* should be replaced by the name of the function you are creating.

All functions need to be thread-safe, so not global or static variables that change can be allocated. Memory is allocated in *x_init()/ and freed in* x_deinit().

Simple Functions

x()

Required for all UDFs; this is where the results are calculated.

C/C++ type	SQL type
char *	STRING

long long	INTEGER
double	REAL

DECIMAL functions return string values, and so should be written accordingly. It is not possible to create ROW functions.

x_init()

Initialization function for x(). Can be used for the following:

- Check the number of arguments to X() (the SQL equivalent).
- Verify the argument types, or to force arguments to be of a particular type after the function is called.
- Specify whether the result can be NULL.
- Specify the maximum result length.
- For REAL functions, specify the maximum number of decimals for the result.
- Allocate any required memory.

x_deinit()

De-initialization function for x(). Used to de-allocate memory that was allocated in x_init().

Description

Each time the SQL function *X()* is called:

- MariaDB will first call the C/C++ initialization function, *x_init()*, assuming it exists. All setup will be performed, and if it returns an error, the SQL statement is aborted and no further functions are called.
- If there is no *x_init()* function, or it has been called and did not return an error, *x()* is then called once per row.
- After all rows have finished processing, *x_deinit()* is called, if present, to clean up by de-allocating any memory that was allocated in *x_init()*.
- See User-defined Functions Calling Sequences for more details on the functions.

Aggregate Functions

The following functions are required for aggregate functions, such as AVG() and SUM(). When using CREATE FUNCTION, the AGGREGATE keyword is required.

x_clear()

Used to reset the current aggregate, but without inserting the argument as the initial aggregate value for the new group.

x_add()

Used to add the argument to the current aggregate.

x_remove()

Starting from MariaDB 10.4, improves the support of window functions (so it is not obligatory to add it) and should remove the argument from the current aggregate.

Description

Each time the aggregate SQL function *X()* is called:

- MariaDB will first call the C/C++ initialization function, *x_init()*, assuming it exists. All setup will be performed, and if it returns an error, the SQL statement is aborted and no further functions are called.
- If there is no *x_init()* function, or it has been called and did not return an error, *x()* is then called once per row.
- After all rows have finished processing, *x_deinit()* is called, if present, to clean up by de-allocating any memory that was allocated in *x_init()*.

- MariaDB will first call the C/C++ initialization function, *x_init()*, assuming it exists. All setup will be performed, and if it returns an error, the SQL statement is aborted and no further functions are called.
- The table is sorted according to the GROUP BY expression.
- *x_clear()* is called for the first row of each new group.
- *x_add()* is called once per row for each row in the same group.
- *x()* is called when the group changes, or after the last row, to get the aggregate result.

- The latter three steps are repeated until all rows have been processed.
- After all rows have finished processing, *x_deinit()* is called, if present, to clean up by de-allocating any memory that was allocated in *x_init()*.

Examples

For an example, see `sql/udf_example.cc` in the source tree. For a collection of existing UDFs see https://github.com/mysqludf.

4.5.2 User-Defined Functions Calling Sequences

Contents

The functions described in Creating User-defined Functions are expanded on this page. They are declared as follows:

Simple Functions

x()

If x() returns an integer, it is declared as follows:

```
long long x(UDF_INIT *initid, UDF_ARGS *args,
            char *is_null, char *error);
```

If x() returns a string (DECIMAL functions also return string values), it is declared as follows:

```
char *x(UDF_INIT *initid, UDF_ARGS *args,
        char *result, unsigned long *length,
        char *is_null, char *error);
```

If x() returns a real, it is declared as follows:

```
double x(UDF_INIT *initid, UDF_ARGS *args,
            char *is_null, char *error);
```

x_init()

```
my_bool x_init(UDF_INIT *initid, UDF_ARGS *args, char *message);
```

x_deinit()

```
void x_deinit(UDF_INIT *initid);
```

Description

initid is a parameter passed to all three functions that points to a *UDF_INIT* structure, used for communicating information between the functions. Its structure members are:

- my_bool maybe_null

    - *maybe_null* should be set to *1* if x_init can return a NULL value, Defaults to *1* if any arguments are declared *maybe_null*.
- unsigned int decimals
    - Number of decimals after the decimal point. The default, if an explicit number of decimals is passed in the arguments to the main function, is the maximum number of decimals, so if *9.5*, *9.55* and *9.555* are passed to the function, the default would be three (based on *9.555*, the maximum). If there are no explicit number of decimals, the default is set to 31, or one more than the maximum for the DOUBLE, FLOAT and DECIMAL types. This default can be changed in the function to suit the actual calculation.
- unsigned int max_length
    - Maximum length of the result. For integers, the default is 21. For strings, the length of the longest argument. For reals, the default is 13 plus the number of decimals indicated by *initid->decimals*. The length includes any signs or decimal points. Can also be set to 65KB or 16MB in order to return a BLOB. The memory remains unallocated, but this is used to decide on the data type to use if the data needs to be temporarily stored.
- char *ptr
    - A pointer for use as required by the function. Commonly, *initid->ptr* is used to communicate allocated memory, with *x_init()* allocating the memory and assigning it to this pointer, *x()* using it, and *x_deinit()* de-allocating it.
- my_bool const_item
    - Should be set to *1* in *x_init()* if *x()* always returns the same value, otherwise *0*.

Aggregate Functions

x_clear()

x_clear() is a required function for aggregate functions, and is declared as follows:

```
void x_clear(UDF_INIT *initid, char *is_null, char *error);
```

It is called when the summary results need to be reset, that is at the beginning of each new group. but also to reset the values when there were no matching rows.

is_null is set to point to CHAR(0) before calling x_clear().

In the case of an error, you can store the value to which the error argument points (a single-byte variable, not a string string buffer) in the variable.

x_reset()

x_reset() is declared as follows:

```
void x_reset(UDF_INIT *initid, UDF_ARGS *args,
             char *is_null, char *error);
```

It is called on finding the first row in a new group. Should reset the summary variables, and then use *UDF_ARGS* as the first value in the group's internal summary value. The function is not required if the UDF interface uses *x_clear()*.

x_add()

x_add() is declared as follows:

```
void x_add(UDF_INIT *initid, UDF_ARGS *args,
           char *is_null, char *error);
```

It is called for all rows belonging to the same group, and should be used to add the value in *UDF_ARGS* to the internal summary variable.

x_remove()

x_remove() was added in MariaDB 10.4 and is declared as follows (same as *x_add()*):

```
void x_remove(UDF_INIT* initid, UDF_ARGS* args,
              char* is_null, char *error );
```

It adds more efficient support of aggregate UDFs as window functions. *x_remove()* should "subtract" the row (reverse *x_add()*). In MariaDB 10.4 aggregate UDFs will work as WINDOW functions without *x_remove()* but it will not be so efficient.

If *x_remove()* supported (defined) detected automatically.

4.5.3 User-Defined Functions Security

The MariaDB server imposes a number of limitations on user-defined functions for security purposes.

- The INSERT privilege for the mysql database is required to run CREATE FUNCTION, as a record will be added to the mysql.func-table.
- The DELETE privilege for the mysql database is required to run DROP FUNCTION as the corresponding record will be removed from the mysql.func-table.
- UDF object files can only be placed in the plugin directory, as specified by the value of the plugin_dir system variable.
- At least one symbol, beyond the required *x()* - corresponding to an SQL function *X())* - is required. These can be *x_init()*, *x_deinit()*, *xxx_reset()*, *x_clear()* and *x_add()* functions (see Creating User-defined Functions). The allow-suspicious-udfs mariadbd option (by default unset) provides a workaround, permitting only one symbol to be used. This is not recommended, as it opens the possibility of loading shared objects that are not legitimate user-defined functions.

1.1.1.3.1.4 CREATE FUNCTION UDF

1.1.1.3.3.4 DROP FUNCTION UDF

1.1.1.2.9.3.5 mysql.func Table

5 Columns, Storage Engines, and Plugins

MariaDB allows for a variety of column data types, characters and collations.

Data Types
Data types for columns in MariaDB tables.

Character Sets and Collations
Setting character set and collation for a language.

Storage Engines
Various storage engines available for MariaDB.

Plugins
Documentation on MariaDB Server plugins.

5.1 Data Types

Data Types in MariaDB

Numeric Data Types

Numeric Data Type Overview
Overview and usage of the numeric data types.

TINYINT
Tiny integer, -128 to 127 signed.

BOOLEAN
Synonym for TINYINT(1).

SMALLINT
Small integer from -32768 to 32767 signed.

MEDIUMINT
Medium integer from -8388608 to 8388607 signed.

INT
Integer from -2147483648 to 2147483647 signed.

INTEGER
Synonym for INT

BIGINT
Large integer.

DECIMAL
A packed "exact" fixed-point number.

DEC, NUMERIC, FIXED
Synonyms for DECIMAL

NUMBER
Synonym for DECIMAL in Oracle mode.

FLOAT
Single-precision floating-point number

DOUBLE
Normal-size (double-precision) floating-point number

DOUBLE PRECISION
REAL and DOUBLE PRECISION are synonyms for DOUBLE.

BIT
Bit field type.

Floating-point Accuracy
Not all floating-point numbers can be stored with exact precision

INT1
A synonym for TINYINT.

INT2
Synonym for SMALLINT.

INT3
Synonym for MEDIUMINT.

INT4
Synonym for INT.

INT8
Synonym for BIGINT.

String Data Types

String Literals
Strings are sequences of characters and are enclosed with quotes.

BINARY
Fixed-length binary byte string.

BLOB
Binary large object up to 65,535 bytes.

BLOB and TEXT Data Types
Binary large object data types and the corresponding TEXT types.

CHAR

Fixed-length string.

CHAR BYTE

Alias for BINARY.

ENUM

Enumeration, or string object that can have one value chosen from a list of values.

INET4

For storage of IPv4 addresses.

INET6

For storage of IPv6 addresses.

JSON Data Type

Compatibility data type that is an alias for LONGTEXT.

MEDIUMBLOB

Medium binary large object up to 16,777,215 bytes.

MEDIUMTEXT

A TEXT column with a maximum length of 16,777,215 characters.

LONGBLOB

Long BLOB holding up to 4GB.

LONG and LONG VARCHAR

LONG and LONG VARCHAR are synonyms for MEDIUMTEXT.

LONGTEXT

A TEXT column with a maximum length of 4,294,967,295 characters.

ROW

Data type for stored procedure variables.

TEXT

A TEXT column with a maximum length of 65,535 characters.

TINYBLOB

Tiny binary large object up to 255 bytes.

TINYTEXT

A TEXT column with a maximum length of 255 characters.

VARBINARY

Variable-length binary byte string.

VARCHAR

Variable-length string.

SET Data Type

Set, or string object that can have 0 or more values chosen from a list of values.

UUID Data Type

Data type intended for the storage of UUID data.

Data Type Storage Requirements

Storage requirements for the various data types.

Supported Character Sets and Collations

MariaDB supports the following character sets and collations.

Character Sets and Collations
Setting character set and collation for a language.

Date and Time Data Types

DATE
The date type YYYY-MM-DD.

TIME
Time format HH:MM:SS.ssssss

DATETIME
Date and time combination displayed as YYYY-MM-DD HH:MM:SS.

TIMESTAMP
YYYY-MM-DD HH:MM:SS

YEAR Data Type
A four-digit year.

Future developments for temporal types
My current project is a forecasting application with dates going out to 263...

How to define a date in order to import an empty date from a CSV file?
I have a CSV file containing amongst other things a couple of date columns....

Which datatypes are supported by MariaDB
I would like to know which datatypes are supported by MariaDB. I'm asking s...

Other Data Types Articles

Geometry Types
Supported geometry types.

AUTO_INCREMENT
Automatic increment.

Data Type Storage Requirements
Storage requirements for the various data types.

AUTO_INCREMENT FAQ
Frequently-asked questions about auto_increment.

NULL Values
NULL represents an unknown value.

There are 8 related questions.

5.1.1 Numeric Data Types

Numeric Data Type Overview
Overview and usage of the numeric data types.

TINYINT
Tiny integer, -128 to 127 signed.

BOOLEAN
Synonym for TINYINT(1).

SMALLINT
Small integer from -32768 to 32767 signed.

MEDIUMINT
Medium integer from -8388608 to 8388607 signed.

INT
Integer from -2147483648 to 2147483647 signed.

INTEGER
Synonym for INT

BIGINT
Large integer.

DECIMAL
A packed "exact" fixed-point number.

DEC, NUMERIC, FIXED
Synonyms for DECIMAL

NUMBER
Synonym for DECIMAL in Oracle mode.

FLOAT
Single-precision floating-point number

DOUBLE
Normal-size (double-precision) floating-point number

DOUBLE PRECISION
REAL and DOUBLE PRECISION are synonyms for DOUBLE.

BIT
Bit field type.

Floating-point Accuracy
Not all floating-point numbers can be stored with exact precision

INT1
A synonym for TINYINT.

INT2
Synonym for SMALLINT.

INT3
Synonym for MEDIUMINT.

INT4
Synonym for INT.

INT8
Synonym for BIGINT.

5.1.1.1 Numeric Data Type Overview

Contents

There are a number of numeric data types:

- TINYINT
- BOOLEAN - Synonym for TINYINT(1)

- INT1 - Synonym for TINYINT
- SMALLINT
- INT2 - Synonym for SMALLINT
- MEDIUMINT
- INT3 - Synonym for MEDIUMINT
- INT, INTEGER
- INT4 - Synonym for INT
- BIGINT
- INT8 - Synonym for BIGINT
- DECIMAL, DEC, NUMERIC, FIXED
- FLOAT
- DOUBLE, DOUBLE PRECISION, REAL
- BIT

See the specific articles for detailed information on each.

SIGNED, UNSIGNED and ZEROFILL

Most numeric types can be defined as `SIGNED`, `UNSIGNED` or `ZEROFILL`, for example:

```
TINYINT[(M)] [SIGNED | UNSIGNED | ZEROFILL]
```

If `SIGNED`, or no attribute, is specified, a portion of the numeric type will be reserved for the sign (plus or minus). For example, a TINYINT SIGNED can range from -128 to 127.

If `UNSIGNED` is specified, no portion of the numeric type is reserved for the sign, so for integer types range can be larger. For example, a TINYINT UNSIGNED can range from 0 to 255. Floating point and fixed-point types also can be `UNSIGNED`, but this only prevents negative values from being stored and doesn't alter the range.

If `ZEROFILL` is specified, the column will be set to UNSIGNED and the spaces used by default to pad the field are replaced with zeros. `ZEROFILL` is ignored in expressions or as part of a UNION. `ZEROFILL` is a non-standard MySQL and MariaDB enhancement.

Note that although the preferred syntax indicates that the attributes are exclusive, more than one attribute can be specified.

Until MariaDB 10.2.7 (MDEV-8659), any combination of the attributes could be used in any order, with duplicates. In this case:

- the presence of `ZEROFILL` makes the column `UNSIGNED ZEROFILL`.
- the presence of `UNSIGNED` makes the column `UNSIGNED`.

From MariaDB 10.2.8, only the following combinations are supported:

- `SIGNED`
- `UNSIGNED`
- `ZEROFILL`
- `UNSIGNED ZEROFILL`
- `ZEROFILL UNSIGNED`

The latter two should be replaced with simply `ZEROFILL`, but are still accepted by the parser.

Examples

```
CREATE TABLE zf (
  i1 TINYINT SIGNED,
  i2 TINYINT UNSIGNED,
  i3 TINYINT ZEROFILL
);

INSERT INTO zf VALUES (2,2,2);

SELECT * FROM zf;
+------+------+------+
| i1   | i2   | i3   |
+------+------+------+
|    2 |    2 |  002 |
+------+------+------+
```

Range

When attempting to add a value that is out of the valid range for the numeric type, MariaDB will react depending on the strict SQL_MODE setting.

If strict_mode has been set (the default from MariaDB 10.2.4), MariaDB will return an error.

If strict_mode has not been set (the default until MariaDB 10.2.3), MariaDB will adjust the number to fit in the field, returning a warning.

Examples

With strict_mode set:

```
SHOW VARIABLES LIKE 'sql_mode';
+---------------+-----------------------------------------------------------------------------------------------
---------------+
| Variable_name | Value
|
+---------------+-----------------------------------------------------------------------------------------------
---------------+
| sql_mode      |
STRICT_TRANS_TABLES,ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION |
+---------------+-----------------------------------------------------------------------------------------------
---------------+

CREATE TABLE ranges (i1 TINYINT, i2 SMALLINT, i3 TINYINT UNSIGNED);

INSERT INTO ranges VALUES (257,257,257);
ERROR 1264 (22003): Out of range value for column 'i1' at row 1

SELECT * FROM ranges;
Empty set (0.10 sec)
```

With strict_mode unset:

```
SHOW VARIABLES LIKE 'sql_mode%';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| sql_mode      |       |
+---------------+-------+

CREATE TABLE ranges (i1 TINYINT, i2 SMALLINT, i3 TINYINT UNSIGNED);

INSERT INTO ranges VALUES (257,257,257);
Query OK, 1 row affected, 2 warnings (0.00 sec)

SHOW WARNINGS;
+---------+------+-----------------------------------------------+
| Level   | Code | Message                                       |
+---------+------+-----------------------------------------------+
| Warning | 1264 | Out of range value for column 'i1' at row 1 |
| Warning | 1264 | Out of range value for column 'i3' at row 1 |
+---------+------+-----------------------------------------------+
2 rows in set (0.00 sec)

SELECT * FROM ranges;
+------+------+------+
| i1   | i2   | i3   |
+------+------+------+
|  127 |  257 |  255 |
+------+------+------+
```

Auto_increment

The `AUTO_INCREMENT` attribute can be used to generate a unique identity for new rows. For more details, see auto_increment.

5.1.1.2 TINYINT

Syntax

```
TINYINT[(M)] [SIGNED | UNSIGNED | ZEROFILL]
```

Contents

Description

A very small integer. The signed range is -128 to 127. The unsigned range is 0 to 255. For details on the attributes, see Numeric Data Type Overview.

INT1 is a synonym for `TINYINT` . BOOL and BOOLEAN are synonyms for `TINYINT(1)` .

Examples

```
CREATE TABLE tinyints (a TINYINT,b TINYINT UNSIGNED,c TINYINT ZEROFILL);
```

With strict_mode set, the default from MariaDB 10.2.4:

```
INSERT INTO tinyints VALUES (-10,-10,-10);
ERROR 1264 (22003): Out of range value for column 'b' at row 1

INSERT INTO tinyints VALUES (-10,10,-10);
ERROR 1264 (22003): Out of range value for column 'c' at row 1

INSERT INTO tinyints VALUES (-10,10,10);

SELECT * FROM tinyints;
+------+------+------+
| a    | b    | c    |
+------+------+------+
|  -10 |   10 |  010 |
+------+------+------+

INSERT INTO tinyints VALUES (128,128,128);
ERROR 1264 (22003): Out of range value for column 'a' at row 1

INSERT INTO tinyints VALUES (127,128,128);

SELECT * FROM tinyints;
+------+------+------+
| a    | b    | c    |
+------+------+------+
|  -10 |   10 |  010 |
|  127 |  128 |  128 |
+------+------+------+
```

With strict_mode unset, the default until MariaDB 10.2.3:

```
INSERT INTO tinyints VALUES (-10,-10,-10);
Query OK, 1 row affected, 2 warnings (0.08 sec)
Warning (Code 1264): Out of range value for column 'b' at row 1
Warning (Code 1264): Out of range value for column 'c' at row 1

INSERT INTO tinyints VALUES (-10,10,-10);
Query OK, 1 row affected, 1 warning (0.11 sec)
Warning (Code 1264): Out of range value for column 'c' at row 1

INSERT INTO tinyints VALUES (-10,10,10);

SELECT * FROM tinyints;
+------+------+------+
| a    | b    | c    |
+------+------+------+
|  -10 |    0 |  000 |
|  -10 |   10 |  000 |
|  -10 |   10 |  010 |
+------+------+------+

INSERT INTO tinyints VALUES (128,128,128);
Query OK, 1 row affected, 1 warning (0.19 sec)
Warning (Code 1264): Out of range value for column 'a' at row 1

INSERT INTO tinyints VALUES (127,128,128);

SELECT * FROM tinyints;
+------+------+------+
| a    | b    | c    |
+------+------+------+
|  -10 |    0 |  000 |
|  -10 |   10 |  000 |
|  -10 |   10 |  010 |
|  127 |  128 |  128 |
|  127 |  128 |  128 |
+------+------+------+
```

5.1.1.3 BOOLEAN

Syntax

```
BOOL, BOOLEAN
```

Description

These types are synonyms for TINYINT(1). A value of zero is considered false. Non-zero values are considered true.

However, the values TRUE and FALSE are merely aliases for 1 and 0. See Boolean Literals, as well as the IS operator for testing values against a boolean.

Examples

```
CREATE TABLE boo (i BOOLEAN);

DESC boo;
+-------+------------+------+-----+---------+-------+
| Field | Type       | Null | Key | Default | Extra |
+-------+------------+------+-----+---------+-------+
| i     | tinyint(1) | YES  |     | NULL    |       |
+-------+------------+------+-----+---------+-------+
```

```
SELECT IF(0, 'true', 'false');
+-----------------------+
| IF(0, 'true', 'false') |
+-----------------------+
| false                 |
+-----------------------+

SELECT IF(1, 'true', 'false');
+-----------------------+
| IF(1, 'true', 'false') |
+-----------------------+
| true                  |
+-----------------------+

SELECT IF(2, 'true', 'false');
+-----------------------+
| IF(2, 'true', 'false') |
+-----------------------+
| true                  |
+-----------------------+
```

TRUE and FALSE as aliases for 1 and 0:

```
SELECT IF(0 = FALSE, 'true', 'false');

+-------------------------------+
| IF(0 = FALSE, 'true', 'false') |
+-------------------------------+
| true                          |
+-------------------------------+

SELECT IF(1 = TRUE, 'true', 'false');
+-------------------------------+
| IF(1 = TRUE, 'true', 'false') |
+-------------------------------+
| true                          |
+-------------------------------+

SELECT IF(2 = TRUE, 'true', 'false');
+-------------------------------+
| IF(2 = TRUE, 'true', 'false') |
+-------------------------------+
| false                         |
+-------------------------------+

SELECT IF(2 = FALSE, 'true', 'false');
+--------------------------------+
| IF(2 = FALSE, 'true', 'false') |
+--------------------------------+
| false                          |
+--------------------------------+
```

The last two statements display the results shown because 2 is equal to neither 1 nor 0.

5.1.1.4 SMALLINT

Syntax

```
SMALLINT[(M)] [SIGNED | UNSIGNED | ZEROFILL]
```

Description

A small integer. The signed range is -32768 to 32767. The unsigned range is 0 to 65535.

If a column has been set to ZEROFILL, all values will be prepended by zeros so that the SMALLINT value contains a number of M digits.

Note: If the ZEROFILL attribute has been specified, the column will automatically become UNSIGNED.

`INT2` is a synonym for `SMALLINT`.

For more details on the attributes, see Numeric Data Type Overview.

Examples

```
CREATE TABLE smallints (a SMALLINT,b SMALLINT UNSIGNED,c SMALLINT ZEROFILL);
```

With strict_mode set, the default from MariaDB 10.2.4:

```
INSERT INTO smallints VALUES (-10,-10,-10);
ERROR 1264 (22003): Out of range value for column 'b' at row 1

INSERT INTO smallints VALUES (-10,10,-10);
ERROR 1264 (22003): Out of range value for column 'c' at row 1

INSERT INTO smallints VALUES (-10,10,10);

INSERT INTO smallints VALUES (32768,32768,32768);
ERROR 1264 (22003): Out of range value for column 'a' at row 1

INSERT INTO smallints VALUES (32767,32768,32768);

SELECT * FROM smallints;
+-------+-------+-------+
| a     | b     | c     |
+-------+-------+-------+
|   -10 |    10 | 00010 |
| 32767 | 32768 | 32768 |
+-------+-------+-------+
```

With strict_mode unset, the default until MariaDB 10.2.3:

```
INSERT INTO smallints VALUES (-10,-10,-10);
Query OK, 1 row affected, 2 warnings (0.09 sec)
Warning (Code 1264): Out of range value for column 'b' at row 1
Warning (Code 1264): Out of range value for column 'c' at row 1

INSERT INTO smallints VALUES (-10,10,-10);
Query OK, 1 row affected, 1 warning (0.08 sec)
Warning (Code 1264): Out of range value for column 'c' at row 1

INSERT INTO smallints VALUES (-10,10,10);

INSERT INTO smallints VALUES (32768,32768,32768);
Query OK, 1 row affected, 1 warning (0.04 sec)
Warning (Code 1264): Out of range value for column 'a' at row 1

INSERT INTO smallints VALUES (32767,32768,32768);

SELECT * FROM smallints;
+-------+-------+-------+
| a     | b     | c     |
+-------+-------+-------+
|   -10 |     0 | 00000 |
|   -10 |    10 | 00000 |
|   -10 |    10 | 00010 |
| 32767 | 32768 | 32768 |
| 32767 | 32768 | 32768 |
+-------+-------+-------+
```

5.1.1.5 MEDIUMINT

Syntax

```
MEDIUMINT[(M)] [SIGNED | UNSIGNED | ZEROFILL]
```

Description

A medium-sized integer. The signed range is -8388608 to 8388607. The unsigned range is 0 to 16777215.

`ZEROFILL` pads the integer with zeroes and assumes `UNSIGNED` (even if `UNSIGNED` is not specified).

`INT3` is a synonym for `MEDIUMINT`.

For details on the attributes, see Numeric Data Type Overview.

Examples

```
CREATE TABLE mediumints (a MEDIUMINT,b MEDIUMINT UNSIGNED,c MEDIUMINT ZEROFILL);

DESCRIBE mediumints;
+-------+---------------------------------+------+-----+---------+-------+
| Field | Type                            | Null | Key | Default | Extra |
+-------+---------------------------------+------+-----+---------+-------+
| a     | mediumint(9)                    | YES  |     | NULL    |       |
| b     | mediumint(8) unsigned           | YES  |     | NULL    |       |
| c     | mediumint(8) unsigned zerofill | YES  |     | NULL    |       |
+-------+---------------------------------+------+-----+---------+-------+
```

With strict_mode set, the default from MariaDB 10.2.4:

```
INSERT INTO mediumints VALUES (-10,-10,-10);
ERROR 1264 (22003): Out of range value for column 'b' at row 1

INSERT INTO mediumints VALUES (-10,10,-10);
ERROR 1264 (22003): Out of range value for column 'c' at row 1

INSERT INTO mediumints VALUES (-10,10,10);

INSERT INTO mediumints VALUES (8388608,8388608,8388608);
ERROR 1264 (22003): Out of range value for column 'a' at row 1

INSERT INTO mediumints VALUES (8388607,8388608,8388608);

SELECT * FROM mediumints;
+---------+---------+----------+
| a       | b       | c        |
+---------+---------+----------+
|     -10 |      10 | 00000010 |
| 8388607 | 8388608 | 08388608 |
+---------+---------+----------+
```

With strict_mode unset, the default until MariaDB 10.2.3:

```
INSERT INTO mediumints VALUES (-10,-10,-10);
Query OK, 1 row affected, 2 warnings (0.05 sec)
Warning (Code 1264): Out of range value for column 'b' at row 1
Warning (Code 1264): Out of range value for column 'c' at row 1

INSERT INTO mediumints VALUES (-10,10,-10);
Query OK, 1 row affected, 1 warning (0.08 sec)
Warning (Code 1264): Out of range value for column 'c' at row 1

INSERT INTO mediumints VALUES (-10,10,10);

INSERT INTO mediumints VALUES (8388608,8388608,8388608);
Query OK, 1 row affected, 1 warning (0.05 sec)
Warning (Code 1264): Out of range value for column 'a' at row 1

INSERT INTO mediumints VALUES (8388607,8388608,8388608);

SELECT * FROM mediumints;
+---------+---------+----------+
| a       | b       | c        |
+---------+---------+----------+
|     -10 |       0 | 00000000 |
|     -10 |       0 | 00000000 |
|     -10 |      10 | 00000000 |
|     -10 |      10 | 00000010 |
| 8388607 | 8388608 | 08388608 |
| 8388607 | 8388608 | 08388608 |
+---------+---------+----------+
```

5.1.1.6 INT

Syntax

```
INT[(M)] [SIGNED | UNSIGNED | ZEROFILL]
INTEGER[(M)] [SIGNED | UNSIGNED | ZEROFILL]
```

Description

A normal-size integer. When marked UNSIGNED, it ranges from 0 to 4294967295, otherwise its range is -2147483648 to 2147483647 (SIGNED is the default). If a column has been set to ZEROFILL, all values will be prepended by zeros so that the INT value contains a number of M digits. INTEGER is a synonym for INT.

Note: If the ZEROFILL attribute has been specified, the column will automatically become UNSIGNED.

`INT4` is a synonym for `INT`.

For details on the attributes, see Numeric Data Type Overview.

Examples

```
CREATE TABLE ints (a INT,b INT UNSIGNED,c INT ZEROFILL);
```

With strict_mode set, the default from MariaDB 10.2.4:

```
INSERT INTO ints VALUES (-10,-10,-10);
ERROR 1264 (22003): Out of range value for column 'b' at row 1

INSERT INTO ints VALUES (-10,10,-10);
ERROR 1264 (22003): Out of range value for column 'c' at row 1

INSERT INTO ints VALUES (-10,10,10);

INSERT INTO ints VALUES (2147483648,2147483648,2147483648);
ERROR 1264 (22003): Out of range value for column 'a' at row 1

INSERT INTO ints VALUES (2147483647,2147483648,2147483648);

SELECT * FROM ints;
+------------+------------+------------+
| a          | b          | c          |
+------------+------------+------------+
|        -10 |         10 | 0000000010 |
| 2147483647 | 2147483648 | 2147483648 |
+------------+------------+------------+
```

With strict_mode unset, the default until MariaDB 10.2.3:

```
INSERT INTO ints VALUES (-10,-10,-10);
Query OK, 1 row affected, 2 warnings (0.10 sec)
Warning (Code 1264): Out of range value for column 'b' at row 1
Warning (Code 1264): Out of range value for column 'c' at row 1

INSERT INTO ints VALUES (-10,10,-10);
Query OK, 1 row affected, 1 warning (0.08 sec)
Warning (Code 1264): Out of range value for column 'c' at row 1

INSERT INTO ints VALUES (-10,10,10);

INSERT INTO ints VALUES (2147483648,2147483648,2147483648);
Query OK, 1 row affected, 1 warning (0.07 sec)
Warning (Code 1264): Out of range value for column 'a' at row 1

INSERT INTO ints VALUES (2147483647,2147483648,2147483648);

SELECT * FROM ints;
+------------+------------+------------+
| a          | b          | c          |
+------------+------------+------------+
|        -10 |          0 | 0000000000 |
|        -10 |         10 | 0000000000 |
|        -10 |         10 | 0000000010 |
| 2147483647 | 2147483648 | 2147483648 |
| 2147483647 | 2147483648 | 2147483648 |
+------------+------------+------------+
```

5.1.1.7 INTEGER

Syntax

```
INTEGER[(M)] [SIGNED | UNSIGNED | ZEROFILL]
```

Description

This type is a synonym for INT.

5.1.1.8 BIGINT

Syntax

```
BIGINT[(M)] [SIGNED | UNSIGNED | ZEROFILL]
```

Description

A large integer. The signed range is `-9223372036854775808` to `9223372036854775807`. The unsigned range is `0` to `18446744073709551615`.

If a column has been set to ZEROFILL, all values will be prepended by zeros so that the BIGINT value contains a number of M digits.

Note: If the ZEROFILL attribute has been specified, the column will automatically become UNSIGNED.

For more details on the attributes, see Numeric Data Type Overview.

`SERIAL` is an alias for:

```
BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE
```

`INT8` is a synonym for `BIGINT`.

Examples

```
CREATE TABLE bigints (a BIGINT,b BIGINT UNSIGNED,c BIGINT ZEROFILL);
```

With strict_mode set, the default from MariaDB 10.2.4:

```
INSERT INTO bigints VALUES (-10,-10,-10);
ERROR 1264 (22003): Out of range value for column 'b' at row 1

INSERT INTO bigints VALUES (-10,10,-10);
ERROR 1264 (22003): Out of range value for column 'c' at row 1

INSERT INTO bigints VALUES (-10,10,10);

INSERT INTO bigints VALUES (9223372036854775808,9223372036854775808,9223372036854775808);
ERROR 1264 (22003): Out of range value for column 'a' at row 1

INSERT INTO bigints VALUES (9223372036854775807,9223372036854775808,9223372036854775808);

SELECT * FROM bigints;
+---------------------+---------------------+----------------------+
| a                   | b                   | c                    |
+---------------------+---------------------+----------------------+
|                 -10 |                  10 | 00000000000000000010 |
| 9223372036854775807 | 9223372036854775808 | 09223372036854775808 |
+---------------------+---------------------+----------------------+
```

With strict_mode unset, the default until MariaDB 10.2.3:

```
INSERT INTO bigints VALUES (-10,-10,-10);
Query OK, 1 row affected, 2 warnings (0.08 sec)
Warning (Code 1264): Out of range value for column 'b' at row 1
Warning (Code 1264): Out of range value for column 'c' at row 1

INSERT INTO bigints VALUES (-10,10,-10);
Query OK, 1 row affected, 1 warning (0.08 sec)
Warning (Code 1264): Out of range value for column 'c' at row 1

INSERT INTO bigints VALUES (-10,10,10);

INSERT INTO bigints VALUES (9223372036854775808,9223372036854775808,9223372036854775808);
Query OK, 1 row affected, 1 warning (0.07 sec)
Warning (Code 1264): Out of range value for column 'a' at row 1

INSERT INTO bigints VALUES (9223372036854775807,9223372036854775808,9223372036854775808);

SELECT * FROM bigints;
+---------------------+---------------------+----------------------+
| a                   | b                   | c                    |
+---------------------+---------------------+----------------------+
|                 -10 |                   0 | 00000000000000000000 |
|                 -10 |                  10 | 00000000000000000000 |
|                 -10 |                  10 | 00000000000000000010 |
| 9223372036854775807 | 9223372036854775808 | 09223372036854775808 |
| 9223372036854775807 | 9223372036854775808 | 09223372036854775808 |
+---------------------+---------------------+----------------------+
```

5.1.1.9 DECIMAL

Syntax

```
DECIMAL[(M[,D])] [SIGNED | UNSIGNED | ZEROFILL]
```

Contents

Description

A packed "exact" fixed-point number. `M` is the total number of digits (the precision) and `D` is the number of digits after the decimal point (the scale).

- The decimal point and (for negative numbers) the "-" sign are not counted in `M`.
- If `D` is `0`, values have no decimal point or fractional part and on INSERT the value will be rounded to the nearest `DECIMAL`.
- The maximum number of digits (`M`) for `DECIMAL` is 65.
- The maximum number of supported decimals (`D`) is `30` before MariadB 10.2.1 and `38` afterwards.
- If `D` is omitted, the default is `0`. If `M` is omitted, the default is `10`.

`UNSIGNED`, if specified, disallows negative values.

`ZEROFILL`, if specified, pads the number with zeros, up to the total number of digits specified by `M`.

All basic calculations (+, -, *, /) with `DECIMAL` columns are done with a precision of 65 digits.

For more details on the attributes, see Numeric Data Type Overview.

`DEC`, `NUMERIC` and `FIXED` are synonyms, as well as `NUMBER` in Oracle mode from MariaDB 10.3.

Examples

```
CREATE TABLE t1 (d DECIMAL UNSIGNED ZEROFILL);

INSERT INTO t1 VALUES (1),(2),(3),(4.0),(5.2),(5.7);
Query OK, 6 rows affected, 2 warnings (0.16 sec)
Records: 6  Duplicates: 0  Warnings: 2

Note (Code 1265): Data truncated for column 'd' at row 5
Note (Code 1265): Data truncated for column 'd' at row 6

SELECT * FROM t1;
+------------+
| d          |
+------------+
| 0000000001 |
| 0000000002 |
| 0000000003 |
| 0000000004 |
| 0000000005 |
| 0000000006 |
+------------+
```

With strict_mode set, the default from MariaDB 10.2.4:

```
INSERT INTO t1 VALUES (-7);
ERROR 1264 (22003): Out of range value for column 'd' at row 1
```

With strict_mode unset, the default until MariaDB 10.2.3:

```
INSERT INTO t1 VALUES (-7);
Query OK, 1 row affected, 1 warning (0.02 sec)
Warning (Code 1264): Out of range value for column 'd' at row 1

SELECT * FROM t1;
+------------+
| d          |
+------------+
| 0000000001 |
| 0000000002 |
| 0000000003 |
| 0000000004 |
| 0000000005 |
| 0000000006 |
| 0000000000 |
+------------+
```

5.1.1.10 DEC, NUMERIC, FIXED

Syntax

```
DEC[(M[,D])] [SIGNED | UNSIGNED | ZEROFILL]

NUMERIC[(M[,D])] [SIGNED | UNSIGNED | ZEROFILL]

FIXED[(M[,D])] [SIGNED | UNSIGNED | ZEROFILL]
```

Description

These types are synonyms for DECIMAL. The `FIXED` synonym is available for compatibility with other database systems.

5.1.1.11 NUMBER

MariaDB starting with 10.3

```
NUMBER[(M[,D])] [SIGNED | UNSIGNED | ZEROFILL]
```

In Oracle mode from MariaDB 10.3, `NUMBER` is a synonym for DECIMAL.

5.1.1.12 FLOAT

Syntax

```
FLOAT[(M,D)] [SIGNED | UNSIGNED | ZEROFILL]
```

Contents

Description

A small (single-precision) floating-point number (see DOUBLE for a regular-size floating point number). Allowable values are:

- -3.402823466E+38 to -1.175494351E-38
- 0
- 1.175494351E-38 to 3.402823466E+38.

These are the theoretical limits, based on the IEEE standard. The actual range might be slightly smaller depending on your hardware or operating system.

M is the total number of digits and D is the number of digits following the decimal point. If M and D are omitted, values are stored to the limits allowed by the hardware. A single-precision floating-point number is accurate to approximately 7 decimal places.

UNSIGNED, if specified, disallows negative values.

Using FLOAT might give you some unexpected problems because all calculations in MariaDB are done with double precision. See Floating Point Accuracy.

For more details on the attributes, see Numeric Data Type Overview.

5.1.1.13 DOUBLE

Syntax

```
DOUBLE[(M,D)] [SIGNED | UNSIGNED | ZEROFILL]
DOUBLE PRECISION[(M,D)] [SIGNED | UNSIGNED | ZEROFILL]
REAL[(M,D)] [SIGNED | UNSIGNED | ZEROFILL]
```

Contents

Description

A normal-size (double-precision) floating-point number (see FLOAT for a single-precision floating-point number).

Allowable values are:

- `-1.7976931348623157E+308` *to* `-2.2250738585072014E-308`
- `0`
- `2.2250738585072014E-308` *to* `1.7976931348623157E+308`

These are the theoretical limits, based on the IEEE standard. The actual range might be slightly smaller depending on your hardware or operating system.

`M` is the total number of digits and `D` is the number of digits following the decimal point. If `M` and `D` are omitted, values are stored to the limits allowed by the hardware. A double-precision floating-point number is accurate to approximately 15

decimal places.

`UNSIGNED`, if specified, disallows negative values.

`ZEROFILL`, if specified, pads the number with zeros, up to the total number of digits specified by `M`.

REAL and DOUBLE PRECISION are synonyms, unless the REAL_AS_FLOAT SQL mode is enabled, in which case REAL is a synonym for FLOAT rather than DOUBLE.

See Floating Point Accuracy for issues when using floating-point numbers.

For more details on the attributes, see Numeric Data Type Overview.

Examples

```
CREATE TABLE t1 (d DOUBLE(5,0) zerofill);

INSERT INTO t1 VALUES (1),(2),(3),(4);

SELECT * FROM t1;
+-------+
| d     |
+-------+
| 00001 |
| 00002 |
| 00003 |
| 00004 |
+-------+
```

5.1.1.14 DOUBLE PRECISION

Syntax

```
DOUBLE PRECISION[(M,D)] [SIGNED | UNSIGNED | ZEROFILL]
REAL[(M,D)] [SIGNED | UNSIGNED | ZEROFILL]
```

Description

REAL and DOUBLE PRECISION are synonyms for DOUBLE.

Exception: If the REAL_AS_FLOAT SQL mode is enabled, REAL is a synonym for FLOAT rather than DOUBLE.

5.1.1.15 BIT

Syntax

```
BIT[(M)]
```

Description

A bit-field type. `M` indicates the number of bits per value, from `1` to `64`. The default is `1` if `M` is omitted.

Bit values can be inserted with `b'value'` notation, where `value` is the bit value in 0's and 1's.

Bit fields are automatically zero-padded from the left to the full length of the bit, so for example in a BIT(4) field, '10' is equivalent to '0010'.

Bits are returned as binary, so to display them, either add 0, or use a function such as HEX, OCT or BIN to convert them.

Examples

```
CREATE TABLE b ( b1 BIT(8) );
```

With strict_mode set, the default from MariaDB 10.2.4:

```
INSERT INTO b VALUES (b'11111111');

INSERT INTO b VALUES (b'01010101');

INSERT INTO b VALUES (b'1111111111111');
ERROR 1406 (22001): Data too long for column 'b1' at row 1

SELECT b1+0, HEX(b1), OCT(b1), BIN(b1) FROM b;
+------+---------+---------+----------+
| b1+0 | HEX(b1) | OCT(b1) | BIN(b1)  |
+------+---------+---------+----------+
|  255 | FF      | 377     | 11111111 |
|   85 | 55      | 125     | 1010101  |
+------+---------+---------+----------+
```

With strict_mode unset, the default until MariaDB 10.2.3:

```
INSERT INTO b VALUES (b'11111111'),(b'01010101'),(b'1111111111111');
Query OK, 3 rows affected, 1 warning (0.10 sec)
Records: 3  Duplicates: 0  Warnings: 1

SHOW WARNINGS;
+---------+------+-------------------------------------------------+
| Level   | Code | Message                                         |
+---------+------+-------------------------------------------------+
| Warning | 1264 | Out of range value for column 'b1' at row 3 |
+---------+------+-------------------------------------------------+

SELECT b1+0, HEX(b1), OCT(b1), BIN(b1) FROM b;
+------+---------+---------+----------+
| b1+0 | HEX(b1) | OCT(b1) | BIN(b1)  |
+------+---------+---------+----------+
|  255 | FF      | 377     | 11111111 |
|   85 | 55      | 125     | 1010101  |
|  255 | FF      | 377     | 11111111 |
+------+---------+---------+----------+
```

5.1.1.16 Floating-point Accuracy

Due to their nature, not all floating-point numbers can be stored with exact precision. Hardware architecture, the CPU or even the compiler version and optimization level may affect the precision.

If you are comparing DOUBLEs or FLOATs with numeric decimals, it is not safe to use the equality operator.

Sometimes, changing a floating-point number from single-precision (FLOAT) to double-precision (DOUBLE) will fix the problem.

Example

f1, f2 and f3 have seemingly identical values across each row, but due to floating point accuracy, the results may be unexpected.

```
CREATE TABLE fpn (id INT, f1 FLOAT, f2 DOUBLE, f3 DECIMAL (10,3));
INSERT INTO fpn VALUES (1,2,2,2),(2,0.1,0.1,0.1);

SELECT * FROM fpn WHERE f1*f1 = f2*f2;
+------+------+------+-------+
| id   | f1   | f2   | f3    |
+------+------+------+-------+
|    1 |    2 |    2 | 2.000 |
+------+------+------+-------+
```

The reason why only one instead of two rows was returned becomes clear when we see how the floating point squares were evaluated.

```
SELECT f1*f1, f2*f2, f3*f3 FROM fpn;
+----------------------+----------------------+----------+
| f1*f1                | f2*f2                | f3*f3    |
+----------------------+----------------------+----------+
|                    4 |                    4 | 4.000000 |
| 0.010000000298023226 | 0.010000000000000002 | 0.010000 |
+----------------------+----------------------+----------+
```

5.1.1.17 INT1

`INT1` is a synonym for TINYINT.

```
CREATE TABLE t1 (x INT1);

DESC t1;
+-------+------------+------+-----+---------+-------+
| Field | Type       | Null | Key | Default | Extra |
+-------+------------+------+-----+---------+-------+
| x     | tinyint(4) | YES  |     | NULL    |       |
+-------+------------+------+-----+---------+-------+
```

5.1.1.18 INT2

`INT2` is a synonym for SMALLINT.

```
CREATE TABLE t1 (x INT2);

DESC t1;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| x     | smallint(6) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
```

5.1.1.19 INT3

`INT3` is a synonym for MEDIUMINT.

```
CREATE TABLE t1 (x INT3);

DESC t1;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| x     | mediumint(9) | YES  |     | NULL    |       |
+-------+--------------+------+-----+---------+-------+
```

5.1.1.20 INT4

`INT4` is a synonym for INT.

```
CREATE TABLE t1 (x INT4);

DESC t1;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| x     | int(11) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
```

5.1.1.21 INT8

`INT8` is a synonym for BIGINT.

```
CREATE TABLE t1 (x INT8);

DESC t1;
+-------+------------+------+-----+---------+-------+
| Field | Type       | Null | Key | Default | Extra |
+-------+------------+------+-----+---------+-------+
| x     | bigint(20) | YES  |     | NULL    |       |
+-------+------------+------+-----+---------+-------+
```

5.1.2 String Data Types

String Literals

Strings are sequences of characters and are enclosed with quotes.

BINARY

Fixed-length binary byte string.

BLOB

Binary large object up to 65,535 bytes.

BLOB and TEXT Data Types

Binary large object data types and the corresponding TEXT types.

CHAR

Fixed-length string.

CHAR BYTE

Alias for BINARY.

ENUM

Enumeration, or string object that can have one value chosen from a list of values.

INET4

For storage of IPv4 addresses.

INET6

For storage of IPv6 addresses.

JSON Data Type

Compatibility data type that is an alias for LONGTEXT.

MEDIUMBLOB

Medium binary large object up to 16,777,215 bytes.

MEDIUMTEXT

A TEXT column with a maximum length of 16,777,215 characters.

LONGBLOB

Long BLOB holding up to 4GB.

LONG and LONG VARCHAR

LONG and LONG VARCHAR are synonyms for MEDIUMTEXT.

LONGTEXT

A TEXT column with a maximum length of 4,294,967,295 characters.

ROW

Data type for stored procedure variables.

TEXT
A TEXT column with a maximum length of 65,535 characters.

TINYBLOB
Tiny binary large object up to 255 bytes.

TINYTEXT
A TEXT column with a maximum length of 255 characters.

VARBINARY
Variable-length binary byte string.

VARCHAR
Variable-length string.

SET Data Type
Set, or string object that can have 0 or more values chosen from a list of values.

UUID Data Type
Data type intended for the storage of UUID data.

Data Type Storage Requirements
Storage requirements for the various data types.

Supported Character Sets and Collations
MariaDB supports the following character sets and collations.

Character Sets and Collations
Setting character set and collation for a language.

1.1.2.13 String Literals

5.1.2.2 BINARY

This page describes the BINARY data type. For details about the operator, see Binary Operator.

Syntax

```
BINARY(M)
```

Contents
1. Description
2. Examples

Description

The `BINARY` type is similar to the `CHAR` type, but stores binary byte strings rather than non-binary character strings. `M` represents the column length in bytes.

It contains no character set, and comparison and sorting are based on the numeric value of the bytes.

If the maximum length is exceeded, and SQL strict mode is not enabled , the extra characters will be dropped with a warning. If strict mode is enabled, an error will occur.

BINARY values are right-padded with `0x00` (the zero byte) to the specified length when inserted. The padding is *not* removed on select, so this needs to be taken into account when sorting and comparing, where all bytes are significant. The zero byte, `0x00` is less than a space for comparison purposes.

Examples

Inserting too many characters, first with strict mode off, then with it on:

```
CREATE TABLE bins (a BINARY(10));

INSERT INTO bins VALUES('12345678901');
Query OK, 1 row affected, 1 warning (0.04 sec)

SELECT * FROM bins;
+------------+
| a          |
+------------+
| 1234567890 |
+------------+

SET sql_mode='STRICT_ALL_TABLES';

INSERT INTO bins VALUES('12345678901');
ERROR 1406 (22001): Data too long for column 'a' at row 1
```

Sorting is performed with the byte value:

```
TRUNCATE bins;

INSERT INTO bins VALUES('A'),('B'),('a'),('b');

SELECT * FROM bins ORDER BY a;
+------+
| a    |
+------+
| A    |
| B    |
| a    |
| b    |
+------+
```

Using CAST to sort as a CHAR instead:

```
SELECT * FROM bins ORDER BY CAST(a AS CHAR);
+------+
| a    |
+------+
| a    |
| A    |
| b    |
| B    |
+------+
```

The field is a BINARY(10), so padding of two '\0's are inserted, causing comparisons that don't take this into account to fail:

```
TRUNCATE bins;

INSERT INTO bins VALUES('12345678');

SELECT a = '12345678', a = '12345678\0\0' from bins;
+----------------+--------------------+
| a = '12345678' | a = '12345678\0\0' |
+----------------+--------------------+
|              0 |                  1 |
+----------------+--------------------+
```

5.1.2.3 BLOB

Syntax

```
BLOB[(M)]
```

Contents

Description

A `BLOB` column with a maximum length of `65,535` ($2^{16} - 1$) bytes. Each `BLOB` value is stored using a two-byte length prefix that indicates the number of bytes in the value.

An optional length `M` can be given for this type. If this is done, MariaDB creates the column as the smallest `BLOB` type large enough to hold values *M* bytes long.

BLOBS can also be used to store dynamic columns.

`BLOB` and `TEXT` columns can both be assigned a DEFAULT value.

Indexing

MariaDB starting with 10.4

From MariaDB 10.4, it is possible to set a unique index on a column that uses the `BLOB` data type. In previous releases this was not possible, as the index would only guarantee the uniqueness of a fixed number of characters.

Oracle Mode

In Oracle mode from MariaDB 10.3, `BLOB` is a synonym for `LONGBLOB`.

5.1.2.4 BLOB and TEXT Data Types

Description

A `BLOB` is a binary large object that can hold a variable amount of data. The four `BLOB` types are

- TINYBLOB,
- BLOB,
- MEDIUMBLOB, and
- LONGBLOB.

These differ only in the maximum length of the values they can hold.

The `TEXT` types are

- TINYTEXT,
- TEXT,
- MEDIUMTEXT, and
- LONGTEXT.
- JSON (alias for LONGTEXT)

These correspond to the four `BLOB` types and have the same maximum lengths and storage requirements.

`BLOB` and `TEXT` columns can have a `DEFAULT` value.

MariaDB starting with 10.4.3

From MariaDB 10.4, it is possible to set a unique index on columns that use the `BLOB` or `TEXT` data types.

5.1.2.5 CHAR

This article covers the CHAR data type. See CHAR Function for the function.

Syntax

```
[NATIONAL] CHAR[(M)] [CHARACTER SET charset_name] [COLLATE collation_name]
```

Contents

Description

A fixed-length string that is always right-padded with spaces to the specified length when stored. `M` represents the column length in characters. The range of `M` is `0` to `255`. If `M` is omitted, the length is `1`.

CHAR(0) columns can contain 2 values: an empty string or NULL. Such columns cannot be part of an index. The CONNECT storage engine does not support CHAR(0).

Note: Trailing spaces are removed when `CHAR` values are retrieved unless the `PAD_CHAR_TO_FULL_LENGTH` SQL mode is enabled.

Before MariaDB 10.2, all collations were of type PADSPACE, meaning that CHAR (as well as VARCHAR and TEXT) values are compared without regard for trailing spaces. This does not apply to the LIKE pattern-matching operator, which takes into account trailing spaces.

If a unique index consists of a column where trailing pad characters are stripped or ignored, inserts into that column where values differ only by the number of trailing pad characters will result in a duplicate-key error.

Examples

Trailing spaces:

```
CREATE TABLE strtest (c CHAR(10));
INSERT INTO strtest VALUES('Maria   ');

SELECT c='Maria',c='Maria   ' FROM strtest;
+-----------+--------------+
| c='Maria' | c='Maria   ' |
+-----------+--------------+
|         1 |            1 |
+-----------+--------------+

SELECT c LIKE 'Maria',c LIKE 'Maria   ' FROM strtest;
+----------------+-------------------+
| c LIKE 'Maria' | c LIKE 'Maria   ' |
+----------------+-------------------+
|              1 |                 0 |
+----------------+-------------------+
```

NO PAD Collations

NO PAD collations regard trailing spaces as normal characters. You can get a list of all NO PAD collations by querying the Information Schema Collations table, for example:

```
SELECT collation_name FROM information_schema.collations
  WHERE collation_name LIKE "%nopad%";
+------------------------------+
| collation_name               |
+------------------------------+
| big5_chinese_nopad_ci        |
| big5_nopad_bin               |
...
```

5.1.2.6 CHAR BYTE

Description

The `CHAR BYTE` data type is an alias for the `BINARY` data type. This is a compatibility feature.

5.1.2.7 ENUM

Syntax

```
ENUM('value1','value2',...) [CHARACTER SET charset_name] [COLLATE collation_name]
```

Contents

Description

An enumeration. A string object that can have only one value, chosen from the list of values 'value1', 'value2', ..., NULL or the special '' error value. In theory, an `ENUM` column can have a maximum of 65,535 distinct values; in practice, the real maximum depends on many factors. `ENUM` values are represented internally as integers.

Trailing spaces are automatically stripped from ENUM values on table creation.

ENUMs require relatively little storage space compared to strings, either one or two bytes depending on the number of enumeration values.

NULL and empty values

An ENUM can also contain NULL and empty values. If the ENUM column is declared to permit NULL values, NULL becomes a valid value, as well as the default value (see below). If strict SQL Mode is not enabled, and an invalid value is inserted into an ENUM, a special empty string, with an index value of zero (see Numeric index, below), is inserted, with a warning. This may be confusing, because the empty string is also a possible value, and the only difference if that is this case its index is not 0. Inserting will fail with an error if strict mode is active.

If a `DEFAULT` clause is missing, the default value will be:

- `NULL` if the column is nullable;
- otherwise, the first value in the enumeration.

Numeric index

ENUM values are indexed numerically in the order they are defined, and sorting will be performed in this numeric order. We suggest not using ENUM to store numerals, as there is little to no storage space benefit, and it is easy to confuse the enum integer with the enum numeral value by leaving out the quotes.

An ENUM defined as ENUM('apple','orange','pear') would have the following index values:

Index	Value
NULL	NULL
0	''
1	'apple'
2	'orange'
3	'pear'

Examples

```
CREATE TABLE fruits (
  id INT NOT NULL auto_increment PRIMARY KEY,
  fruit ENUM('apple','orange','pear'),
  bushels INT);

DESCRIBE fruits;
+---------+-------------------------------+------+-----+---------+----------------+
| Field   | Type                          | Null | Key | Default | Extra          |
+---------+-------------------------------+------+-----+---------+----------------+
| id      | int(11)                       | NO   | PRI | NULL    | auto_increment |
| fruit   | enum('apple','orange','pear') | YES  |     | NULL    |                |
| bushels | int(11)                       | YES  |     | NULL    |                |
+---------+-------------------------------+------+-----+---------+----------------+

INSERT INTO fruits
    (fruit,bushels) VALUES
    ('pear',20),
    ('apple',100),
    ('orange',25);

INSERT INTO fruits
    (fruit,bushels) VALUES
    ('avocado',10);
ERROR 1265 (01000): Data truncated for column 'fruit' at row 1

SELECT * FROM fruits;
+----+--------+---------+
| id | fruit  | bushels |
+----+--------+---------+
|  1 | pear   |      20 |
|  2 | apple  |     100 |
|  3 | orange |      25 |
+----+--------+---------+
```

Selecting by numeric index:

```
SELECT * FROM fruits WHERE fruit=2;
+----+--------+---------+
| id | fruit  | bushels |
+----+--------+---------+
|  3 | orange |      25 |
+----+--------+---------+
```

Sorting is according to the index value:

```
CREATE TABLE enums (a ENUM('2','1'));

INSERT INTO enums VALUES ('1'),('2');

SELECT * FROM enums ORDER BY a ASC;
+------+
| a    |
+------+
| 2    |
| 1    |
+------+
```

It's easy to get confused between returning the enum integer with the stored value, so we don't suggest using ENUM to store numerals. The first example returns the 1st indexed field ('2' has an index value of 1, as it's defined first), while the second example returns the string value '1'.

```
SELECT * FROM enums WHERE a=1;
+------+
| a    |
+------+
| 2    |
+------+

SELECT * FROM enums WHERE a='1';
+------+
| a    |
+------+
| 1    |
+------+
```

5.1.2.8 INET4

MariaDB starting with 10.10.0
The INET4 data type was added in MariaDB 10.10.0

Syntax

```
INET4
```

Contents

Description

`INET4` is a data type to store IPv4 addresses, as 4-byte binary strings.

From MariaDB 11.3.0, casting from INET4 data types to INET6 is permitted, allowing for example comparisons between the two data types, and for INET 4 values to be inserted into INET6 columns.

Examples

```
CREATE OR REPLACE TABLE t1 (a INET4);

INSERT INTO t1 VALUES('0.0.0.0'), ('255.10.0.0'), ('255.255.255.255');

INSERT INTO t1 VALUES (0xa0000001);
INSERT INTO t1 VALUES (0xf0000000);
INSERT INTO t1 VALUES (0xff000001);

SELECT HEX(a), a FROM t1 ORDER BY a;
+----------+-----------------+
| HEX(a)   | a               |
+----------+-----------------+
| 00000000 | 0.0.0.0         |
| A0000001 | 160.0.0.1       |
| F0000000 | 240.0.0.0       |
| FF000001 | 255.0.0.1       |
| FF0A0000 | 255.10.0.0      |
| FFFFFFFF | 255.255.255.255 |
+----------+-----------------+
```

Casting from INET4 to INET6 is permitted, allowing direct inserts.

Before MariaDB 11.3:

```
CREATE TABLE t1 (a INET6);

INSERT INTO t1 VALUES('0.0.0.0'), ('255.10.0.0'), ('255.255.255.255');
ERROR 1292 (22007): Incorrect inet6 value: '0.0.0.0' for column `test`.`t1`.`a` at row 1
```

From MariaDB 11.3:

```
CREATE TABLE t1 (a INET6);

INSERT INTO t1 VALUES('0.0.0.0'), ('255.10.0.0'), ('255.255.255.255');
Query OK, 3 rows affected (0.027 sec)
```

Comparisons are also permitted from MariaDB 11.3:

```
CREATE OR REPLACE TABLE t1 (i4 INET4, i6 INET6);
INSERT INTO t1 VALUES('10.10.10.10','::ffff:192.168.0.1');

SELECT LEAST(i4,i6) FROM t1;
+--------------------+
| LEAST(i4,i6)       |
+--------------------+
| ::ffff:10.10.10.10 |
+--------------------+
```

5.1.2.9 INET6

MariaDB starting with 10.5.0

The INET6 data type was added in MariaDB 10.5.0

Syntax

```
INET6
```

Contents

Description

The `INET6` data type is intended for storage of IPv6 addresses, as well as IPv4 addresses assuming conventional mapping of IPv4 addresses into IPv6 addresses.

Both short and long IPv6 notation are permitted, according to RFC-5952.

- Values are stored as a 16-byte fixed length binary string, with most significant byte first.
- Storage engines see INET6 as BINARY(16).
- Clients see INET6 as CHAR(39) and get text representation on retrieval.

The IPv4-compatible notation is considered as deprecated. It is supported for compatibility with the INET6_ATON function,

which also understands this format. It's recommended to use the mapped format to store IPv4 addresses in INET6.

When an IPv4 mapped (or compatible) value is stored in INET6, it still occupies 16 bytes:

Retrieval

On retrieval, in the client-server text protocol, INET6 values are converted to the short text representation, according to RFC-5952, that is with all leading zeroes in each group removed and with consequent zero groups compressed.

Besides creating one's own stored function, there is no a way to retrieve an INET6 value using long text representation.

Casting

- CAST from a character string to INET6 understands addresses in short or long text notation (including IPv4 mapped and compatible addresses). NULL is returned if the format is not understood.
- CAST from a binary string to INET6 requires a 16-byte string as an argument. NULL is returned if the argument length is not equal to 16.
- CAST from other data types to INET6 first converts data to a character string, then CAST from character string to INET6 is applied.
- CAST from INET6 to CHAR returns short text address notation.
- CAST from INET6 to BINARY returns its 16-byte binary string representation.
- CAST from INET6 to data types other than CHAR (e.g. SIGNED, UNSIGNED, TIME, etc) returns an error.

Comparisons

An INET6 expression can be compared to:

- another INET6 expression
- a character string expression with a text (short or long) address representation:
- a 16-byte binary string expression:

Attempting to compare INET6 to an expression of any other data type returns an error.

Mixing INET6 Values for Result

An INET6 expression can be mixed for result (i.e. UNION, CASE..THEN, COALESCE etc) with:

- another INET6 expression. The resulting data type is INET6.
- a character string in text (short or long) address representation. The result data type is INET6. The character string counterpart is automatically converted to INET6. If the string format is not understood, it's converted with a warning to either NULL or to '::', depending on the NULL-ability of the result.
- a 16-byte binary string. The resulting data type is INET6. The binary string counterpart is automatically converted to INET6. If the length of the binary string is not equal to 16, it's converted with a warning to NULL or to '::' depending on the NULL-ability of the result.

Attempts to mix INET6 for result with other data types will return an error.

Mixing INET6 with other data types for LEAST and GREATEST, when mixing for comparison and mixing for result are involved at the same time, uses the same rules with mixing for result, described in the previous paragraphs.

Functions and Operators

- HEX() with an INET6 argument returns a hexadecimal representation of the underlying 16-byte binary string
- Arithmetic operators (+,-,*,/,MOD,DIV) are not supported for INET6. This may change in the future.
- The INET6_ATON function now understands INET6 values as an argument
- The prototypes of the IS_IPV4_COMPAT and IS_IPV4_MAPPED functions have changed from `a BINARY(16)` to `a INET6`,
- When the argument for these two functions is not INET6, automatic implicit CAST to INET6 is applied. As a consequence, both functions now understand arguments in both text representation and binary(16) representation. Before MariaDB 10.5.0, these functions understood only binary(16) representation.

Prepared Statement Parameters

INET6 understands both text and binary(16) address representation in prepared statement parameters (PREPARE..EXECUTE and EXECUTE IMMEDIATE statements).

Migration between BINARY(16) and INET6

Before MariaDB 10.5.0, you may have used BINARY(16) as a storage for IPv6 internet addresses, in combination with INET6_ATON and INET6_NTOA to respectively insert and retrieve data.

From 10.5, you can ALTER BINARY(16) columns storing IPv6 addresses to INET6. After such an alter, there is no a need to use INET6_ATON() and INET6_NTOA(). Addresses can be inserted and retrieved directly.

It is also possible to convert INET6 columns to BINARY(16) and continue using the data in combination with INET6_NTOA() and INET6_ATON().

Examples

```
CREATE TABLE t1 (a INET6);
```

Inserting using short text address notation:

```
INSERT INTO t1 VALUES ('2001:db8::ff00:42:8329');
```

Long text address notation:

```
INSERT INTO t1 VALUES ('2001:0db8:0000:0000:0000:ff00:0042:8329');
```

16-byte binary string notation:

```
INSERT INTO t1 VALUES (0x20010DB8000000000000FF0000428329);
INSERT INTO t1 VALUES (UNHEX('20010DB8000000000000FF0000428329'));
```

IPv4 addresses, using IPv4-mapped and IPv4-compatible notations:

```
INSERT INTO t1 VALUES ('::ffff:192.0.2.128'); -- mapped
INSERT INTO t1 VALUES ('::192.0.2.128'); -- compatible
```

```
SELECT * FROM t1;
+-------------------------+
| a                       |
+-------------------------+
| 2001:db8::ff00:42:8329  |
| 2001:db8::ff00:42:8329  |
| 2001:db8::ff00:42:8329  |
| 2001:db8::ff00:42:8329  |
| ::ffff:192.0.2.128      |
| ::192.0.2.128           |
+-------------------------+
```

IPv4 mapped (or compatible) values still occupy 16 bytes:

```
CREATE OR REPLACE TABLE t1 (a INET6);

INSERT INTO t1 VALUES ('::ffff:192.0.2.128');

SELECT * FROM t1;
+--------------------+
| a                  |
+--------------------+
| ::ffff:192.0.2.128 |
+--------------------+

SELECT HEX(a) FROM t1;
+----------------------------------+
| HEX(a)                           |
+----------------------------------+
| 00000000000000000000FFFFC0000280 |
+----------------------------------+
```

Casting from INET6 to anything other than CHAR returns an error:

```
SELECT CAST(a AS DECIMAL) FROM t1;

ERROR 4079 (HY000): Illegal parameter data type inet6 for operation 'decimal_typecast'
```

Comparison Examples

Comparison with another INET6 expression:

```
    CREATE OR REPLACE TABLE t1 (a INET6);
    CREATE OR REPLACE TABLE t2 (a INET6);

    INSERT INTO t1 VALUES ('2001:db8::ff00:42:8328'),('2001:db8::ff00:42:8329');
    INSERT INTO t2 VALUES ('2001:db8::ff00:42:832a'),('2001:db8::ff00:42:8329');

    SELECT t1.* FROM t1,t2 WHERE t1.a=t2.a;
    +------------------------+
    | a                      |
    +------------------------+
    | 2001:db8::ff00:42:8329 |
    +------------------------+
```

With a character string expression with a text (short or long) address representation:

```
    CREATE OR REPLACE TABLE t1 (a INET6);

    INSERT INTO t1 VALUES ('2001:db8::ff00:42:8329');

    SELECT * FROM t1 WHERE a='2001:db8::ff00:42:8329';
    +------------------------+
    | a                      |
    +------------------------+
    | 2001:db8::ff00:42:8329 |
    +------------------------+
```

With a 16-byte binary string expression:

```
    CREATE OR REPLACE TABLE t1 (a INET6);

    INSERT INTO t1 VALUES ('2001:db8::ff00:42:8329');

    SELECT * FROM t1 WHERE a=X'20010DB8000000000000FF0000428329';
    +------------------------+
    | a                      |
    +------------------------+
    | 2001:db8::ff00:42:8329 |
    +------------------------+
```

With an expression of another data type:

```
SELECT * FROM t1 WHERE a=1;
ERROR 4078 (HY000): Illegal parameter data types inet6 and int for operation '='
```

Mixing for Result Examples

Mixed with another INET6 expression, returning an INET6 data type:

```
CREATE OR REPLACE TABLE t1 (a INET6, b INET6);

INSERT INTO t1 VALUES (NULL,'2001:db8::ff00:42:8329');

SELECT a FROM t1 UNION SELECT b FROM t1;
+------------------------+
| a                      |
+------------------------+
| NULL                   |
| 2001:db8::ff00:42:8329 |
+------------------------+

SELECT COALESCE(a, b) FROM t1;
+------------------------+
| COALESCE(a, b)         |
+------------------------+
| 2001:db8::ff00:42:8329 |
+------------------------+
```

Mixed with a character string in text (short or long) address representation:

```
CREATE OR REPLACE TABLE t1 (a INET6, b VARCHAR(64));

INSERT INTO t1 VALUES (NULL,'2001:db8::ff00:42:8328');

INSERT INTO t1 VALUES (NULL,'2001:db8::ff00:42:832a garbage');

SELECT COALESCE(a,b) FROM t1;
+------------------------+
| COALESCE(a,b)          |
+------------------------+
| 2001:db8::ff00:42:8328 |
| NULL                   |
+------------------------+
2 rows in set, 1 warning (0.001 sec)

SHOW WARNINGS;
+---------+------+-----------------------------------------------------------+
| Level   | Code | Message                                                   |
+---------+------+-----------------------------------------------------------+
| Warning | 1292 | Incorrect inet6 value: '2001:db8::ff00:42:832a garbage' |
+---------+------+-----------------------------------------------------------+
```

Mixed with a 16-byte binary string:

```
CREATE OR REPLACE TABLE t1 (a INET6, b VARBINARY(16));

INSERT INTO t1 VALUES (NULL,CONCAT(0xFFFF,REPEAT(0x0000,6),0xFFFF));

INSERT INTO t1 VALUES (NULL,0x00/*garbage*/);

SELECT COALESCE(a,b) FROM t1;
+---------------+
| COALESCE(a,b) |
+---------------+
| ffff::ffff    |
| NULL          |
+---------------+
2 rows in set, 1 warning (0.001 sec)

SHOW WARNINGS;
+---------+------+-------------------------------+
| Level   | Code | Message                       |
+---------+------+-------------------------------+
| Warning | 1292 | Incorrect inet6 value: '\x00' |
+---------+------+-------------------------------+
```

Mixing with other data types:

```
SELECT CAST('ffff::ffff' AS INET6) UNION SELECT 1;
ERROR 4078 (HY000): Illegal parameter data types inet6 and int for operation 'UNION'
```

Functions and Operators Examples

HEX with an INET6 argument returning a hexadecimal representation:

```
SELECT HEX(CAST('2001:db8::ff00:42:8329' AS INET6));
+--------------------------------------------+
| HEX(CAST('2001:db8::ff00:42:8329' AS INET6)) |
+--------------------------------------------+
| 20010DB8000000000000FF0000428329           |
+--------------------------------------------+
```

INET6_ATON now understands INET6 values as an argument:

```
CREATE OR REPLACE TABLE t1 (a INET6);

INSERT INTO t1 VALUES ('2001:db8::ff00:42:8329');

SELECT a, HEX(INET6_ATON(a)) FROM t1;
+-----------------------+----------------------------------+
| a                     | HEX(INET6_ATON(a))               |
+-----------------------+----------------------------------+
| 2001:db8::ff00:42:8329 | 20010DB8000000000000FF0000428329 |
+-----------------------+----------------------------------+
```

IS_IPV4_COMPAT and IS_IPV4_MAPPED prototype now `a BINARY(16))` :

```
CREATE OR REPLACE TABLE t1 (a INET6);

INSERT INTO t1 VALUES ('2001:db8::ff00:42:8329');
INSERT INTO t1 VALUES ('::ffff:192.168.0.1');
INSERT INTO t1 VALUES ('::192.168.0.1');

SELECT a, IS_IPV4_MAPPED(a), IS_IPV4_COMPAT(a) FROM t1;
+-----------------------+------------------+------------------+
| a                     | IS_IPV4_MAPPED(a) | IS_IPV4_COMPAT(a) |
+-----------------------+------------------+------------------+
| 2001:db8::ff00:42:8329 |                0 |                0 |
| ::ffff:192.168.0.1    |                1 |                0 |
| ::192.168.0.1         |                0 |                1 |
+-----------------------+------------------+------------------+
```

Automatic implicit CAST to INET6:

```
CREATE OR REPLACE TABLE t1 (
  a INET6,
  b VARCHAR(39) DEFAULT a
);

INSERT INTO t1 (a) VALUES ('ffff::ffff'),('::ffff:192.168.0.1');

SELECT a, IS_IPV4_MAPPED(a), b, IS_IPV4_MAPPED(b) FROM t1;
+-------------------+-------------------+-------------------+-------------------+
| a                 | IS_IPV4_MAPPED(a) | b                 | IS_IPV4_MAPPED(b) |
+-------------------+-------------------+-------------------+-------------------+
| ffff::ffff        |                 0 | ffff::ffff        |                 0 |
| ::ffff:192.168.0.1 |                 1 | ::ffff:192.168.0.1 |                 1 |
+-------------------+-------------------+-------------------+-------------------+

CREATE OR REPLACE TABLE t1 (
  a INET6,
  b BINARY(16) DEFAULT UNHEX(HEX(a))
);

INSERT INTO t1 (a) VALUES ('ffff::ffff'),('::ffff:192.168.0.1');

SELECT a, IS_IPV4_MAPPED(a), HEX(b), IS_IPV4_MAPPED(b) FROM t1;
+-------------------+-------------------+----------------------------------+----------------
| a                 | IS_IPV4_MAPPED(a) | HEX(b)                           | IS_IPV4_MAPPED(
+-------------------+-------------------+----------------------------------+----------------
| ffff::ffff        |                 0 | FFFF00000000000000000000FFFF     |
| ::ffff:192.168.0.1 |                 1 | 00000000000000000000FFFFC0A80001 |
+-------------------+-------------------+----------------------------------+----------------
```

Prepared Statement Parameters Examples

```
CREATE OR REPLACE TABLE t1 (a INET6);

EXECUTE IMMEDIATE 'INSERT INTO t1 VALUES (?)' USING 'ffff::fffe';
EXECUTE IMMEDIATE 'INSERT INTO t1 VALUES (?)' USING X'FFFF00000000000000000000FFFF';

SELECT * FROM t1;
+------------+
| a          |
+------------+
| ffff::fffe |
| ffff::ffff |
+------------+

EXECUTE IMMEDIATE 'SELECT * FROM t1 WHERE a=?' USING 'ffff::fffe';
+------------+
| a          |
+------------+
| ffff::fffe |
+------------+

EXECUTE IMMEDIATE 'SELECT * FROM t1 WHERE a=?' USING X'FFFF00000000000000000000FFFF';
+------------+
| a          |
+------------+
| ffff::ffff |
+------------+
```

Migration between BINARY(16) and INET6 Examples

Before MariaDB 10.5:

```
CREATE OR REPLACE TABLE t1 (a BINARY(16));

INSERT INTO t1 VALUES (INET6_ATON('ffff::ffff'));

SELECT INET6_NTOA(a) FROM t1;
+---------------+
| INET6_NTOA(a) |
+---------------+
| ffff::ffff    |
+---------------+
```

Migrating to INET6, from MariaDB 10.5:

```
ALTER TABLE t1 MODIFY a INET6;

INSERT INTO t1 VALUES ('ffff::fffe');

SELECT * FROM t1;
+------------+
| a          |
+------------+
| ffff::ffff |
| ffff::fffe |
+------------+
```

Migration from INET6 to BINARY(16):

```
CREATE OR REPLACE TABLE t1 (a INET6);

INSERT INTO t1 VALUES ('2001:db8::ff00:42:8329');
INSERT INTO t1 VALUES ('::ffff:192.168.0.1');
INSERT INTO t1 VALUES ('::192.168.0.1');

ALTER TABLE t1 MODIFY a BINARY(16);

SELECT INET6_NTOA(a) FROM t1;
+------------------------+
| INET6_NTOA(a)          |
+------------------------+
| 2001:db8::ff00:42:8329 |
| ::ffff:192.168.0.1     |
| ::192.168.0.1          |
+------------------------+
```

Casting from INET4 to INET6

From MariaDB 11.3.0, casting from INET4 data types to INET6 is permitted, allowing INET4 values to be inserted into INET6 columns.

Before MariaDB 11.3:

```
CREATE TABLE t1 (a INET6);

INSERT INTO t1 VALUES('0.0.0.0'), ('255.10.0.0'), ('255.255.255.255');
ERROR 1292 (22007): Incorrect inet6 value: '0.0.0.0' for column `test`.`t1`.`a` at row 1
```

From MariaDB 11.3:

```
CREATE TABLE t1 (a INET6);

INSERT INTO t1 VALUES('0.0.0.0'), ('255.10.0.0'), ('255.255.255.255');
Query OK, 3 rows affected (0.027 sec)
```

5.1.2.10 JSON Data Type

The JSON alias was added to make it possible to use JSON columns in statement based replication from MySQL to MariaDB and to make it possible for MariaDB to read mysqldumps from MySQL.

Contents

JSON is an alias for LONGTEXT introduced for compatibility reasons with MySQL's JSON data type. MariaDB implements this as a LONGTEXT rather, as the JSON data type contradicts the SQL standard, and MariaDB's benchmarks indicate that performance is at least equivalent.

In order to ensure that a a valid json document is inserted, the JSON_VALID function can be used as a CHECK constraint. This constraint is automatically included for types using the JSON alias from MariaDB 10.4.3.

Examples

```
CREATE TABLE t (j JSON);

DESC t;
+-------+----------+------+-----+---------+-------+
| Field | Type     | Null | Key | Default | Extra |
+-------+----------+------+-----+---------+-------+
| j     | longtext | YES  |     | NULL    |       |
+-------+----------+------+-----+---------+-------+
```

With validation:

```
CREATE TABLE t2 (
  j JSON
  CHECK (JSON_VALID(j))
);

INSERT INTO t2 VALUES ('invalid');
ERROR 4025 (23000): CONSTRAINT `j` failed for `test`.`t2`

INSERT INTO t2 VALUES ('{"id": 1, "name": "Monty"}');
Query OK, 1 row affected (0.13 sec)
```

Replicating JSON Data Between MySQL and MariaDB

The JSON type in MySQL stores the JSON object in a compact form, not as LONGTEXT as in MariaDB. This means that row based replication will not work for JSON types from MySQL to MariaDB.

There are a a few different ways to solve this:

- Use statement based replication.
- Change the JSON column to type TEXT in MySQL
- If you must use row-based replication and cannot change the MySQL master from JSON to TEXT, you can try to introduce an intermediate MySQL slave and change the column type from JSON to TEXT on it. Then you replicate from this intermediate slave to MariaDB.

Converting a MySQL TABLE with JSON Fields to MariaDB

MariaDB can't directly access MySQL's JSON format.

There are a a few different ways to move the table to MariaDB:

- From MariaDB 10.5.7, see the you can use the mysql_json plugin. See Making MariaDB understand MySQL JSON.
- Change the JSON column to type TEXT in MySQL. After this, MariaDB can directly use the table without any need for a dump and restore.
- Use mysqldump to copy the table.

Differences Between MySQL JSON Strings and MariaDB JSON Strings

- In MySQL, JSON is an object and is compared according to json values. In MariaDB JSON strings are normal strings and compared as strings. One exception is when using JSON_EXTRACT() in which case strings are unescaped before comparison.

5.1.2.11 MEDIUMBLOB

Syntax

```
MEDIUMBLOB
```

Description

A BLOB column with a maximum length of 16,777,215 (2^{24} - 1) bytes. Each MEDIUMBLOB value is stored using a three-byte length prefix that indicates the number of bytes in the value.

5.1.2.12 MEDIUMTEXT

Syntax

```
MEDIUMTEXT [CHARACTER SET charset_name] [COLLATE collation_name]
```

Description

A TEXT column with a maximum length of 16,777,215 ($2^{24} - 1$) characters. The effective maximum length is less if the value contains multi-byte characters. Each MEDIUMTEXT value is stored using a three-byte length prefix that indicates the number of bytes in the value.

5.1.2.13 LONGBLOB

Syntax

```
LONGBLOB
```

Description

A BLOB column with a maximum length of 4,294,967,295 bytes or 4GB (2^{32} - 1). The effective maximum length of LONGBLOB columns depends on the configured maximum packet size in the client/server protocol and available memory. Each LONGBLOB value is stored using a four-byte length prefix that indicates the number of bytes in the value.

Oracle Mode

In Oracle mode from MariaDB 10.3, `BLOB` is a synonym for `LONGBLOB`.

5.1.2.14 LONG and LONG VARCHAR

`LONG` and `LONG VARCHAR` are synonyms for MEDIUMTEXT.

```
CREATE TABLE t1 (a LONG, b LONG VARCHAR);

DESC t1;
+-------+------------+------+-----+---------+-------+
| Field | Type       | Null | Key | Default | Extra |
+-------+------------+------+-----+---------+-------+
| a     | mediumtext | YES  |     | NULL    |       |
| b     | mediumtext | YES  |     | NULL    |       |
+-------+------------+------+-----+---------+-------+
```

5.1.2.15 LONGTEXT

Syntax

```
LONGTEXT [CHARACTER SET charset_name] [COLLATE collation_name]
```

Contents

Description

A TEXT column with a maximum length of 4,294,967,295 or 4GB ($2^{32} - 1$) characters. The effective maximum length is less if the value contains multi-byte characters. The effective maximum length of LONGTEXT columns also depends on the configured maximum packet size in the client/server protocol and available memory. Each LONGTEXT value is stored using a four-byte length prefix that indicates the number of bytes in the value.

From MariaDB 10.2.7, JSON is an alias for LONGTEXT. See JSON Data Type for details.

Oracle Mode

In Oracle mode from MariaDB 10.3, `CLOB` is a synonym for `LONGTEXT`.

5.1.2.16 ROW

Syntax

```
ROW (<field name> <data type> [{, <field name> <data type>}... ])
```

Contents

Description

`ROW` is a data type for stored procedure variables.

Features

ROW fields as normal variables

`ROW` fields (members) act as normal variables, and are able to appear in all query parts where a stored procedure variable is allowed:

- Assignment is using the `:=` operator and the SET command:

```
a.x:= 10;
a.x:= b.x;
SET a.x= 10, a.y=20, a.z= b.z;
```

- Passing to functions and operators:

```
SELECT f1(rec.a), rec.a<10;
```

- Clauses (select list, WHERE, HAVING, LIMIT, etc...,):

```
SELECT var.a, t1.b FROM t1 WHERE t1.b=var.b LIMIT var.c;
```

- `INSERT` values:

```
INSERT INTO t1 VALUES (rec.a, rec.b, rec.c);
```

- `SELECT .. INTO` targets

```
SELECT a,b INTO rec.a, rec.b FROM t1 WHERE t1.id=10;
```

- Dynamic SQL out parameters (EXECUTE and EXECUTE IMMEDIATE)

```
EXECUTE IMMEDIATE 'CALL proc_with_out_param(?)' USING rec.a;
```

ROW type variables as FETCH targets

`ROW` type variables are allowed as FETCH targets:

```
FETCH cur INTO rec;
```

where `cur` is a `CURSOR` and `rec` is a `ROW` type stored procedure variable.

Note, currently an attempt to use `FETCH` for a `ROW` type variable returns this error:

```
ERROR 1328 (HY000): Incorrect number of FETCH variables
```

`FETCH` from a cursor `cur` into a `ROW` variable `rec` works as follows:

- The number of fields in `cur` must match the number of fields in `rec`. Otherwise, an error is reported.
- Assignment is done from left to right. The first cursor field is assigned to the first variable field, the second cursor field is assigned to the second variable field, etc.
- Field names in `rec` are not important and can differ from field names in `cur`.

See FETCH Examples (below) for examples of using this with `sql_mode=ORACLE` and `sql_mode=DEFAULT`.

ROW type variables as `SELECT...INTO` targets

`ROW` type variables are allowed as `SELECT..INTO` targets with some differences depending on which `sql_mode` is in use.

- When using `sql_mode=ORACLE`, `table%ROWTYPE` and `cursor%ROWTYPE` variables can be used as `SELECT...INTO` targets.

- Using multiple `ROW` variables in the `SELECT..INTO` list will report an error.
- Using `ROW` variables with a different column count than in the `SELECT..INTO` list will report an error.

See SELECT...INTO Examples (below) for examples of using this with `sql_mode=ORACLE` and `sql_mode=DEFAULT`.

Features not implemented

The following features are planned, but not implemented yet:

- Returning a ROW type expression from a stored function (see MDEV-12252). This will need some grammar change to support field names after parentheses:

```
SELECT f1().x FROM DUAL;
```

- Returning a ROW type expression from a built-in hybrid type function, such as `CASE`, `IF`, etc.
- ROW of ROWs

Examples

Declaring a ROW in a stored procedure

```
DELIMITER $$
CREATE PROCEDURE p1()
BEGIN
  DECLARE r ROW (c1 INT, c2 VARCHAR(10));
  SET r.c1= 10;
  SET r.c2= 'test';
  INSERT INTO t1 VALUES (r.c1, r.c2);
END;
$$
DELIMITER ;
CALL p1();
```

FETCH Examples

A complete `FETCH` example for `sql_mode=ORACLE`:

```
DROP TABLE IF EXISTS t1;
CREATE TABLE t1 (a INT, b VARCHAR(32));
INSERT INTO t1 VALUES (10,'b10');
INSERT INTO t1 VALUES (20,'b20');
INSERT INTO t1 VALUES (30,'b30');

SET sql_mode=oracle;
DROP PROCEDURE IF EXISTS p1;
DELIMITER $$
CREATE PROCEDURE p1 AS
  rec ROW(a INT, b VARCHAR(32));
  CURSOR c IS SELECT a,b FROM t1;
BEGIN
  OPEN c;
  LOOP
    FETCH c INTO rec;
    EXIT WHEN c%NOTFOUND;
    SELECT ('rec=(' || rec.a ||','|| rec.b||')');
  END LOOP;
  CLOSE c;
END;
$$
DELIMITER ;
CALL p1();
```

A complete `FETCH` example for `sql_mode=DEFAULT`:

```
DROP TABLE IF EXISTS t1;
CREATE TABLE t1 (a INT, b VARCHAR(32));
INSERT INTO t1 VALUES (10,'b10');
INSERT INTO t1 VALUES (20,'b20');
INSERT INTO t1 VALUES (30,'b30');

SET sql_mode=DEFAULT;
DROP PROCEDURE IF EXISTS p1;
DELIMITER $$
CREATE PROCEDURE p1()
BEGIN
  DECLARE done INT DEFAULT FALSE;
  DECLARE rec ROW(a INT, b VARCHAR(32));
  DECLARE c CURSOR FOR SELECT a,b FROM t1;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
  OPEN c;
read_loop:
  LOOP
    FETCH c INTO rec;
    IF done THEN
      LEAVE read_loop;
    END IF;
    SELECT CONCAT('rec=(',rec.a,',',rec.b,')');
  END LOOP;
  CLOSE c;
END;
$$
DELIMITER ;
CALL p1();
```

SELECT...INTO Examples

A `SELECT...INTO` example for `sql_mode=DEFAULT`:

```
SET sql_mode=DEFAULT;
DROP TABLE IF EXISTS t1;
DROP PROCEDURE IF EXISTS p1;
CREATE TABLE t1 (a INT, b VARCHAR(32));
INSERT INTO t1 VALUES (10,'b10');
DELIMITER $$
CREATE PROCEDURE p1()
BEGIN
  DECLARE rec1 ROW(a INT, b VARCHAR(32));
  SELECT * FROM t1 INTO rec1;
  SELECT rec1.a, rec1.b;
END;
$$
DELIMITER ;
CALL p1();
```

The above example returns:

```
+--------+--------+
| rec1.a | rec1.b |
+--------+--------+
|     10 | b10    |
+--------+--------+
```

A `SELECT...INTO` example for `sql_mode=ORACLE`:

```
SET sql_mode=ORACLE;
DROP TABLE IF EXISTS t1;
DROP PROCEDURE IF EXISTS p1;
CREATE TABLE t1 (a INT, b VARCHAR(32));
INSERT INTO t1 VALUES (10,'b10');
DELIMITER $$
CREATE PROCEDURE p1 AS
  rec1 ROW(a INT, b VARCHAR(32));
BEGIN
  SELECT * FROM t1 INTO rec1;
  SELECT rec1.a, rec1.b;
END;
$$
DELIMITER ;
CALL p1();
```

The above example returns:

```
+--------+--------+
| rec1.a | rec1.b |
+--------+--------+
|     10 | b10    |
+--------+--------+
```

An example for `sql_mode=ORACLE` using `table%ROWTYPE` variables as `SELECT..INTO` targets:

```
SET sql_mode=ORACLE;
DROP TABLE IF EXISTS t1;
DROP PROCEDURE IF EXISTS p1;
CREATE TABLE t1 (a INT, b VARCHAR(32));
INSERT INTO t1 VALUES (10,'b10');
DELIMITER $$
CREATE PROCEDURE p1 AS
  rec1 t1%ROWTYPE;
BEGIN
  SELECT * FROM t1 INTO rec1;
  SELECT rec1.a, rec1.b;
END;
$$
DELIMITER ;
CALL p1();
```

The above example returns:

```
+--------+--------+
| rec1.a | rec1.b |
+--------+--------+
|     10 | b10    |
+--------+--------+
```

An example for `sql_mode=ORACLE` using `cursor%ROWTYPE` variables as `SELECT..INTO` targets:

```
SET sql_mode=ORACLE;
DROP TABLE IF EXISTS t1;
DROP PROCEDURE IF EXISTS p1;
CREATE TABLE t1 (a INT, b VARCHAR(32));
INSERT INTO t1 VALUES (10,'b10');
DELIMITER $$
CREATE PROCEDURE p1 AS
  CURSOR cur1 IS SELECT * FROM t1;
  rec1 cur1%ROWTYPE;
BEGIN
  SELECT * FROM t1 INTO rec1;
  SELECT rec1.a, rec1.b;
END;
$$
DELIMITER ;
CALL p1();
```

The above example returns:

```
+--------+--------+
| rec1.a | rec1.b |
+--------+--------+
|     10 | b10    |
+--------+--------+
```

5.1.2.17 TEXT

Syntax

```
TEXT[(M)] [CHARACTER SET charset_name] [COLLATE collation_name]
```

Contents

Description

A `TEXT` column with a maximum length of `65,535` ($2^{16} - 1$) characters. The effective maximum length is less if the value contains multi-byte characters. Each `TEXT` value is stored using a two-byte length prefix that indicates the number of bytes in the value. If you need a bigger storage, consider using MEDIUMTEXT instead.

An optional length `M` can be given for this type. If this is done, MariaDB creates the column as the smallest `TEXT` type large enough to hold values `M` characters long.

Before MariaDB 10.2, all MariaDB collations were of type PADSPACE, meaning that TEXT (as well as VARCHAR and CHAR values) are compared without regard for trailing spaces. This does not apply to the LIKE pattern-matching operator, which takes into account trailing spaces.

Before MariaDB 10.2.1, `BLOB` and `TEXT` columns could not be assigned a DEFAULT value. This restriction was lifted in MariaDB 10.2.1.

Examples

Trailing spaces:

```
CREATE TABLE strtest (d TEXT(10));
INSERT INTO strtest VALUES('Maria   ');

SELECT d='Maria',d='Maria   ' FROM strtest;
+-----------+--------------+
| d='Maria' | d='Maria   ' |
+-----------+--------------+
|         1 |            1 |
+-----------+--------------+

SELECT d LIKE 'Maria',d LIKE 'Maria   ' FROM strtest;
+----------------+-------------------+
| d LIKE 'Maria' | d LIKE 'Maria   ' |
+----------------+-------------------+
|              0 |                 1 |
+----------------+-------------------+
```

Indexing

`TEXT` columns can only be indexed over a specified length. This means that they cannot be used as the primary key of a table norm until MariaDB 10.4, can a unique index be created on them.

MariaDB starting with 10.4

Starting with MariaDB 10.4, a unique index can be created on a `TEXT` column.

Internally, this uses hash indexing to quickly check the values and if a hash collision is found, the actual stored values are compared in order to retain the uniqueness.

Difference between VARCHAR and TEXT

- VARCHAR columns can be fully indexed. `TEXT` columns can only be indexed over a specified length.
- Using TEXT or BLOB in a SELECT query that uses temporary tables for storing intermediate results will force the temporary table to be disk based (using the Aria storage engine instead of the memory storage engine, which is a bit slower. This is not that bad as the Aria storage engine caches the rows in memory. To get the benefit of this, one should ensure that the aria_pagecache_buffer_size variable is big enough to hold most of the row and index data for temporary tables.

For Storage Engine Developers

- Internally the full length of the VARCHAR column is allocated inside each TABLE objects record[] structure. As there are three such buffers, each open table will allocate 3 times max-length-to-store-varchar bytes of memory.
- `TEXT` and `BLOB` columns are stored with a pointer (4 or 8 bytes) + a 1-4 bytes length. The `TEXT` data is only stored once. This means that internally `TEXT` uses less memory for each open table but instead has the additional overhead that each `TEXT` object needs to be allocated and freed for each row access (with some caching in between).

5.1.2.18 TINYBLOB

Syntax

```
TINYBLOB
```

Description

A BLOB column with a maximum length of 255 (2^8 - 1) bytes. Each TINYBLOB value is stored using a one-byte length prefix that indicates the number of bytes in the value.

5.1.2.19 TINYTEXT

Syntax

```
TINYTEXT [CHARACTER SET charset_name] [COLLATE collation_name]
```

Description

A TEXT column with a maximum length of 255 (2^8 - 1) characters. The effective maximum length is less if the value contains multi-byte characters. Each TINYTEXT value is stored using a one-byte length prefix that indicates the number of bytes in the value.

5.1.2.20 VARBINARY

Syntax

```
VARBINARY(M)
```

Contents

Description

The VARBINARY type is similar to the VARCHAR type, but stores binary byte strings rather than non-binary character strings. `M` represents the maximum column length in bytes.

It contains no character set, and comparison and sorting are based on the numeric value of the bytes.

If the maximum length is exceeded, and SQL strict mode is not enabled , the extra characters will be dropped with a warning. If strict mode is enabled, an error will occur.

Unlike BINARY values, VARBINARYs are not right-padded when inserting.

Oracle Mode

In Oracle mode from MariaDB 10.3, `RAW` is a synonym for `VARBINARY` .

Examples

Inserting too many characters, first with strict mode off, then with it on:

```
CREATE TABLE varbins (a VARBINARY(10));

INSERT INTO varbins VALUES('12345678901');
Query OK, 1 row affected, 1 warning (0.04 sec)

SELECT * FROM varbins;
+------------+
| a          |
+------------+
| 1234567890 |
+------------+

SET sql_mode='STRICT_ALL_TABLES';

INSERT INTO varbins VALUES('12345678901');
ERROR 1406 (22001): Data too long for column 'a' at row 1
```

Sorting is performed with the byte value:

```
TRUNCATE varbins;

INSERT INTO varbins VALUES('A'),('B'),('a'),('b');

SELECT * FROM varbins ORDER BY a;
+------+
| a    |
+------+
| A    |
| B    |
| a    |
| b    |
+------+
```

Using CAST to sort as a CHAR instead:

```
SELECT * FROM varbins ORDER BY CAST(a AS CHAR);
+------+
| a    |
+------+
| a    |
| A    |
| b    |
| B    |
+------+
```

5.1.2.21 VARCHAR

Syntax

```
[NATIONAL] VARCHAR(M) [CHARACTER SET charset_name] [COLLATE collation_name]
```

Contents

Description

A variable-length string. M represents the maximum column length in characters. The range of M is 0 to 65,532. The effective maximum length of a VARCHAR is subject to the maximum row size and the character set used. For example, utf8 characters can require up to three bytes per character, so a VARCHAR column that uses the utf8 character set can be declared to be a maximum of 21,844 characters.

Note: For the ColumnStore engine, M represents the maximum column length in bytes.

MariaDB stores VARCHAR values as a one-byte or two-byte length prefix plus data. The length prefix indicates the number of bytes in the value. A VARCHAR column uses one length byte if values require no more than 255 bytes, two length bytes if values may require more than 255 bytes.

MariaDB follows the standard SQL specification, and does not remove trailing spaces from VARCHAR values.

VARCHAR(0) columns can contain 2 values: an empty string or NULL. Such columns cannot be part of an index. The CONNECT storage engine does not support VARCHAR(0).

VARCHAR is shorthand for CHARACTER VARYING. NATIONAL VARCHAR is the standard SQL way to define that a VARCHAR column should use some predefined character set. MariaDB uses utf8 as this predefined character set, as does MySQL 4.1 and up. NVARCHAR is shorthand for NATIONAL VARCHAR.

Before MariaDB 10.2, all MariaDB collations were of type `PADSPACE`, meaning that VARCHAR (as well as CHAR and TEXT values) are compared without regard for trailing spaces. This does not apply to the LIKE pattern-matching operator, which takes into account trailing spaces. From MariaDB 10.2, a number of NO PAD collations are available.

If a unique index consists of a column where trailing pad characters are stripped or ignored, inserts into that column where values differ only by the number of trailing pad characters will result in a duplicate-key error.

Examples

The following are equivalent:

```
VARCHAR(30) CHARACTER SET utf8
NATIONAL VARCHAR(30)
NVARCHAR(30)
NCHAR VARCHAR(30)
NATIONAL CHARACTER VARYING(30)
NATIONAL CHAR VARYING(30)
```

Trailing spaces:

```
CREATE TABLE strtest (v VARCHAR(10));
INSERT INTO strtest VALUES('Maria   ');

SELECT v='Maria',v='Maria   ' FROM strtest;
+-----------+--------------+
| v='Maria' | v='Maria   ' |
+-----------+--------------+
|         1 |            1 |
+-----------+--------------+

SELECT v LIKE 'Maria',v LIKE 'Maria   ' FROM strtest;
+----------------+-------------------+
| v LIKE 'Maria' | v LIKE 'Maria   ' |
+----------------+-------------------+
|              0 |                 1 |
+----------------+-------------------+
```

Truncation

- Depending on whether or not strict sql mode is set, you will either get a warning or an error if you try to insert a string that is too long into a VARCHAR column. If the extra characters are spaces, the spaces that can't fit will be removed and you will always get a warning, regardless of the sql mode setting.

Difference Between VARCHAR and TEXT

- VARCHAR columns can be fully indexed. TEXT columns can only be indexed over a specified length.
- Using TEXT or BLOB in a SELECT query that uses temporary tables for storing intermediate results will force the temporary table to be disk based (using the Aria storage engine instead of the memory storage engine, which is a bit slower. This is not that bad as the Aria storage engine caches the rows in memory. To get the benefit of this, one should ensure that the aria_pagecache_buffer_size variable is big enough to hold most of the row and index data for temporary tables.

Oracle Mode

In Oracle mode from MariaDB 10.3, `VARCHAR2` is a synonym.

For Storage Engine Developers

- Internally the full length of the VARCHAR column is allocated inside each TABLE objects record[] structure. As there are three such buffers, each open table will allocate 3 times max-length-to-store-varchar bytes of memory.
- TEXT and BLOB columns are stored with a pointer (4 or 8 bytes) + a 1-4 bytes length. The TEXT data is only stored once. This means that internally `TEXT` uses less memory for each open table but instead has the additional overhead that each `TEXT` object needs to be allocated and freed for each row access (with some caching in between).

5.1.2.22 SET Data Type

Syntax

```
SET('value1','value2',...) [CHARACTER SET charset_name] [COLLATE collation_name]
```

Description

A set. A string object that can have zero or more values, each of which must be chosen from the list of values 'value1', 'value2', ... A SET column can have a maximum of 64 members. SET values are represented internally as integers.

SET values cannot contain commas.

If a SET contains duplicate values, an error will be returned if strict mode is enabled, or a warning if strict mode is not enabled.

5.1.2.23 UUID Data Type

MariaDB starting with 10.7
The UUID data type was added in MariaDB 10.7.

Syntax

```
UUID
```

Contents

Description

The `UUID` data type is intended for the storage of 128-bit UUID (Universally Unique Identifier) data. See the UUID function page for more details on UUIDs themselves.

Retrieval

Data retrieved by this data type is in the string representation defined in RFC4122.

Casting

String literals of hexadecimal characters and CHAR/VARCHAR/TEXT can be cast to the UUID data type. Likewise hexadecimal literals, binary-literals, and BINARY/VARBINARY/BLOB types can also be cast to UUID.

The data type will not accept a short UUID generated with the UUID_SHORT function, but will accept a value without the `-` character generated by the SYS_GUID function (or inserted directly). Hyphens can be partially omitted as well, or included after any group of two digits.

The type does not accept UUIDs in braces, permitted by some implementations.

Storage

UUID are stored in an index friendly manner, the order of a UUID of llllllll-mmmm-Vhhh-vsss-nnnnnnnnnnnn is stored as:

```
nnnnnnnnnnnn-vsss-Vhhh-mmmm-llllllll
```

This provides a sorting order, assuming a UUIDv1 (node and timestamp) is used, of the node, followed by the timestamp. The key aspect is the timestamps are sequential.

MariaDB starting with 10.10
Starting from MariaDB 10.10.6 and MariaDB 10.11.5, taking into account that UUIDv7 and other versions are designed around time ordering, UUIDs version >= 6 are now stored without byte-swapping, and UUIDs with version >=8 and variant=0 are now considered invalid (as the standard expects)

Examples

```
CREATE TABLE t1 (id UUID);
```

Directly Inserting via string literals:

```
INSERT INTO t1 VALUES('123e4567-e89b-12d3-a456-426655440000');
```

Directly Inserting via hexadecimal literals:

```
INSERT INTO t1 VALUES (x'fffffffffffffffffffffffffffffffe');
```

Generating and inserting via the UUID function.

```
INSERT INTO t1 VALUES (UUID());
```

Retrieval:

```
SELECT * FROM t1;
+--------------------------------------+
| id                                   |
+--------------------------------------+
| 123e4567-e89b-12d3-a456-426655440000 |
| ffffffff-ffff-ffff-ffff-fffffffffffe |
| 93aac041-1a14-11ec-ab4e-f859713e4be4 |
+--------------------------------------+
```

The UUID_SHORT function does not generate valid full-length UUID:

```
INSERT INTO t1 VALUES (UUID_SHORT());
ERROR 1292 (22007): Incorrect uuid value: '99440417627439104'
  for column `test`.`t1`.`id` at row 1
```

Accepting a value without the - character, either directly or generated by the SYS_GUID function:

```
INSERT INTO t1 VALUES (SYS_GUID());

SELECT * FROM t1;
+--------------------------------------+
| id                                   |
+--------------------------------------+
| 123e4567-e89b-12d3-a456-426655440000 |
| ffffffff-ffff-ffff-ffff-fffffffffffe |
| 93aac041-1a14-11ec-ab4e-f859713e4be4 |
| ea0368d3-1a14-11ec-ab4e-f859713e4be4 |
+--------------------------------------+

SELECT SYS_GUID();
+----------------------------------+
| SYS_GUID()                       |
+----------------------------------+
| ff5b6bcc1a1411ecab4ef859713e4be4 |
+----------------------------------+

INSERT INTO t1 VALUES ('ff5b6bcc1a1411ecab4ef859713e4be4');

SELECT * FROM t1;
+--------------------------------------+
| id                                   |
+--------------------------------------+
| 123e4567-e89b-12d3-a456-426655440000 |
| ffffffff-ffff-ffff-ffff-fffffffffffe |
| 93aac041-1a14-11ec-ab4e-f859713e4be4 |
| ea0368d3-1a14-11ec-ab4e-f859713e4be4 |
| ff5b6bcc-1a14-11ec-ab4e-f859713e4be4 |
+--------------------------------------+
```

Valid and invalid hyphen and brace usage:

```
TRUNCATE t1;

INSERT INTO t1 VALUES ('f8aa-ed66-1a1b-11ec-ab4e-f859-713e-4be4');

INSERT INTO t1 VALUES ('1b80667f1a1c-11ecab4ef859713e4be4');

INSERT INTO t1 VALUES ('2fd6c945-1a-1c-11ec-ab4e-f859713e4be4');

INSERT INTO t1 VALUES ('49-c9-f9-59-1a-1c-11ec-ab4e-f859713e4be4');

INSERT INTO t1 VALUES ('57-96-da-c1-1a-1c-11-ec-ab-4e-f8-59-71-3e-4b-e4');

INSERT INTO t1 VALUES ('6-eb74f8f-1a1c-11ec-ab4e-f859713e4be4');

INSERT INTO t1 VALUES ('{29bad136-1a1d-11ec-ab4e-f859713e4be4}');
ERROR 1292 (22007): Incorrect uuid value: '{29bad136-1a1d-11ec-ab4e-f859713e4be4}'
  for column `test`.`t1`.`id` at row 1

SELECT * FROM t1;
+--------------------------------------+
| id                                   |
+--------------------------------------+
| f8aaed66-1a1b-11ec-ab4e-f859713e4be4 |
| 1b80667f-1a1c-11ec-ab4e-f859713e4be4 |
| 2fd6c945-1a1c-11ec-ab4e-f859713e4be4 |
| 49c9f959-1a1c-11ec-ab4e-f859713e4be4 |
| 5796dac1-1a1c-11ec-ab4e-f859713e4be4 |
| 6eb74f8f-1a1c-11ec-ab4e-f859713e4be4 |
+--------------------------------------+
```

5.1.2.24 Data Type Storage Requirements

Contents

The following tables indicate the approximate data storage requirements for each data type.

Numeric Data Types

Data Type	Storage Requirement
TINYINT	1 byte
SMALLINT	2 bytes
MEDIUMINT	3 bytes
INT	4 bytes
BIGINT	8 bytes
FLOAT(p)	4 bytes if p <= 24, otherwise 8 bytes
DOUBLE	8 bytes
DECIMAL	See table below
BIT(M)	(M+7)/8 bytes

Note that MEDIUMINT columns will require 4 bytes in memory (for example, in InnoDB buffer pool).

Decimal

Decimals are stored using a binary format, with the integer and the fraction stored separately. Each nine-digit multiple requires 4 bytes, followed by a number of bytes for whatever remains, as follows:

Remaining digits	Storage Requirement
0	0 bytes
1	1 byte
2	1 byte
3	2 bytes
4	2 bytes
5	3 bytes
6	3 bytes
7	4 bytes
8	4 bytes

String Data Types

In the descriptions below, `M` is the declared column length (in characters or in bytes), while `len` is the actual length in bytes of the value.

Data Type	Storage Requirement
ENUM	1 byte for up to 255 enum values, 2 bytes for 256 to 65,535 enum values
CHAR(M)	M × w bytes, where w is the number of bytes required for the maximum-length character in the character set
BINARY(M)	M bytes
VARCHAR(M), VARBINARY(M)	len + 1 bytes if column is 0 – 255 bytes, len + 2 bytes if column may require more than 255 bytes
TINYBLOB, TINYTEXT	len + 1 bytes
BLOB, TEXT	len + 2 bytes
MEDIUMBLOB, MEDIUMTEXT	len + 3 bytes
LONGBLOB, LONGTEXT	len + 4 bytes
SET	Given M members of the set, (M+7)/8 bytes, rounded up to 1, 2, 3, 4, or 8 bytes
INET6	16 bytes
UUID	16 bytes

In some character sets, not all characters use the same number of bytes. utf8 encodes characters with one to three bytes per character, while utf8mb4 requires one to four bytes per character.

When using field the COMPRESSED attribute, 1 byte is reserved for metadata. For example, VARCHAR(255) will use +2 bytes instead of +1.

Examples

Assuming a single-byte character-set:

Value	CHAR(2)	Storage Required	VARCHAR(2)	Storage Required
''	' '	2 bytes	''	1 byte
'1'	'1 '	2 bytes	'1'	2 bytes
'12'	'12'	2 bytes	'12'	3 bytes

Date and Time Data Types

Data Type	Storage Requirement
DATE	3 bytes
TIME	3 bytes
DATETIME	8 bytes
TIMESTAMP	4 bytes
YEAR	1 byte

Microseconds

MariaDB 5.3 and MySQL 5.6 introduced microseconds. The underlying storage implementations were different, but from MariaDB 10.1, MariaDB defaults to the MySQL format (by means of the mysql56_temporal_format variable). Microseconds have the following additional storage requirements:

MySQL 5.6+ and MariaDB 10.1+

Precision	Storage Requirement
0	0 bytes
1,2	1 byte
3,4	2 bytes
5,6	3 bytes

MariaDB 5.3 - MariaDB 10.0

Precision	Storage Requirement
0	0 bytes
1,2	1 byte
3,4,5	2 bytes
6	3 bytes

NULLs

For the InnoDB COMPACT, DYNAMIC and COMPRESSED row formats, a number of bytes will be allocated in the record header for the nullable fields. If there are between 1 and 8 nullable fields, 1 such byte will be allocated. In the record payload area, no space will be reserved for values that are NULL.

For the InnoDB REDUNDANT row format, the overhead is 1 bit in the record header (as a part of the 1-byte or 2-byte "end of field" pointer). In that format, a NULL fixed-length field will consume the same amount of space as any NOT NULL value in the record payload area. The motivation is that it is possible to update in place between NOT NULL and NULL values.

In other formats, NULL values usually require 1 bit in the data file, 1 byte in the index file.

5.1.2.25 Supported Character Sets and Collations

Contents

Character Sets

You can see which character sets are available in a particular version by running the SHOW CHARACTER SET statement or by querying the Information Schema CHARACTER_SETS Table.

From MariaDB 11.2, it is possible to change the default collation associated with a character set. See Changing Default Collation

MariaDB supports the following character sets:

Charset	Description	Default collation	Maxlen
armscii8	ARMSCII-8 Armenian	armscii8_general_ci	1
ascii	US ASCII	ascii_general_ci	1
big5	Big5 Traditional Chinese	big5_chinese_ci	2
binary	Binary pseudo charset	binary	1
cp1250	Windows Central European	cp1250_general_ci	1
cp1251	Windows Cyrillic	cp1251_general_ci	1
cp1256	Windows Arabic	cp1256_general_ci	1
cp1257	Windows Baltic	cp1257_general_ci	1
cp850	DOS West European	cp850_general_ci	1
cp852	DOS Central European	cp852_general_ci	1
cp866	DOS Russian	cp866_general_ci	1
cp932	SJIS for Windows Japanese	cp932_japanese_ci	2
dec8	DEC West European	dec8_swedish_ci	1
eucjpms	UJIS for Windows Japanese	eucjpms_japanese_ci	3
euckr	EUC-KR Korean	euckr_korean_ci	2
gb2312	GB2312 Simplified Chinese	gb2312_chinese_ci	2
gbk	GBK Simplified Chinese	gbk_chinese_ci	2
geostd8	GEOSTD8 Georgian	geostd8_general_ci	1
greek	ISO 8859-7 Greek	greek_general_ci	1
hebrew	ISO 8859-8 Hebrew	hebrew_general_ci	1
hp8	HP West European	hp8_english_ci	1
keybcs2	DOS Kamenicky Czech-Slovak	keybcs2_general_ci	1
koi8r	KOI8-R Relcom Russian	koi8r_general_ci	1
koi8u	KOI8-U Ukrainian	koi8u_general_ci	1
latin1	cp1252 West European	latin1_swedish_ci	1
latin2	ISO 8859-2 Central European	latin2_general_ci	1
latin5	ISO 8859-9 Turkish	latin5_turkish_ci	1
latin7	ISO 8859-13 Baltic	latin7_general_ci	1
macce	Mac Central European	macce_general_ci	1
macroman	Mac West European	macroman_general_ci	1
sjis	Shift-JIS Japanese	sjis_japanese_ci	2
swe7	7bit Swedish	swe7_swedish_ci	1
tis620	TIS620 Thai	tis620_thai_ci	1
ucs2	UCS-2 Unicode	ucs2_general_ci	2
ujis	EUC-JP Japanese	ujis_japanese_ci	3
utf8	UTF-8 Unicode	utf8_general_ci	3/4 (see OLD_MODE)

utf16	UTF-16 Unicode	utf16_general_ci	4
utf16le	UTF-16LE Unicode	utf16le_general_ci	4
utf32	UTF-32 Unicode	utf32_general_ci	4
utf8mb3	UTF-8 Unicode	utf8mb3_general_ci	3
utf8mb4	UTF-8 Unicode	utf8mb4_general_ci	4

Note that the Mroonga Storage Engine only supports a limited number of character sets. See Mroonga available character sets.

Collations

MariaDB supports the following collations:

```
show collation;
+--------------------------------+----------+------+---------+----------+---------+
| Collation                      | Charset  | Id   | Default | Compiled | Sortlen |
+--------------------------------+----------+------+---------+----------+---------+
| big5_chinese_ci                | big5     |    1 | Yes     | Yes      |       1 |
| big5_bin                       | big5     |   84 |         | Yes      |       1 |
| big5_chinese_nopad_ci          | big5     | 1025 |         | Yes      |       1 |
| big5_nopad_bin                 | big5     | 1108 |         | Yes      |       1 |
| dec8_swedish_ci                | dec8     |    3 | Yes     | Yes      |       1 |
| dec8_bin                       | dec8     |   69 |         | Yes      |       1 |
| dec8_swedish_nopad_ci          | dec8     | 1027 |         | Yes      |       1 |
| dec8_nopad_bin                 | dec8     | 1093 |         | Yes      |       1 |
| cp850_general_ci               | cp850    |    4 | Yes     | Yes      |       1 |
| cp850_bin                      | cp850    |   80 |         | Yes      |       1 |
| cp850_general_nopad_ci         | cp850    | 1028 |         | Yes      |       1 |
| cp850_nopad_bin                | cp850    | 1104 |         | Yes      |       1 |
| hp8_english_ci                 | hp8      |    6 | Yes     | Yes      |       1 |
| hp8_bin                        | hp8      |   72 |         | Yes      |       1 |
| hp8_english_nopad_ci           | hp8      | 1030 |         | Yes      |       1 |
| hp8_nopad_bin                  | hp8      | 1096 |         | Yes      |       1 |
| koi8r_general_ci               | koi8r    |    7 | Yes     | Yes      |       1 |
| koi8r_bin                      | koi8r    |   74 |         | Yes      |       1 |
| koi8r_general_nopad_ci         | koi8r    | 1031 |         | Yes      |       1 |
| koi8r_nopad_bin                | koi8r    | 1098 |         | Yes      |       1 |
| latin1_german1_ci              | latin1   |    5 |         | Yes      |       1 |
| latin1_swedish_ci              | latin1   |    8 | Yes     | Yes      |       1 |
| latin1_danish_ci               | latin1   |   15 |         | Yes      |       1 |
| latin1_german2_ci              | latin1   |   31 |         | Yes      |       2 |
| latin1_bin                     | latin1   |   47 |         | Yes      |       1 |
| latin1_general_ci              | latin1   |   48 |         | Yes      |       1 |
| latin1_general_cs              | latin1   |   49 |         | Yes      |       1 |
| latin1_spanish_ci              | latin1   |   94 |         | Yes      |       1 |
| latin1_swedish_nopad_ci        | latin1   | 1032 |         | Yes      |       1 |
| latin1_nopad_bin               | latin1   | 1071 |         | Yes      |       1 |
| latin2_czech_cs                | latin2   |    2 |         | Yes      |       4 |
| latin2_general_ci              | latin2   |    9 | Yes     | Yes      |       1 |
| latin2_hungarian_ci            | latin2   |   21 |         | Yes      |       1 |
| latin2_croatian_ci             | latin2   |   27 |         | Yes      |       1 |
| latin2_bin                     | latin2   |   77 |         | Yes      |       1 |
| latin2_general_nopad_ci        | latin2   | 1033 |         | Yes      |       1 |
| latin2_nopad_bin               | latin2   | 1101 |         | Yes      |       1 |
| swe7_swedish_ci                | swe7     |   10 | Yes     | Yes      |       1 |
| swe7_bin                       | swe7     |   82 |         | Yes      |       1 |
| swe7_swedish_nopad_ci          | swe7     | 1034 |         | Yes      |       1 |
| swe7_nopad_bin                 | swe7     | 1106 |         | Yes      |       1 |
| ascii_general_ci               | ascii    |   11 | Yes     | Yes      |       1 |
| ascii_bin                      | ascii    |   65 |         | Yes      |       1 |
| ascii_general_nopad_ci         | ascii    | 1035 |         | Yes      |       1 |
| ascii_nopad_bin                | ascii    | 1089 |         | Yes      |       1 |
| ujis_japanese_ci               | ujis     |   12 | Yes     | Yes      |       1 |
| ujis_bin                       | ujis     |   91 |         | Yes      |       1 |
| ujis_japanese_nopad_ci         | ujis     | 1036 |         | Yes      |       1 |
| ujis_nopad_bin                 | ujis     | 1115 |         | Yes      |       1 |
| sjis_japanese_ci               | sjis     |   13 | Yes     | Yes      |       1 |
| sjis_bin                       | sjis     |   88 |         | Yes      |       1 |
| sjis_japanese_nopad_ci         | sjis     | 1037 |         | Yes      |       1 |
| sjis_nopad_bin                 | sjis     | 1112 |         | Yes      |       1 |
```

hebrew_general_ci	hebrew	16	Yes	Yes	1
hebrew_bin	hebrew	71		Yes	1
hebrew_general_nopad_ci	hebrew	1040		Yes	1
hebrew_nopad_bin	hebrew	1095		Yes	1
tis620_thai_ci	tis620	18	Yes	Yes	4
tis620_bin	tis620	89		Yes	1
tis620_thai_nopad_ci	tis620	1042		Yes	4
tis620_nopad_bin	tis620	1113		Yes	1
euckr_korean_ci	euckr	19	Yes	Yes	1
euckr_bin	euckr	85		Yes	1
euckr_korean_nopad_ci	euckr	1043		Yes	1
euckr_nopad_bin	euckr	1109		Yes	1
koi8u_general_ci	koi8u	22	Yes	Yes	1
koi8u_bin	koi8u	75		Yes	1
koi8u_general_nopad_ci	koi8u	1046		Yes	1
koi8u_nopad_bin	koi8u	1099		Yes	1
gb2312_chinese_ci	gb2312	24	Yes	Yes	1
gb2312_bin	gb2312	86		Yes	1
gb2312_chinese_nopad_ci	gb2312	1048		Yes	1
gb2312_nopad_bin	gb2312	1110		Yes	1
greek_general_ci	greek	25	Yes	Yes	1
greek_bin	greek	70		Yes	1
greek_general_nopad_ci	greek	1049		Yes	1
greek_nopad_bin	greek	1094		Yes	1
cp1250_general_ci	cp1250	26	Yes	Yes	1
cp1250_czech_cs	cp1250	34		Yes	2
cp1250_croatian_ci	cp1250	44		Yes	1
cp1250_bin	cp1250	66		Yes	1
cp1250_polish_ci	cp1250	99		Yes	1
cp1250_general_nopad_ci	cp1250	1050		Yes	1
cp1250_nopad_bin	cp1250	1090		Yes	1
gbk_chinese_ci	gbk	28	Yes	Yes	1
gbk_bin	gbk	87		Yes	1
gbk_chinese_nopad_ci	gbk	1052		Yes	1
gbk_nopad_bin	gbk	1111		Yes	1
latin5_turkish_ci	latin5	30	Yes	Yes	1
latin5_bin	latin5	78		Yes	1
latin5_turkish_nopad_ci	latin5	1054		Yes	1
latin5_nopad_bin	latin5	1102		Yes	1
armscii8_general_ci	armscii8	32	Yes	Yes	1
armscii8_bin	armscii8	64		Yes	1
armscii8_general_nopad_ci	armscii8	1056		Yes	1
armscii8_nopad_bin	armscii8	1088		Yes	1
utf8mb3_general_ci	utf8mb3	33	Yes	Yes	1
utf8mb3_bin	utf8mb3	83		Yes	1
utf8mb3_unicode_ci	utf8mb3	192		Yes	8
utf8mb3_icelandic_ci	utf8mb3	193		Yes	8
utf8mb3_latvian_ci	utf8mb3	194		Yes	8
utf8mb3_romanian_ci	utf8mb3	195		Yes	8
utf8mb3_slovenian_ci	utf8mb3	196		Yes	8
utf8mb3_polish_ci	utf8mb3	197		Yes	8
utf8mb3_estonian_ci	utf8mb3	198		Yes	8
utf8mb3_spanish_ci	utf8mb3	199		Yes	8
utf8mb3_swedish_ci	utf8mb3	200		Yes	8
utf8mb3_turkish_ci	utf8mb3	201		Yes	8
utf8mb3_czech_ci	utf8mb3	202		Yes	8
utf8mb3_danish_ci	utf8mb3	203		Yes	8
utf8mb3_lithuanian_ci	utf8mb3	204		Yes	8
utf8mb3_slovak_ci	utf8mb3	205		Yes	8
utf8mb3_spanish2_ci	utf8mb3	206		Yes	8
utf8mb3_roman_ci	utf8mb3	207		Yes	8
utf8mb3_persian_ci	utf8mb3	208		Yes	8
utf8mb3_esperanto_ci	utf8mb3	209		Yes	8
utf8mb3_hungarian_ci	utf8mb3	210		Yes	8
utf8mb3_sinhala_ci	utf8mb3	211		Yes	8
utf8mb3_german2_ci	utf8mb3	212		Yes	8
utf8mb3_croatian_mysql561_ci	utf8mb3	213		Yes	8
utf8mb3_unicode_520_ci	utf8mb3	214		Yes	8
utf8mb3_vietnamese_ci	utf8mb3	215		Yes	8
utf8mb3_general_mysql500_ci	utf8mb3	223		Yes	1
utf8mb3_croatian_ci	utf8mb3	576		Yes	8
utf8mb3_myanmar_ci	utf8mb3	577		Yes	8
utf8mb3_thai_520_w2	utf8mb3	578		Yes	4
utf8mb3_general_nopad_ci	utf8mb3	1057		Yes	1
utf8mb3_nopad_bin	utf8mb3	1107		Yes	1

utf8mb3_unicode_nopad_ci	utf8mb3	1216		Yes	8
utf8mb3_unicode_520_nopad_ci	utf8mb3	1238		Yes	8
ucs2_general_ci	ucs2	35	Yes	Yes	1
ucs2_bin	ucs2	90		Yes	1
ucs2_unicode_ci	ucs2	128		Yes	8
ucs2_icelandic_ci	ucs2	129		Yes	8
ucs2_latvian_ci	ucs2	130		Yes	8
ucs2_romanian_ci	ucs2	131		Yes	8
ucs2_slovenian_ci	ucs2	132		Yes	8
ucs2_polish_ci	ucs2	133		Yes	8
ucs2_estonian_ci	ucs2	134		Yes	8
ucs2_spanish_ci	ucs2	135		Yes	8
ucs2_swedish_ci	ucs2	136		Yes	8
ucs2_turkish_ci	ucs2	137		Yes	8
ucs2_czech_ci	ucs2	138		Yes	8
ucs2_danish_ci	ucs2	139		Yes	8
ucs2_lithuanian_ci	ucs2	140		Yes	8
ucs2_slovak_ci	ucs2	141		Yes	8
ucs2_spanish2_ci	ucs2	142		Yes	8
ucs2_roman_ci	ucs2	143		Yes	8
ucs2_persian_ci	ucs2	144		Yes	8
ucs2_esperanto_ci	ucs2	145		Yes	8
ucs2_hungarian_ci	ucs2	146		Yes	8
ucs2_sinhala_ci	ucs2	147		Yes	8
ucs2_german2_ci	ucs2	148		Yes	8
ucs2_croatian_mysql561_ci	ucs2	149		Yes	8
ucs2_unicode_520_ci	ucs2	150		Yes	8
ucs2_vietnamese_ci	ucs2	151		Yes	8
ucs2_general_mysql500_ci	ucs2	159		Yes	1
ucs2_croatian_ci	ucs2	640		Yes	8
ucs2_myanmar_ci	ucs2	641		Yes	8
ucs2_thai_520_w2	ucs2	642		Yes	4
ucs2_general_nopad_ci	ucs2	1059		Yes	1
ucs2_nopad_bin	ucs2	1114		Yes	1
ucs2_unicode_nopad_ci	ucs2	1152		Yes	8
ucs2_unicode_520_nopad_ci	ucs2	1174		Yes	8
cp866_general_ci	cp866	36	Yes	Yes	1
cp866_bin	cp866	68		Yes	1
cp866_general_nopad_ci	cp866	1060		Yes	1
cp866_nopad_bin	cp866	1092		Yes	1
keybcs2_general_ci	keybcs2	37	Yes	Yes	1
keybcs2_bin	keybcs2	73		Yes	1
keybcs2_general_nopad_ci	keybcs2	1061		Yes	1
keybcs2_nopad_bin	keybcs2	1097		Yes	1
macce_general_ci	macce	38	Yes	Yes	1
macce_bin	macce	43		Yes	1
macce_general_nopad_ci	macce	1062		Yes	1
macce_nopad_bin	macce	1067		Yes	1
macroman_general_ci	macroman	39	Yes	Yes	1
macroman_bin	macroman	53		Yes	1
macroman_general_nopad_ci	macroman	1063		Yes	1
macroman_nopad_bin	macroman	1077		Yes	1
cp852_general_ci	cp852	40	Yes	Yes	1
cp852_bin	cp852	81		Yes	1
cp852_general_nopad_ci	cp852	1064		Yes	1
cp852_nopad_bin	cp852	1105		Yes	1
latin7_estonian_cs	latin7	20		Yes	1
latin7_general_ci	latin7	41	Yes	Yes	1
latin7_general_cs	latin7	42		Yes	1
latin7_bin	latin7	79		Yes	1
latin7_general_nopad_ci	latin7	1065		Yes	1
latin7_nopad_bin	latin7	1103		Yes	1
utf8mb4_general_ci	utf8mb4	45	Yes	Yes	1
utf8mb4_bin	utf8mb4	46		Yes	1
utf8mb4_unicode_ci	utf8mb4	224		Yes	8
utf8mb4_icelandic_ci	utf8mb4	225		Yes	8
utf8mb4_latvian_ci	utf8mb4	226		Yes	8
utf8mb4_romanian_ci	utf8mb4	227		Yes	8
utf8mb4_slovenian_ci	utf8mb4	228		Yes	8
utf8mb4_polish_ci	utf8mb4	229		Yes	8
utf8mb4_estonian_ci	utf8mb4	230		Yes	8
utf8mb4_spanish_ci	utf8mb4	231		Yes	8
utf8mb4_swedish_ci	utf8mb4	232		Yes	8
utf8mb4_turkish_ci	utf8mb4	233		Yes	8
utf8mb4_czech_ci	utf8mb4	234		Yes	8

utf8mb4_danish_ci	utf8mb4	235		Yes	8
utf8mb4_lithuanian_ci	utf8mb4	236		Yes	8
utf8mb4_slovak_ci	utf8mb4	237		Yes	8
utf8mb4_spanish2_ci	utf8mb4	238		Yes	8
utf8mb4_roman_ci	utf8mb4	239		Yes	8
utf8mb4_persian_ci	utf8mb4	240		Yes	8
utf8mb4_esperanto_ci	utf8mb4	241		Yes	8
utf8mb4_hungarian_ci	utf8mb4	242		Yes	8
utf8mb4_sinhala_ci	utf8mb4	243		Yes	8
utf8mb4_german2_ci	utf8mb4	244		Yes	8
utf8mb4_croatian_mysql561_ci	utf8mb4	245		Yes	8
utf8mb4_unicode_520_ci	utf8mb4	246		Yes	8
utf8mb4_vietnamese_ci	utf8mb4	247		Yes	8
utf8mb4_croatian_ci	utf8mb4	608		Yes	8
utf8mb4_myanmar_ci	utf8mb4	609		Yes	8
utf8mb4_thai_520_w2	utf8mb4	610		Yes	4
utf8mb4_general_nopad_ci	utf8mb4	1069		Yes	1
utf8mb4_nopad_bin	utf8mb4	1070		Yes	1
utf8mb4_unicode_nopad_ci	utf8mb4	1248		Yes	8
utf8mb4_unicode_520_nopad_ci	utf8mb4	1270		Yes	8
uca1400_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_as_ci	NULL	NULL	NULL	Yes	8
uca1400_as_cs	NULL	NULL	NULL	Yes	8
uca1400_nopad_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_nopad_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_nopad_as_ci	NULL	NULL	NULL	Yes	8
uca1400_nopad_as_cs	NULL	NULL	NULL	Yes	8
uca1400_icelandic_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_icelandic_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_icelandic_as_ci	NULL	NULL	NULL	Yes	8
uca1400_icelandic_as_cs	NULL	NULL	NULL	Yes	8
uca1400_icelandic_nopad_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_icelandic_nopad_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_icelandic_nopad_as_ci	NULL	NULL	NULL	Yes	8
uca1400_icelandic_nopad_as_cs	NULL	NULL	NULL	Yes	8
uca1400_latvian_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_latvian_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_latvian_as_ci	NULL	NULL	NULL	Yes	8
uca1400_latvian_as_cs	NULL	NULL	NULL	Yes	8
uca1400_latvian_nopad_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_latvian_nopad_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_latvian_nopad_as_ci	NULL	NULL	NULL	Yes	8
uca1400_latvian_nopad_as_cs	NULL	NULL	NULL	Yes	8
uca1400_romanian_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_romanian_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_romanian_as_ci	NULL	NULL	NULL	Yes	8
uca1400_romanian_as_cs	NULL	NULL	NULL	Yes	8
uca1400_romanian_nopad_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_romanian_nopad_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_romanian_nopad_as_ci	NULL	NULL	NULL	Yes	8
uca1400_romanian_nopad_as_cs	NULL	NULL	NULL	Yes	8
uca1400_slovenian_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_slovenian_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_slovenian_as_ci	NULL	NULL	NULL	Yes	8
uca1400_slovenian_as_cs	NULL	NULL	NULL	Yes	8
uca1400_slovenian_nopad_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_slovenian_nopad_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_slovenian_nopad_as_ci	NULL	NULL	NULL	Yes	8
uca1400_slovenian_nopad_as_cs	NULL	NULL	NULL	Yes	8
uca1400_polish_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_polish_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_polish_as_ci	NULL	NULL	NULL	Yes	8
uca1400_polish_as_cs	NULL	NULL	NULL	Yes	8
uca1400_polish_nopad_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_polish_nopad_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_polish_nopad_as_ci	NULL	NULL	NULL	Yes	8
uca1400_polish_nopad_as_cs	NULL	NULL	NULL	Yes	8
uca1400_estonian_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_estonian_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_estonian_as_ci	NULL	NULL	NULL	Yes	8
uca1400_estonian_as_cs	NULL	NULL	NULL	Yes	8
uca1400_estonian_nopad_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_estonian_nopad_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_estonian_nopad_as_ci	NULL	NULL	NULL	Yes	8

uca1400_estonian_nopad_as_ci	NULL	NULL	NULL	Yes	8
uca1400_estonian_nopad_as_cs	NULL	NULL	NULL	Yes	8
uca1400_spanish_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_spanish_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_spanish_as_ci	NULL	NULL	NULL	Yes	8
uca1400_spanish_as_cs	NULL	NULL	NULL	Yes	8
uca1400_spanish_nopad_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_spanish_nopad_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_spanish_nopad_as_ci	NULL	NULL	NULL	Yes	8
uca1400_spanish_nopad_as_cs	NULL	NULL	NULL	Yes	8
uca1400_swedish_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_swedish_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_swedish_as_ci	NULL	NULL	NULL	Yes	8
uca1400_swedish_as_cs	NULL	NULL	NULL	Yes	8
uca1400_swedish_nopad_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_swedish_nopad_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_swedish_nopad_as_ci	NULL	NULL	NULL	Yes	8
uca1400_swedish_nopad_as_cs	NULL	NULL	NULL	Yes	8
uca1400_turkish_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_turkish_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_turkish_as_ci	NULL	NULL	NULL	Yes	8
uca1400_turkish_as_cs	NULL	NULL	NULL	Yes	8
uca1400_turkish_nopad_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_turkish_nopad_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_turkish_nopad_as_ci	NULL	NULL	NULL	Yes	8
uca1400_turkish_nopad_as_cs	NULL	NULL	NULL	Yes	8
uca1400_czech_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_czech_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_czech_as_ci	NULL	NULL	NULL	Yes	8
uca1400_czech_as_cs	NULL	NULL	NULL	Yes	8
uca1400_czech_nopad_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_czech_nopad_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_czech_nopad_as_ci	NULL	NULL	NULL	Yes	8
uca1400_czech_nopad_as_cs	NULL	NULL	NULL	Yes	8
uca1400_danish_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_danish_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_danish_as_ci	NULL	NULL	NULL	Yes	8
uca1400_danish_as_cs	NULL	NULL	NULL	Yes	8
uca1400_danish_nopad_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_danish_nopad_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_danish_nopad_as_ci	NULL	NULL	NULL	Yes	8
uca1400_danish_nopad_as_cs	NULL	NULL	NULL	Yes	8
uca1400_lithuanian_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_lithuanian_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_lithuanian_as_ci	NULL	NULL	NULL	Yes	8
uca1400_lithuanian_as_cs	NULL	NULL	NULL	Yes	8
uca1400_lithuanian_nopad_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_lithuanian_nopad_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_lithuanian_nopad_as_ci	NULL	NULL	NULL	Yes	8
uca1400_lithuanian_nopad_as_cs	NULL	NULL	NULL	Yes	8
uca1400_slovak_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_slovak_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_slovak_as_ci	NULL	NULL	NULL	Yes	8
uca1400_slovak_as_cs	NULL	NULL	NULL	Yes	8
uca1400_slovak_nopad_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_slovak_nopad_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_slovak_nopad_as_ci	NULL	NULL	NULL	Yes	8
uca1400_slovak_nopad_as_cs	NULL	NULL	NULL	Yes	8
uca1400_spanish2_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_spanish2_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_spanish2_as_ci	NULL	NULL	NULL	Yes	8
uca1400_spanish2_as_cs	NULL	NULL	NULL	Yes	8
uca1400_spanish2_nopad_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_spanish2_nopad_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_spanish2_nopad_as_ci	NULL	NULL	NULL	Yes	8
uca1400_spanish2_nopad_as_cs	NULL	NULL	NULL	Yes	8
uca1400_roman_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_roman_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_roman_as_ci	NULL	NULL	NULL	Yes	8
uca1400_roman_as_cs	NULL	NULL	NULL	Yes	8
uca1400_roman_nopad_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_roman_nopad_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_roman_nopad_as_ci	NULL	NULL	NULL	Yes	8
uca1400_roman_nopad_as_cs	NULL	NULL	NULL	Yes	8
uca1400_persian_ai_ci	NULL	NULL	NULL	Yes	8

uca1400_persian_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_persian_as_ci	NULL	NULL	NULL	Yes	8
uca1400_persian_as_cs	NULL	NULL	NULL	Yes	8
uca1400_persian_nopad_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_persian_nopad_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_persian_nopad_as_ci	NULL	NULL	NULL	Yes	8
uca1400_persian_nopad_as_cs	NULL	NULL	NULL	Yes	8
uca1400_esperanto_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_esperanto_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_esperanto_as_ci	NULL	NULL	NULL	Yes	8
uca1400_esperanto_as_cs	NULL	NULL	NULL	Yes	8
uca1400_esperanto_nopad_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_esperanto_nopad_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_esperanto_nopad_as_ci	NULL	NULL	NULL	Yes	8
uca1400_esperanto_nopad_as_cs	NULL	NULL	NULL	Yes	8
uca1400_hungarian_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_hungarian_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_hungarian_as_ci	NULL	NULL	NULL	Yes	8
uca1400_hungarian_as_cs	NULL	NULL	NULL	Yes	8
uca1400_hungarian_nopad_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_hungarian_nopad_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_hungarian_nopad_as_ci	NULL	NULL	NULL	Yes	8
uca1400_hungarian_nopad_as_cs	NULL	NULL	NULL	Yes	8
uca1400_sinhala_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_sinhala_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_sinhala_as_ci	NULL	NULL	NULL	Yes	8
uca1400_sinhala_as_cs	NULL	NULL	NULL	Yes	8
uca1400_sinhala_nopad_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_sinhala_nopad_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_sinhala_nopad_as_ci	NULL	NULL	NULL	Yes	8
uca1400_sinhala_nopad_as_cs	NULL	NULL	NULL	Yes	8
uca1400_german2_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_german2_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_german2_as_ci	NULL	NULL	NULL	Yes	8
uca1400_german2_as_cs	NULL	NULL	NULL	Yes	8
uca1400_german2_nopad_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_german2_nopad_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_german2_nopad_as_ci	NULL	NULL	NULL	Yes	8
uca1400_german2_nopad_as_cs	NULL	NULL	NULL	Yes	8
uca1400_vietnamese_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_vietnamese_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_vietnamese_as_ci	NULL	NULL	NULL	Yes	8
uca1400_vietnamese_as_cs	NULL	NULL	NULL	Yes	8
uca1400_vietnamese_nopad_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_vietnamese_nopad_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_vietnamese_nopad_as_ci	NULL	NULL	NULL	Yes	8
uca1400_vietnamese_nopad_as_cs	NULL	NULL	NULL	Yes	8
uca1400_croatian_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_croatian_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_croatian_as_ci	NULL	NULL	NULL	Yes	8
uca1400_croatian_as_cs	NULL	NULL	NULL	Yes	8
uca1400_croatian_nopad_ai_ci	NULL	NULL	NULL	Yes	8
uca1400_croatian_nopad_ai_cs	NULL	NULL	NULL	Yes	8
uca1400_croatian_nopad_as_ci	NULL	NULL	NULL	Yes	8
uca1400_croatian_nopad_as_cs	NULL	NULL	NULL	Yes	8
cp1251_bulgarian_ci	cp1251	14		Yes	1
cp1251_ukrainian_ci	cp1251	23		Yes	1
cp1251_bin	cp1251	50		Yes	1
cp1251_general_ci	cp1251	51	Yes	Yes	1
cp1251_general_cs	cp1251	52		Yes	1
cp1251_nopad_bin	cp1251	1074		Yes	1
cp1251_general_nopad_ci	cp1251	1075		Yes	1
utf16_general_ci	utf16	54	Yes	Yes	1
utf16_bin	utf16	55		Yes	1
utf16_unicode_ci	utf16	101		Yes	8
utf16_icelandic_ci	utf16	102		Yes	8
utf16_latvian_ci	utf16	103		Yes	8
utf16_romanian_ci	utf16	104		Yes	8
utf16_slovenian_ci	utf16	105		Yes	8
utf16_polish_ci	utf16	106		Yes	8
utf16_estonian_ci	utf16	107		Yes	8
utf16_spanish_ci	utf16	108		Yes	8
utf16_swedish_ci	utf16	109		Yes	8
utf16_turkish_ci	utf16	110		Yes	8
utf16_czech_ci	utf16	111		Yes	8

utf16_danish_ci	utf16	112		Yes	8
utf16_lithuanian_ci	utf16	113		Yes	8
utf16_slovak_ci	utf16	114		Yes	8
utf16_spanish2_ci	utf16	115		Yes	8
utf16_roman_ci	utf16	116		Yes	8
utf16_persian_ci	utf16	117		Yes	8
utf16_esperanto_ci	utf16	118		Yes	8
utf16_hungarian_ci	utf16	119		Yes	8
utf16_sinhala_ci	utf16	120		Yes	8
utf16_german2_ci	utf16	121		Yes	8
utf16_croatian_mysql561_ci	utf16	122		Yes	8
utf16_unicode_520_ci	utf16	123		Yes	8
utf16_vietnamese_ci	utf16	124		Yes	8
utf16_croatian_ci	utf16	672		Yes	8
utf16_myanmar_ci	utf16	673		Yes	8
utf16_thai_520_w2	utf16	674		Yes	4
utf16_general_nopad_ci	utf16	1078		Yes	1
utf16_nopad_bin	utf16	1079		Yes	1
utf16_unicode_nopad_ci	utf16	1125		Yes	8
utf16_unicode_520_nopad_ci	utf16	1147		Yes	8
utf16le_general_ci	utf16le	56	Yes	Yes	1
utf16le_bin	utf16le	62		Yes	1
utf16le_general_nopad_ci	utf16le	1080		Yes	1
utf16le_nopad_bin	utf16le	1086		Yes	1
cp1256_general_ci	cp1256	57	Yes	Yes	1
cp1256_bin	cp1256	67		Yes	1
cp1256_general_nopad_ci	cp1256	1081		Yes	1
cp1256_nopad_bin	cp1256	1091		Yes	1
cp1257_lithuanian_ci	cp1257	29		Yes	1
cp1257_bin	cp1257	58		Yes	1
cp1257_general_ci	cp1257	59	Yes	Yes	1
cp1257_nopad_bin	cp1257	1082		Yes	1
cp1257_general_nopad_ci	cp1257	1083		Yes	1
utf32_general_ci	utf32	60	Yes	Yes	1
utf32_bin	utf32	61		Yes	1
utf32_unicode_ci	utf32	160		Yes	8
utf32_icelandic_ci	utf32	161		Yes	8
utf32_latvian_ci	utf32	162		Yes	8
utf32_romanian_ci	utf32	163		Yes	8
utf32_slovenian_ci	utf32	164		Yes	8
utf32_polish_ci	utf32	165		Yes	8
utf32_estonian_ci	utf32	166		Yes	8
utf32_spanish_ci	utf32	167		Yes	8
utf32_swedish_ci	utf32	168		Yes	8
utf32_turkish_ci	utf32	169		Yes	8
utf32_czech_ci	utf32	170		Yes	8
utf32_danish_ci	utf32	171		Yes	8
utf32_lithuanian_ci	utf32	172		Yes	8
utf32_slovak_ci	utf32	173		Yes	8
utf32_spanish2_ci	utf32	174		Yes	8
utf32_roman_ci	utf32	175		Yes	8
utf32_persian_ci	utf32	176		Yes	8
utf32_esperanto_ci	utf32	177		Yes	8
utf32_hungarian_ci	utf32	178		Yes	8
utf32_sinhala_ci	utf32	179		Yes	8
utf32_german2_ci	utf32	180		Yes	8
utf32_croatian_mysql561_ci	utf32	181		Yes	8
utf32_unicode_520_ci	utf32	182		Yes	8
utf32_vietnamese_ci	utf32	183		Yes	8
utf32_croatian_ci	utf32	736		Yes	8
utf32_myanmar_ci	utf32	737		Yes	8
utf32_thai_520_w2	utf32	738		Yes	4
utf32_general_nopad_ci	utf32	1084		Yes	1
utf32_nopad_bin	utf32	1085		Yes	1
utf32_unicode_nopad_ci	utf32	1184		Yes	8
utf32_unicode_520_nopad_ci	utf32	1206		Yes	8
binary	binary	63	Yes	Yes	1
geostd8_general_ci	geostd8	92	Yes	Yes	1
geostd8_bin	geostd8	93		Yes	1
geostd8_general_nopad_ci	geostd8	1116		Yes	1
geostd8_nopad_bin	geostd8	1117		Yes	1
cp932_japanese_ci	cp932	95	Yes	Yes	1
cp932_bin	cp932	96		Yes	1
cp932_japanese_nopad_ci	cp932	1119		Yes	1
cp932_nopad_bin	cp932	1120		Yes	1

```
| eucjpms_japanese_ci              | eucjpms   |   97 | Yes      | Yes      |       1 |
| eucjpms_bin                      | eucjpms   |   98 |          | Yes      |       1 |
| eucjpms_japanese_nopad_ci        | eucjpms   | 1121 |          | Yes      |       1 |
| eucjpms_nopad_bin                | eucjpms   | 1122 |          | Yes      |       1 |
+----------------------------------+-----------+------+----------+----------+---------+
506 rows in set (0.002 sec)
```

This is from MariaDB 10.11.5 including the UCA-14.0.0 collations added in MariaDB 10.10.1.

Before MariaDB 10.6.1, the `utf8mb3*` collations listed above were named `utf8*`.

Case Sensitivity

A '`ci`' at the end of a collation name indicates the collation is case insensitive. A '`cs`' at the end of a collation name indicates the collation is case sensitive.

NO PAD Collations

`NO PAD` collations regard trailing spaces as normal characters. You can get a list of all of these by querying the Information Schema COLLATIONS Table as follows:

```
SELECT collation_name FROM information_schema.COLLATIONS
WHERE collation_name LIKE "%nopad%";
+------------------------------+
| collation_name               |
+------------------------------+
| big5_chinese_nopad_ci        |
| big5_nopad_bin               |
...
```

Accent Insensitivity

An accent insensitive collation is one where the accented and unaccented versions of a letter are considered to be identical for sorting purposes.

MariaDB 10.10 added the accent insensitivity flag, and new collations are marked with '_ai' or '_as' in the name to indicate this, for example:

```
...
| uca1400_spanish2_ai_ci           |
| uca1400_spanish2_ai_cs           |
| uca1400_spanish2_as_ci           |
| uca1400_spanish2_as_cs           |
...
```

Changes

- MariaDB 10.10 added 184 UCA-14.0.0 collations. Unicode-14.0.0 was released in September 2021. They contain 939 built-in contractions. Old collations based on UCA-4.0.0 and UCA-5.2.0 did not support built-in contractions. This is a step towards better Unicode Collation Algorithm compliance. With built-in contractions, some languages (e.g. Thai) won't need specific collations and will just work with the default "root" collation.
- MariaDB 10.6.1 changed the `utf8` character set by default to be an alias for utf8mb3 rather than the other way around. It can be set to imply `utf8mb4` by changing the value of the old_mode system variable.
- MariaDB 10.2.2 added 88 `NO PAD` collations.
- MariaDB 10.1.15 added the `utf8_thai_520_w2`, `utf8mb4_thai_520_w2`, `ucs2_thai_520_w2`, `utf16_thai_520_w2` and `utf32_thai_520_w2` collations.
- MariaDB 10.0.7 added the `utf8_myanmar_ci`, `ucs2_myanmar_ci`, `utf8mb4_myanmar_ci`, `utf16_myanmar_ci` and `utf32_myanmar_ci` collations.
- MariaDB 10.0.5 added the `utf8_german2_ci`, `utf8mb4_german2_ci`, `ucs2_german2_ci`, `utf16_german2_ci` and `utf32_german2_ci` collations.
- MariaDB 5.1.41 added a Croatian collation patch from Alexander Barkov to fix some problems with the Croatian character set and `LIKE` queries. This patch added `utf8_croatian_ci` and `ucs2_croatian_ci` collations to MariaDB.

5.1.3 Date and Time Data Types

DATE

The date type YYYY-MM-DD.

TIME

Time format HH:MM:SS.ssssss

DATETIME

Date and time combination displayed as YYYY-MM-DD HH:MM:SS.

TIMESTAMP

YYYY-MM-DD HH:MM:SS

YEAR Data Type

A four-digit year.

There are 3 related questions.

5.1.3.1 DATE

Contents

Syntax

```
DATE
```

Description

A date. The supported range is '`1000-01-01`' to '`9999-12-31`'. MariaDB displays `DATE` values in '`YYYY-MM-DD`' format, but can be assigned dates in looser formats, including strings or numbers, as long as they make sense. These include a short year, `YY-MM-DD`, no delimiters, `YYMMDD`, or any other acceptable delimiter, for example `YYYY/MM/DD`. For details, see date and time literals.

'`0000-00-00`' is a permitted special value (zero-date), unless the NO_ZERO_DATE SQL_MODE is used. Also, individual components of a date can be set to 0 (for example: '`2015-00-12`'), unless the NO_ZERO_IN_DATE SQL_MODE is used. In many cases, the result of en expression involving a zero-date, or a date with zero-parts, is `NULL`. If the ALLOW_INVALID_DATES SQL_MODE is enabled, if the day part is in the range between 1 and 31, the date does not produce any error, even for months that have less than 31 days.

Oracle Mode

MariaDB starting with 10.3

In Oracle mode from MariaDB 10.3, `DATE` with a time portion is a synonym for DATETIME. See also mariadb_schema.

Examples

```
CREATE TABLE t1 (d DATE);

INSERT INTO t1 VALUES ("2010-01-12"), ("2011-2-28"), ('120314'),('13*04*21');

SELECT * FROM t1;
+------------+
| d          |
+------------+
| 2010-01-12 |
| 2011-02-28 |
| 2012-03-14 |
| 2013-04-21 |
+------------+
```

5.1.3.2 TIME

Syntax

```
TIME [(<microsecond precision>)]
```

Contents

Description

A time. The range is `'-838:59:59.999999'` to `'838:59:59.999999'`. Microsecond precision can be from 0-6; if not specified 0 is used. Microseconds have been available since MariaDB 5.3.

MariaDB displays `TIME` values in `'HH:MM:SS.ssssss'` format, but allows assignment of times in looser formats, including 'D HH:MM:SS', 'HH:MM:SS', 'HH:MM', 'D HH:MM', 'D HH', 'SS', or 'HHMMSS', as well as permitting dropping of any leading zeros when a delimiter is provided, for example '3:9:10'. For details, see date and time literals.

MariaDB 10.1.2 introduced the --mysql56-temporal-format option, on by default, which allows MariaDB to store TIMEs using the same low-level format MySQL 5.6 uses.

Internal Format

In MariaDB 10.1.2 a new temporal format was introduced from MySQL 5.6 that alters how the `TIME`, `DATETIME` and `TIMESTAMP` columns operate at lower levels. These changes allow these temporal data types to have fractional parts and negative values. You can disable this feature using the mysql56_temporal_format system variable.

Tables that include `TIMESTAMP` values that were created on an older version of MariaDB or that were created while the mysql56_temporal_format system variable was disabled continue to store data using the older data type format.

In order to update table columns from the older format to the newer format, execute an ALTER TABLE... MODIFY COLUMN statement that changes the column to the *same* data type. This change may be needed if you want to export the table's tablespace and import it onto a server that has `mysql56_temporal_format=ON` set (see MDEV-15225).

For instance, if you have a `TIME` column in your table:

```
SHOW VARIABLES LIKE 'mysql56_temporal_format';

+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| mysql56_temporal_format | ON    |
+--------------------------+-------+

ALTER TABLE example_table MODIFY ts_col TIME;
```

When MariaDB executes the `ALTER TABLE` statement, it converts the data from the older temporal format to the newer one.

In the event that you have several tables and columns using temporal data types that you want to switch over to the new format, make sure the system variable is enabled, then perform a dump and restore using `mariadb-dump`. The columns using relevant temporal data types are restored using the new temporal format.

Starting from MariaDB 10.5.1 columns with old temporal formats are marked with a `/* mariadb-5.3 */` comment in the output of SHOW CREATE TABLE, SHOW COLUMNS, DESCRIBE statements, as well as in the `COLUMN_TYPE` column of the INFORMATION_SCHEMA.COLUMNS Table.

```
SHOW CREATE TABLE mariadb5312_time\G
*************************** 1. row ***************************
       Table: mariadb5312_time
Create Table: CREATE TABLE `mariadb5312_time` (
  `t0` time /* mariadb-5.3 */ DEFAULT NULL,
  `t6` time(6) /* mariadb-5.3 */ DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1
```

Note, columns with the current format are not marked with a comment.

Examples

```
INSERT INTO time VALUES ('90:00:00'), ('800:00:00'), (800), (22), (151413), ('9:6:3'), ('12 09');

SELECT * FROM time;
+-----------+
| t         |
+-----------+
| 90:00:00  |
| 800:00:00 |
| 00:08:00  |
| 00:00:22  |
| 15:14:13  |
| 09:06:03  |
| 297:00:00 |
+-----------+
```

5.1.3.3 DATETIME

Contents

Syntax

```
DATETIME [(microsecond precision)]
```

Description

A date and time combination.

MariaDB displays `DATETIME` values in '`YYYY-MM-DD HH:MM:SS.ffffff`' format, but allows assignment of values to `DATETIME` columns using either strings or numbers. For details, see date and time literals.

DATETIME columns also accept CURRENT_TIMESTAMP as the default value.

MariaDB 10.1.2 introduced the --mysql56-temporal-format option, on by default, which allows MariaDB to store DATETMEs using the same low-level format MySQL 5.6 uses. For more information, see Internal Format, below.

For storage requirements, see Data Type Storage Requirements.

Supported Values

MariaDB stores values that use the `DATETIME` data type in a format that supports values between `1000-01-01 00:00:00.000000` and `9999-12-31 23:59:59.999999`.

MariaDB can also store microseconds with a precision between 0 and 6. If no microsecond precision is specified, then 0 is used by default.

MariaDB also supports '`0000-00-00`' as a special *zero-date* value, unless NO_ZERO_DATE is specified in the SQL_MODE. Similarly, individual components of a date can be set to `0` (for example: '`2015-00-12`'), unless NO_ZERO_IN_DATE is specified in the SQL_MODE. In many cases, the result of en expression involving a zero-date, or a date with zero-parts, is `NULL`. If the ALLOW_INVALID_DATES SQL_MODE is enabled, if the day part is in the range between 1 and 31, the date does not produce any error, even for months that have less than 31 days.

Oracle Mode

MariaDB starting with 10.3

In Oracle mode from MariaDB 10.3, `DATE` with a time portion is a synonym for `DATETIME`. See also mariadb_schema.

Internal Format

In MariaDB 10.1.2 a new temporal format was introduced from MySQL 5.6 that alters how the `TIME`, `DATETIME` and `TIMESTAMP` columns operate at lower levels. These changes allow these temporal data types to have fractional parts and negative values. You can disable this feature using the mysql56_temporal_format system variable.

Tables that include `TIMESTAMP` values that were created on an older version of MariaDB or that were created while the mysql56_temporal_format system variable was disabled continue to store data using the older data type format.

In order to update table columns from the older format to the newer format, execute an ALTER TABLE... MODIFY COLUMN statement that changes the column to the *same* data type. This change may be needed if you want to export the table's tablespace and import it onto a server that has `mysql56_temporal_format=ON` set (see MDEV-15225).

For instance, if you have a `DATETIME` column in your table:

```
SHOW VARIABLES LIKE 'mysql56_temporal_format';

+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| mysql56_temporal_format | ON    |
+--------------------------+-------+

ALTER TABLE example_table MODIFY ts_col DATETIME;
```

When MariaDB executes the ALTER TABLE statement, it converts the data from the older temporal format to the newer one.

In the event that you have several tables and columns using temporal data types that you want to switch over to the new format, make sure the system variable is enabled, then perform a dump and restore using `mysqldump`. The columns using relevant temporal data types are restored using the new temporal format.

Starting from MariaDB 10.5.1 columns with old temporal formats are marked with a `/* mariadb-5.3 */` comment in the output of SHOW CREATE TABLE, SHOW COLUMNS, DESCRIBE statements, as well as in the `COLUMN_TYPE` column of the INFORMATION_SCHEMA.COLUMNS Table.

```
SHOW CREATE TABLE mariadb5312_datetime\G
*************************** 1. row ***************************
       Table: mariadb5312_datetime
Create Table: CREATE TABLE `mariadb5312_datetime` (
  `dt0` datetime /* mariadb-5.3 */ DEFAULT NULL,
  `dt6` datetime(6) /* mariadb-5.3 */ DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1
```

Examples

```
CREATE TABLE t1 (d DATETIME);

INSERT INTO t1 VALUES ("2011-03-11"), ("2012-04-19 13:08:22"),
 ("2013-07-18 13:44:22.123456");

SELECT * FROM t1;
+---------------------+
| d                   |
+---------------------+
| 2011-03-11 00:00:00 |
| 2012-04-19 13:08:22 |
| 2013-07-18 13:44:22 |
+---------------------+
```

```
CREATE TABLE t2 (d DATETIME(6));

INSERT INTO t2 VALUES ("2011-03-11"), ("2012-04-19 13:08:22"),
 ("2013-07-18 13:44:22.123456");

SELECT * FROM t2;
+----------------------------+
| d                          |
+----------------------------+
| 2011-03-11 00:00:00.000000 |
| 2012-04-19 13:08:22.000000 |
| 2013-07-18 13:44:22.123456 |
+----------------------------++
```

Strings used in datetime context are automatically converted to datetime(6). If you want to have a datetime without seconds, you should use CONVERT(..,datetime).

```
SELECT CONVERT('2007-11-30 10:30:19',datetime);
+---------------------------------------+
| CONVERT('2007-11-30 10:30:19',datetime) |
+---------------------------------------+
| 2007-11-30 10:30:19                   |
+---------------------------------------+

SELECT CONVERT('2007-11-30 10:30:19',datetime(6));
+-------------------------------------------+
| CONVERT('2007-11-30 10:30:19',datetime(6)) |
+-------------------------------------------+
| 2007-11-30 10:30:19.000000                |
+-------------------------------------------+
```

5.1.3.4 TIMESTAMP

Syntax

```
TIMESTAMP [(<microsecond precision)]
```

Contents

Description

A timestamp in the format `YYYY-MM-DD HH:MM:SS.ffffff`.

The timestamp field is generally used to define at which moment in time a row was added or updated and by default will automatically be assigned the current datetime when a record is inserted or updated. The automatic properties only apply to the first TIMESTAMP in the record; subsequent TIMESTAMP columns will not be changed.

MariaDB includes the --mysql56-temporal-format option, on by default, which allows MariaDB to store TIMESTAMPs using the same low-level format MySQL 5.6 uses.

For more information, see Internal Format.

Supported Values

MariaDB stores values that use the `TIMESTAMP` data type as the number of seconds since '1970-01-01 00:00:00' (UTC). This means that the `TIMESTAMP` data type can hold values between '1970-01-01 00:00:01' (UTC) and '2038-01-19 03:14:07' (UTC).

MariaDB can also store microseconds with a precision between 0 and 6. If no microsecond precision is specified, then 0 is used by default.

Automatic Values

MariaDB has special behavior for the first column that uses the `TIMESTAMP` data type in a specific table when the system variable explicit_defaults_for_timestamp is not set (which is the default until MariaDB 10.10). For the first column that uses the `TIMESTAMP` data type in a specific table, MariaDB automatically assigns the following properties to the column:

- `DEFAULT CURRENT_TIMESTAMP`
- `ON UPDATE CURRENT_TIMESTAMP`

This means that if the column is not explicitly assigned a value in an `INSERT` or `UPDATE` query, then MariaDB will automatically initialize the column's value with the current date and time.

This automatic initialization for `INSERT` and `UPDATE` queries can also be **explicitly enabled** for a column that uses the `TIMESTAMP` data type by specifying the `DEFAULT CURRENT_TIMESTAMP` and `ON UPDATE CURRENT_TIMESTAMP` clauses for the column. In these clauses, any synonym of `CURRENT_TIMESTAMP` is accepted, including `CURRENT_TIMESTAMP()`, `NOW()`, `LOCALTIME`, `LOCALTIME()`, `LOCALTIMESTAMP`, and `LOCALTIMESTAMP()`.

This automatic initialization for `INSERT` queries can also be **explicitly disabled** for a column that uses the `TIMESTAMP` data type by specifying a constant `DEFAULT` value. For example, `DEFAULT 0`.

This automatic initialization for `UPDATE` queries can also be **explicitly disabled** for a column that uses the `TIMESTAMP` data type by specifying a `DEFAULT` clause for the column, but no `ON UPDATE` clause. If a `DEFAULT` clause is explicitly specified for a column that uses the `TIMESTAMP` data type, but an `ON UPDATE` clause is not specified for the column, then the timestamp value will not automatically change when an `UPDATE` statement is executed.

MariaDB also has special behavior if `NULL` is assigned to column that uses the `TIMESTAMP` data type. If the column is assigned the `NULL` value in an `INSERT` or `UPDATE` query, then MariaDB will automatically initialize the column's value with the current date and time. For details, see NULL values in MariaDB.

This automatic initialization for `NULL` values can also be **explicitly disabled** for a column that uses the `TIMESTAMP` data type by specifying the `NULL` attribute for the column. In this case, if the column's value is set to `NULL`, then the column's value will actually be set to `NULL`.

Time Zones

If a column uses the `TIMESTAMP` data type, then any inserted values are converted from the session's time zone to Coordinated Universal Time (UTC) when stored, and converted back to the session's time zone when retrieved.

MariaDB validates `TIMESTAMP` literals against the session's time zone. For example, if a specific time range never occurred in a specific time zone due to daylight savings time, then `TIMESTAMP` values within that range would be invalid for that time zone.

MariaDB does not currently store any time zone identifier with the value of the `TIMESTAMP` data type. See MDEV-10018 for more information.

MariaDB does not currently support time zone literals that contain time zone identifiers. See MDEV-11829 for more information.

Limitations

- Because the TIMESTAMP value is stored as Epoch Seconds, the timestamp value '1970-01-01 00:00:00' (UTC) is reserved since the second #0 is used to represent '0000-00-00 00:00:00'.
- In MariaDB 5.5 and before there could only be one TIMESTAMP column per table that had CURRENT_TIMESTAMP defined as its default value. This limit has no longer applied since MariaDB 10.0.

SQL_MODE=MAXDB

If the SQL_MODE is set to `MAXDB`, TIMESTAMP fields will be silently converted to DATETIME.

Internal Format

In MariaDB 10.1.2 a new temporal format was introduced from MySQL 5.6 that alters how the `TIME`, `DATETIME` and `TIMESTAMP` columns operate at lower levels. These changes allow these temporal data types to have fractional parts and negative values. You can disable this feature using the `mysql56_temporal_format` system variable.

Tables that include `TIMESTAMP` values that were created on an older version of MariaDB or that were created while the `mysql56_temporal_format` system variable was disabled continue to store data using the older data type format.

In order to update table columns from the older format to the newer format, execute an `ALTER TABLE... MODIFY COLUMN` statement that changes the column to the *same* data type. This change may be needed if you want to export the table's tablespace and import it onto a server that has `mysql56_temporal_format=ON` set (see MDEV-15225).

For instance, if you have a `TIMESTAMP` column in your table:

```
SHOW VARIABLES LIKE 'mysql56_temporal_format';

+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| mysql56_temporal_format | ON    |
+--------------------------+-------+

ALTER TABLE example_table MODIFY ts_col TIMESTAMP;
```

When MariaDB executes the `ALTER TABLE` statement, it converts the data from the older temporal format to the newer one.

In the event that you have several tables and columns using temporal data types that you want to switch over to the new format, make sure the system variable is enabled, then perform a dump and restore using `mysqldump`. The columns using relevant temporal data types are restored using the new temporal format.

Starting from MariaDB 10.5.1 columns with old temporal formats are marked with a `/* mariadb-5.3 */` comment in the output of `SHOW CREATE TABLE`, `SHOW COLUMNS`, `DESCRIBE` statements, as well as in the `COLUMN_TYPE` column of the `INFORMATION_SCHEMA.COLUMNS Table`.

```
SHOW CREATE TABLE mariadb5312_timestamp\G
*************************** 1. row ***************************
       Table: mariadb5312_timestamp
Create Table: CREATE TABLE `mariadb5312_timestamp` (
  `ts0` timestamp /* mariadb-5.3 */ NOT NULL DEFAULT current_timestamp() ON UPDATE current_timest
  `ts6` timestamp(6) /* mariadb-5.3 */ NOT NULL DEFAULT '0000-00-00 00:00:00.000000'
) ENGINE=MyISAM DEFAULT CHARSET=latin1
```

Note: Prior to MySQL 4.1 a different format for the TIMESTAMP datatype was used. This format is unsupported in MariaDB 5.1 and upwards.

Examples

```
CREATE TABLE t (id INT, ts TIMESTAMP);

DESC t;
+-------+-----------+------+-----+-------------------+-----------------------------+
| Field | Type      | Null | Key | Default           | Extra                       |
+-------+-----------+------+-----+-------------------+-----------------------------+
| id    | int(11)   | YES  |     | NULL              |                             |
| ts    | timestamp | NO   |     | CURRENT_TIMESTAMP | on update CURRENT_TIMESTAMP |
+-------+-----------+------+-----+-------------------+-----------------------------+

INSERT INTO t(id)  VALUES (1),(2);

SELECT * FROM t;
+------+---------------------+
| id   | ts                  |
+------+---------------------+
|    1 | 2013-07-22 12:50:05 |
|    2 | 2013-07-22 12:50:05 |
+------+---------------------+

INSERT INTO t  VALUES (3,NULL),(4,'2001-07-22 12:12:12');

SELECT * FROM t;
+------+---------------------+
| id   | ts                  |
+------+---------------------+
|    1 | 2013-07-22 12:50:05 |
|    2 | 2013-07-22 12:50:05 |
|    3 | 2013-07-22 12:51:56 |
|    4 | 2001-07-22 12:12:12 |
+------+---------------------+
```

Converting to Unix epoch:

```
SELECT ts, UNIX_TIMESTAMP(ts) FROM t;
+---------------------+--------------------+
| ts                  | UNIX_TIMESTAMP(ts) |
+---------------------+--------------------+
| 2013-07-22 12:50:05 |         1374490205 |
| 2013-07-22 12:50:05 |         1374490205 |
| 2013-07-22 12:51:56 |         1374490316 |
| 2001-07-22 12:12:12 |          995796732 |
+---------------------+--------------------+
```

Update also changes the timestamp:

```
UPDATE t set id=5 WHERE id=1;

SELECT * FROM t;
+------+---------------------+
| id   | ts                  |
+------+---------------------+
|    5 | 2013-07-22 14:52:33 |
|    2 | 2013-07-22 12:50:05 |
|    3 | 2013-07-22 12:51:56 |
|    4 | 2001-07-22 12:12:12 |
+------+---------------------+
```

Default NULL:

```
CREATE TABLE t2 (id INT, ts TIMESTAMP NULL ON UPDATE CURRENT_TIMESTAMP);

INSERT INTO t(id)  VALUES (1),(2);

SELECT * FROM t2;

INSERT INTO t2(id)  VALUES (1),(2);

SELECT * FROM t2;
+------+------+
| id   | ts   |
+------+------+
|    1 | NULL |
|    2 | NULL |
+------+------+

UPDATE t2 SET id=3 WHERE id=1;

SELECT * FROM t2;
+------+---------------------+
| id   | ts                  |
+------+---------------------+
|    3 | 2013-07-22 15:32:22 |
|    2 | NULL                |
+------+---------------------+
```

Only the first timestamp is automatically inserted and updated:

```
CREATE TABLE t3 (id INT, ts1 TIMESTAMP, ts2 TIMESTAMP);

INSERT INTO t3(id)  VALUES (1),(2);

SELECT * FROM t3;
+------+---------------------+---------------------+
| id   | ts1                 | ts2                 |
+------+---------------------+---------------------+
|    1 | 2013-07-22 15:35:07 | 0000-00-00 00:00:00 |
|    2 | 2013-07-22 15:35:07 | 0000-00-00 00:00:00 |
+------+---------------------+---------------------+

DESC t3;
+-------+-----------+------+-----+---------------------+-----------------------------+
| Field | Type      | Null | Key | Default             | Extra                       |
+-------+-----------+------+-----+---------------------+-----------------------------+
| id    | int(11)   | YES  |     | NULL                |                             |
| ts1   | timestamp | NO   |     | CURRENT_TIMESTAMP   | on update CURRENT_TIMESTAMP |
| ts2   | timestamp | NO   |     | 0000-00-00 00:00:00 |                             |
+-------+-----------+------+-----+---------------------+-----------------------------+
```

Explicitly setting a timestamp with the CURRENT_TIMESTAMP function:

```
INSERT INTO t3(id,ts2)  VALUES (3,CURRENT_TIMESTAMP());

SELECT * FROM t3;
+------+---------------------+---------------------+
| id   | ts1                 | ts2                 |
+------+---------------------+---------------------+
|    1 | 2013-07-22 15:35:07 | 0000-00-00 00:00:00 |
|    2 | 2013-07-22 15:35:07 | 0000-00-00 00:00:00 |
|    3 | 2013-07-22 15:38:52 | 2013-07-22 15:38:52 |
+------+---------------------+---------------------+
```

Specifying the timestamp as NOT NULL:

```
CREATE TABLE t4 (id INT, ts TIMESTAMP NOT NULL);

INSERT INTO t4(id)  VALUES (1);
SELECT SLEEP(1);
INSERT INTO t4(id,ts) VALUES (2,NULL);

SELECT * FROM t4;
```

5.1.3.5 YEAR Data Type

Syntax

```
YEAR[(4)]
```

Description

A year in two-digit or four-digit format. The default is four-digit format. Note that the two-digit format has been deprecated since MariaDB 5.5.27.

In four-digit format, the allowable values are 1901 to 2155, and 0000. In two-digit format, the allowable values are 70 to 69, representing years from 1970 to 2069. MariaDB displays YEAR values in YYYY format, but allows you to assign values to YEAR columns using either strings or numbers.

Inserting numeric zero has a different result for YEAR(4) and YEAR(2). For YEAR(2), the value `00` reflects the year 2000. For YEAR(4), the value `0000` reflects the year zero. This only applies to numeric zero. String zero always reflects the year 2000.

Examples

Accepting a string or a number:

```
CREATE TABLE y(y YEAR);

INSERT INTO y VALUES (1990),('2012');

SELECT * FROM y;
+------+
| y    |
+------+
| 1990 |
| 2012 |
+------+
```

With strict_mode set, the default from MariaDB 10.2.4:

Out of range:

```
INSERT INTO y VALUES (1005),('3080');
ERROR 1264 (22003): Out of range value for column 'y' at row 1

INSERT INTO y VALUES ('2013-12-12');
ERROR 1265 (01000): Data truncated for column 'y' at row 1

SELECT * FROM y;
+------+
| y    |
+------+
| 1990 |
| 2012 |
+------+
```

With strict_mode unset, the default until MariaDB 10.2.3:

Out of range:

```
INSERT INTO y VALUES (1005),('3080');
Query OK, 2 rows affected, 2 warnings (0.05 sec)
Records: 2  Duplicates: 0  Warnings: 2

SHOW WARNINGS;
+---------+------+-----------------------------------------+
| Level   | Code | Message                                 |
+---------+------+-----------------------------------------+
| Warning | 1264 | Out of range value for column 'y' at row 1 |
| Warning | 1264 | Out of range value for column 'y' at row 2 |
+---------+------+-----------------------------------------+

SELECT * FROM y;
+------+
| y    |
+------+
| 1990 |
| 2012 |
| 0000 |
| 0000 |
+------+
```

Truncating:

```
INSERT INTO y VALUES ('2013-12-12');
Query OK, 1 row affected, 1 warning (0.05 sec)

SHOW WARNINGS;
+---------+------+---------------------------------------+
| Level   | Code | Message                               |
+---------+------+---------------------------------------+
| Warning | 1265 | Data truncated for column 'y' at row 1 |
+---------+------+---------------------------------------+

SELECT * FROM y;
+------+
| y    |
+------+
| 1990 |
| 2012 |
| 0000 |
| 0000 |
| 2013 |
+------+
```

Difference between YEAR(2) and YEAR(4), and string and numeric zero:

```
CREATE TABLE y2(y YEAR(4), y2 YEAR(2));
Query OK, 0 rows affected, 1 warning (0.40 sec)

Note (Code 1287): 'YEAR(2)' is deprecated and will be removed in a future release.
 Please use YEAR(4) instead

INSERT INTO y2 VALUES(0,0),('0','0');

SELECT YEAR(y),YEAR(y2) FROM y2;
+---------+----------+
| YEAR(y) | YEAR(y2) |
+---------+----------+
|       0 |     2000 |
|    2000 |     2000 |
+---------+----------+
```

1.1.3.3 Geometry Types

5.1.5 AUTO_INCREMENT

Contents

Description

The `AUTO_INCREMENT` attribute can be used to generate a unique identity for new rows. When you insert a new record to the table (or upon adding an AUTO_INCREMENT attribute with the ALTER TABLE statement), and the auto_increment field is NULL or DEFAULT (in the case of an INSERT), the value will automatically be incremented. This also applies to 0, unless the NO_AUTO_VALUE_ON_ZERO SQL_MODE is enabled.

`AUTO_INCREMENT` columns start from 1 by default. The automatically generated value can never be lower than 0.

Each table can have only one `AUTO_INCREMENT` column. It must defined as a key (not necessarily the `PRIMARY KEY` or `UNIQUE` key). In some storage engines (including the default InnoDB), if the key consists of multiple columns, the `AUTO_INCREMENT` column must be the first column. Storage engines that permit the column to be placed elsewhere are Aria, MyISAM, MERGE, Spider, TokuDB, BLACKHOLE, FederatedX and Federated.

```
CREATE TABLE animals (
     id MEDIUMINT NOT NULL AUTO_INCREMENT,
     name CHAR(30) NOT NULL,
     PRIMARY KEY (id)
 );

INSERT INTO animals (name) VALUES
    ('dog'),('cat'),('penguin'),
    ('fox'),('whale'),('ostrich');
```

```
SELECT * FROM animals;
+----+---------+
| id | name    |
+----+---------+
|  1 | dog     |
|  2 | cat     |
|  3 | penguin |
|  4 | fox     |
|  5 | whale   |
|  6 | ostrich |
+----+---------+
```

`SERIAL` is an alias for `BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE`.

```
CREATE TABLE t (id SERIAL, c CHAR(1)) ENGINE=InnoDB;

SHOW CREATE TABLE t \G
*************************** 1. row ***************************
       Table: t
Create Table: CREATE TABLE `t` (
  `id` bigint(20) unsigned NOT NULL AUTO_INCREMENT,
  `c` char(1) DEFAULT NULL,
  UNIQUE KEY `id` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

Setting or Changing the Auto_Increment Value

You can use an `ALTER TABLE` statement to assign a new value to the `auto_increment` table option, or set the insert_id server system variable to change the next `AUTO_INCREMENT` value inserted by the current session.

`LAST_INSERT_ID()` can be used to see the last `AUTO_INCREMENT` value inserted by the current session.

```
ALTER TABLE animals AUTO_INCREMENT=8;

INSERT INTO animals (name) VALUES ('aardvark');

SELECT * FROM animals;
+----+-----------+
| id | name      |
+----+-----------+
|  1 | dog       |
|  2 | cat       |
|  3 | penguin   |
|  4 | fox       |
|  5 | whale     |
|  6 | ostrich   |
|  8 | aardvark  |
+----+-----------+

SET insert_id=12;

INSERT INTO animals (name) VALUES ('gorilla');

SELECT * FROM animals;
+----+-----------+
| id | name      |
+----+-----------+
|  1 | dog       |
|  2 | cat       |
|  3 | penguin   |
|  4 | fox       |
|  5 | whale     |
|  6 | ostrich   |
|  8 | aardvark  |
| 12 | gorilla   |
+----+-----------+
```

InnoDB

AUTO_INCREMENT is persistent in InnoDB. Prior to MariaDB 10.2.3, InnoDB used an auto-increment counter that was stored in memory. When the server restarted, the counter was re-initialized to the highest value used in the table, which canceled the effects of any AUTO_INCREMENT = N option in the table statements).

See also AUTO_INCREMENT Handling in InnoDB.

Setting Explicit Values

It is possible to specify a value for an `AUTO_INCREMENT` column. If the key is primary or unique, the value must not already exist in the key.

If the new value is higher than the current maximum value, the `AUTO_INCREMENT` value is updated, so the next value will be higher. If the new value is lower than the current maximum value, the `AUTO_INCREMENT` value remains unchanged.

The following example demonstrates these behaviors:

```
CREATE TABLE t (id INTEGER UNSIGNED AUTO_INCREMENT PRIMARY KEY) ENGINE = InnoDB;

INSERT INTO t VALUES (NULL);
SELECT id FROM t;
+----+
| id |
+----+
|  1 |
+----+

INSERT INTO t VALUES (10); -- higher value
SELECT id FROM t;
+----+
| id |
+----+
|  1 |
| 10 |
+----+

INSERT INTO t VALUES (2); -- lower value
INSERT INTO t VALUES (NULL); -- auto value
SELECT id FROM t;
+----+
| id |
+----+
|  1 |
|  2 |
| 10 |
| 11 |
+----+
```

The ARCHIVE storage engine does not allow to insert a value that is lower than the current maximum.

Missing Values

An AUTO_INCREMENT column normally has missing values. This happens because if a row is deleted, or an AUTO_INCREMENT value is explicitly updated, old values are never re-used. The REPLACE statement also deletes a row, and its value is wasted. With InnoDB, values can be reserved by a transaction; but if the transaction fails (for example, because of a ROLLBACK) the reserved value will be lost.

Thus AUTO_INCREMENT values can be used to sort results in a chronological order, but not to create a numeric sequence.

Replication

To make master-master or Galera safe to use `AUTO_INCREMENT` one should use the system variables auto_increment_increment and auto_increment_offset to generate unique values for each server.

```
SET @@auto_increment_increment=3;

SHOW VARIABLES LIKE 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 3     |
| auto_increment_offset    | 1     |
+--------------------------+-------+

CREATE TABLE t (c INT NOT NULL AUTO_INCREMENT PRIMARY KEY);

INSERT INTO t VALUES (NULL), (NULL), (NULL);

SELECT * FROM t;
+---+
| c |
+---+
| 1 |
| 4 |
| 7 |
+---+

CREATE TABLE t2 (c INT NOT NULL AUTO_INCREMENT PRIMARY KEY);

SET @@auto_increment_offset=2;

SHOW VARIABLES LIKE 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 3     |
| auto_increment_offset    | 2     |
+--------------------------+-------+

INSERT INTO t2 VALUES (NULL), (NULL), (NULL);

SELECT * FROM t2;
+---+
| c |
+---+
| 2 |
| 5 |
| 8 |
+---+
```

If `auto_increment_offset` is larger than `auto_increment_increment`, the value of `auto_increment_offset` is ignored, and the offset reverts to the default of 1 instead:

```
SET @@auto_increment_offset=5;

SHOW VARIABLES LIKE 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 3     |
| auto_increment_offset    | 5     |
+--------------------------+-------+

CREATE TABLE t3 (c INT NOT NULL AUTO_INCREMENT PRIMARY KEY);

INSERT INTO t3 VALUES (NULL), (NULL), (NULL);

SELECT * FROM t3;
+---+
| c |
+---+
| 1 |
| 4 |
| 5 |
+---+

+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 3     |
| auto_increment_offset    | 3     |
+--------------------------+-------+

INSERT INTO t4 VALUES (NULL), (NULL), (NULL);

SELECT * FROM t4;
+---+
| c |
+---+
| 3 |
| 6 |
| 9 |
+---+
```

CHECK Constraints, DEFAULT Values and Virtual Columns

auto_increment columns are not permitted in CHECK constraints, DEFAULT value expressions and virtual columns. They were permitted until MariaDB 10.2.6, but did not work correctly. See MDEV-11117.

Generating Auto_Increment Values When Adding the Attribute

```
CREATE OR REPLACE TABLE t1 (a INT);
INSERT t1 VALUES (0),(0),(0);
ALTER TABLE t1 MODIFY a INT NOT NULL AUTO_INCREMENT PRIMARY KEY;
SELECT * FROM t1;
+---+
| a |
+---+
| 1 |
| 2 |
| 3 |
+---+
```

```
CREATE OR REPLACE TABLE t1 (a INT);
INSERT t1 VALUES (5),(0),(8),(0);
ALTER TABLE t1 MODIFY a INT NOT NULL AUTO_INCREMENT PRIMARY KEY;
SELECT * FROM t1;
+---+
| a |
+---+
| 5 |
| 6 |
| 8 |
| 9 |
+---+
```

If the NO_AUTO_VALUE_ON_ZERO SQL_MODE is set, zero values will not be automatically incremented:

```
SET SQL_MODE='no_auto_value_on_zero';
CREATE OR REPLACE TABLE t1 (a INT);
INSERT t1 VALUES (3), (0);
ALTER TABLE t1 MODIFY a INT NOT NULL AUTO_INCREMENT PRIMARY KEY;
SELECT * FROM t1;
+---+
| a |
+---+
| 0 |
| 3 |
+---+
```

5.1.2.24 Data Type Storage Requirements

5.1.7 AUTO_INCREMENT FAQ

Contents

How do I get the last inserted auto_increment value?

Use the LAST_INSERT_ID() function:

```
SELECT LAST_INSERT_ID();
```

What if someone else inserts before I select my id?

LAST_INSERT_ID() is connection specific, so there is no problem from race conditions.

How do I get the next value to be inserted?

You don't. Insert, then find out what you did with LAST_INSERT_ID().

How do I change what number auto_increment starts

with?

`ALTER TABLE yourTable AUTO_INCREMENT = x;` — Next insert will contain `x` or `MAX(autoField) + 1`, whichever is higher

or

`INSERT INTO yourTable (autoField) VALUES (x);` — Next insert will contain `x+1` or `MAX(autoField) + 1`, whichever is higher

Issuing TRUNCATE TABLE will delete all the rows in the table, and will reset the auto_increment value to 0 in most cases (some earlier versions mapped TRUNCATE to DELETE for InnoDB tables, meaning the auto_increment value would not be reset).

How do I renumber rows once I've deleted some in the middle?

Typically, you don't want to. Gaps are hardly ever a problem; if your application can't handle gaps in the sequence, you probably should rethink your application.

Can I do group-wise auto_increment?

Yes, if you use the MyISAM engine.

How do I get the auto_increment value in a BEFORE INSERT trigger?

You don't. It's only available after insert.

How do I assign two fields the same auto_increment value in one query?

You can't, not even with an AFTER INSERT trigger. Insert, then go back and update using `LAST_INSERT_ID()`. Those two statements could be wrapped into one stored procedure if you wish.

However, you can mimic this behavior with a BEFORE INSERT trigger and a second table to store the sequence position:

```
CREATE TABLE sequence (table_name VARCHAR(255), position INT UNSIGNED);
INSERT INTO sequence VALUES ('testTable', 0);
CREATE TABLE testTable (firstAuto INT UNSIGNED, secondAuto INT UNSIGNED);
DELIMITER //
CREATE TRIGGER testTable_BI BEFORE INSERT ON testTable FOR EACH ROW BEGIN
  UPDATE sequence SET position = LAST_INSERT_ID(position + 1) WHERE table_name = 'testTable';
  SET NEW.firstAuto = LAST_INSERT_ID();
  SET NEW.secondAuto = LAST_INSERT_ID();
END//
DELIMITER ;
INSERT INTO testTable VALUES (NULL, NULL), (NULL, NULL);
SELECT * FROM testTable;

+-----------+------------+
| firstAuto | secondAuto |
+-----------+------------+
|         1 |          1 |
|         2 |          2 |
+-----------+------------+
```

The same sequence table can maintain separate sequences for multiple tables (or separate sequences for different fields in the same table) by adding extra rows.

Does the auto_increment field have to be primary key?

No, it only has to be indexed. It doesn't even have to be unique.

InnoDB and AUTO_INCREMENT

See AUTO_INCREMENT handling in InnoDB

General Information To Read

AUTO_INCREMENT

Manual Notes

There can be only one `AUTO_INCREMENT` column per table, it must be indexed, and it cannot have a `DEFAULT` value. An `AUTO_INCREMENT` column works properly only if it contains only positive values. Inserting a negative number is regarded as inserting a very large positive number. This is done to avoid precision problems when numbers wrap over from positive to negative and also to ensure that you do not accidentally get an `AUTO_INCREMENT` column that contains 0.

How to start a table with a set AUTO_INCREMENT value?

```
CREATE TABLE autoinc_test (
  h INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
  m INT UNSIGNED
) AUTO_INCREMENT = 100;

INSERT INTO autoinc_test ( m ) VALUES ( 1 );

SELECT * FROM autoinc_test;
+-----+------+
| h   | m    |
+-----+------+
| 100 |    1 |
+-----+------+
```

5.1.8 NULL Values

Contents

NULL represents an unknown value. It is *not* an empty string (by default), or a zero value. These are all valid values, and are not NULLs.

When a table is created or the format altered, columns can be specified as accepting NULL values, or not accepting them, with the `NULL` and `NOT NULL` clauses respectively.

For example, a customer table could contain dates of birth. For some customers, this information is unknown, so the value could be NULL.

The same system could allocate a customer ID for each customer record, and in this case a NULL value would not be permitted.

```
CREATE TABLE customer (
 id INT NOT NULL,
 date_of_birth DATE NULL
...
)
```

User-defined variables are NULL until a value is explicitly assigned.

Stored routines parameters and local variables can always be set to NULL. If no DEFAULT value is specified for a local variable, its initial value will be NULL. If no value is assigned to an OUT parameter in a stored procedure, NULL is assigned at the end of the procedure.

Syntax

The case of `NULL` is not relevant. `\N` (uppercase) is an alias for `NULL`.

The `IS` operator accepts `UNKNOWN` as an alias for `NULL`, which is meant for boolean contexts.

Comparison Operators

NULL values cannot be used with most comparison operators. For example, =, >, >=, <=, <, or != cannot be used, as any comparison with a NULL always returns a NULL value, never true (1) or false (0).

```
SELECT NULL = NULL;
+-------------+
| NULL = NULL |
+-------------+
|        NULL |
+-------------+

SELECT 99 = NULL;
+-----------+
| 99 = NULL |
+-----------+
|      NULL |
+-----------+
```

To overcome this, certain operators are specifically designed for use with NULL values. To cater for testing equality between two values that may contain NULLs, there's <=>, NULL-safe equal.

```
SELECT 99 <=> NULL, NULL <=> NULL;
+-------------+---------------+
| 99 <=> NULL | NULL <=> NULL |
+-------------+---------------+
|           0 |             1 |
+-------------+---------------+
```

Other operators for working with NULLs include IS NULL and IS NOT NULL, ISNULL (for testing an expression) and COALESCE (for returning the first non-NULL parameter).

Ordering

When you order by a field that may contain NULL values, any NULLs are considered to have the lowest value. So ordering in DESC order will see the NULLs appearing last. To force NULLs to be regarded as highest values, one can add another column which has a higher value when the main field is NULL. Example:

```
SELECT col1 FROM tab ORDER BY ISNULL(col1), col1;
```

Descending order, with NULLs first:

```
SELECT col1 FROM tab ORDER BY IF(col1 IS NULL, 0, 1), col1 DESC;
```

All NULL values are also regarded as equivalent for the purposes of the DISTINCT and GROUP BY clauses.

Functions

In most cases, functions will return NULL if any of the parameters are NULL. There are also functions specifically for handling NULLs. These include IFNULL(), NULLIF() and COALESCE().

```
SELECT IFNULL(1,0);
+-------------+
| IFNULL(1,0) |
+-------------+
|           1 |
+-------------+

SELECT IFNULL(NULL,10);
+-----------------+
| IFNULL(NULL,10) |
+-----------------+
|              10 |
+-----------------+

SELECT COALESCE(NULL,NULL,1);
+-----------------------+
| COALESCE(NULL,NULL,1) |
+-----------------------+
|                     1 |
+-----------------------+
```

Aggregate functions, such as SUM and AVG ignore NULLs.

```
CREATE TABLE t(x INT);

INSERT INTO t VALUES (1),(9),(NULL);

SELECT SUM(x) FROM t;
+--------+
| SUM(x) |
+--------+
|     10 |
+--------+

SELECT AVG(x) FROM t;
+--------+
| AVG(x) |
+--------+
| 5.0000 |
+--------+
```

The one exception is COUNT(*), which counts rows, and doesn't look at whether a value is NULL or not. Compare for example, COUNT(x), which ignores the NULL, and COUNT(*), which counts it:

```
SELECT COUNT(x) FROM t;
+----------+
| COUNT(x) |
+----------+
|        2 |
+----------+

SELECT COUNT(*) FROM t;
+----------+
| COUNT(*) |
+----------+
|        3 |
+----------+
```

AUTO_INCREMENT, TIMESTAMP and Virtual Columns

MariaDB handles NULL values in a special way if the field is an AUTO_INCREMENT, a TIMESTAMP or a virtual column. Inserting a NULL value into a numeric AUTO_INCREMENT column will result in the next number in the auto increment sequence being inserted instead. This technique is frequently used with AUTO_INCREMENT fields, which are left to take care of themselves.

```
CREATE TABLE t2(id INT PRIMARY KEY AUTO_INCREMENT, letter CHAR(1));

INSERT INTO t2(letter) VALUES ('a'),('b');

SELECT * FROM t2;
+----+--------+
| id | letter |
+----+--------+
|  1 | a      |
|  2 | b      |
+----+--------+
```

Similarly, if a NULL value is assigned to a TIMESTAMP field, the current date and time is assigned instead.

```
CREATE TABLE t3 (x INT, ts TIMESTAMP);

INSERT INTO t3(x) VALUES (1),(2);
```

After a pause,

```
INSERT INTO t3(x) VALUES (3);

SELECT* FROM t3;
+------+---------------------+
| x    | ts                  |
+------+---------------------+
|    1 | 2013-09-05 10:14:18 |
|    2 | 2013-09-05 10:14:18 |
|    3 | 2013-09-05 10:14:29 |
+------+---------------------+
```

If a `NULL` is assigned to a `VIRTUAL` or `PERSISTENT` column, the default value is assigned instead.

```
CREATE TABLE virt (c INT, v INT AS (c+10) PERSISTENT) ENGINE=InnoDB;

INSERT INTO virt VALUES (1, NULL);

SELECT c, v FROM virt;
+------+------+
| c    | v    |
+------+------+
|    1 |   11 |
+------+------+
```

In all these special cases, `NULL` is equivalent to the `DEFAULT` keyword.

Inserting

If a NULL value is single-row inserted into a column declared as NOT NULL, an error will be returned. However, if the SQL mode is not strict (default until MariaDB 10.2.3), if a NULL value is multi-row inserted into a column declared as NOT NULL, the implicit default for the column type will be inserted (and NOT the default value in the table definition). The implicit defaults are an empty string for string types, and the zero value for numeric, date and time types.

Since MariaDB 10.2.4, by default both cases will result in an error.

Examples

```
CREATE TABLE nulltest (
  a INT(11),
  x VARCHAR(10) NOT NULL DEFAULT 'a',
  y INT(11) NOT NULL DEFAULT 23
);
```

Single-row insert:

```
INSERT INTO nulltest (a,x,y) VALUES (1,NULL,NULL);
ERROR 1048 (23000): Column 'x' cannot be null
```

Multi-row insert with SQL mode not strict (default until MariaDB 10.2.3):

```
show variables like 'sql_mode%';
+---------------+--------------------------------------------+
| Variable_name | Value                                      |
+---------------+--------------------------------------------+
| sql_mode      | NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION |
+---------------+--------------------------------------------+

INSERT INTO nulltest (a,x,y) VALUES (1,NULL,NULL),(2,NULL,NULL);
Query OK, 2 rows affected, 4 warnings (0.08 sec)
Records: 2  Duplicates: 0  Warnings: 4
```

The specified defaults have not been used; rather, the implicit column type defaults have been inserted

```
SELECT * FROM nulltest;
+------+---+---+
| a    | x | y |
+------+---+---+
|    1 |   | 0 |
|    2 |   | 0 |
+------+---+---+
```

Primary Keys and UNIQUE Indexes

UNIQUE indexes can contain multiple NULL values.

Primary keys are never nullable.

MariaDB starting with 10.3

Oracle Compatibility

In Oracle mode, NULL can be used as a statement:

```
IF a=10 THEN NULL; ELSE NULL; END IF
```

In Oracle mode, CONCAT and the Logical OR operator || ignore NULL.

When setting sql_mode=EMPTY_STRING_IS_NULL, empty strings and NULLs are the same thing. For example:

```
SET sql_mode=EMPTY_STRING_IS_NULL;
SELECT '' IS NULL; -- returns TRUE
INSERT INTO t1 VALUES (''); -- inserts NULL
```

5.2 Character Sets and Collations

Simply put, a character set defines how and which characters are stored to support a particular language or languages. A collation, on the other hand, defines the order used when comparing strings (i.e. the position of any given character within the alphabet of that language)

Character Set and Collation Overview

Introduction to character sets and collations.

Supported Character Sets and Collations

MariaDB supports the following character sets and collations.

Setting Character Sets and Collations

Changing from the default character set and collation.

Unicode

Unicode support.

SHOW CHARACTER SET

Available character sets.

SHOW COLLATION

Supported collations.

Information Schema CHARACTER_SETS Table

Supported character sets.

Information Schema COLLATIONS Table

Supported collations.

Internationalization and Localization

Character sets, collations, time zones and locales.

SET CHARACTER SET

Maps all strings sent between the current client and the server with the given mapping.

SET NAMES

The character set used to send statements to the server, and results back to the client.

There are 6 related questions.

5.2.1 Character Set and Collation Overview

Contents

What Are Character Sets and Collations

A character set is a set of characters while a collation is the rules for comparing and sorting a particular character set.

For example, a subset of a character set could consist of the letters `A`, `B` and `C`. A default collation could define these as appearing in an ascending order of `A, B, C`.

If we consider different case characters, more complexity is added. A binary collation would evaluate the characters `A` and `a` differently, ordering them in a particular way. A case-insensitive collation would evaluate `A` and `a` equivalently, while the German phone book collation evaluates the characters `ue` and `ü` equivalently.

A character set can have many collations associated with it, while each collation is only associated with one character set. In MariaDB, the character set name is always part of the collation name. For example, the `latin1_german1_ci` collation applies only to the `latin1` character set. Each character set also has one default collation. The `latin1` default collation is `latin1_swedish_ci`.

As an example, by default, the character `y` comes between `x` and `z`, while in Lithuanian, it's sorted between `i` and `k`. Similarly, the German phone book order is different to the German dictionary order, so while they share the same character set, the collation is different.

Viewing Character Sets and Collations

In MariaDB, the default character set is latin1, and the default collation is latin1_swedish_ci (however this may differ in some distros, see for example Differences in MariaDB in Debian). You can view a full list of character sets and collations supported by MariaDB at Supported Character Sets and Collations, or see what's supported on your server with the SHOW CHARACTER SET and SHOW COLLATION commands.

By default, `A` comes before `Z`, so the following evaluates to true:

```
 SELECT "A" < "Z";
+-----------+
| "A" < "Z" |
+-----------+
|         1 |
+-----------+
```

By default, comparisons are case-insensitive:

```
SELECT "A" < "a", "A" = "a";
+-----------+-----------+
| "A" < "a" | "A" = "a" |
+-----------+-----------+
|         0 |         1 |
+-----------+-----------+
```

Changing Character Sets and Collations

Character sets and collations can be set from the server level right down to the column level, as well as for client-server communication.

For example, `ue` and `ü` are by default evaluated differently.

```
SELECT 'Mueller' = 'Müller';
+----------------------+
| 'Müller' = 'Mueller' |
+----------------------+
|                    0 |
+----------------------+
```

By using the collation_connection system variable to change the connection character set to `latin1_german2_ci` , or German phone book, the same two characters will evaluate as equivalent.

```
SET collation_connection = latin1_german2_ci;

SELECT 'Mueller' = 'Müller';
+-----------------------+
| 'Mueller' = 'Müller'  |
+-----------------------+
|                     1 |
+-----------------------+
```

See Setting Character Sets and Collations for more.

5.1.2.25 Supported Character Sets and Collations

5.2.3 Setting Character Sets and Collations

Contents

In MariaDB, the default character set is latin1, and the default collation is latin1_swedish_ci (however this may differ in some distros, see for example Differences in MariaDB in Debian). Both character sets and collations can be specified from the server right down to the column level, as well as for client-server connections. When changing a character set and not

specifying a collation, the default collation for the new character set is always used.

Character sets and collations always cascade down, so a column without a specified collation will look for the table default, the table for the database, and the database for the server. It's therefore possible to have extremely fine-grained control over all the character sets and collations used in your data.

Default collations for each character set can be viewed with the SHOW COLLATION statement, for example, to find the default collation for the latin2 character set:

```
SHOW COLLATION LIKE 'latin2%';
+--------------------+---------+----+---------+----------+---------+
| Collation          | Charset | Id | Default | Compiled | Sortlen |
+--------------------+---------+----+---------+----------+---------+
| latin2_czech_cs    | latin2  |  2 |         | Yes      |       4 |
| latin2_general_ci  | latin2  |  9 | Yes     | Yes      |       1 |
| latin2_hungarian_ci| latin2  | 21 |         | Yes      |       1 |
| latin2_croatian_ci | latin2  | 27 |         | Yes      |       1 |
| latin2_bin         | latin2  | 77 |         | Yes      |       1 |
+--------------------+---------+----+---------+----------+---------+
```

Server Level

The character_set_server system variable can be used to change the default server character set. It can be set both on startup or dynamically, with the SET command:

```
SET character_set_server = 'latin2';
```

Similarly, the collation_server variable is used for setting the default server collation.

```
SET collation_server = 'latin2_czech_cs';
```

Database Level

The CREATE DATABASE and ALTER DATABASE statements have optional character set and collation clauses. If these are left out, the server defaults are used.

```
CREATE DATABASE czech_slovak_names
  CHARACTER SET = 'keybcs2'
  COLLATE = 'keybcs2_bin';
```

```
ALTER DATABASE czech_slovak_names COLLATE = 'keybcs2_general_ci';
```

To determine the default character set used by a database, use:

```
SHOW CREATE DATABASE czech_slovak_names;
+--------------------+-----------------------------------------------------------------------
---------+
| Database           | Create Database
|
+--------------------+-----------------------------------------------------------------------
---------+
| czech_slovak_names | CREATE DATABASE `czech_slovak_names` /*!40100 DEFAULT CHARACTER SET
keybcs2 */ |
+--------------------+-----------------------------------------------------------------------
---------+
```

or alternatively, for the character set and collation:

```
SELECT * FROM INFORMATION_SCHEMA.SCHEMATA;
+--------------+--------------------+----------------------------+------------------------+--
--------+
| CATALOG_NAME | SCHEMA_NAME        | DEFAULT_CHARACTER_SET_NAME | DEFAULT_COLLATION_NAME |
SQL_PATH |
+--------------+--------------------+----------------------------+------------------------+--
--------+
| def          | czech_slovak_names | keybcs2                    | keybcs2_general_ci     |
NULL     |
| def          | information_schema | utf8                       | utf8_general_ci        |
NULL     |
| def          | mysql              | latin1                     | latin1_swedish_ci      |
NULL     |
| def          | performance_schema | utf8                       | utf8_general_ci        |
NULL     |
| def          | test               | latin1                     | latin1_swedish_ci      |
NULL     |
+--------------+--------------------+----------------------------+------------------------+--
--------+
```

It is also possible to specify only the collation, and, since each collation only applies to one character set, the associated character set will automatically be specified.

```
CREATE DATABASE danish_names COLLATE 'utf8_danish_ci';

SHOW CREATE DATABASE danish_names;
+--------------+-----------------------------------------------------------------------------
-----------------+
| Database     | Create Database
|
+--------------+-----------------------------------------------------------------------------
-----------------+
| danish_names | CREATE DATABASE `danish_names` /*!40100 DEFAULT CHARACTER SET utf8 COLLATE
utf8_danish_ci */ |
+--------------+-----------------------------------------------------------------------------
-----------------+
```

Although there are character_set_database and collation_database system variables which can be set dynamically, these are used for determining the character set and collation for the default database, and should only be set by the server.

Table Level

The CREATE TABLE and ALTER TABLE statements support optional character set and collation clauses, a MariaDB and MySQL extension to standard SQL.

```
CREATE TABLE english_names (id INT, name VARCHAR(40))
  CHARACTER SET 'utf8'
  COLLATE 'utf8_icelandic_ci';
```

If neither character set nor collation is provided, the database default will be used. If only the character set is provided, the default collation for that character set will be used . If only the collation is provided, the associated character set will be used. See Supported Character Sets and Collations.

```
ALTER TABLE table_name
 CONVERT TO CHARACTER SET charset_name [COLLATE collation_name];
```

If no collation is provided, the collation will be set to the default collation for that character set. See Supported Character Sets and Collations.

For VARCHAR or TEXT columns, CONVERT TO CHARACTER SET changes the data type if needed to ensure the new column is long enough to store as many characters as the original column.

For example, an ascii TEXT column requires a single byte per character, so the column can hold up to 65,535 characters. If the column is converted to utf8, 3 bytes can be required for each character, so the column will be converted to MEDIUMTEXT to be able to hold the same number of characters.

`CONVERT TO CHARACTER SET binary` will convert CHAR, VARCHAR and TEXT columns to BINARY, VARBINARY and BLOB respectively, and from that point will no longer have a character set, or be affected by future `CONVERT TO CHARACTER SET` statements.

To avoid data type changes resulting from `CONVERT TO CHARACTER SET`, use `MODIFY` on the individual columns instead. For example:

```
ALTER TABLE table_name MODIFY ascii_text_column TEXT CHARACTER SET utf8;
ALTER TABLE table_name MODIFY ascii_varchar_column VARCHAR(M) CHARACTER SET utf8;
```

Column Level

Character sets and collations can also be specified for columns that are character types CHAR, TEXT or VARCHAR. The CREATE TABLE and ALTER TABLE statements support optional character set and collation clauses for this purpose - unlike those at the table level, the column level definitions are standard SQL.

```
CREATE TABLE european_names (
  croatian_names VARCHAR(40) COLLATE 'cp1250_croatian_ci',
  greek_names VARCHAR(40) CHARACTER SET 'greek');
```

If neither collation nor character set is provided, the table default is used. If only the character set is specified, that character set's default collation is used, while if only the collation is specified, the associated character set is used.

When using ALTER TABLE to change a column's character set, you need to ensure the character sets are compatible with your data. MariaDB will map the data as best it can, but it's possible to lose data if care is not taken.

The SHOW CREATE TABLE statement or INFORMATION SCHEMA database can be used to determine column character sets and collations.

```
SHOW CREATE TABLE european_names\G
*************************** 1. row ***************************
       Table: european_names
Create Table: CREATE TABLE `european_names` (
  `croatian_names` varchar(40) CHARACTER SET cp1250 COLLATE cp1250_croatian_ci DEFAULT NULL,
  `greek_names` varchar(40) CHARACTER SET greek DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_danish_ci
```

```
SELECT * FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME LIKE 'european%'\G
*************************** 1. row ***************************
           TABLE_CATALOG: def
            TABLE_SCHEMA: danish_names
              TABLE_NAME: european_names
             COLUMN_NAME: croatian_names
        ORDINAL_POSITION: 1
          COLUMN_DEFAULT: NULL
             IS_NULLABLE: YES
               DATA_TYPE: varchar
CHARACTER_MAXIMUM_LENGTH: 40
  CHARACTER_OCTET_LENGTH: 40
       NUMERIC_PRECISION: NULL
           NUMERIC_SCALE: NULL
      DATETIME_PRECISION: NULL
      CHARACTER_SET_NAME: cp1250
          COLLATION_NAME: cp1250_croatian_ci
             COLUMN_TYPE: varchar(40)
              COLUMN_KEY:
                   EXTRA:
              PRIVILEGES: select,insert,update,references
          COLUMN_COMMENT:
*************************** 2. row ***************************
           TABLE_CATALOG: def
            TABLE_SCHEMA: danish_names
              TABLE_NAME: european_names
             COLUMN_NAME: greek_names
        ORDINAL_POSITION: 2
          COLUMN_DEFAULT: NULL
             IS_NULLABLE: YES
               DATA_TYPE: varchar
CHARACTER_MAXIMUM_LENGTH: 40
  CHARACTER_OCTET_LENGTH: 40
       NUMERIC_PRECISION: NULL
           NUMERIC_SCALE: NULL
      DATETIME_PRECISION: NULL
      CHARACTER_SET_NAME: greek
          COLLATION_NAME: greek_general_ci
             COLUMN_TYPE: varchar(40)
              COLUMN_KEY:
                   EXTRA:
              PRIVILEGES: select,insert,update,references
          COLUMN_COMMENT:
```

Filenames

Since MariaDB 5.1, the character_set_filesystem system variable has controlled interpretation of file names that are given as literal strings. This affects the following statements and functions:

- SELECT INTO DUMPFILE
- SELECT INTO OUTFILE
- LOAD DATA INFILE
- LOAD XML
- LOAD_FILE()

Literals

By default, the character set and collation used for literals is determined by the character_set_connection and collation_connection system variables. However, they can also be specified explicitly:

```
[_charset_name]'string' [COLLATE collation_name]
```

The character set of string literals that do not have a character set introducer is determined by the character_set_connection system variable.

This query:

```
SELECT CHARSET('a'), @@character_set_connection;
```

always returns the same character set name in both columns.

character_set_client and character_set_connection are normally (e.g. during handshake, or after a SET NAMES query) are set to equal values. However, it's possible to set to different values.

Examples

Examples when setting @@character_set_client and @@character_set_connection to different values can be useful:

Example 1:

Suppose, we have a utf8 database with this table:

```
CREATE TABLE t1 (a VARCHAR(10)) CHARACTER SET utf8 COLLATE utf8_general_ci;
INSERT INTO t1 VALUES ('oe'),('ö');
```

Now we connect to it using "mysql.exe", which uses the DOS character set (cp850 on a West European machine), and want to fetch all records that are equal to 'ö' according to the German phonebook rules.

It's possible with the following:

```
SET @@character_set_client=cp850, @@character_set_connection=utf8;
SELECT a FROM t1 WHERE a='ö' COLLATE utf8_german2_ci;
```

This will return:

```
+------+
| a    |
+------+
| oe   |
| ö    |
+------+
```

It works as follows:

1. The client sends the query using cp850.
2. The server, when parsing the query, creates a utf8 string literal by converting 'ö' from @@character_set_client (cp850) to @@character_set_connection (utf8)
3. The server applies the collation "utf8_german2_ci" to this string literal.
4. The server uses utf8_german2_ci for comparison.

Note, if we rewrite the script like this:

```
SET NAMES cp850;
SELECT a FROM t1 WHERE a='ö' COLLATE utf8_german2_ci;
```

we'll get an error:

```
ERROR 1253 (42000): COLLATION 'utf8_german2_ci' is not valid for CHARACTER SET 'cp850'
```

because:

- on step #2, the literal is not converted to utf8 any more and is created using cp850.
- on step #3, the server fails to apply utf8_german2_ci to an cp850 string literal.

Example 2:

Suppose we have a utf8 database and use "mysql.exe" from a West European machine again.

We can do this:

```
SET @@character_set_client=cp850, @@character_set_connection=utf8;
CREATE TABLE t2 AS SELECT 'ö';
```

It will create a table with a column of the type `VARCHAR(1) CHARACTER SET utf8`.

Note, if we rewrite the query like this:

```
SET NAMES cp850;
CREATE TABLE t2 AS SELECT 'ö';
```

It will create a table with a column of the type `VARCHAR(1) CHARACTER SET cp850`, which is probably not a good idea.

N

Also, `N` or `n` can be used as prefix to convert a literal into the National Character set (which in MariaDB is always utf8).

For example:

```
SELECT _latin2 'Müller';
+-----------+
| MÄźller   |
+-----------+
| MÄźller   |
+-----------+
```

```
SELECT CHARSET(N'a string');
+----------------------+
| CHARSET(N'a string') |
+----------------------+
| utf8                 |
+----------------------+
```

```
SELECT 'Mueller' = 'Müller' COLLATE 'latin1_german2_ci';
+-----------------------------------------------------+
| 'Mueller' = 'Müller' COLLATE 'latin1_german2_ci'  |
+-----------------------------------------------------+
|                                                   1 |
+-----------------------------------------------------+
```

Stored Programs and Views

The literals which occur in stored programs and views, by default, use the character set and collation which was specified by the character_set_connection and collation_connection system variables when the stored program was created. These values can be seen using the SHOW CREATE statements. To change the character sets used for literals in an existing stored program, it is necessary to drop and recreate the stored program.

For stored routines parameters and return values, a character set and a collation can be specified via the CHARACTER SET and COLLATE clauses. Before 5.5, specifying a collation was not supported.

The following example shows that the character set and collation are determined at the time of creation:

```
SET @@local.character_set_connection='latin1';

DELIMITER ||
CREATE PROCEDURE `test`.`x`()
BEGIN
 SELECT CHARSET('x');
END;
||
Query OK, 0 rows affected (0.00 sec)

DELIMITER ;
SET @@local.character_set_connection='utf8';

CALL `test`.`x`();
+--------------+
| CHARSET('x') |
+--------------+
| latin1       |
+--------------+
```

The following example shows how to specify a function parameters character set and collation:

```
CREATE FUNCTION `test`.`y`(`str` TEXT CHARACTER SET utf8 COLLATE utf8_bin)
 RETURNS TEXT CHARACTER SET latin1 COLLATE latin1_bin
BEGIN
 SET @param_coll = COLLATION(`str`);
 RETURN `str`;
END;

-- return value's collation:
SELECT COLLATION(`test`.`y`('Hello, planet!'));
+---------------------------------------+
| COLLATION(`test`.`y`('Hello, planet!')) |
+---------------------------------------+
| latin1_bin                            |
+---------------------------------------+

-- parameter's collation:
SELECT @param_coll;
+-------------+
| @param_coll |
+-------------+
| utf8_bin    |
+-------------+
```

Changing Default Collation

MariaDB starting with 11.2.1

From MariaDB 11.2, it is possible to change the default collation associated with a particular character set. The character_set_collations system variable accepts a comma-delimited list of character sets and new default collations, for example:

```
SET @@character_set_collations = 'utf8mb4=uca1400_ai_ci, latin2=latin2_hungarian_ci';
```

The new variable will take effect in all cases where a character set is explicitly or implicitly specified without an explicit COLLATE clause, including but not limited to:

- Column collation
- Table collation
- Database collation
- CHAR(expr USING csname)
- CONVERT(expr USING csname)
- CAST(expr AS CHAR CHARACTER SET csname)
- '' - character string literal
- _utf8mb3'text' - a character string literal with an introducer
- _utf8mb3 X'61' - a character string literal with an introducer with hex notation
- _utf8mb3 0x61 - a character string literal with an introducer with hex hybrid notation
- @@collation_connection after a SET NAMES without COLLATE

Example: Changing the Default Character Set To UTF-8

To change the default character set from latin1 to UTF-8, the following settings should be specified in the my.cnf configuration file.

```
[mysql]
...
default-character-set=utf8mb4
...
[mysqld]
...
collation-server = utf8mb4_unicode_ci
init-connect='SET NAMES utf8mb4'
character-set-server = utf8mb4
...
```

Note that the `default-character-set` option is a client option, not a server option.

5.2.4 Unicode

Unicode is a standard for encoding text across multiple writing systems. MariaDB supports a number of character sets for storing Unicode data:

Character Set	Description
ucs2	UCS-2, each character is represented by a 2-byte code with the most significant byte first. Fixed-length 16-bit encoding.
utf8	Until MariaDB 10.5, this was a UTF-8 encoding using one to three bytes per character. Basic Latin letters, numbers and punctuation use one byte. European and Middle East letters mostly fit into 2 bytes. Korean, Chinese, and Japanese ideographs use 3-bytes. No supplementary characters are stored. From MariaDB 10.6, `utf8` is an alias for `utf8mb3`, but this can changed to `utf8mb4` by changing the default value of the old_mode system variable.
utf8mb3	UTF-8 encoding using one to three bytes per character. Basic Latin letters, numbers and punctuation use one byte. European and Middle East letters mostly fit into 2 bytes. Korean, Chinese, and Japanese ideographs use 3-bytes. No supplementary characters are stored. Until MariaDB 10.5, this was an alias for `utf8`. From MariaDB 10.6, `utf8` is by default an alias for `utf8mb3`, but this can changed to `utf8mb4` by changing the default value of the old_mode system variable.
utf8mb4	UTF-8 encoding the same as `utf8mb3` but which stores supplementary characters in four bytes.
utf16	UTF-16, same as ucs2, but stores supplementary characters in 32 bits. 16 or 32-bits.
utf32	UTF-32, fixed-length 32-bit encoding.

1.1.1.2.8.6 SHOW CHARACTER SET

5.2.6 SHOW COLLATION

Syntax

```
SHOW COLLATION
    [LIKE 'pattern' | WHERE expr]
```

Contents

Description

The output from `SHOW COLLATION` includes all available collations. The `LIKE` clause, if present on its own, indicates which collation names to match. The `WHERE` and `LIKE` clauses can be given to select rows using more general conditions, as discussed in Extended SHOW.

The same information can be queried from the Information Schema COLLATIONS table.

See Setting Character Sets and Collations for details on specifying the collation at the server, database, table and column levels.

Examples

```
SHOW COLLATION LIKE 'latin1%';
+-------------------------+---------+------+---------+----------+---------+
| Collation               | Charset | Id   | Default | Compiled | Sortlen |
+-------------------------+---------+------+---------+----------+---------+
| latin1_german1_ci       | latin1  |    5 |         | Yes      |       1 |
| latin1_swedish_ci       | latin1  |    8 | Yes     | Yes      |       1 |
| latin1_danish_ci        | latin1  |   15 |         | Yes      |       1 |
| latin1_german2_ci       | latin1  |   31 |         | Yes      |       2 |
| latin1_bin              | latin1  |   47 |         | Yes      |       1 |
| latin1_general_ci       | latin1  |   48 |         | Yes      |       1 |
| latin1_general_cs       | latin1  |   49 |         | Yes      |       1 |
| latin1_spanish_ci       | latin1  |   94 |         | Yes      |       1 |
| latin1_swedish_nopad_ci | latin1  | 1032 |         | Yes      |       1 |
| latin1_nopad_bin        | latin1  | 1071 |         | Yes      |       1 |
+-------------------------+---------+------+---------+----------+---------+
```

```
SHOW COLLATION WHERE Sortlen LIKE '8' AND Charset LIKE 'utf8mb4';
+-------------------------------+---------+------+---------+----------+---------+
| Collation                     | Charset | Id   | Default | Compiled | Sortlen |
+-------------------------------+---------+------+---------+----------+---------+
| utf8mb4_unicode_ci            | utf8mb4 |  224 |         | Yes      |       8 |
| utf8mb4_icelandic_ci          | utf8mb4 |  225 |         | Yes      |       8 |
| utf8mb4_latvian_ci            | utf8mb4 |  226 |         | Yes      |       8 |
| utf8mb4_romanian_ci           | utf8mb4 |  227 |         | Yes      |       8 |
| utf8mb4_slovenian_ci          | utf8mb4 |  228 |         | Yes      |       8 |
| utf8mb4_polish_ci             | utf8mb4 |  229 |         | Yes      |       8 |
| utf8mb4_estonian_ci           | utf8mb4 |  230 |         | Yes      |       8 |
| utf8mb4_spanish_ci            | utf8mb4 |  231 |         | Yes      |       8 |
| utf8mb4_swedish_ci            | utf8mb4 |  232 |         | Yes      |       8 |
| utf8mb4_turkish_ci            | utf8mb4 |  233 |         | Yes      |       8 |
| utf8mb4_czech_ci              | utf8mb4 |  234 |         | Yes      |       8 |
| utf8mb4_danish_ci             | utf8mb4 |  235 |         | Yes      |       8 |
| utf8mb4_lithuanian_ci         | utf8mb4 |  236 |         | Yes      |       8 |
| utf8mb4_slovak_ci             | utf8mb4 |  237 |         | Yes      |       8 |
| utf8mb4_spanish2_ci           | utf8mb4 |  238 |         | Yes      |       8 |
| utf8mb4_roman_ci              | utf8mb4 |  239 |         | Yes      |       8 |
| utf8mb4_persian_ci            | utf8mb4 |  240 |         | Yes      |       8 |
| utf8mb4_esperanto_ci          | utf8mb4 |  241 |         | Yes      |       8 |
| utf8mb4_hungarian_ci          | utf8mb4 |  242 |         | Yes      |       8 |
| utf8mb4_sinhala_ci            | utf8mb4 |  243 |         | Yes      |       8 |
| utf8mb4_german2_ci            | utf8mb4 |  244 |         | Yes      |       8 |
| utf8mb4_croatian_mysql561_ci  | utf8mb4 |  245 |         | Yes      |       8 |
| utf8mb4_unicode_520_ci        | utf8mb4 |  246 |         | Yes      |       8 |
| utf8mb4_vietnamese_ci         | utf8mb4 |  247 |         | Yes      |       8 |
| utf8mb4_croatian_ci           | utf8mb4 |  608 |         | Yes      |       8 |
| utf8mb4_myanmar_ci            | utf8mb4 |  609 |         | Yes      |       8 |
| utf8mb4_unicode_nopad_ci      | utf8mb4 | 1248 |         | Yes      |       8 |
| utf8mb4_unicode_520_nopad_ci  | utf8mb4 | 1270 |         | Yes      |       8 |
+-------------------------------+---------+------+---------+----------+---------+
```

1.1.1.2.9.1.1.6 Information Schema CHARACTER_SETS Table

1.1.1.2.9.1.1.10 Information Schema COLLATIONS Table

5.2.9 Internationalization and Localization

Server Locale

Server locale.

Time Zones

MariaDB time zones.

Setting the Language for Error Messages

Specifying the language for the server error messages.

Coordinated Universal Time
Coordinated Universal Time.

Locales Plugin
List compiled-in locales.

mariadb-tzinfo-to-sql
Load time zones into the time zone tables.

2.4.3 Setting the Language for Error Messages

5.4.11.2 Locales plugin

1.3.40 mariadb-tzinfo-to-sql

1.1.1.2.7.2 SET CHARACTER SET

1.1.1.2.7.4 SET NAMES

5.3 Storage Engines

Information on storage engines available for MariaDB.

Choosing the Right Storage Engine
Quickly choose the most suitable storage engine for your needs.

InnoDB
The general-purpose InnoDB storage engine.

MariaDB ColumnStore
Uses a massively parallel architecture, ideal for systems that scale to petabytes of data.

Aria
Aria is a crash safe MyISAM and more.

Archive
Stores data in compressed (gzip) format.

BLACKHOLE
Storage engine that accepts data without storing it.

CONNECT
The CONNECT storage engine enables MariaDB to access external local or remote data.

CSV
Works with files stored in CSV (comma-separated-values) format.

FederatedX
Allows you to access tables in other MariaDB or MySQL servers.

MEMORY Storage Engine
Storage engine stored in memory rather than on disk.

MERGE
Allows you to access a collection of identical MyISAM tables as one.

Mroonga
Provides fast CJK-ready full text searching using column store.

MyISAM
Non-transactional storage engine with good performance and small data footprint.

MyRocks
Adds RocksDB, an LSM database with a great compression ratio that is optimized for flash storage.

OQGRAPH
Open Query GRAPH computation engine for handling hierarchies (tree structures) and complex graphs.

S3 Storage Engine
A read-only storage engine that stores its data in Amazon S3.

Sequence Storage Engine
Allows ascending or descending sequences of numbers.

SphinxSE
Storage engine that talks to searchd to enable text searching.

Spider
Supports partitioning and xa transactions and allows tables of different in...

TokuDB
For use in high-performance and write-intensive environments.

Information Schema ENGINES Table
Storage engine information.

PERFORMANCE_SCHEMA Storage Engine
PERFORMANCE_SCHEMA storage engine, a mechanism for implementing the feature.

Legacy Storage Engines
Storage engines that are no longer maintained.

Storage Engine Development
Storage Engine Development.

Converting Tables from MyISAM to InnoDB
Issues when converting tables from MyISAM to InnoDB.

Machine Learning with MindsDB
A 3rd-party tool interfacing with MariaDB Server to provide Machine Learning capabilities.

There are 5 related questions.

5.3.1 Choosing the Right Storage Engine

Contents

A high-level overview of the main reasons for choosing a particular storage engine:

Topic List

General Purpose

- InnoDB is a good general transaction storage engine, and, from MariaDB 10.2, the best choice in most cases. It is the default storage engine from MariaDB 10.2. For earlier releases, XtraDB was a performance enhanced fork of InnoDB and is usually preferred.
- Aria, MariaDB's more modern improvement on MyISAM, has a small footprint and allows for easy copying between systems.
- MyISAM has a small footprint and allows for easy copying between systems. MyISAM is MySQL's oldest storage engine. There is usually little reason to use it except for legacy purposes. Aria is MariaDB's more modern improvement.
- XtraDB was the best choice in MariaDB 10.1 and earlier in the majority of cases. It was a performance-enhanced fork of InnoDB and was MariaDB's default engine until MariaDB 10.1.

Scaling, Partitioning

When you want to split your database load on several servers or optimize for scaling. We also suggest looking at Galera, a synchronous multi-master cluster.

- Spider uses partitioning to provide data sharding through multiple servers.
- ColumnStore utilizes a massively parallel distributed data architecture and is designed for big data scaling to process petabytes of data.
- The MERGE storage engine is a collection of identical MyISAM tables that can be used as one. "Identical" means that all tables have identical column and index information.
- TokuDB is a transactional storage engine which is optimized for workloads that do not fit in memory, and provides a good compression ratio. TokuDB has been deprecated by its upstream developers, and is disabled in MariaDB 10.5, and removed in MariaDB 10.6

Compression / Archive

- MyRocks enables greater compression than InnoDB, as well as less write amplification giving better endurance of flash storage and improving overall throughput.
- The Archive storage engine is, unsurprisingly, best used for archiving.
- TokuDB is a transactional storage engine which is optimized for workloads that do not fit in memory, and provides a good compression ratio. TokuDB has been deprecated by its upstream developers, and is disabled in MariaDB 10.5, and removed in MariaDB 10.6

Connecting to Other Data Sources

When you want to use data not stored in a MariaDB database.

- CONNECT allows access to different kinds of text files and remote resources as if they were regular MariaDB tables.
- The CSV storage engine can read and append to files stored in CSV (comma-separated-values) format. However, since MariaDB 10.0, CONNECT is a better choice and is more flexibly able to read and write such files.
- FederatedX uses libmysql to talk to the data source, the data source being a remote RDBMS. Currently, since FederatedX only uses libmysql, it can only talk to another MySQL RDBMS.
- CassandraSE is a storage engine allowing access to an older version of Apache Cassandra NoSQL DBMS. It was relatively experimental, is no longer being actively developed and has been removed in MariaDB 10.6.

Search Optimized

Search engines optimized for search.

- SphinxSE is used as a proxy to run statements on a remote Sphinx database server (mainly useful for advanced fulltext searches).
- Mroonga provides fast CJK-ready full text searching using column store.

Cache, Read-only

- MEMORY does not write data on-disk (all rows are lost on crash) and is best-used for read-only caches of data from other tables, or for temporary work areas. With the default InnoDB and other storage engines having good caching, there is less need for this engine than in the past.

Other Specialized Storage Engines

- S3 Storage Engine is a read-only storage engine that stores its data in Amazon S3.
- Sequence allows the creation of ascending or descending sequences of numbers (positive integers) with a given

starting value, ending value and increment, creating virtual, ephemeral tables automatically when you need them.
- The BLACKHOLE storage engine accepts data but does not store it and always returns an empty result. This can be useful in replication environments, for example, if you want to run complex filtering rules on a slave without incurring any overhead on a master.
- OQGRAPH allows you to handle hierarchies (tree structures) and complex graphs (nodes having many connections in several directions).

Alphabetical List

- The Archive storage engine is, unsurprisingly, best used for archiving.
- Aria, MariaDB's more modern improvement on MyISAM, has a small footprint and allows for easy copy between systems.
- The BLACKHOLE storage engine accepts data but does not store it and always returns an empty result. This can be useful in replication environments, for example, if you want to run complex filtering rules on a slave without incurring any overhead on a master.
- CassandraSE is a storage engine allowing access to an older version of Apache Cassandra NoSQL DBMS. It was relatively experimental, is no longer being actively developed and has been removed in MariaDB 10.6.
- ColumnStore utilizes a massively parallel distributed data architecture and is designed for big data scaling to process petabytes of data.
- CONNECT allows access to different kinds of text files and remote resources as if they were regular MariaDB tables.
- The CSV storage engine can read and append to files stored in CSV (comma-separated-values) format. However, since MariaDB 10.0, CONNECT is a better choice and is more flexibly able to read and write such files.
- FederatedX uses libmysql to talk to the data source, the data source being a remote RDBMS. Currently, since FederatedX only uses libmysql, it can only talk to another MySQL RDBMS.
- InnoDB is a good general transaction storage engine, and, from MariaDB 10.2, the best choice in most cases. It is the default storage engine from MariaDB 10.2. For earlier releases, XtraDB was a performance enhanced fork of InnoDB and is usually preferred.
- The MERGE storage engine is a collection of identical MyISAM tables that can be used as one. "Identical" means that all tables have identical column and index information.
- MEMORY does not write data on-disk (all rows are lost on crash) and is best-used for read-only caches of data from other tables, or for temporary work areas. With the default InnoDB and other storage engines having good caching, there is less need for this engine than in the past.
- Mroonga provides fast CJK-ready full text searching using column store.
- MyISAM has a small footprint and allows for easy copying between systems. MyISAM is MySQL's oldest storage engine. There is usually little reason to use it except for legacy purposes. Aria is MariaDB's more modern improvement.
- MyRocks enables greater compression than InnoDB, as well as less write amplification giving better endurance of flash storage and improving overall throughput.
- OQGRAPH allows you to handle hierarchies (tree structures) and complex graphs (nodes having many connections in several directions).
- S3 Storage Engine is a read-only storage engine that stores its data in Amazon S3.
- Sequence allows the creation of ascending or descending sequences of numbers (positive integers) with a given starting value, ending value and increment, creating virtual, ephemeral tables automatically when you need them.
- SphinxSE is used as a proxy to run statements on a remote Sphinx database server (mainly useful for advanced fulltext searches).
- Spider uses partitioning to provide data sharding through multiple servers.
- TokuDB is a transactional storage engine which is optimized for workloads that do not fit in memory, and provides a good compression ratio. TokuDB has been deprecated by its upstream developers, and is disabled in MariaDB 10.5, and removed in MariaDB 10.6
- XtraDB was the best choice in MariaDB 10.1 and earlier in the majority of cases. It was a performance-enhanced fork of InnoDB and was MariaDB's default engine until MariaDB 10.1.

5.3.2 InnoDB

InnoDB is a general-purpose storage engine, and the default in MariaDB.

InnoDB Versions

From MariaDB 10.2, InnoDB is the default storage engine.

InnoDB Limitations

The InnoDB storage engine has the following limitations.

InnoDB Troubleshooting

Guidelines when troubleshooting problems with InnoDB .

InnoDB System Variables

List and description of InnoDB-related server system variables.

InnoDB Server Status Variables

List and description of InnoDB status variables.

AUTO_INCREMENT Handling in InnoDB

AUTO_INCREMENT handling in InnoDB and the lock modes.

InnoDB Buffer Pool

The most important memory buffer used by InnoDB.

InnoDB Change Buffering

Buffering INSERT, UPDATE and DELETE statements for greater efficiency.

InnoDB Doublewrite Buffer

Buffer used for recovering from half-written pages.

InnoDB Tablespaces

Information on tablespaces in InnoDB, including an overview, system tablesp...

InnoDB File Format

Description of the file formats supported by InnoDB.

InnoDB Row Formats

InnoDB's row formats are REDUNDANT, COMPACT, DYNAMIC, and COMPRESSED.

InnoDB Strict Mode

InnoD strict mode makes InnoDB more reliable.

InnoDB Redo Log

The redo log is used by InnoDB during crash recovery.

InnoDB Undo Log

InnoDB Undo log.

InnoDB Page Flushing

Configuring when and how InnoDB flushes dirty pages to disk.

InnoDB Purge

Purge process to remove old versions of a row from the undo log.

Information Schema InnoDB Tables

All InnoDB-specific Information Schema tables.

InnoDB Online DDL

InnoDB tables support online DDL in certain circumstances.

Binary Log Group Commit and InnoDB Flushing Performance

Improvement for group commit for InnoDB transactions with the binary log enabled.

InnoDB Page Compression

InnoDB page compression, which is more sophisticated than the COMPRESSED row format.

InnoDB Data Scrubbing

Ensuring data is completely removed when deleted.

InnoDB Lock Modes

InnoDB supports a number of lock modes to ensure that concurrent write operations never collide.

InnoDB Monitors

Standard Monitor, Lock Monitor, Tablespace Monitor and the Table Monitor.

InnoDB Encryption Overview

Data-at-rest encryption for tables that use the InnoDB storage engine.

InnoDB - Unmaintained

Articles that apply only to old, unmaintained versions of MariaDB.

There are 11 related questions.

5.3.2.1 InnoDB Versions

MariaDB starting with 10.3.7

In MariaDB 10.3.7 and later, the InnoDB implementation has diverged substantially from the InnoDB in MySQL. Therefore, in these versions, the InnoDB version is no longer associated with a MySQL release version.

MariaDB starting with 10.2

In MariaDB 10.2 and later, the default InnoDB implementation is based on InnoDB from MySQL 5.7. See Why MariaDB uses InnoDB instead of XtraDB from MariaDB 10.2 for more information.

MariaDB until 10.1

In MariaDB 10.1 and before, the default InnoDB implementation is based on Percona's XtraDB. XtraDB is a performance enhanced fork of InnoDB. For compatibility reasons, the system variables still retain their original `innodb` prefixes. If the documentation says that something applies to InnoDB, then it usually also applies to the XtraDB fork, unless explicitly stated otherwise. In these versions, it is still possible to use InnoDB instead of XtraDB. See Using InnoDB instead of XtraDB for more information.

Divergences

Some examples of divergences between MariaDB's InnoDB and MySQL's InnoDB are:

- MariaDB 10.1 (which is based on MySQL 5.6) included encryption and variable-size page compression before MySQL 5.7 introduced them.
- MariaDB 10.2 (based on MySQL 5.7) introduced persistent AUTO_INCREMENT (MDEV-6076) in a GA release before MySQL 8.0.
- MariaDB 10.3 (based on MySQL 5.7) introduced instant ADD COLUMN (MDEV-11369) before MySQL.

InnoDB Versions Included in MariaDB Releases

MariaDB 10.3

InnoDB Version	Introduced
No longer reported	MariaDB 10.3.7
InnoDB 5.7.20	MariaDB 10.3.3
InnoDB 5.7.19	MariaDB 10.3.1
InnoDB 5.7.14	MariaDB 10.3.0

MariaDB 10.2

InnoDB Version	Introduced
InnoDB 5.7.29	MariaDB 10.2.33
InnoDB 5.7.23	MariaDB 10.2.17
InnoDB 5.7.22	MariaDB 10.2.15
InnoDB 5.7.21	MariaDB 10.2.13

InnoDB 5.7.20	MariaDB 10.2.10
InnoDB 5.7.19	MariaDB 10.2.8
InnoDB 5.7.18	MariaDB 10.2.7
InnoDB 5.7.14	MariaDB 10.2.2

MariaDB 10.1

InnoDB Version	Introduced
InnoDB 5.6.49	MariaDB 10.1.46
InnoDB 5.6.47	MariaDB 10.1.44
InnoDB 5.6.44	MariaDB 10.1.39
InnoDB 5.6.42	MariaDB 10.1.37
InnoDB 5.6.39	MariaDB 10.1.31
InnoDB 5.6.37	MariaDB 10.1.26
InnoDB 5.6.36	MariaDB 10.1.24
InnoDB 5.6.35	MariaDB 10.1.21
InnoDB 5.6.33	MariaDB 10.1.18
InnoDB 5.6.32	MariaDB 10.1.17
InnoDB 5.6.31	MariaDB 10.1.16
InnoDB 5.6.30	MariaDB 10.1.14
InnoDB 5.6.29	MariaDB 10.1.12

MariaDB 10.0

InnoDB Version	Introduced
InnoDB 5.6.43	MariaDB 10.0.38
InnoDB 5.6.42	MariaDB 10.0.37
InnoDB 5.6.40	MariaDB 10.0.35
InnoDB 5.6.39	MariaDB 10.0.34
InnoDB 5.6.38	MariaDB 10.0.33
InnoDB 5.6.37	MariaDB 10.0.32
InnoDB 5.6.36	MariaDB 10.0.31
InnoDB 5.6.35	MariaDB 10.0.29
InnoDB 5.6.33	MariaDB 10.0.28
InnoDB 5.6.32	MariaDB 10.0.27
InnoDB 5.6.31	MariaDB 10.0.26
InnoDB 5.6.30	MariaDB 10.0.25
InnoDB 5.6.29	MariaDB 10.0.24
InnoDB 5.6.28	MariaDB 10.0.23
InnoDB 5.6.27	MariaDB 10.0.22
InnoDB 5.6.26	MariaDB 10.0.21
InnoDB 5.6.25	MariaDB 10.0.20
InnoDB 5.6.24	MariaDB 10.0.18
InnoDB 5.6.23	MariaDB 10.0.17
InnoDB 5.6.22	MariaDB 10.0.16

InnoDB 5.6.21	MariaDB 10.0.15
InnoDB 5.6.20	MariaDB 10.0.14
InnoDB 5.6.19	MariaDB 10.0.13
InnoDB 5.6.17	MariaDB 10.0.11
InnoDB 5.6.15	MariaDB 10.0.9
InnoDB 5.6.14	MariaDB 10.0.8

5.3.2.2 InnoDB Limitations

Contents

The InnoDB storage engine has the following limitations.

Limitations on Schema

- InnoDB tables can have a maximum of 1,017 columns. This includes virtual generated columns.
- InnoDB tables can have a maximum of 64 secondary indexes.
- A multicolumn index on InnoDB can use a maximum of 16 columns. If you attempt to create a multicolumn index that uses more than 16 columns, MariaDB returns an Error 1070.

Limitations on Size

- With the exception of variable-length columns (that is, VARBINARY, VARCHAR, BLOB and TEXT), rows in InnoDB have a maximum length of roughly half the page size for 4KB, 8KB, 16KB and 32KB page sizes.
- The maximum size for BLOB and TEXT columns is 4GB. This also applies to LONGBLOB and LONGTEXT.
- MariaDB imposes a row-size limit of 65,535 bytes for the combined sizes of all columns. If the table contains BLOB or TEXT columns, these only count for 9 - 12 bytes in this calculation, given that their content is stored separately.
- 32-bit operating systems have a maximum file size limit of 2GB. When working with large tables using this architecture, configure InnoDB to use smaller data files.
- The maximum size for the combined InnoDB log files is 512GB.
- With tablespaces, the minimum size is 10MB, the maximum varies depending on the InnoDB Page Size.

InnoDB Page Size	Maximum Tablespace Size
4KB	16TB
8KB	32TB
16KB	64TB
32KB	128TB
64KB	256TB

Page Sizes

Using the innodb_page_size system variable, you can configure the size in bytes for InnoDB pages. Pages default to 16KB. There are certain limitations on how you use this variable.

- MariaDB instances using one page size cannot use data files or log files from an instance using a different page size.
- When using a Page Size of 4KB or 8KB, the maximum index key length is lowered proportionately.

InnoDB Page Size	Index Key Length

4KB	768B
8KB	1536B
16KB	3072B

Large Prefix Size

Until MariaDB 10.3.1, the innodb_large_prefix system variable enabled large prefix sizes. That is, when enabled (the default from MariaDB 10.2), InnoDB uses 3072B index key prefixes for `DYNAMIC` and `COMPRESSED` row formats. When disabled, it uses 787B key prefixes for tables of any row format. Using an index key that exceeds this limit throws an error.

From MariaDB 10.3.1, InnoDB always uses large index key prefixes.

Limitations on Tables

InnoDB has the following table-specific limitations.

- When you issue a DELETE statement, InnoDB doesn't regenerate the table, rather it deletes each row from the table one by one.
- When running MariaDB on Windows, InnoDB stores databases and tables in lowercase. When moving databases and tables in a binary format from Windows to a Unix-like system or from a Unix system to Windows, you need to rename these to use lowercase.
- When using cascading foreign keys, operations in the cascade don't activate triggers.

Table Analysis

MariaDB supports the use of the ANALYZE TABLE SQL statement to analyze and store table key distribution. When MariaDB executes this statement, it calculates index cardinality through random dives on index trees. This makes it fast, but not always accurate as it does not check all rows. The data is only an estimate and repeated executions of this statement may return different results.

> In situations where accurate data from ANALYZE TABLE statements is important, enable the innodb_stats_persistent system variable. Additionally, you can use the innodb_stats_transient_sample_pages system variable to change the number of random dives it performs.

When running ANALYZE TABLE twice on a table in which statements or transactions are running, MariaDB blocks the second ANALYZE TABLE until the statement or transaction is complete. This occurs because the statement or transaction blocks the second ANALYZE TABLE statement from reloading the table definition, which it must do since the old one was marked as obsolete after the first statement.

Table Status

Similar to the ANALYZE TABLE statement, SHOW TABLE STATUS statements do not provide accurate statistics for InnoDB, except for the physical table size.

The InnoDB storage engine does not maintain internal row counts. Transactions isolate writes, which means that concurrent transactions will not have the same row counts.

Auto-incrementing Columns

- When defining an index on an auto-incrementing column, it must be defined in a way that allows the equivalent of `SELECT MAX(col)` lookups on the table.
- Restarting MariaDB may cause InnoDB to reuse old auto-increment values, such as in the case of a transaction that was rolled back.
- When auto-incrementing columns run out of values, INSERT statements generate duplicate-key errors.

Transactions and Locks

- You can modify data on a maximum of 96 * 1023 concurrent transactions that generate undo records.
- Of the 128 rollback segments, InnoDB assigns 32 to non-redo logs for transactions that modify temporary tables and related objects, reducing the maximum number of concurrent data-modifying transactions to 96,000, from 128.000.
- The limit is 32,000 concurrent transactions when all data-modifying transactions also modify temporary tables.
- Issuing a LOCK TABLES statement sets two locks on each table when the innodb_table_locks system variable is

enabled (the default).
- When you commit or roll back a transaction, any locks set in the transaction are released. You don't need to issue LOCK TABLES statements when the autocommit variable is enabled, as InnoDB would immediately release the table locks.

5.3.2.3 InnoDB Troubleshooting

Guidelines when troubleshooting problems with InnoDB.

InnoDB Troubleshooting Overview

Overview of InnoDB errors experienced and steps to take.

InnoDB Data Dictionary Troubleshooting

Troubleshooting the InnoDB Data Dictionary.

InnoDB Recovery Modes

Modes for recovering from emergency situations in InnoDB.

Troubleshooting Row Size Too Large Errors with InnoDB

Fixing "Row size too large (> 8126). Changing some columns to TEXT or BLOB may help."

5.3.2.3.1 InnoDB Troubleshooting Overview

Contents

As with most errors, first take a look at the contents of the MariaDB error log. If dealing with a deadlock, setting the innodb_print_all_deadlocks option (off by default) will output details of all deadlocks to the error log.

It can also help to enable the various InnoDB Monitors relating to the problem you are experiencing. There are four types: the standard InnoDB monitor, the InnoDB Lock Monitor, InnoDB Tablespace Monitor and the InnoDB Table Monitor.

Running CHECK TABLE will help determine whether there are errors in the table.

For problems with the InnoDB Data Dictionary, see InnoDB Data Dictionary Troubleshooting.

5.3.2.3.2 InnoDB Data Dictionary Troubleshooting

Contents

Can't Open File

If InnoDB returns something like the following error:

```
ERROR 1016: Can't open file: 'x.ibd'. (errno: 1)
```

it may be that an orphan `.frm` file exists. Something like the following may also appear in the error log:

```
InnoDB: Cannot find table test/x from the internal data dictionary
InnoDB: of InnoDB though the .frm file for the table exists. Maybe you
InnoDB: have deleted and recreated InnoDB data files but have forgotten
InnoDB: to delete the corresponding .frm files of InnoDB tables?
```

If this is the case, as the text describes, delete the orphan `.frm` file on the filesystem.

Removing Orphan Intermediate Tables

An orphan intermediate table may prevent you from dropping the tablespace even if it is otherwise empty, and generally takes up unnecessary space.

It may come about if MariaDB exits in the middle of an ALTER TABLE ... ALGORITHM=INPLACE operation. They will be

listed in the INFORMATION_SCHEMA.INNODB_SYS_TABLES table, and always start with an `#sql-ib` prefix. The accompanying `.frm` file also begins with `#sql-`, but has a different name.

To identify orphan tables, run:

```
SELECT * FROM INFORMATION_SCHEMA.INNODB_SYS_TABLES WHERE NAME LIKE '%#sql%';
```

When innodb_file_per_table is set, the `#sql-*.ibd` file will also be visible in the database directory.

To remove an orphan intermediate table:

- Rename the `#sql-*.frm` file (in the database directory) to match the base name of the orphan intermediate table, for example:

```
mv #sql-36ab_2.frm #sql-ib87-856498050.frm
```

- Drop the table, for example:

```
DROP TABLE `#mysql50##sql-ib87-856498050`;
```

5.3.2.3.3 InnoDB Recovery Modes

The InnoDB recovery mode is a mode used for recovering from emergency situations. You should ensure you have a backup of your database before making changes in case you need to restore it. The innodb_force_recovery server system variable sets the recovery mode. A mode of 0 is normal use, while the higher the mode, the more stringent the restrictions. Higher modes incorporate all limitations of the lower modes.

> The recovery mode should never be set to a value other than zero except in an emergency situation.

> Please note that recovery mode does not repair corruptions. The corrupted files remain corrupted regardless of recovery mode. The sole purpose of recovery mode is to allow read access to the data, if at all possible.

Generally, it is best to start with a recovery mode of 1, and increase in single increments if needs be. With a recovery mode < 4, only corrupted pages should be lost. With 4, secondary indexes could be corrupted. With 5, results could be inconsistent and secondary indexes could be corrupted (even if they were not with 4). A value of 6 leaves pages in an obsolete state, which might cause more corruption.

Until MariaDB 10.2.7, mode `0` was the only mode permitting changes to the data. From MariaDB 10.2.7, write transactions are permitted with mode `3` or less.

To recover the tables, you can execute SELECTs to dump data, and DROP TABLE (when write transactions are permitted) to remove corrupted tables.

The following modes are available:

Recovery Modes

Recovery mode behaviour differs per version (server/storage/innobase/include/srv0srv.h)

MariaDB 10.4 and before:

Mode	Description
0	The default mode while InnoDB is running normally. Until MariaDB 10.2.7, it was the only mode permitting changes to the data. From MariaDB 10.2.7, write transactions are permitted with innodb_force_recovery<=3.
1	(SRV_FORCE_IGNORE_CORRUPT) allows the the server to keep running even if corrupt pages are detected. It does so by making redo log based recovery ignore certain errors, such as missing data files or corrupted data pages. Any redo log for affected files or pages will be skipped. You can facilitate dumping tables by getting the SELECT * FROM table_name statement to jump over corrupt indexes and pages.
2	(SRV_FORCE_NO_BACKGROUND) stops the master thread from running, preventing a crash that occurs during a purge. No purge will be performed, so the undo logs will keep growing.

3	(SRV_FORCE_NO_TRX_UNDO) does not roll back transactions after the crash recovery. Does not affect rollback of currently active transactions. Starting with MariaDB 10.2.7, will also prevent some undo-generating background tasks from running. These tasks could hit a lock wait due to the recovered incomplete transactions whose rollback is being prevented.
4	(SRV_FORCE_NO_IBUF_MERGE) does not calculate tables statistics and prevents insert buffer merges.
5	(SRV_FORCE_NO_UNDO_LOG_SCAN) treats incomplete transactions as committed, and does not look at the undo logs when starting.
6	(SRV_FORCE_NO_LOG_REDO) does not perform redo log roll-forward as part of recovery. Running queries that require indexes are likely to fail with this mode active. However, if a table dump still causes a crash, you can try using a `SELECT * FROM tab ORDER BY primary_key DESC` to dump all the data portion after the corrupted part.

From MariaDB 10.5 to MariaDB 10.6.4:

Mode	Description
0	The default mode while InnoDB is running normally. Write transactions are permitted with innodb_force_recovery<=4.
1	(SRV_FORCE_IGNORE_CORRUPT) allows the the server to keep running even if corrupt pages are detected. It does so by making redo log based recovery ignore certain errors, such as missing data files or corrupted data pages. Any redo log for affected files or pages will be skipped. You can facilitate dumping tables by getting the SELECT * FROM table_name statement to jump over corrupt indexes and pages.
2	(SRV_FORCE_NO_BACKGROUND) stops the master thread from running, preventing a crash that occurs during a purge. No purge will be performed, so the undo logs will keep growing.
3	(SRV_FORCE_NO_TRX_UNDO) does not roll back transactions after the crash recovery. Does not affect rollback of currently active transactions. Will also prevent some undo-generating background tasks from running. These tasks could hit a lock wait due to the recovered incomplete transactions whose rollback is being prevented.
4	(SRV_FORCE_NO_IBUF_MERGE) The same as 3.
5	(SRV_FORCE_NO_UNDO_LOG_SCAN) treats incomplete transactions as committed, and does not look at the undo logs when starting.
6	(SRV_FORCE_NO_LOG_REDO) does not perform redo log roll-forward as part of recovery. Running queries that require indexes are likely to fail with this mode active. However, if a table dump still causes a crash, you can try using a `SELECT * FROM tab ORDER BY primary_key DESC` to dump all the data portion after the corrupted part.

From MariaDB 10.6.5

Mode	Description
0	The default mode while InnoDB is running normally. Write transactions are permitted with innodb_force_recovery<=4.
1	(SRV_FORCE_IGNORE_CORRUPT) allows the the server to keep running even if corrupt pages are detected. It does so by making redo log based recovery ignore certain errors, such as missing data files or corrupted data pages. Any redo log for affected files or pages will be skipped. You can facilitate dumping tables by getting the SELECT * FROM table_name statement to jump over corrupt indexes and pages.
2	(SRV_FORCE_NO_BACKGROUND) stops the master thread from running, preventing a crash that occurs during a purge. No purge will be performed, so the undo logs will keep growing.
3	(SRV_FORCE_NO_TRX_UNDO) does not roll back DML transactions after the crash recovery. Does not affect rollback of currently active DML transactions. Will also prevent some undo-generating background tasks from running. These tasks could hit a lock wait due to the recovered incomplete transactions whose rollback is being prevented.
4	(SRV_FORCE_NO_DDL_UNDO) does not roll back transactions after the crash recovery. Does not affect rollback of currently active transactions. Will also prevent some undo-generating background tasks from running. These tasks could hit a lock wait due to the recovered incomplete transactions whose rollback is being prevented.
5	(SRV_FORCE_NO_UNDO_LOG_SCAN) treats incomplete transactions as committed, and does not look at the undo logs when starting. Any DDL log for InnoDB tables will be essentially ignored by InnoDB, but the server will start up

6	(SRV_FORCE_NO_LOG_REDO) does not perform redo log roll-forward as part of recovery. Running queries that require indexes are likely to fail with this mode active. However, if a table dump still causes a crash, you can try using a `SELECT * FROM tab ORDER BY primary_key DESC` to dump all the data portion after the corrupted part.

Note also that XtraDB (<= MariaDB 10.2.6) by default will crash the server when it detects corrupted data in a single-table tablespace. This behaviour can be changed - see the innodb_corrupt_table_action system variable.

Fixing Things

Try to set innodb_force_recovery to 1 and start mariadb. If that fails, try a value of "2". If a value of 2 works, then there is a chance the only corruption you have experienced is within the innodb "undo logs". If that gets mariadb started, you should be able to dump your database with mariadb-dump. You can verify any other issues with any tables by running mariadb-check --all-databases.

If you were able to successfully dump your databases, or had previously known good backups, drop your database(s) from the mariadb command line like "DROP DATABASE yourdatabase". Stop mariadb. Go to /var/lib/mysql (or whereever your mysql data directory is located) and "rm -i ib*". Start mariadb, create the database(s) you dropped ("CREATE DATABASE yourdatabase"), and then import your most recent dumps: "mysql < mydatabasedump.sql"

5.3.2.3.4 Troubleshooting Row Size Too Large Errors with InnoDB

With InnoDB, users can see the following message as an error or warning:

```
ERROR 1118 (42000): Row size too large (> 8126). Changing some columns to
TEXT or BLOB may help. In current row format, BLOB prefix of 0 bytes is stored
inline.
```

And they can also see the following message as an error or warning in the error log:

```
[Warning] InnoDB: Cannot add field col in table db1.tab because after adding it,
the row size is 8478 which is greater than maximum allowed size (8126) for a
record on index leaf page.
```

Contents

These messages indicate that the table's definition allows rows that the table's InnoDB row format can't actually store.

These messages are raised in the following cases:

- If InnoDB strict mode is **enabled** and if a DDL statement is executed that touches the table, such as CREATE TABLE or ALTER TABLE, then InnoDB will raise an **error** with this message
- If InnoDB strict mode is **disabled** and if a DDL statement is executed that touches the table, such as CREATE TABLE or ALTER TABLE, then InnoDB will raise a **warning** with this message.
- Regardless of whether InnoDB strict mode is enabled, if a DML statement is executed that attempts to write a row that the table's InnoDB row format can't store, then InnoDB will raise an **error** with this message.

Example of the Problem

Here is an example of the problem:

```
SET GLOBAL innodb_default_row_format='dynamic';

SET SESSION innodb_strict_mode=ON;

CREATE OR REPLACE TABLE tab (
   col1 varchar(40) NOT NULL,
   col2 varchar(40) NOT NULL,
   col3 varchar(40) NOT NULL,
   col4 varchar(40) NOT NULL,
   col5 varchar(40) NOT NULL,
   col6 varchar(40) NOT NULL,
   col7 varchar(40) NOT NULL,
   col8 varchar(40) NOT NULL,
   col9 varchar(40) NOT NULL,
   col10 varchar(40) NOT NULL,
   col11 varchar(40) NOT NULL,
   col12 varchar(40) NOT NULL,
   col13 varchar(40) NOT NULL,
   col14 varchar(40) NOT NULL,
   col15 varchar(40) NOT NULL,
   col16 varchar(40) NOT NULL,
   col17 varchar(40) NOT NULL,
   col18 varchar(40) NOT NULL,
   col19 varchar(40) NOT NULL,
   col20 varchar(40) NOT NULL,
   col21 varchar(40) NOT NULL,
   col22 varchar(40) NOT NULL,
   col23 varchar(40) NOT NULL,
   col24 varchar(40) NOT NULL,
   col25 varchar(40) NOT NULL,
   col26 varchar(40) NOT NULL,
   col27 varchar(40) NOT NULL,
   col28 varchar(40) NOT NULL,
   col29 varchar(40) NOT NULL,
   col30 varchar(40) NOT NULL,
   col31 varchar(40) NOT NULL,
   col32 varchar(40) NOT NULL,
   col33 varchar(40) NOT NULL,
   col34 varchar(40) NOT NULL,
   col35 varchar(40) NOT NULL,
   col36 varchar(40) NOT NULL,
   col37 varchar(40) NOT NULL,
   col38 varchar(40) NOT NULL,
   col39 varchar(40) NOT NULL,
   col40 varchar(40) NOT NULL,
   col41 varchar(40) NOT NULL,
   col42 varchar(40) NOT NULL,
   col43 varchar(40) NOT NULL,
   col44 varchar(40) NOT NULL,
   col45 varchar(40) NOT NULL,
   col46 varchar(40) NOT NULL,
   col47 varchar(40) NOT NULL,
   col48 varchar(40) NOT NULL,
   col49 varchar(40) NOT NULL,
   col50 varchar(40) NOT NULL,
   col51 varchar(40) NOT NULL,
   col52 varchar(40) NOT NULL,
   col53 varchar(40) NOT NULL,
   col54 varchar(40) NOT NULL,
   col55 varchar(40) NOT NULL,
   col56 varchar(40) NOT NULL,
   col57 varchar(40) NOT NULL,
   col58 varchar(40) NOT NULL,
   col59 varchar(40) NOT NULL,
   col60 varchar(40) NOT NULL,
   col61 varchar(40) NOT NULL,
   col62 varchar(40) NOT NULL,
   col63 varchar(40) NOT NULL,
   col64 varchar(40) NOT NULL,
   col65 varchar(40) NOT NULL,
```

```
    col66 varchar(40) NOT NULL,
    col67 varchar(40) NOT NULL,
    col68 varchar(40) NOT NULL,
    col69 varchar(40) NOT NULL,
    col70 varchar(40) NOT NULL,
    col71 varchar(40) NOT NULL,
    col72 varchar(40) NOT NULL,
    col73 varchar(40) NOT NULL,
    col74 varchar(40) NOT NULL,
    col75 varchar(40) NOT NULL,
    col76 varchar(40) NOT NULL,
    col77 varchar(40) NOT NULL,
    col78 varchar(40) NOT NULL,
    col79 varchar(40) NOT NULL,
    col80 varchar(40) NOT NULL,
    col81 varchar(40) NOT NULL,
    col82 varchar(40) NOT NULL,
    col83 varchar(40) NOT NULL,
    col84 varchar(40) NOT NULL,
    col85 varchar(40) NOT NULL,
    col86 varchar(40) NOT NULL,
    col87 varchar(40) NOT NULL,
    col88 varchar(40) NOT NULL,
    col89 varchar(40) NOT NULL,
    col90 varchar(40) NOT NULL,
    col91 varchar(40) NOT NULL,
    col92 varchar(40) NOT NULL,
    col93 varchar(40) NOT NULL,
    col94 varchar(40) NOT NULL,
    col95 varchar(40) NOT NULL,
    col96 varchar(40) NOT NULL,
    col97 varchar(40) NOT NULL,
    col98 varchar(40) NOT NULL,
    col99 varchar(40) NOT NULL,
    col100 varchar(40) NOT NULL,
    col101 varchar(40) NOT NULL,
    col102 varchar(40) NOT NULL,
    col103 varchar(40) NOT NULL,
    col104 varchar(40) NOT NULL,
    col105 varchar(40) NOT NULL,
    col106 varchar(40) NOT NULL,
    col107 varchar(40) NOT NULL,
    col108 varchar(40) NOT NULL,
    col109 varchar(40) NOT NULL,
    col110 varchar(40) NOT NULL,
    col111 varchar(40) NOT NULL,
    col112 varchar(40) NOT NULL,
    col113 varchar(40) NOT NULL,
    col114 varchar(40) NOT NULL,
    col115 varchar(40) NOT NULL,
    col116 varchar(40) NOT NULL,
    col117 varchar(40) NOT NULL,
    col118 varchar(40) NOT NULL,
    col119 varchar(40) NOT NULL,
    col120 varchar(40) NOT NULL,
    col121 varchar(40) NOT NULL,
    col122 varchar(40) NOT NULL,
    col123 varchar(40) NOT NULL,
    col124 varchar(40) NOT NULL,
    col125 varchar(40) NOT NULL,
    col126 varchar(40) NOT NULL,
    col127 varchar(40) NOT NULL,
    col128 varchar(40) NOT NULL,
    col129 varchar(40) NOT NULL,
    col130 varchar(40) NOT NULL,
    col131 varchar(40) NOT NULL,
    col132 varchar(40) NOT NULL,
    col133 varchar(40) NOT NULL,
    col134 varchar(40) NOT NULL,
    col135 varchar(40) NOT NULL,
    col136 varchar(40) NOT NULL,
    col137 varchar(40) NOT NULL,
    col138 varchar(40) NOT NULL,
    col139 varchar(40) NOT NULL,
    col140 varchar(40) NOT NULL,
```

```
    col141 varchar(40) NOT NULL,
    col142 varchar(40) NOT NULL,
    col143 varchar(40) NOT NULL,
    col144 varchar(40) NOT NULL,
    col145 varchar(40) NOT NULL,
    col146 varchar(40) NOT NULL,
    col147 varchar(40) NOT NULL,
    col148 varchar(40) NOT NULL,
    col149 varchar(40) NOT NULL,
    col150 varchar(40) NOT NULL,
    col151 varchar(40) NOT NULL,
    col152 varchar(40) NOT NULL,
    col153 varchar(40) NOT NULL,
    col154 varchar(40) NOT NULL,
    col155 varchar(40) NOT NULL,
    col156 varchar(40) NOT NULL,
    col157 varchar(40) NOT NULL,
    col158 varchar(40) NOT NULL,
    col159 varchar(40) NOT NULL,
    col160 varchar(40) NOT NULL,
    col161 varchar(40) NOT NULL,
    col162 varchar(40) NOT NULL,
    col163 varchar(40) NOT NULL,
    col164 varchar(40) NOT NULL,
    col165 varchar(40) NOT NULL,
    col166 varchar(40) NOT NULL,
    col167 varchar(40) NOT NULL,
    col168 varchar(40) NOT NULL,
    col169 varchar(40) NOT NULL,
    col170 varchar(40) NOT NULL,
    col171 varchar(40) NOT NULL,
    col172 varchar(40) NOT NULL,
    col173 varchar(40) NOT NULL,
    col174 varchar(40) NOT NULL,
    col175 varchar(40) NOT NULL,
    col176 varchar(40) NOT NULL,
    col177 varchar(40) NOT NULL,
    col178 varchar(40) NOT NULL,
    col179 varchar(40) NOT NULL,
    col180 varchar(40) NOT NULL,
    col181 varchar(40) NOT NULL,
    col182 varchar(40) NOT NULL,
    col183 varchar(40) NOT NULL,
    col184 varchar(40) NOT NULL,
    col185 varchar(40) NOT NULL,
    col186 varchar(40) NOT NULL,
    col187 varchar(40) NOT NULL,
    col188 varchar(40) NOT NULL,
    col189 varchar(40) NOT NULL,
    col190 varchar(40) NOT NULL,
    col191 varchar(40) NOT NULL,
    col192 varchar(40) NOT NULL,
    col193 varchar(40) NOT NULL,
    col194 varchar(40) NOT NULL,
    col195 varchar(40) NOT NULL,
    col196 varchar(40) NOT NULL,
    col197 varchar(40) NOT NULL,
    col198 varchar(40) NOT NULL,
    PRIMARY KEY (col1)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
ERROR 1118 (42000): Row size too large (> 8126). Changing some columns to
TEXT or BLOB may help. In current row format, BLOB prefix of 0 bytes is stored inline.
```

Root Cause of the Problem

The root cause is that InnoDB has a maximum row size that is roughly equivalent to half of the value of the innodb_page_size system variable. See InnoDB Row Formats Overview: Maximum Row Size for more information.

InnoDB's row formats work around this limit by storing certain kinds of variable-length columns on overflow pages. However, different row formats can store different types of data on overflow pages. Some row formats can store more data in overflow pages than others. For example, the DYNAMIC and COMPRESSED row formats can store the most data in overflow pages. To learn how the various InnoDB row formats use overflow pages, see the following pages:

- InnoDB REDUNDANT Row Format: Overflow Pages with the REDUNDANT Row Format
- InnoDB COMPACT Row Format: Overflow Pages with the COMPACT Row Format
- InnoDB DYNAMIC Row Format: Overflow Pages with the DYNAMIC Row Format
- InnoDB COMPRESSED Row Format: Overflow Pages with the COMPRESSED Row Format

Checking Existing Tables for the Problem

InnoDB does not currently have an easy way to check all existing tables to determine which tables have this problem. See MDEV-20400 for more information.

One method to check a single existing table for this problem is to enable InnoDB strict mode, and then try to create a duplicate of the table with CREATE TABLE ... LIKE. If the table has this problem, then the operation will fail. For example:

```
SET SESSION innodb_strict_mode=ON;

CREATE TABLE tab_dup LIKE tab;
ERROR 1118 (42000): Row size too large (> 8126). Changing some columns to
TEXT or BLOB may help. In current row format, BLOB prefix of 0 bytes is stored inline.
```

Finding All Tables That Currently Have the Problem

The following shell script will read through a MariaDB server to identify every table that has a row size definition that is too large for its row format and the server's page size. It runs on most common distributions of Linux.

To run the script, copy the code below to a shell-script named `rowsize.sh`, make it executable with the command `chmod 755 ./rowsize.sh`, and invoke it with the following parameters:

```
./rowsize.sh host user password
```

When the script runs, it displays the name of the temporary database it creates, so that if the script is interrupted before cleaning up, the database can be easily identified and removed manually.

As the script runs it will output one line reporting the database and tablename for each table it finds that has the oversize row problem. If it finds none, it will print the following message: "No tables with rows size too big found."

In either case, the script prints one final line to announce when it's done: `./rowsize.sh done.`

```
#!/bin/bash

[ -z "$3" ] && echo "Usage: $0 host user password" >&2 && exit 1

dt="tmp_$RANDOM$RANDOM"

mysql -h $1 -u $2 -p$3 -ABNe "create database $dt;"
[ $? -ne 0 ] && echo "Error: $0 terminating" >&2 exit 1

echo
echo "Created temporary database ${dt} on host $1"
echo

c=0
for d in $(mysql -h $1 -u $2 -p$3 -ABNe "show databases;" | egrep -iv
"information_schema|mysql|performance_schema|$dt")
do
 for t in $(mysql -h $1 -u $2 -p$3 -ABNe "show tables;" $d)
 do
  tc=$(mysql -h $1 -u $2 -p$3 -ABNe "show create table $t\\G" $d | egrep -iv "^\*|^$t")

  echo $tc | grep -iq "ROW_FORMAT"
  if [ $? -ne 0 ]
  then
   tf=$(mysql -h $1 -u $2 -p$3 -ABNe "select row_format from
information_schema.innodb_sys_tables where name = '${d}/${t}';")
   tc="$tc ROW_FORMAT=$tf"
  fi

  ef="/tmp/e$RANDOM$RANDOM"
  mysql -h $1 -u $2 -p$3 -ABNe "set innodb_strict_mode=1; set foreign_key_checks=0; ${tc};" $dt
>/dev/null  2>$ef
  [ $? -ne 0 ] && cat $ef | grep -q "Row size too large" && echo "${d}.${t}" && let c++ ||
mysql -h $1 -u $2 -p$3 -ABNe "drop table if exists ${t};" $dt
  rm -f $ef
 done
done
mysql -h $1 -u $2 -p$3 -ABNe "set innodb_strict_mode=1; drop database $dt;"
[ $c -eq 0 ] && echo "No tables with rows size too large found." || echo && echo "$c tables
found with row size too large."
echo
echo "$0 done."
```

Solving the Problem

There are several potential solutions available to solve this problem.

Converting the Table to the DYNAMIC Row Format

If the table is using either the REDUNDANT or the COMPACT row format, then one potential solution to this problem is to convert the table to use the DYNAMIC row format instead.

If your tables were originally created on an older version of MariaDB or MySQL, then your table may be using one of InnoDB's older row formats:

- In MariaDB 10.1 and before, and in MySQL 5.6 and before, the COMPACT row format was the default row format.
- In MySQL 4.1 and before, the REDUNDANT row format was the default row format.

The DYNAMIC row format can store more data on overflow pages than these older row formats, so this row format may actually be able to store the table's data safely. See InnoDB DYNAMIC Row Format: Overflow Pages with the DYNAMIC Row Format for more information.

Therefore, a potential solution to the *Row size too large* error is to convert the table to use the DYNAMIC row format. For example:

```
ALTER TABLE tab ROW_FORMAT=DYNAMIC;
```

You can use the INNODB_SYS_TABLES table in the information_schema database to find all tables that use the REDUNDANT or the COMPACT row formats. This is helpful if you would like to convert all of your tables that you still use the older row formats to the DYNAMIC row format. For example, the following query can find those tables, while excluding InnoDB's internal system tables:

```
SELECT NAME, ROW_FORMAT
FROM information_schema.INNODB_SYS_TABLES
WHERE ROW_FORMAT IN('Redundant', 'Compact')
AND NAME NOT IN('SYS_DATAFILES', 'SYS_FOREIGN', 'SYS_FOREIGN_COLS', 'SYS_TABLESPACES', 'SYS_VIRTU
```

In MariaDB 10.2 and later, the DYNAMIC row format is the default row format. If your tables were originally created on one of these newer versions, then they may already be using this row format. In that case, you may need to try the next solution.

Fitting More Columns on Overflow Pages

If the table is already using the DYNAMIC row format, then another potential solution to this problem is to change the table schema, so that the row format can store more columns on overflow pages.

In order for InnoDB to store some variable-length columns on overflow pages, the length of those columns may need to be increased.

Therefore, a counter-intuitive solution to the *Row size too large* error in a lot of cases is actually to **increase** the length of some variable-length columns, so that InnoDB's row format can store them on overflow pages.

Some possible ways to change the table schema are listed below.

Converting Some Columns to `BLOB` or `TEXT`

For BLOB and TEXT columns, the DYNAMIC row format can store these columns on overflow pages. See InnoDB DYNAMIC Row Format: Overflow Pages with the DYNAMIC Row Format for more information.

Therefore, a potential solution to the *Row size too large* error is to convert some columns to the BLOB or TEXT data types.

Increasing the Length of `VARBINARY` Columns

For VARBINARY columns, the DYNAMIC row format can only store these columns on overflow pages if the maximum length of the column is 256 bytes or longer. See InnoDB DYNAMIC Row Format: Overflow Pages with the DYNAMIC Row Format for more information.

Therefore, a potential solution to the *Row size too large* error is to ensure that all VARBINARY columns are at least as long as `varbinary(256)`.

Increasing the Length of `VARCHAR` Columns

For VARCHAR columns, the DYNAMIC row format can only store these columns on overflow pages if the maximum length of the column is 256 bytes or longer. See InnoDB DYNAMIC Row Format: Overflow Pages with the DYNAMIC Row Format for more information.

The original table schema shown earlier on this page causes the *Row size too large* error, because all of the table's VARCHAR columns are smaller than 256 bytes, which means that they have to be stored on the row's main data page.

Therefore, a potential solution to the *Row size too large* error is to ensure that all VARCHAR columns are at least as long as 256 bytes. The number of characters required to reach the 256 byte limit depends on the character set used by the column.

For example, when using InnoDB's DYNAMIC row format and a default character set of latin1 (which requires up to 1 byte per character), the 256 byte limit means that a VARCHAR column will only be stored on overflow pages if it is at least as large as a `varchar(256)`:

```
SET GLOBAL innodb_default_row_format='dynamic';
SET SESSION innodb_strict_mode=ON;
CREATE OR REPLACE TABLE tab (
   col1 varchar(256) NOT NULL,
   col2 varchar(256) NOT NULL,
   col3 varchar(256) NOT NULL,
   col4 varchar(256) NOT NULL,
   col5 varchar(256) NOT NULL,
   col6 varchar(256) NOT NULL,
   col7 varchar(256) NOT NULL,
   col8 varchar(256) NOT NULL,
   col9 varchar(256) NOT NULL,
   col10 varchar(256) NOT NULL,
   col11 varchar(256) NOT NULL,
   col12 varchar(256) NOT NULL,
   col13 varchar(256) NOT NULL,
   col14 varchar(256) NOT NULL,
   col15 varchar(256) NOT NULL,
```

```
    col16 varchar(256) NOT NULL,
    col17 varchar(256) NOT NULL,
    col18 varchar(256) NOT NULL,
    col19 varchar(256) NOT NULL,
    col20 varchar(256) NOT NULL,
    col21 varchar(256) NOT NULL,
    col22 varchar(256) NOT NULL,
    col23 varchar(256) NOT NULL,
    col24 varchar(256) NOT NULL,
    col25 varchar(256) NOT NULL,
    col26 varchar(256) NOT NULL,
    col27 varchar(256) NOT NULL,
    col28 varchar(256) NOT NULL,
    col29 varchar(256) NOT NULL,
    col30 varchar(256) NOT NULL,
    col31 varchar(256) NOT NULL,
    col32 varchar(256) NOT NULL,
    col33 varchar(256) NOT NULL,
    col34 varchar(256) NOT NULL,
    col35 varchar(256) NOT NULL,
    col36 varchar(256) NOT NULL,
    col37 varchar(256) NOT NULL,
    col38 varchar(256) NOT NULL,
    col39 varchar(256) NOT NULL,
    col40 varchar(256) NOT NULL,
    col41 varchar(256) NOT NULL,
    col42 varchar(256) NOT NULL,
    col43 varchar(256) NOT NULL,
    col44 varchar(256) NOT NULL,
    col45 varchar(256) NOT NULL,
    col46 varchar(256) NOT NULL,
    col47 varchar(256) NOT NULL,
    col48 varchar(256) NOT NULL,
    col49 varchar(256) NOT NULL,
    col50 varchar(256) NOT NULL,
    col51 varchar(256) NOT NULL,
    col52 varchar(256) NOT NULL,
    col53 varchar(256) NOT NULL,
    col54 varchar(256) NOT NULL,
    col55 varchar(256) NOT NULL,
    col56 varchar(256) NOT NULL,
    col57 varchar(256) NOT NULL,
    col58 varchar(256) NOT NULL,
    col59 varchar(256) NOT NULL,
    col60 varchar(256) NOT NULL,
    col61 varchar(256) NOT NULL,
    col62 varchar(256) NOT NULL,
    col63 varchar(256) NOT NULL,
    col64 varchar(256) NOT NULL,
    col65 varchar(256) NOT NULL,
    col66 varchar(256) NOT NULL,
    col67 varchar(256) NOT NULL,
    col68 varchar(256) NOT NULL,
    col69 varchar(256) NOT NULL,
    col70 varchar(256) NOT NULL,
    col71 varchar(256) NOT NULL,
    col72 varchar(256) NOT NULL,
    col73 varchar(256) NOT NULL,
    col74 varchar(256) NOT NULL,
    col75 varchar(256) NOT NULL,
    col76 varchar(256) NOT NULL,
    col77 varchar(256) NOT NULL,
    col78 varchar(256) NOT NULL,
    col79 varchar(256) NOT NULL,
    col80 varchar(256) NOT NULL,
    col81 varchar(256) NOT NULL,
    col82 varchar(256) NOT NULL,
    col83 varchar(256) NOT NULL,
    col84 varchar(256) NOT NULL,
    col85 varchar(256) NOT NULL,
    col86 varchar(256) NOT NULL,
    col87 varchar(256) NOT NULL,
    col88 varchar(256) NOT NULL,
    col89 varchar(256) NOT NULL,
    col90 varchar(256) NOT NULL,
```

```
    col91 varchar(256) NOT NULL,
    col92 varchar(256) NOT NULL,
    col93 varchar(256) NOT NULL,
    col94 varchar(256) NOT NULL,
    col95 varchar(256) NOT NULL,
    col96 varchar(256) NOT NULL,
    col97 varchar(256) NOT NULL,
    col98 varchar(256) NOT NULL,
    col99 varchar(256) NOT NULL,
    col100 varchar(256) NOT NULL,
    col101 varchar(256) NOT NULL,
    col102 varchar(256) NOT NULL,
    col103 varchar(256) NOT NULL,
    col104 varchar(256) NOT NULL,
    col105 varchar(256) NOT NULL,
    col106 varchar(256) NOT NULL,
    col107 varchar(256) NOT NULL,
    col108 varchar(256) NOT NULL,
    col109 varchar(256) NOT NULL,
    col110 varchar(256) NOT NULL,
    col111 varchar(256) NOT NULL,
    col112 varchar(256) NOT NULL,
    col113 varchar(256) NOT NULL,
    col114 varchar(256) NOT NULL,
    col115 varchar(256) NOT NULL,
    col116 varchar(256) NOT NULL,
    col117 varchar(256) NOT NULL,
    col118 varchar(256) NOT NULL,
    col119 varchar(256) NOT NULL,
    col120 varchar(256) NOT NULL,
    col121 varchar(256) NOT NULL,
    col122 varchar(256) NOT NULL,
    col123 varchar(256) NOT NULL,
    col124 varchar(256) NOT NULL,
    col125 varchar(256) NOT NULL,
    col126 varchar(256) NOT NULL,
    col127 varchar(256) NOT NULL,
    col128 varchar(256) NOT NULL,
    col129 varchar(256) NOT NULL,
    col130 varchar(256) NOT NULL,
    col131 varchar(256) NOT NULL,
    col132 varchar(256) NOT NULL,
    col133 varchar(256) NOT NULL,
    col134 varchar(256) NOT NULL,
    col135 varchar(256) NOT NULL,
    col136 varchar(256) NOT NULL,
    col137 varchar(256) NOT NULL,
    col138 varchar(256) NOT NULL,
    col139 varchar(256) NOT NULL,
    col140 varchar(256) NOT NULL,
    col141 varchar(256) NOT NULL,
    col142 varchar(256) NOT NULL,
    col143 varchar(256) NOT NULL,
    col144 varchar(256) NOT NULL,
    col145 varchar(256) NOT NULL,
    col146 varchar(256) NOT NULL,
    col147 varchar(256) NOT NULL,
    col148 varchar(256) NOT NULL,
    col149 varchar(256) NOT NULL,
    col150 varchar(256) NOT NULL,
    col151 varchar(256) NOT NULL,
    col152 varchar(256) NOT NULL,
    col153 varchar(256) NOT NULL,
    col154 varchar(256) NOT NULL,
    col155 varchar(256) NOT NULL,
    col156 varchar(256) NOT NULL,
    col157 varchar(256) NOT NULL,
    col158 varchar(256) NOT NULL,
    col159 varchar(256) NOT NULL,
    col160 varchar(256) NOT NULL,
    col161 varchar(256) NOT NULL,
    col162 varchar(256) NOT NULL,
    col163 varchar(256) NOT NULL,
    col164 varchar(256) NOT NULL,
```

```
   col165 varchar(256) NOT NULL,
   col166 varchar(256) NOT NULL,
   col167 varchar(256) NOT NULL,
   col168 varchar(256) NOT NULL,
   col169 varchar(256) NOT NULL,
   col170 varchar(256) NOT NULL,
   col171 varchar(256) NOT NULL,
   col172 varchar(256) NOT NULL,
   col173 varchar(256) NOT NULL,
   col174 varchar(256) NOT NULL,
   col175 varchar(256) NOT NULL,
   col176 varchar(256) NOT NULL,
   col177 varchar(256) NOT NULL,
   col178 varchar(256) NOT NULL,
   col179 varchar(256) NOT NULL,
   col180 varchar(256) NOT NULL,
   col181 varchar(256) NOT NULL,
   col182 varchar(256) NOT NULL,
   col183 varchar(256) NOT NULL,
   col184 varchar(256) NOT NULL,
   col185 varchar(256) NOT NULL,
   col186 varchar(256) NOT NULL,
   col187 varchar(256) NOT NULL,
   col188 varchar(256) NOT NULL,
   col189 varchar(256) NOT NULL,
   col190 varchar(256) NOT NULL,
   col191 varchar(256) NOT NULL,
   col192 varchar(256) NOT NULL,
   col193 varchar(256) NOT NULL,
   col194 varchar(256) NOT NULL,
   col195 varchar(256) NOT NULL,
   col196 varchar(256) NOT NULL,
   col197 varchar(256) NOT NULL,
   col198 varchar(256) NOT NULL,
   PRIMARY KEY (col1)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

And when using InnoDB's DYNAMIC row format and a default character set of utf8 (which requires up to 3 bytes per character), the 256 byte limit means that a VARCHAR column will only be stored on overflow pages if it is at least as large as a `varchar(86)`:

```
SET GLOBAL innodb_default_row_format='dynamic';
SET SESSION innodb_strict_mode=ON;
CREATE OR REPLACE TABLE tab (
   col1 varchar(86) NOT NULL,
   col2 varchar(86) NOT NULL,
   col3 varchar(86) NOT NULL,
   col4 varchar(86) NOT NULL,
   col5 varchar(86) NOT NULL,
   col6 varchar(86) NOT NULL,
   col7 varchar(86) NOT NULL,
   col8 varchar(86) NOT NULL,
   col9 varchar(86) NOT NULL,
   col10 varchar(86) NOT NULL,
   col11 varchar(86) NOT NULL,
   col12 varchar(86) NOT NULL,
   col13 varchar(86) NOT NULL,
   col14 varchar(86) NOT NULL,
   col15 varchar(86) NOT NULL,
   col16 varchar(86) NOT NULL,
   col17 varchar(86) NOT NULL,
   col18 varchar(86) NOT NULL,
   col19 varchar(86) NOT NULL,
   col20 varchar(86) NOT NULL,
   col21 varchar(86) NOT NULL,
   col22 varchar(86) NOT NULL,
   col23 varchar(86) NOT NULL,
   col24 varchar(86) NOT NULL,
   col25 varchar(86) NOT NULL,
   col26 varchar(86) NOT NULL,
   col27 varchar(86) NOT NULL,
   col28 varchar(86) NOT NULL,
   col29 varchar(86) NOT NULL,
   col30 varchar(86) NOT NULL,
```

```
    col31 varchar(86) NOT NULL,
    col32 varchar(86) NOT NULL,
    col33 varchar(86) NOT NULL,
    col34 varchar(86) NOT NULL,
    col35 varchar(86) NOT NULL,
    col36 varchar(86) NOT NULL,
    col37 varchar(86) NOT NULL,
    col38 varchar(86) NOT NULL,
    col39 varchar(86) NOT NULL,
    col40 varchar(86) NOT NULL,
    col41 varchar(86) NOT NULL,
    col42 varchar(86) NOT NULL,
    col43 varchar(86) NOT NULL,
    col44 varchar(86) NOT NULL,
    col45 varchar(86) NOT NULL,
    col46 varchar(86) NOT NULL,
    col47 varchar(86) NOT NULL,
    col48 varchar(86) NOT NULL,
    col49 varchar(86) NOT NULL,
    col50 varchar(86) NOT NULL,
    col51 varchar(86) NOT NULL,
    col52 varchar(86) NOT NULL,
    col53 varchar(86) NOT NULL,
    col54 varchar(86) NOT NULL,
    col55 varchar(86) NOT NULL,
    col56 varchar(86) NOT NULL,
    col57 varchar(86) NOT NULL,
    col58 varchar(86) NOT NULL,
    col59 varchar(86) NOT NULL,
    col60 varchar(86) NOT NULL,
    col61 varchar(86) NOT NULL,
    col62 varchar(86) NOT NULL,
    col63 varchar(86) NOT NULL,
    col64 varchar(86) NOT NULL,
    col65 varchar(86) NOT NULL,
    col66 varchar(86) NOT NULL,
    col67 varchar(86) NOT NULL,
    col68 varchar(86) NOT NULL,
    col69 varchar(86) NOT NULL,
    col70 varchar(86) NOT NULL,
    col71 varchar(86) NOT NULL,
    col72 varchar(86) NOT NULL,
    col73 varchar(86) NOT NULL,
    col74 varchar(86) NOT NULL,
    col75 varchar(86) NOT NULL,
    col76 varchar(86) NOT NULL,
    col77 varchar(86) NOT NULL,
    col78 varchar(86) NOT NULL,
    col79 varchar(86) NOT NULL,
    col80 varchar(86) NOT NULL,
    col81 varchar(86) NOT NULL,
    col82 varchar(86) NOT NULL,
    col83 varchar(86) NOT NULL,
    col84 varchar(86) NOT NULL,
    col85 varchar(86) NOT NULL,
    col86 varchar(86) NOT NULL,
    col87 varchar(86) NOT NULL,
    col88 varchar(86) NOT NULL,
    col89 varchar(86) NOT NULL,
    col90 varchar(86) NOT NULL,
    col91 varchar(86) NOT NULL,
    col92 varchar(86) NOT NULL,
    col93 varchar(86) NOT NULL,
    col94 varchar(86) NOT NULL,
    col95 varchar(86) NOT NULL,
    col96 varchar(86) NOT NULL,
    col97 varchar(86) NOT NULL,
    col98 varchar(86) NOT NULL,
    col99 varchar(86) NOT NULL,
    col100 varchar(86) NOT NULL,
    col101 varchar(86) NOT NULL,
    col102 varchar(86) NOT NULL,
    col103 varchar(86) NOT NULL,
    col104 varchar(86) NOT NULL,
    col105 varchar(86) NOT NULL,
```

```
    col106 varchar(86) NOT NULL,
    col107 varchar(86) NOT NULL,
    col108 varchar(86) NOT NULL,
    col109 varchar(86) NOT NULL,
    col110 varchar(86) NOT NULL,
    col111 varchar(86) NOT NULL,
    col112 varchar(86) NOT NULL,
    col113 varchar(86) NOT NULL,
    col114 varchar(86) NOT NULL,
    col115 varchar(86) NOT NULL,
    col116 varchar(86) NOT NULL,
    col117 varchar(86) NOT NULL,
    col118 varchar(86) NOT NULL,
    col119 varchar(86) NOT NULL,
    col120 varchar(86) NOT NULL,
    col121 varchar(86) NOT NULL,
    col122 varchar(86) NOT NULL,
    col123 varchar(86) NOT NULL,
    col124 varchar(86) NOT NULL,
    col125 varchar(86) NOT NULL,
    col126 varchar(86) NOT NULL,
    col127 varchar(86) NOT NULL,
    col128 varchar(86) NOT NULL,
    col129 varchar(86) NOT NULL,
    col130 varchar(86) NOT NULL,
    col131 varchar(86) NOT NULL,
    col132 varchar(86) NOT NULL,
    col133 varchar(86) NOT NULL,
    col134 varchar(86) NOT NULL,
    col135 varchar(86) NOT NULL,
    col136 varchar(86) NOT NULL,
    col137 varchar(86) NOT NULL,
    col138 varchar(86) NOT NULL,
    col139 varchar(86) NOT NULL,
    col140 varchar(86) NOT NULL,
    col141 varchar(86) NOT NULL,
    col142 varchar(86) NOT NULL,
    col143 varchar(86) NOT NULL,
    col144 varchar(86) NOT NULL,
    col145 varchar(86) NOT NULL,
    col146 varchar(86) NOT NULL,
    col147 varchar(86) NOT NULL,
    col148 varchar(86) NOT NULL,
    col149 varchar(86) NOT NULL,
    col150 varchar(86) NOT NULL,
    col151 varchar(86) NOT NULL,
    col152 varchar(86) NOT NULL,
    col153 varchar(86) NOT NULL,
    col154 varchar(86) NOT NULL,
    col155 varchar(86) NOT NULL,
    col156 varchar(86) NOT NULL,
    col157 varchar(86) NOT NULL,
    col158 varchar(86) NOT NULL,
    col159 varchar(86) NOT NULL,
    col160 varchar(86) NOT NULL,
    col161 varchar(86) NOT NULL,
    col162 varchar(86) NOT NULL,
    col163 varchar(86) NOT NULL,
    col164 varchar(86) NOT NULL,
    col165 varchar(86) NOT NULL,
    col166 varchar(86) NOT NULL,
    col167 varchar(86) NOT NULL,
    col168 varchar(86) NOT NULL,
    col169 varchar(86) NOT NULL,
    col170 varchar(86) NOT NULL,
    col171 varchar(86) NOT NULL,
    col172 varchar(86) NOT NULL,
    col173 varchar(86) NOT NULL,
    col174 varchar(86) NOT NULL,
    col175 varchar(86) NOT NULL,
    col176 varchar(86) NOT NULL,
    col177 varchar(86) NOT NULL,
    col178 varchar(86) NOT NULL,
    col179 varchar(86) NOT NULL,
    col180 varchar(86) NOT NULL,
```

```
   col181 varchar(86) NOT NULL,
   col182 varchar(86) NOT NULL,
   col183 varchar(86) NOT NULL,
   col184 varchar(86) NOT NULL,
   col185 varchar(86) NOT NULL,
   col186 varchar(86) NOT NULL,
   col187 varchar(86) NOT NULL,
   col188 varchar(86) NOT NULL,
   col189 varchar(86) NOT NULL,
   col190 varchar(86) NOT NULL,
   col191 varchar(86) NOT NULL,
   col192 varchar(86) NOT NULL,
   col193 varchar(86) NOT NULL,
   col194 varchar(86) NOT NULL,
   col195 varchar(86) NOT NULL,
   col196 varchar(86) NOT NULL,
   col197 varchar(86) NOT NULL,
   col198 varchar(86) NOT NULL,
   PRIMARY KEY (col1)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

And when using InnoDB's DYNAMIC row format and a default character set of utf8mb4 (which requires up to 4 bytes per character), the 256 byte limit means that a VARCHAR column will only be stored on overflow pages if it is at least as large as a `varchar(64)` :

```
SET GLOBAL innodb_default_row_format='dynamic';
SET SESSION innodb_strict_mode=ON;
CREATE OR REPLACE TABLE tab (
   col1 varchar(64) NOT NULL,
   col2 varchar(64) NOT NULL,
   col3 varchar(64) NOT NULL,
   col4 varchar(64) NOT NULL,
   col5 varchar(64) NOT NULL,
   col6 varchar(64) NOT NULL,
   col7 varchar(64) NOT NULL,
   col8 varchar(64) NOT NULL,
   col9 varchar(64) NOT NULL,
   col10 varchar(64) NOT NULL,
   col11 varchar(64) NOT NULL,
   col12 varchar(64) NOT NULL,
   col13 varchar(64) NOT NULL,
   col14 varchar(64) NOT NULL,
   col15 varchar(64) NOT NULL,
   col16 varchar(64) NOT NULL,
   col17 varchar(64) NOT NULL,
   col18 varchar(64) NOT NULL,
   col19 varchar(64) NOT NULL,
   col20 varchar(64) NOT NULL,
   col21 varchar(64) NOT NULL,
   col22 varchar(64) NOT NULL,
   col23 varchar(64) NOT NULL,
   col24 varchar(64) NOT NULL,
   col25 varchar(64) NOT NULL,
   col26 varchar(64) NOT NULL,
   col27 varchar(64) NOT NULL,
   col28 varchar(64) NOT NULL,
   col29 varchar(64) NOT NULL,
   col30 varchar(64) NOT NULL,
   col31 varchar(64) NOT NULL,
   col32 varchar(64) NOT NULL,
   col33 varchar(64) NOT NULL,
   col34 varchar(64) NOT NULL,
   col35 varchar(64) NOT NULL,
   col36 varchar(64) NOT NULL,
   col37 varchar(64) NOT NULL,
   col38 varchar(64) NOT NULL,
   col39 varchar(64) NOT NULL,
   col40 varchar(64) NOT NULL,
   col41 varchar(64) NOT NULL,
   col42 varchar(64) NOT NULL,
   col43 varchar(64) NOT NULL,
   col44 varchar(64) NOT NULL,
   col45 varchar(64) NOT NULL,
```

```
    col46 varchar(64) NOT NULL,
    col47 varchar(64) NOT NULL,
    col48 varchar(64) NOT NULL,
    col49 varchar(64) NOT NULL,
    col50 varchar(64) NOT NULL,
    col51 varchar(64) NOT NULL,
    col52 varchar(64) NOT NULL,
    col53 varchar(64) NOT NULL,
    col54 varchar(64) NOT NULL,
    col55 varchar(64) NOT NULL,
    col56 varchar(64) NOT NULL,
    col57 varchar(64) NOT NULL,
    col58 varchar(64) NOT NULL,
    col59 varchar(64) NOT NULL,
    col60 varchar(64) NOT NULL,
    col61 varchar(64) NOT NULL,
    col62 varchar(64) NOT NULL,
    col63 varchar(64) NOT NULL,
    col64 varchar(64) NOT NULL,
    col65 varchar(64) NOT NULL,
    col66 varchar(64) NOT NULL,
    col67 varchar(64) NOT NULL,
    col68 varchar(64) NOT NULL,
    col69 varchar(64) NOT NULL,
    col70 varchar(64) NOT NULL,
    col71 varchar(64) NOT NULL,
    col72 varchar(64) NOT NULL,
    col73 varchar(64) NOT NULL,
    col74 varchar(64) NOT NULL,
    col75 varchar(64) NOT NULL,
    col76 varchar(64) NOT NULL,
    col77 varchar(64) NOT NULL,
    col78 varchar(64) NOT NULL,
    col79 varchar(64) NOT NULL,
    col80 varchar(64) NOT NULL,
    col81 varchar(64) NOT NULL,
    col82 varchar(64) NOT NULL,
    col83 varchar(64) NOT NULL,
    col84 varchar(64) NOT NULL,
    col85 varchar(64) NOT NULL,
    col86 varchar(64) NOT NULL,
    col87 varchar(64) NOT NULL,
    col88 varchar(64) NOT NULL,
    col89 varchar(64) NOT NULL,
    col90 varchar(64) NOT NULL,
    col91 varchar(64) NOT NULL,
    col92 varchar(64) NOT NULL,
    col93 varchar(64) NOT NULL,
    col94 varchar(64) NOT NULL,
    col95 varchar(64) NOT NULL,
    col96 varchar(64) NOT NULL,
    col97 varchar(64) NOT NULL,
    col98 varchar(64) NOT NULL,
    col99 varchar(64) NOT NULL,
    col100 varchar(64) NOT NULL,
    col101 varchar(64) NOT NULL,
    col102 varchar(64) NOT NULL,
    col103 varchar(64) NOT NULL,
    col104 varchar(64) NOT NULL,
    col105 varchar(64) NOT NULL,
    col106 varchar(64) NOT NULL,
    col107 varchar(64) NOT NULL,
    col108 varchar(64) NOT NULL,
    col109 varchar(64) NOT NULL,
    col110 varchar(64) NOT NULL,
    col111 varchar(64) NOT NULL,
    col112 varchar(64) NOT NULL,
    col113 varchar(64) NOT NULL,
    col114 varchar(64) NOT NULL,
    col115 varchar(64) NOT NULL,
    col116 varchar(64) NOT NULL,
    col117 varchar(64) NOT NULL,
    col118 varchar(64) NOT NULL,
    col119 varchar(64) NOT NULL,
    col120 varchar(64) NOT NULL,
```

```
    col121 varchar(64) NOT NULL,
    col122 varchar(64) NOT NULL,
    col123 varchar(64) NOT NULL,
    col124 varchar(64) NOT NULL,
    col125 varchar(64) NOT NULL,
    col126 varchar(64) NOT NULL,
    col127 varchar(64) NOT NULL,
    col128 varchar(64) NOT NULL,
    col129 varchar(64) NOT NULL,
    col130 varchar(64) NOT NULL,
    col131 varchar(64) NOT NULL,
    col132 varchar(64) NOT NULL,
    col133 varchar(64) NOT NULL,
    col134 varchar(64) NOT NULL,
    col135 varchar(64) NOT NULL,
    col136 varchar(64) NOT NULL,
    col137 varchar(64) NOT NULL,
    col138 varchar(64) NOT NULL,
    col139 varchar(64) NOT NULL,
    col140 varchar(64) NOT NULL,
    col141 varchar(64) NOT NULL,
    col142 varchar(64) NOT NULL,
    col143 varchar(64) NOT NULL,
    col144 varchar(64) NOT NULL,
    col145 varchar(64) NOT NULL,
    col146 varchar(64) NOT NULL,
    col147 varchar(64) NOT NULL,
    col148 varchar(64) NOT NULL,
    col149 varchar(64) NOT NULL,
    col150 varchar(64) NOT NULL,
    col151 varchar(64) NOT NULL,
    col152 varchar(64) NOT NULL,
    col153 varchar(64) NOT NULL,
    col154 varchar(64) NOT NULL,
    col155 varchar(64) NOT NULL,
    col156 varchar(64) NOT NULL,
    col157 varchar(64) NOT NULL,
    col158 varchar(64) NOT NULL,
    col159 varchar(64) NOT NULL,
    col160 varchar(64) NOT NULL,
    col161 varchar(64) NOT NULL,
    col162 varchar(64) NOT NULL,
    col163 varchar(64) NOT NULL,
    col164 varchar(64) NOT NULL,
    col165 varchar(64) NOT NULL,
    col166 varchar(64) NOT NULL,
    col167 varchar(64) NOT NULL,
    col168 varchar(64) NOT NULL,
    col169 varchar(64) NOT NULL,
    col170 varchar(64) NOT NULL,
    col171 varchar(64) NOT NULL,
    col172 varchar(64) NOT NULL,
    col173 varchar(64) NOT NULL,
    col174 varchar(64) NOT NULL,
    col175 varchar(64) NOT NULL,
    col176 varchar(64) NOT NULL,
    col177 varchar(64) NOT NULL,
    col178 varchar(64) NOT NULL,
    col179 varchar(64) NOT NULL,
    col180 varchar(64) NOT NULL,
    col181 varchar(64) NOT NULL,
    col182 varchar(64) NOT NULL,
    col183 varchar(64) NOT NULL,
    col184 varchar(64) NOT NULL,
    col185 varchar(64) NOT NULL,
    col186 varchar(64) NOT NULL,
    col187 varchar(64) NOT NULL,
    col188 varchar(64) NOT NULL,
    col189 varchar(64) NOT NULL,
    col190 varchar(64) NOT NULL,
    col191 varchar(64) NOT NULL,
    col192 varchar(64) NOT NULL,
    col193 varchar(64) NOT NULL,
    col194 varchar(64) NOT NULL,
    col195 varchar(64) NOT NULL,
```

```
   col196 varchar(64) NOT NULL,
   col197 varchar(64) NOT NULL,
   col198 varchar(64) NOT NULL,
   PRIMARY KEY (col1)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Working Around the Problem

There are a few ways to work around this problem.

If you would like a solution for the problem instead of just working around it, then see the solutions mentioned in the previous section.

Refactoring the Table into Multiple Tables

A *safe* workaround is to refactor the single wide table, so that its columns are spread among multiple tables.

This workaround can even work if your table is so wide that the previous solutions have failed to solve them problem for your table.

Refactoring Some Columns into JSON

A *safe* workaround is to refactor some of the columns into a JSON document.

The JSON document can be queried and manipulated using MariaDB's JSON functions.

The JSON document can be stored in a column that uses one of the following data types:

- TEXT: The maximum size of a TEXT column is 64 KB.
- MEDIUMTEXT: The maximum size of a MEDIUMTEXT column is 16 MB.
- LONGTEXT: The maximum size of a LONGTEXT column is 4 GB.
- JSON: This is just an alias for the LONGTEXT data type.

This workaround can even work if your table is so wide that the previous solutions have failed to solve them problem for your table.

Disabling InnoDB Strict Mode

An *unsafe* workaround is to disable InnoDB strict mode. InnoDB strict mode can be disabled by setting the innodb_strict_mode system variable to `OFF`.

For example, even though the following table schema is too large for most InnoDB row formats to store, it can still be created when InnoDB strict mode is disabled:

```
SET GLOBAL innodb_default_row_format='dynamic';
SET SESSION innodb_strict_mode=OFF;
CREATE OR REPLACE TABLE tab (
   col1 varchar(40) NOT NULL,
   col2 varchar(40) NOT NULL,
   col3 varchar(40) NOT NULL,
   col4 varchar(40) NOT NULL,
   col5 varchar(40) NOT NULL,
   col6 varchar(40) NOT NULL,
   col7 varchar(40) NOT NULL,
   col8 varchar(40) NOT NULL,
   col9 varchar(40) NOT NULL,
   col10 varchar(40) NOT NULL,
   col11 varchar(40) NOT NULL,
   col12 varchar(40) NOT NULL,
   col13 varchar(40) NOT NULL,
   col14 varchar(40) NOT NULL,
   col15 varchar(40) NOT NULL,
   col16 varchar(40) NOT NULL,
   col17 varchar(40) NOT NULL,
   col18 varchar(40) NOT NULL,
   col19 varchar(40) NOT NULL,
   col20 varchar(40) NOT NULL,
   col21 varchar(40) NOT NULL,
   col22 varchar(40) NOT NULL,
   col23 varchar(40) NOT NULL,
   col24 varchar(40) NOT NULL,
   col25 varchar(40) NOT NULL,
```

```
    col126 varchar(40) NOT NULL,
    col127 varchar(40) NOT NULL,
    col128 varchar(40) NOT NULL,
    col129 varchar(40) NOT NULL,
    col130 varchar(40) NOT NULL,
    col131 varchar(40) NOT NULL,
    col132 varchar(40) NOT NULL,
    col133 varchar(40) NOT NULL,
    col134 varchar(40) NOT NULL,
    col135 varchar(40) NOT NULL,
    col136 varchar(40) NOT NULL,
    col137 varchar(40) NOT NULL,
    col138 varchar(40) NOT NULL,
    col139 varchar(40) NOT NULL,
    col140 varchar(40) NOT NULL,
    col141 varchar(40) NOT NULL,
    col142 varchar(40) NOT NULL,
    col143 varchar(40) NOT NULL,
    col144 varchar(40) NOT NULL,
    col145 varchar(40) NOT NULL,
    col146 varchar(40) NOT NULL,
    col147 varchar(40) NOT NULL,
    col148 varchar(40) NOT NULL,
    col149 varchar(40) NOT NULL,
    col150 varchar(40) NOT NULL,
    col151 varchar(40) NOT NULL,
    col152 varchar(40) NOT NULL,
    col153 varchar(40) NOT NULL,
    col154 varchar(40) NOT NULL,
    col155 varchar(40) NOT NULL,
    col156 varchar(40) NOT NULL,
    col157 varchar(40) NOT NULL,
    col158 varchar(40) NOT NULL,
    col159 varchar(40) NOT NULL,
    col160 varchar(40) NOT NULL,
    col161 varchar(40) NOT NULL,
    col162 varchar(40) NOT NULL,
    col163 varchar(40) NOT NULL,
    col164 varchar(40) NOT NULL,
    col165 varchar(40) NOT NULL,
    col166 varchar(40) NOT NULL,
    col167 varchar(40) NOT NULL,
    col168 varchar(40) NOT NULL,
    col169 varchar(40) NOT NULL,
    col170 varchar(40) NOT NULL,
    col171 varchar(40) NOT NULL,
    col172 varchar(40) NOT NULL,
    col173 varchar(40) NOT NULL,
    col174 varchar(40) NOT NULL,
    col175 varchar(40) NOT NULL,
    col176 varchar(40) NOT NULL,
    col177 varchar(40) NOT NULL,
    col178 varchar(40) NOT NULL,
    col179 varchar(40) NOT NULL,
    col180 varchar(40) NOT NULL,
    col181 varchar(40) NOT NULL,
    col182 varchar(40) NOT NULL,
    col183 varchar(40) NOT NULL,
    col184 varchar(40) NOT NULL,
    col185 varchar(40) NOT NULL,
    col186 varchar(40) NOT NULL,
    col187 varchar(40) NOT NULL,
    col188 varchar(40) NOT NULL,
    col189 varchar(40) NOT NULL,
    col190 varchar(40) NOT NULL,
    col191 varchar(40) NOT NULL,
    col192 varchar(40) NOT NULL,
    col193 varchar(40) NOT NULL,
    col194 varchar(40) NOT NULL,
    col195 varchar(40) NOT NULL,
    col196 varchar(40) NOT NULL,
    col197 varchar(40) NOT NULL,
    col198 varchar(40) NOT NULL,
    col199 varchar(40) NOT NULL,
```

```
col100 varchar(40) NOT NULL,
col101 varchar(40) NOT NULL,
col102 varchar(40) NOT NULL,
col103 varchar(40) NOT NULL,
col104 varchar(40) NOT NULL,
col105 varchar(40) NOT NULL,
col106 varchar(40) NOT NULL,
col107 varchar(40) NOT NULL,
col108 varchar(40) NOT NULL,
col109 varchar(40) NOT NULL,
col110 varchar(40) NOT NULL,
col111 varchar(40) NOT NULL,
col112 varchar(40) NOT NULL,
col113 varchar(40) NOT NULL,
col114 varchar(40) NOT NULL,
col115 varchar(40) NOT NULL,
col116 varchar(40) NOT NULL,
col117 varchar(40) NOT NULL,
col118 varchar(40) NOT NULL,
col119 varchar(40) NOT NULL,
col120 varchar(40) NOT NULL,
col121 varchar(40) NOT NULL,
col122 varchar(40) NOT NULL,
col123 varchar(40) NOT NULL,
col124 varchar(40) NOT NULL,
col125 varchar(40) NOT NULL,
col126 varchar(40) NOT NULL,
col127 varchar(40) NOT NULL,
col128 varchar(40) NOT NULL,
col129 varchar(40) NOT NULL,
col130 varchar(40) NOT NULL,
col131 varchar(40) NOT NULL,
col132 varchar(40) NOT NULL,
col133 varchar(40) NOT NULL,
col134 varchar(40) NOT NULL,
col135 varchar(40) NOT NULL,
col136 varchar(40) NOT NULL,
col137 varchar(40) NOT NULL,
col138 varchar(40) NOT NULL,
col139 varchar(40) NOT NULL,
col140 varchar(40) NOT NULL,
col141 varchar(40) NOT NULL,
col142 varchar(40) NOT NULL,
col143 varchar(40) NOT NULL,
col144 varchar(40) NOT NULL,
col145 varchar(40) NOT NULL,
col146 varchar(40) NOT NULL,
col147 varchar(40) NOT NULL,
col148 varchar(40) NOT NULL,
col149 varchar(40) NOT NULL,
col150 varchar(40) NOT NULL,
col151 varchar(40) NOT NULL,
col152 varchar(40) NOT NULL,
col153 varchar(40) NOT NULL,
col154 varchar(40) NOT NULL,
col155 varchar(40) NOT NULL,
col156 varchar(40) NOT NULL,
col157 varchar(40) NOT NULL,
col158 varchar(40) NOT NULL,
col159 varchar(40) NOT NULL,
col160 varchar(40) NOT NULL,
col161 varchar(40) NOT NULL,
col162 varchar(40) NOT NULL,
col163 varchar(40) NOT NULL,
col164 varchar(40) NOT NULL,
col165 varchar(40) NOT NULL,
col166 varchar(40) NOT NULL,
col167 varchar(40) NOT NULL,
col168 varchar(40) NOT NULL,
col169 varchar(40) NOT NULL,
col170 varchar(40) NOT NULL,
col171 varchar(40) NOT NULL,
col172 varchar(40) NOT NULL,
col173 varchar(40) NOT NULL,
col174 varchar(40) NOT NULL,
```

```
    col175 varchar(40) NOT NULL,
    col176 varchar(40) NOT NULL,
    col177 varchar(40) NOT NULL,
    col178 varchar(40) NOT NULL,
    col179 varchar(40) NOT NULL,
    col180 varchar(40) NOT NULL,
    col181 varchar(40) NOT NULL,
    col182 varchar(40) NOT NULL,
    col183 varchar(40) NOT NULL,
    col184 varchar(40) NOT NULL,
    col185 varchar(40) NOT NULL,
    col186 varchar(40) NOT NULL,
    col187 varchar(40) NOT NULL,
    col188 varchar(40) NOT NULL,
    col189 varchar(40) NOT NULL,
    col190 varchar(40) NOT NULL,
    col191 varchar(40) NOT NULL,
    col192 varchar(40) NOT NULL,
    col193 varchar(40) NOT NULL,
    col194 varchar(40) NOT NULL,
    col195 varchar(40) NOT NULL,
    col196 varchar(40) NOT NULL,
    col197 varchar(40) NOT NULL,
    col198 varchar(40) NOT NULL,
    PRIMARY KEY (col1)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

But as mentioned above, if InnoDB strict mode is **disabled** and if a DDL statement is executed, then InnoDB will still raise a **warning** with this message. The SHOW WARNINGS statement can be used to view the warning. For example:

```
SHOW WARNINGS;
+---------+------+----------------------------------------------------------------------------
--------------------------------------------------------------------+
| Level | Code | Message |
+---------+------+----------------------------------------------------------------------------
--------------------------------------------------------------------+
| Warning | 139 | Row size too large (> 8126). Changing some columns to TEXT or BLOB may help.
In current row format, BLOB prefix of 0 bytes is stored inline. |
+---------+------+----------------------------------------------------------------------------
--------------------------------------------------------------------+
1 row in set (0.000 sec)
```

As mentioned above, even though InnoDB is allowing the table to be created, there is still an opportunity for errors. Regardless of whether InnoDB strict mode is enabled, if a DML statement is executed that attempts to write a row that the table's InnoDB row format can't store, then InnoDB will raise an **error** with this message. This creates a somewhat *unsafe* situation, because it means that the application has the chance to encounter an additional error while executing DML.

5.3.2.4 InnoDB System Variables

Contents

20. innodb_background_scrub_data_check_interval
21. innodb_background_scrub_data_compressed
22. innodb_background_scrub_data_interval
23. innodb_background_scrub_data_uncompressed
24. innodb_blocking_buffer_pool_restore
25. innodb_buf_dump_status_frequency
26. innodb_buffer_pool_chunk_size
27. innodb_buffer_pool_dump_at_shutdown
28. innodb_buffer_pool_dump_now
29. innodb_buffer_pool_dump_pct
30. innodb_buffer_pool_evict
31. innodb_buffer_pool_filename
32. innodb_buffer_pool_instances
33. innodb_buffer_pool_load_abort
34. innodb_buffer_pool_load_at_startup
35. innodb_buffer_pool_load_now
36. innodb_buffer_pool_load_pages_abort
37. innodb_buffer_pool_populate
38. innodb_buffer_pool_restore_at_startup
39. innodb_buffer_pool_shm_checksum
40. innodb_buffer_pool_shm_key
41. innodb_buffer_pool_size
42. innodb_change_buffer_dump
43. innodb_change_buffer_max_size
44. innodb_change_buffering
45. innodb_change_buffering_debug
46. innodb_checkpoint_age_target
47. innodb_checksum_algorithm
48. innodb_checksums
49. innodb_cleaner_lsn_age_factor
50. innodb_cmp_per_index_enabled
51. innodb_commit_concurrency
52. innodb_compression_algorithm
53. innodb_compression_default
54. innodb_compression_failure_threshold_pct
55. innodb_compression_level
56. innodb_compression_pad_pct_max
57. innodb_concurrency_tickets
58. innodb_corrupt_table_action
59. innodb_data_file_buffering
60. innodb_data_file_path
61. innodb_data_file_write_through
62. innodb_data_home_dir
63. innodb_deadlock_detect
64. innodb_deadlock_report
65. innodb_default_page_encryption_key
66. innodb_default_encryption_key_id
67. innodb_default_row_format
68. innodb_defragment
69. innodb_defragment_fill_factor
70. innodb_defragment_fill_factor_n_recs
71. innodb_defragment_frequency
72. innodb_defragment_n_pages
73. innodb_defragment_stats_accuracy
74. innodb_dict_size_limit
75. innodb_disable_sort_file_cache
76. innodb_disallow_writes
77. innodb_doublewrite
78. innodb_doublewrite_file
79. innodb_empty_free_list_algorithm
80. innodb_enable_unsafe_group_commit
81. innodb_encrypt_log
82. innodb_encrypt_tables
83. innodb_encrypt_temporary_tables
84. innodb_encryption_rotate_key_age
85. innodb_encryption_rotation_iops
86. innodb_encryption_threads

87. innodb_extra_rsegments
88. innodb_extra_undoslots
89. innodb_fake_changes
90. innodb_fast_checksum
91. innodb_fast_shutdown
92. innodb_fatal_semaphore_wait_threshold
93. innodb_file_format
94. innodb_file_format_check
95. innodb_file_format_max
96. innodb_file_per_table
97. innodb_fill_factor
98. innodb_flush_log_at_timeout
99. innodb_flush_log_at_trx_commit
100. innodb_flush_method
101. innodb_flush_neighbor_pages
102. innodb_flush_neighbors
103. innodb_flush_sync
104. innodb_flushing_avg_loops
105. innodb_force_load_corrupted
106. innodb_force_primary_key
107. innodb_force_recovery
108. innodb_foreground_preflush
109. innodb_ft_aux_table
110. innodb_ft_cache_size
111. innodb_ft_enable_diag_print
112. innodb_ft_enable_stopword
113. innodb_ft_max_token_size
114. innodb_ft_min_token_size
115. innodb_ft_num_word_optimize
116. innodb_ft_result_cache_limit
117. innodb_ft_server_stopword_table
118. innodb_ft_sort_pll_degree
119. innodb_ft_total_cache_size
120. innodb_ft_user_stopword_table
121. innodb_ibuf_accel_rate
122. innodb_ibuf_active_contract
123. innodb_ibuf_max_size
124. innodb_idle_flush_pct
125. innodb_immediate_scrub_data_uncompressed
126. innodb_import_table_from_xtrabackup
127. innodb_instant_alter_column_allowed
128. innodb_instrument_semaphores
129. innodb_io_capacity
130. innodb_io_capacity_max
131. innodb_kill_idle_transaction
132. innodb_large_prefix
133. innodb_lazy_drop_table
134. innodb_lock_schedule_algorithm
135. innodb_lock_wait_timeout
136. innodb_locking_fake_changes
137. innodb_locks_unsafe_for_binlog
138. innodb_log_arch_dir
139. innodb_log_arch_expire_sec
140. innodb_log_archive
141. innodb_log_block_size
142. innodb_log_buffer_size
143. innodb_log_checksum_algorithm
144. innodb_log_checksums
145. innodb_log_compressed_pages
146. innodb_log_file_buffering
147. innodb_log_file_size
148. innodb_log_file_write_through
149. innodb_log_files_in_group
150. innodb_log_group_home_dir
151. innodb_log_optimize_ddl
152. innodb_log_write_ahead_size
153. innodb_lru_flush_size
154. innodb_lru_scan_depth

155. innodb_max_bitmap_file_size
156. innodb_max_changed_pages
157. innodb_max_dirty_pages_pct
158. innodb_max_dirty_pages_pct_lwm
159. innodb_max_purge_lag
160. innodb_max_purge_lag_delay
161. innodb_max_purge_lag_wait
162. innodb_max_undo_log_size
163. innodb_merge_sort_block_size
164. innodb_mirrored_log_groups
165. innodb_mtflush_threads
166. innodb_monitor_disable
167. innodb_monitor_enable
168. innodb_monitor_reset
169. innodb_monitor_reset_all
170. innodb_numa_interleave
171. innodb_old_blocks_pct
172. innodb_old_blocks_time
173. innodb_online_alter_log_max_size
174. innodb_open_files
175. innodb_optimize_fulltext_only
176. innodb_page_cleaners
177. innodb_page_size
178. innodb_pass_corrupt_table
179. innodb_prefix_index_cluster_optimization
180. innodb_print_all_deadlocks
181. innodb_purge_batch_size
182. innodb_purge_rseg_truncate_frequency
183. innodb_purge_threads
184. innodb_random_read_ahead
185. innodb_read_ahead
186. innodb_read_ahead_threshold
187. innodb_read_io_threads
188. innodb_read_only
189. innodb_read_only_compressed
190. innodb_recovery_stats
191. innodb_recovery_update_relay_log
192. innodb_replication_delay
193. innodb_rollback_on_timeout
194. innodb_rollback_segments
195. innodb_safe_truncate
196. innodb_scrub_log
197. innodb_scrub_log_interval
198. innodb_scrub_log_speed
199. innodb_sched_priority_cleaner
200. innodb_show_locks_held
201. innodb_show_verbose_locks
202. innodb_simulate_comp_failures
203. innodb_sort_buffer_size
204. innodb_spin_wait_delay
205. innodb_stats_auto_recalc
206. innodb_stats_auto_update
207. innodb_stats_include_delete_marked
208. innodb_stats_method
209. innodb_stats_modified_counter
210. innodb_stats_on_metadata
211. innodb_stats_persistent
212. innodb_stats_persistent_sample_pages
213. innodb_stats_sample_pages
214. innodb_stats_traditional
215. innodb_stats_transient_sample_pages
216. innodb_stats_update_need_lock
217. innodb_status_output
218. innodb_status_output_locks
219. innodb_strict_mode
220. innodb_support_xa
221. innodb_sync_array_size

222. innodb_sync_spin_loops
223. innodb_table_locks
224. innodb_thread_concurrency
225. innodb_thread_concurrency_timer_based
226. innodb_thread_sleep_delay
227. innodb_temp_data_file_path
228. innodb_tmpdir
229. innodb_track_changed_pages
230. innodb_track_redo_log_now
231. innodb_truncate_temporary_tablespace_now
232. innodb_undo_directory
233. innodb_undo_log_truncate
234. innodb_undo_logs
235. innodb_undo_tablespaces
236. innodb_use_atomic_writes
237. innodb_use_fallocate
238. innodb_use_global_flush_log_at_trx_commit
239. innodb_use_mtflush
240. innodb_use_native_aio
241. innodb_use_purge_thread
242. innodb_use_stacktrace
243. innodb_use_sys_malloc
244. innodb_use_sys_stats_table
245. innodb_use_trim
246. innodb_version
247. innodb_write_io_threads

This page documents system variables related to the InnoDB storage engine. For options that are not system variables, see InnoDB Options.

See Server System Variables for a complete list of system variables and instructions on setting them.

Also see the Full list of MariaDB options, system and status variables.

have_innodb

- **Description:** If the server supports InnoDB tables, will be set to `YES`, otherwise will be set to `NO`. Removed in MariaDB 10.0, use the Information Schema PLUGINS table or SHOW ENGINES instead.
- **Scope:** Global
- **Dynamic:** No
- **Removed:** MariaDB 10.0

ignore_builtin_innodb

- **Description:** Setting this to `1` results in the built-in InnoDB storage engine being ignored. In some versions of MariaDB, XtraDB is the default and is always present, so this variable is ignored and setting it results in a warning. From MariaDB 10.0.1 to MariaDB 10.0.8, when InnoDB was the default instead of XtraDB, this variable needed to be set. Usually used in conjunction with the plugin-load=innodb=ha_innodb option to use the InnoDB plugin.
- **Commandline:** `--ignore-builtin-innodb`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`

innodb_adaptive_checkpoint

- **Description:** Replaced with innodb_adaptive_flushing_method. Controls adaptive checkpointing. InnoDB's fuzzy checkpointing can cause stalls, as many dirty blocks are flushed at once as the checkpoint age nears the maximum. Adaptive checkpointing aims for more consistent flushing, approximately `modified age / maximum checkpoint age`. Can result in larger transaction log files
  - `reflex` Similar to innodb_max_dirty_pages_pct flushing but flushes blocks constantly and contiguously based on the oldest modified age. If the age exceeds 1/2 of the maximum age capacity, flushing will be weak contiguous. If the age exceeds 3/4, flushing will be strong. Strength can be adjusted by the variable innodb_io_capacity.
  - `estimate` The default, and independent of innodb_io_capacity. If the oldest modified age exceeds 1/2 of the

maximum age capacity, blocks will be flushed every second at a rate determined by the number of modified blocks, LSN progress speed and the average age of all modified blocks.
    - `keep_average` Attempts to keep the I/O rate constant by using a shorter loop cycle of one tenth of a second. Designed for SSD cards.
- **Commandline:** `--innodb-adaptive-checkpoint=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `estimate`
- **Valid Values:** `none` or `0`, `reflex` or `1`, `estimate` or `2`, `keep_average` or `3`
- **Removed:** XtraDB 5.5 - replaced with innodb_adaptive_flushing_method

---

innodb_adaptive_flushing

- **Description:** If set to `1`, the default, the server will dynamically adjust the flush rate of dirty pages in the InnoDB buffer pool. This assists to reduce brief bursts of I/O activity. If set to `0`, adaptive flushing will only take place when the limit specified by innodb_adaptive_flushing_lwm is reached.
- **Commandline:** `--innodb-adaptive-flushing={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `ON`

---

innodb_adaptive_flushing_lwm

- **Description:** Adaptive flushing is enabled when this low water mark percentage of the InnoDB redo log capacity is reached. Takes effect even if innodb_adaptive_flushing is disabled.
- **Commandline:** `--innodb-adaptive-flushing-lwm=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `double`
- **Default Value:** `10.000000`
- **Range:** `0` to `70`

---

innodb_adaptive_flushing_method

- **Description:** Determines the method of flushing dirty blocks from the InnoDB buffer pool. If set to `native` or `0`, the original InnoDB method is used. The maximum checkpoint age is determined by the total length of all transaction log files. When the checkpoint age reaches the maximum checkpoint age, blocks are flushed. This can cause lag if there are many updates per second and many blocks with an almost identical age need to be flushed. If set to `estimate` or `1`, the default, the oldest modified age will be compared with the maximum age capacity. If it's more than 1/4 of this age, blocks are flushed every second. The number of blocks flushed is determined by the number of modified blocks, the LSN progress speed and the average age of all modified blocks. It's therefore independent of the innodb_io_capacity for the 1-second loop, but not entirely so for the 10-second loop. If set to `keep_average` or `2`, designed specifically for SSD cards, a shorter loop cycle is used in an attempt to keep the I/O rate constant. Removed in MariaDB 10.0/XtraDB 5.6 and replaced with InnoDB flushing method from MySQL 5.6.
- **Commandline:** `innodb-adaptive-flushing-method=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `enumeration`
- **Default Value:** `estimate`
- **Valid Values:** `native` or `0`, `estimate` or `1`, `keep_average` or `2`
- **Removed:** MariaDB 10.0 - replaced with InnoDB flushing method from MySQL 5.6

---

innodb_adaptive_hash_index

- **Description:** If set to `1`, the default until MariaDB 10.5, the InnoDB hash index is enabled. Based on performance testing (MDEV-17492), the InnoDB adaptive hash index helps performance in mostly read-only workloads, and could slow down performance in other environments, especially DROP TABLE, TRUNCATE TABLE, ALTER TABLE, or DROP INDEX operations.
- **Commandline:** `--innodb-adaptive-hash-index={0|1}`

- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF` (>= MariaDB 10.5), `ON` (<= MariaDB 10.4)

---

innodb_adaptive_hash_index_partitions

- **Description:** Specifies the number of partitions for use in adaptive searching. If set to `1`, no extra partitions are created. XtraDB-only. From MariaDB 10.2.6 (which uses InnoDB as default instead of XtraDB), this is an alias for innodb_adaptive_hash_index_parts to allow for easier upgrades.
- **Commandline:** `innodb-adaptive-hash-index-partitions=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `1`
- **Range:** `1` to `64`

---

innodb_adaptive_hash_index_parts

- **Description:** Specifies the number of partitions for use in adaptive searching. If set to `1`, no extra partitions are created.
- **Commandline:** `innodb-adaptive-hash-index-parts=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `8`
- **Range:** `1` to `512`

---

innodb_adaptive_max_sleep_delay

- **Description:** Maximum time in microseconds to automatically adjust the innodb_thread_sleep_delay value to, based on the workload. Useful in extremely busy systems with hundreds of thousands of simultaneous connections. `0` disables any limit. Deprecated and ignored from MariaDB 10.5.5.
- **Commandline:** `--innodb-adaptive-max-sleep-delay=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:**
  - `0` (>= MariaDB 10.5.5)
  - `150000` (<= MariaDB 10.5.4)
- **Range:** `0` to `1000000`
- **Introduced:** MariaDB 10.0
- **Deprecated:** MariaDB 10.5.5
- **Removed:** MariaDB 10.6.0

---

innodb_additional_mem_pool_size

- **Description:** Size in bytes of the InnoDB memory pool used for storing information about internal data structures. Defaults to 8MB, if your application has many tables and a large structure, and this is exceeded, operating system memory will be allocated and warning messages written to the error log, in which case you should increase this value. Deprecated in MariaDB 10.0 and removed in MariaDB 10.2 along with InnoDB's internal memory allocator.
- **Commandline:** `--innodb-additional-mem-pool-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `8388608`
- **Range:** `2097152` to `4294967295`
- **Deprecated:** MariaDB 10.0
- **Removed:** MariaDB 10.2.2

---

innodb_api_bk_commit_interval

- **Description:** Time in seconds between auto-commits for idle connections using the InnoDB memcached interface (not implemented in MariaDB).
- **Commandline:** `--innodb-api-bk-commit-interval=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `5`
- **Range:** `1` to `1073741824`
- **Introduced:** MariaDB 10.0
- **Removed:** MariaDB 10.2.4

---

innodb_api_disable_rowlock

- **Description:** For use with MySQL's memcached (not implemented in MariaDB)
- **Commandline:** `--innodb-api-disable-rowlock={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB 10.0
- **Removed:** MariaDB 10.2.4

---

innodb_api_enable_binlog

- **Description:** For use with MySQL's memcached (not implemented in MariaDB)
- **Commandline:** `--innodb-api-enable-binlog={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB 10.0
- **Removed:** MariaDB 10.2.4

---

innodb_api_enable_mdl

- **Description:** For use with MySQL's memcached (not implemented in MariaDB)
- **Commandline:** `--innodb-api-enable-mdl={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB 10.0
- **Removed:** MariaDB 10.2.4

---

innodb_api_trx_level

- **Description:** For use with MySQL's memcached (not implemented in MariaDB)
- **Commandline:** `--innodb-api-trx-level=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Introduced:** MariaDB 10.0
- **Removed:** MariaDB 10.2.4

---

innodb_auto_lru_dump

- **Description:** Renamed innodb_buffer_pool_restore_at_startup since XtraDB 5.5.10-20.1, which was in turn replaced

by innodb_buffer_pool_load_at_startup in MariaDB 10.0.
- **Commandline:** `--innodb-auto-lru-dump=#`
- **Removed:** XtraDB 5.5.10-20.1

---

innodb_autoextend_increment

- **Description:** Size in MB to increment an auto-extending shared tablespace file when it becomes full. If innodb_file_per_table was set to `1`, this setting does not apply to the resulting per-table tablespace files, which are automatically extended in their own way.
- **Commandline:** `--innodb-autoextend-increment=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `64` (from MariaDB 10.0) `8` (before MariaDB 10.0),
- **Range:** `1` to `1000`

---

innodb_autoinc_lock_mode

- **Description:** The lock mode that is used when generating `AUTO_INCREMENT` values for InnoDB tables.
  - Valid values are:
    - `0` is the traditional lock mode.
    - `1` is the consecutive lock mode.
    - `2` is the interleaved lock mode.
  - In order to use Galera Cluster, the lock mode needs to be set to `2`.
  - See AUTO_INCREMENT Handling in InnoDB: AUTO_INCREMENT Lock Modes for more information.
- **Commandline:** `--innodb-autoinc-lock-mode=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `1`
- **Range:** `0` to `2`

---

innodb_background_scrub_data_check_interval

- **Description:** Check if spaces needs scrubbing every innodb_background_scrub_data_check_interval seconds. See Data Scrubbing. Deprecated and ignored from MariaDB 10.5.2.
- **Commandline:** `--innodb-background-scrub-data-check-interval=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `3600`
- **Range:** `1` to `4294967295`
- **Deprecated:** MariaDB 10.5.2
- **Removed:** MariaDB 10.6.0

---

innodb_background_scrub_data_compressed

- **Description:** Enable scrubbing of compressed data by background threads (same as encryption_threads). See Data Scrubbing. Deprecated and ignored from MariaDB 10.5.2.
- **Commandline:** `--innodb-background-scrub-data-compressed={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `0`
- **Deprecated:** MariaDB 10.5.2
- **Removed:** MariaDB 10.6.0

---

innodb_background_scrub_data_interval

- **Description:** Scrub spaces that were last scrubbed longer than this number of seconds ago. See Data Scrubbing.

Deprecated and ignored from MariaDB 10.5.2.

- **Commandline:** `--innodb-background-scrub-data-interval=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `604800`
- **Range:** `1` to `4294967295`
- **Deprecated:** MariaDB 10.5.2
- **Removed:** MariaDB 10.6.0

---

innodb_background_scrub_data_uncompressed

- **Description:** Enable scrubbing of uncompressed data by background threads (same as encryption_threads). See Data Scrubbing. Deprecated and ignored from MariaDB 10.5.2.
- **Commandline:** `--innodb-background-scrub-data-uncompressed={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `0`
- **Deprecated:** MariaDB 10.5.2
- **Removed:** MariaDB 10.6.0

---

innodb_blocking_buffer_pool_restore

- **Description:** If set to `1` (`0` is default), XtraDB will wait until the least-recently used (LRU) dump is completely restored upon restart before reporting back to the server that it has successfully started up. Available with XtraDB only, not InnoDB.
- **Commandline:** `innodb-blocking-buffer-pool-restore={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Removed:** MariaDB 10.0.0

---

innodb_buf_dump_status_frequency

- **Description:** Determines how often (as a percent) the buffer pool dump status should be printed in the logs. For example, `10` means that the buffer pool dump status is printed when every 10% of the number of buffer pool pages are dumped. The default is `0` (only start and end status is printed).
- **Commandline:** `--innodb-buf-dump-status-frequency=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `100`

---

innodb_buffer_pool_chunk_size

- **Description:** Chunk size used for dynamically resizing the buffer pool. Note that changing this setting can change the size of the buffer pool. When large-pages is used this value is effectively rounded up to the next multiple of large-page-size. See Setting Innodb Buffer Pool Size Dynamically. From MariaDB 10.8.0, the variable is autosized based on the buffer pool size.
- **Commandline:** `--innodb-buffer-pool-chunk-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:**
  - `autosize (0)` , resulting in innodb_buffer_pool_size/64, if large_pages round down to multiple of largest page size, with 1MiB minimum (>= MariaDB 10.8.1)
  - `134217728` (<= MariaDB 10.8.0)
- **Range:**

    - `0`, as autosize, and then `1048576` to `18446744073709551615` (>= MariaDB 10.8)
    - `1048576` to innodb_buffer_pool_size/innodb_buffer_pool_instances (<= MariaDB 10.7)

---

innodb_buffer_pool_dump_at_shutdown

- **Description:** Whether to record pages cached in the buffer pool on server shutdown, which reduces the length of the warmup the next time the server starts. The related innodb_buffer_pool_load_at_startup specifies whether the buffer pool is automatically warmed up at startup.
- **Commandline:** `--innodb-buffer-pool-dump-at-shutdown={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:**
    - `ON`

---

innodb_buffer_pool_dump_now

- **Description:** Immediately records pages stored in the buffer pool. The related innodb_buffer_pool_load_now does the reverse, and will immediately warm up the buffer pool.
- **Commandline:** `--innodb-buffer-pool-dump-now={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB 10.0

---

innodb_buffer_pool_dump_pct

- **Description:** Dump only the hottest N% of each buffer pool.
- **Commandline:** `--innodb-buffer-pool-dump-pct={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:**
    - `25`
- **Range:** `1` to `100`

---

innodb_buffer_pool_evict

- **Description:** Evict pages from the buffer pool. If set to "uncompressed" then all uncompressed pages are evicted from the buffer pool. Variable to be used only for testing. Only exists in DEBUG builds.
- **Commandline:** `--innodb-buffer-pool-evict=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `""`
- **Valid Values:** "" or "uncompressed"

---

innodb_buffer_pool_filename

- **Description:** The file that holds the buffer pool list of page numbers set by innodb_buffer_pool_dump_at_shutdown and innodb_buffer_pool_dump_now.
- **Commandline:** `--innodb-buffer-pool-filename=file`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `ib_buffer_pool`
- **Introduced:** MariaDB 10.0

---

innodb_buffer_pool_instances

- **Description:** If innodb_buffer_pool_size is set to more than 1GB, innodb_buffer_pool_instances divides the InnoDB buffer pool into the specified number of instances. The default was 1 in MariaDB 5.5, but for large systems with buffer pools of many gigabytes, many instances could help reduce contention concurrency through MariaDB 10.2. The default is 8 in MariaDB 10 (except on Windows 32-bit, where it varies according to innodb_buffer_pool_size, or from MariaDB 10.2.2, where it is set to 1 if innodb_buffer_pool_size < 1GB). Each instance manages its own data structures and takes an equal portion of the total buffer pool size, so for example if innodb_buffer_pool_size is 4GB and innodb_buffer_pool_instances is set to 4, each instance will be 1GB. Each instance should ideally be at least 1GB in size. Starting with MariaDB 10.3, performance improvements intended to reduce the overhead of context-switching between buffer pools changed the recommended number of innodb_buffer_pool_instances to one for every 128GB of buffer pool size. Based on these changes, the variable is deprecated and ignored from MariaDB 10.5.1, where the buffer pool runs in a single instance regardless of size.
- **Commandline:** `--innodb-buffer-pool-instances=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** >= MariaDB 10.0.4: `8`, `1` (>= MariaDB 10.2.2 if innodb_buffer_pool_size < 1GB), or dependent on innodb_buffer_pool_size (Windows 32-bit)
- **Deprecated:** MariaDB 10.5.1
- **Removed:** MariaDB 10.6.0

---

innodb_buffer_pool_load_abort

- **Description:** Aborts the process of restoring buffer pool contents started by innodb_buffer_pool_load_at_startup or innodb_buffer_pool_load_now.
- **Commandline:** `--innodb-buffer-pool-load-abort={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

innodb_buffer_pool_load_at_startup

- **Description:** Specifies whether the buffer pool is automatically warmed up when the server starts by loading the pages held earlier. The related innodb_buffer_pool_dump_at_shutdown specifies whether pages are saved at shutdown. If the buffer pool is large and taking a long time to load, increasing innodb_io_capacity at startup may help.
- **Commandline:** `--innodb-buffer-pool-load-at-startup={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:**
  - `ON`

---

innodb_buffer_pool_load_now

- **Description:** Immediately warms up the buffer pool by loading the stored data pages. The related innodb_buffer_pool_dump_now does the reverse, and immediately records pages stored in the buffer pool.
- **Commandline:** `--innodb-buffer-pool-load-now={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB 10.0

---

innodb_buffer_pool_load_pages_abort

- **Description:** Number of pages during a buffer pool load to process before signaling innodb_buffer_pool_load_abort=1. Debug builds only.
- **Commandline:** `--innodb-buffer-pool-load-pages-abort=#`
- **Scope:** Global
- **Dynamic:** Yes

- **Data Type:** `numeric`
- **Default Value:** `9223372036854775807`
- **Range:** `1` to `9223372036854775807`
- **Introduced:** MariaDB 10.3

---

innodb_buffer_pool_populate

- **Description:** When set to `1` (`0` is default), XtraDB will preallocate pages in the buffer pool on starting up so that NUMA allocation decisions are made while the buffer cache is still clean. XtraDB only. This option was made ineffective in MariaDB 10.0.23. Added as a deprecated and ignored option in MariaDB 10.2.6 (which uses InnoDB as default instead of XtraDB) to allow for easier upgrades.
- **Commandline:** `innodb-buffer-pool-populate={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Deprecated:** MariaDB 10.0.23
- **Removed:** MariaDB 10.3.0

---

innodb_buffer_pool_restore_at_startup

- **Description:** Time in seconds between automatic buffer pool dumps. If set to a non-zero value, XtraDB will also perform an automatic restore of the buffer pool at startup. If set to `0`, automatic dumps are not performed, nor automatic restores on startup. Replaced by innodb_buffer_pool_load_at_startup in MariaDB 10.0.
- **Commandline:** `innodb-buffer-pool-restore-at-startup`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range - 32 bit:** `0` to `4294967295`
- **Range - 64 bit:** `0` to `18446744073709547520`
- **Removed:** MariaDB 10.0 - replaced by innodb_buffer_pool_load_at_startup

---

innodb_buffer_pool_shm_checksum

- **Description:** Used with Percona's SHM buffer pool patch in XtraDB 5.5. Was shortly deprecated and removed in XtraDB 5.6. XtraDB only.
- **Commandline:** `innodb-buffer-pool-shm-checksum={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `ON`
- **Removed:** MariaDB 10.0

---

innodb_buffer_pool_shm_key

- **Description:** Used with Percona's SHM buffer pool patch in XtraDB 5.5. Later deprecated in XtraDB 5.5, and removed in XtraDB 5.6.
- **Commandline:** `innodb-buffer-pool-shm-key={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `0`
- **Removed:** MariaDB 10.0

---

innodb_buffer_pool_size

- **Description:** InnoDB buffer pool size in bytes. The primary value to adjust on a database server with entirely/primarily InnoDB tables, can be set up to 80% of the total memory in these environments. See the InnoDB Buffer Pool for more on setting this variable, and also Setting InnoDB Buffer Pool Size Dynamically if doing so

dynamically.

- **Commandline:** `--innodb-buffer-pool-size=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `134217728` (128iMB)
- **Range:**
  - Minimum: `5242880` (5MiB) for InnoDB Page Size <= 16k otherwise `25165824` (24MiB) for InnoDB Page Size > 16k (for versions less than next line)
  - Minimium: `2MiB` InnoDB Page Size = 4k, `3MiB` InnoDB Page Size = 8k, `5MiB` InnoDB Page Size = 16k, `10MiB` InnoDB Page Size = 32k, `20MiB` InnoDB Page Size = 64k, (>= MariaDB 10.2.42, >= MariaDB 10.3.33, >= MariaDB 10.4.23, >= MariaDB 10.5.14, >= MariaDB 10.6.6, >= MariaDB 10.7.2)
  - Minimum: 1GiB for innodb_buffer_pool_instances > 1 (<= MariaDB 10.7)
  - Maximium: `9223372036854775807` (8192PB) (all versions)

---

innodb_change_buffer_dump

- **Description:** If set, causes the contents of the InnoDB change buffer to be dumped to the server error log at startup. Only available in debug builds.
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB 10.2.28, MariaDB 10.3.19, MariaDB 10.4.9

---

innodb_change_buffer_max_size

- **Description:** Maximum size of the InnoDB Change Buffer as a percentage of the total buffer pool. The default is 25%, and this can be increased up to 50% for servers with high write activity, and lowered down to 0 for servers used exclusively for reporting.
- **Commandline:** `--innodb-change-buffer-max-size=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `25`
- **Range:** `0` to `50`
- **Introduced:** MariaDB 10.0
- **Deprecated:** MariaDB 10.9.0
- **Removed:** MariaDB 11.0.0

---

innodb_change_buffering

- **Description:** Sets how InnoDB change buffering is performed. See InnoDB Change Buffering for details on the settings. Deprecated and ignored from MariaDB 10.9.0.
- **Commandline:** `--innodb-change-buffering=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `enumeration` (>= MariaDB 10.3.7), `string` (<= MariaDB 10.3.6)
- **Default Value:**
  - >= MariaDB 10.5.15, MariaDB 10.6.7, MariaDB 10.7.3, MariaDB 10.8.2: `none`
  - <= MariaDB 10.5.14, MariaDB 10.6.6, MariaDB 10.7.2, MariaDB 10.8.1: `all`
- **Valid Values:** `inserts` , `none` , `deletes` , `purges` , `changes` , `all`
- **Deprecated:** MariaDB 10.9.0
- **Removed:** MariaDB 11.0.0

---

innodb_change_buffering_debug

- **Description:** If set to `1` , an InnoDB Change Buffering debug flag is set. `1` forces all changes to the change buffer, while `2` causes a crash at merge. `0` , the default, indicates no flag is set. Only available in debug builds.
- **Commandline:** `--innodb-change-buffering-debug=#`
- **Scope:** Global

- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `2`

---

innodb_checkpoint_age_target

- **Description:** The maximum value of the checkpoint age. If set to `0`, has no effect. Removed in MariaDB 10.0/XtraDB 5.6 and replaced with InnoDB flushing method from MySQL 5.6.
- **Commandline:** `innodb-checkpoint-age-target=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` upwards
- **Removed:** MariaDB 10.0 - replaced with InnoDB flushing method from MySQL 5.6.

---

innodb_checksum_algorithm

- **Description:** Specifies how the InnoDB tablespace checksum is generated and verified.
  - `innodb`: Backwards compatible with earlier versions (<= MariaDB 5.5). Deprecated in MariaDB 10.3.29, MariaDB 10.4.19, MariaDB 10.5.10 and removed in MariaDB 10.6. If really needed, data files can still be converted with innochecksum.
  - `crc32`: A newer, faster algorithm, but incompatible with earlier versions. Tablespace blocks will be converted to the new format over time, meaning that a mix of checksums may be present.
  - `full_crc32` and `strict_full_crc32`: From MariaDB 10.4.3. Permits encryption to be supported over a SPATIAL INDEX, which `crc32` does not support. Newly-created data files will carry a flag that indicates that all pages of the file will use a full CRC-32C checksum over the entire page contents (excluding the bytes where the checksum is stored, at the very end of the page). Such files will always use that checksum, no matter what parameter `innodb_checksum_algorithm` is assigned to. Even if `innodb_checksum_algorithm` is modified later, the same checksum will continue to be used. A special flag will be set in the FSP_SPACE_FLAGS in the first data page to indicate the new format of checksum and encryption/page_compressed. ROW_FORMAT=COMPRESSED tables will only use the old format. These tables do not support new features, such as larger innodb_page_size or instant ADD/DROP COLUMN. Also cleans up the MariaDB tablespace flags - flags are reserved to store the page_compressed compression algorithm, and to store the compressed payload length, so that checksum can be computed over the compressed (and possibly encrypted) stream and can be validated without decrypting or decompressing the page. In the full_crc32 format, there no longer are separate before-encryption and after-encryption checksums for pages. The single checksum is computed on the page contents that is written to the file.See MDEV-12026 for details.
  - `none`: Writes a constant rather than calculate a checksum. Deprecated in MariaDB 10.3.29, MariaDB 10.4.19, MariaDB 10.5.10 and removed in MariaDB 10.6 as was mostly used to disable the original, slow, page checksum for benchmarking purposes.
  - `strict_crc32`, `strict_innodb` and `strict_none`: The options are the same as the regular options, but InnoDB will halt if it comes across a mix of checksum values. These are faster, as both new and old checksum values are not required, but can only be used when setting up tablespaces for the first time.
- **Commandline:** `--innodb-checksum-algorithm=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `enumeration`
- **Default Value:**
  - `full_crc32` (>= MariaDB 10.5.0)
  - `crc32` (>= MariaDB 10.2.2 to <= MariaDB 10.4)
  - `innodb` (<= MariaDB 10.2.1)
- **Valid Values:**
  - >= MariaDB 10.6.0: `crc32`, `full_crc32`, `strict_crc32`, `strict_full_crc32`
  - MariaDB 10.5, >= MariaDB 10.4.3: `innodb`, `crc32`, `full_crc32`, `none`, `strict_innodb`, `strict_crc32`, `strict_none`, `strict_full_crc32`
  - <= MariaDB 10.4.2: `innodb`, `crc32`, `none`, `strict_innodb`, `strict_crc32`, `strict_none`

---

innodb_checksums

- **Description:** By default, InnoDB performs checksum validation on all pages read from disk, which provides extra fault tolerance. You would usually want this set to `1` in production environments, although setting it to `0` can provide marginal performance improvements. Deprecated and functionality replaced by innodb_checksum_algorithm in MariaDB 10.0, and should be removed to avoid conflicts. `ON` is equivalent to `--innodb_checksum_algorithm=innodb` and `OFF` to `--innodb_checksum_algorithm=none`.
- **Commandline:** `--innodb-checksums`, `--skip-innodb-checksums`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `ON`
- **Deprecated:** MariaDB 10.0
- **Removed:** MariaDB 10.5.0

---

innodb_cleaner_lsn_age_factor

- **Description:** XtraDB has enhanced page cleaner heuristics, and with these in place, the default InnoDB adaptive flushing may be too aggressive. As a result, a new LSN age factor formula has been introduced, controlled by this variable. The default setting, `high_checkpoint`, uses the new formula, while the alternative, `legacy`, uses the original algorithm. XtraDB only. Added as a deprecated and ignored option in MariaDB 10.2.6 (which uses InnoDB as default instead of XtraDB) to allow for easier upgrades.
- **Commandline:** `--innodb-cleaner-lsn-age-factor=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `enum`
- **Default Value:**
  - `deprecated`
- **Valid Values:**
  - `deprecated`, `high_checkpoint`, `legacy`
- **Deprecated:** MariaDB 10.2.6
- **Removed:** MariaDB 10.3.0

---

innodb_cmp_per_index_enabled

- **Description:** If set to `ON` (`OFF` is default), per-index compression statistics are stored in the INFORMATION_SCHEMA.INNODB_CMP_PER_INDEX table. These are expensive to record, so this setting should only be changed with care, such as for performance tuning on development or replica servers.
- **Commandline:** `--innodb-cmp-per-index-enabled={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB 10.0

---

innodb_commit_concurrency

- **Description:** Limit to the number of transaction threads that can can commit simultaneously. 0, the default, imposes no limit. While you can change from one positive limit to another at runtime, you cannot set this variable to 0, or change it from 0, while the server is running. Deprecated and ignored from MariaDB 10.5.5.
- **Commandline:** `--innodb-commit-concurrency=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `1000`
- **Deprecated:** MariaDB 10.5.5
- **Removed:** MariaDB 10.6.0

---

innodb_compression_algorithm

- **Description:** Compression algorithm used for InnoDB page compression. The supported values are:
  - `none`: Pages are not compressed.

    - `zlib` : Pages are compressed using the bundled `zlib` compression algorithm.
    - `lz4` : Pages are compressed using the `lz4` compression algorithm.
    - `lzo` : Pages are compressed using the `lzo` compression algorithm.
    - `lzma` : Pages are compressed using the `lzma` compression algorithm.
    - `bzip2` : Pages are compressed using the `bzip2` compression algorithm.
    - `snappy` : Pages are compressed using the `snappy` algorithm.
    - On many distributions, MariaDB may not support all page compression algorithms by default. From MariaDB 10.7, libraries can be installed as a plugin. See Compression Plugins.
    - See InnoDB Page Compression: Configuring the InnoDB Page Compression Algorithm for more information.
- **Commandline:** `--innodb-compression-algorithm=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `enum`
- **Default Value:** `zlib`
- **Valid Values:** `none`, `zlib`, `lz4`, `lzo`, `lzma`, `bzip2` or `snappy`

---

innodb_compression_default

- **Description:** Whether or not InnoDB page compression is enabled by default for new tables.
    - The default value is `OFF`, which means new tables are not compressed.
    - See InnoDB Page Compression: Enabling InnoDB Page Compression by Default for more information.
- **Commandline:** `--innodb-compression-default={0|1}`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

innodb_compression_failure_threshold_pct

- **Description:** Specifies the percentage cutoff for expensive compression failures during updates to a table that uses InnoDB page compression, after which free space is added to each new compressed page, dynamically adjusted up to the level set by innodb_compression_pad_pct_max. Zero disables checking of compression efficiency and adjusting padding.
    - See InnoDB Page Compression: Configuring the Failure Threshold and Padding for more information.
- **Commandline:** `--innodb-compression-failure-threshold-pct=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `5`
- **Range:** `0` to `100`
- **Introduced:** MariaDB 10.0

---

innodb_compression_level

- **Description:** Specifies the default level of compression for tables that use InnoDB page compression.
    - Only a subset of InnoDB page compression algorithms support compression levels. If an InnoDB page compression algorithm does not support compression levels, then the compression level value is ignored.
    - The compression level can be set to any value between `1` and `9`. The default compression level is `6`. The range goes from the fastest to the most compact, which means that `1` is the fastest and `9` is the most compact.
    - See InnoDB Page Compression: Configuring the Default Compression Level for more information.
- **Commandline:** `--innodb-compression-level=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `6`
- **Range:** `1` to `9`
- **Introduced:** MariaDB 10.0

---

innodb_compression_pad_pct_max

- **Description:** The maximum percentage of reserved free space within each compressed page for tables that use InnoDB page compression. Reserved free space is used when the page's data is reorganized and might be recompressed. Only used when innodb_compression_failure_threshold_pct is not zero, and the rate of compression failures exceeds its setting.
  - See InnoDB Page Compression: Configuring the Failure Threshold and Padding for more information.
- **Commandline:** `--innodb-compression-pad-pct-max=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `50`
- **Range:** `0` to `75`
- **Introduced:** MariaDB 10.0

---

innodb_concurrency_tickets

- **Description:** Number of times a newly-entered thread can enter and leave InnoDB until it is again subject to the limitations of innodb_thread_concurrency and may possibly be queued. Deprecated and ignored from MariaDB 10.5.5.
- **Commandline:** `--innodb-concurrency-tickets=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:**
  - `0` (>= MariaDB 10.5.5)
  - `5000` (<= MariaDB 10.5.4)
- **Range:** `1` to `18446744073709551615`
- **Deprecated:** MariaDB 10.5.5
- **Removed:** MariaDB 10.6.0

---

innodb_corrupt_table_action

- **Description:** What action to perform when a corrupt table is found. XtraDB only.
  - When set to `assert`, the default, XtraDB will intentionally crash the server when it detects corrupted data in a single-table tablespace, with an assertion failure.
  - When set to `warn`, it will pass corruption as corrupt table instead of crashing, and disable all further I/O (except for deletion) on the table file.
  - If set to `salvage`, read access is permitted, but corrupted pages are ignored. innodb_file_per_table must be enabled for this option. Previously named `innodb_pass_corrupt_table`.
  - Added as a deprecated and ignored option in MariaDB 10.2.6 (which uses InnoDB as default instead of XtraDB) to allow for easier upgrades.
- **Commandline:** `innodb-corrupt-table-action=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `enumeration`
- **Default Value:**
  - `assert` (<= MariaDB 10.1)
  - `deprecated` (<= MariaDB 10.2.6)
- **Valid Values:**
  - `deprecated`, `assert`, `warn`, `salvage`
- **Deprecated:** MariaDB 10.2.6
- **Removed:** MariaDB 10.3.0

---

innodb_data_file_buffering

- **Description:** Whether to enable the file system cache for data files. Set to `OFF` by default, will be set to `ON` if innodb_flush_method is set to `fsync`, `littlesync`, `nosync`, or (Windows specific) `normal`.
- **Commandline:** `--innodb-data-file-buffering={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB 11.0.0

innodb_data_file_path

- **Description:** Individual InnoDB data files, paths and sizes. The value of innodb_data_home_dir is joined to each path specified by innodb_data_file_path to get the full directory path. If innodb_data_home_dir is an empty string, absolute paths can be specified here. A file size is specified with K for kilobytes, M for megabytes and G for gigabytes, and whether or not to autoextend the data file is also specified.
- **Commandline:** `--innodb-data-file-path=name`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `ibdata1:12M:autoextend` (from MariaDB 10.0), `ibdata1:10M:autoextend` (before MariaDB 10.0)

innodb_data_file_write_through

- **Description:** Whether writes to InnoDB data files (including the temporary tablespace) are write through. Set to `OFF` by default, will be set to `ON` if innodb_flush_method is set to `O_DSYNC`. On systems that support FUA it may make sense to enable write-through, to avoid extra system calls.
- **Commandline:** `--innodb-data-file-write-through={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB 11.0.0

innodb_data_home_dir

- **Description:** Directory path for all InnoDB data files in the shared tablespace (assuming innodb_file_per_table is not enabled). File-specific information can be added in innodb_data_file_path, as well as absolute paths if innodb_data_home_dir is set to an empty string.
- **Commandline:** `--innodb-data-home-dir=path`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `directory name`
- **Default Value:** `The MariaDB data directory`

innodb_deadlock_detect

- **Description:** By default, the InnoDB deadlock detector is enabled. If set to off, deadlock detection is disabled and MariaDB will rely on innodb_lock_wait_timeout instead. This may be more efficient in systems with high concurrency as deadlock detection can cause a bottleneck when a number of threads have to wait for the same lock.
- **Commandline:** `--innodb-deadlock-detect`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `1`

innodb_deadlock_report

- **Description:** How to report deadlocks (if innodb_deadlock_detect=ON).
  - `off` : Do not report any details of deadlocks.
  - `basic` : Report transactions and waiting locks.
  - `full` : Default. Report transactions, waiting locks and blocking locks.
- **Commandline:** `--innodb-deadlock-report=val`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `enum`
- **Default Value:** `full`
- **Valid Values:** `off`, `basic`, `full`
- **Introduced:** MariaDB 10.6.0

innodb_default_page_encryption_key

- **Description:** Encryption key used for page encryption.
  - See Data-at-Rest Encryption and InnoDB Encryption Keys for more information.
- **Commandline:** `--innodb-default-page-encryption-key=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1`
- **Range:** `1` to `255`
- **Introduced:** MariaDB 10.1.3
- **Removed:** MariaDB 10.1.4

innodb_default_encryption_key_id

- **Description:** ID of encryption key used by default to encrypt InnoDB tablespaces.
  - See Data-at-Rest Encryption and InnoDB Encryption Keys for more information.
- **Commandline:** `--innodb-default-encryption-key-id=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1`
- **Range:** `1` to `4294967295`

innodb_default_row_format

- **Description:** Specifies the default row format to be used for InnoDB tables. The compressed row format cannot be set as the default.
  - See InnoDB Row Formats Overview: Default Row Format for more information.
- **Commandline:** `--innodb-default-row-format=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `enum`
- **Default Value:** `dynamic`
- **Valid Values:** `redundant`, `compact` or `dynamic`

innodb_defragment

- **Description:** When set to `1` (the default is `0`), InnoDB defragmentation is enabled. When set to FALSE, all existing defragmentation will be paused and new defragmentation commands will fail. Paused defragmentation commands will resume when this variable is set to true again. See Defragmenting InnoDB Tablespaces.
- **Commandline:** `--innodb-defragment={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Deprecated:** MariaDB 11.0.1
- **Removed:** MariaDB 11.1.0

innodb_defragment_fill_factor

- **Description:**. Indicates how full defragmentation should fill a page. Together with innodb_defragment_fill_factor_n_recs ensures defragmentation won't pack the page too full and cause page split on the next insert on every page. The variable indicating more defragmentation gain is the one effective. See Defragmenting InnoDB Tablespaces.
- **Commandline:** `--innodb-defragment-fill-factor=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `double`
- **Default Value:** `0.9`

- **Range:** `0.7` to `1`
- **Deprecated:** MariaDB 11.0.1
- **Removed:** MariaDB 11.1.0

---

innodb_defragment_fill_factor_n_recs

- **Description:** Number of records of space that defragmentation should leave on the page. This variable, together with innodb_defragment_fill_factor, is introduced so defragmentation won't pack the page too full and cause page split on the next insert on every page. The variable indicating more defragmentation gain is the one effective. See Defragmenting InnoDB Tablespaces.
- **Commandline:** `--innodb-defragment-fill-factor-n-recs=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `20`
- **Range:** `1` to `100`
- **Deprecated:** MariaDB 11.0.1
- **Removed:** MariaDB 11.1.0

---

innodb_defragment_frequency

- **Description:** Maximum times per second for defragmenting a single index. This controls the number of times the defragmentation thread can request X_LOCK on an index. The defragmentation thread will check whether 1/defragment_frequency (s) has passed since it last worked on this index, and put the index back in the queue if not enough time has passed. The actual frequency can only be lower than this given number. See Defragmenting InnoDB Tablespaces.
- **Commandline:** `--innodb-defragment-frequency=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `integer`
- **Default Value:** `40`
- **Range:** `1` to `1000`
- **Deprecated:** MariaDB 11.0.1
- **Removed:** MariaDB 11.1.0

---

innodb_defragment_n_pages

- **Description:** Number of pages considered at once when merging multiple pages to defragment. See Defragmenting InnoDB Tablespaces.
- **Commandline:** `--innodb-defragment-n-pages=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `7`
- **Range:** `2` to `32`
- **Deprecated:** MariaDB 11.0.1
- **Removed:** MariaDB 11.1.0

---

innodb_defragment_stats_accuracy

- **Description:** Number of defragment stats changes there are before the stats are written to persistent storage. Defaults to zero, meaning disable defragment stats tracking. See Defragmenting InnoDB Tablespaces.
- **Commandline:** `--innodb-defragment-stats-accuracy=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `4294967295`
- **Deprecated:** MariaDB 11.0.1
- **Removed:** MariaDB 11.1.0

---

innodb_dict_size_limit

- **Description:** Size in bytes of a soft limit the memory used by tables in the data dictionary. Once this limit is reached, XtraDB will attempt to remove unused entries. If set to `0`, the default and standard InnoDB behavior, there is no limit to memory usage. Removed in MariaDB 10.0/XtraDB 5.6 and replaced by MySQL 5.6's new table_definition_cache implementation.
- **Commandline:** `innodb-dict-size-limit=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Default Value - 32 bit:** `2147483648`
- **Default Value - 64 bit:** `9223372036854775807`
- **Removed:** MariaDB 10.0 - replaced by MySQL 5.6's table_definition_cache implementation.

---

innodb_disable_sort_file_cache

- **Description:** If set to `1` (`0` is default), the operating system file system cache for merge-sort temporary files is disabled.
- **Commandline:** `--innodb-disable-sort-file-cache={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

innodb_disallow_writes

- **Description:** Tell InnoDB to stop any writes to disk.
- **Commandline:** None
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Removed:** MariaDB 10.3.35, MariaDB 10.4.25, MariaDB 10.5.16, MariaDB 10.6.8, MariaDB 10.7.4

---

innodb_doublewrite

- **Description:** If set to `1`, the default, to improve fault tolerance InnoDB first stores data to a doublewrite buffer before writing it to data file. Disabling will provide a marginal peformance improvement.
- **Commandline:** `--innodb-doublewrite`, `--skip-innodb-doublewrite`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `ON`

---

innodb_doublewrite_file

- **Description:** The absolute or relative path and filename to a dedicated tablespace for the doublewrite buffer. In heavy workloads, the doublewrite buffer can impact heavily on the server, and moving it to a different drive will reduce contention on random reads. Since the doublewrite buffer is mostly sequential writes, a traditional HDD is a better choice than SSD. This Percona XtraDB variable has not been ported to XtraDB 5.6.
- **Commandline:** `innodb-doublewrite-file=filename`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `filename`
- **Default Value:** `NULL`
- **Removed:** MariaDB 10.0

---

innodb_empty_free_list_algorithm

- **Description:** XtraDB 5.6.13-61 introduced an algorithm to assist with reducing mutex contention when the buffer pool free list is empty, controlled by this variable. If set to `backoff`, the default until MariaDB 10.1.24, the new algorithm will be used. If set to `legacy`, the original InnoDB algorithm will be used. XtraDB only. Added as a deprecated and ignored option in MariaDB 10.2.6 (which uses InnoDB as default instead of XtraDB) to allow for easier upgrades. See #1651657 for the reasons this was changed back to `legacy` in XtraDB 5.6.36-82.0. When upgrading from 10.0 to 10.1 (>= 10.1.24), for large buffer pools the default will remain `backoff`, while for small ones it will be changed to `legacy`.
- **Commandline:** `innodb-empty-free-list-algorithm=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `enum`
- **Default Value:**
  - `deprecated`
- **Valid Values:**
  - `deprecated`, `backoff`, `legacy`
- **Deprecated:** MariaDB 10.2.6
- **Removed:** MariaDB 10.3.0

---

innodb_enable_unsafe_group_commit

- **Description:** Unneeded after XtraDB 1.0.5. If set to `0`, the default, InnoDB will keep transactions between the transaction log and binary logs in the same order. Safer, but slower. If set to `1`, transactions can be group-committed, but there is no guarantee of the order being kept, and a small risk of the two logs getting out of sync. In write-intensive environments, can lead to a significant improvement in performance.
- **Commandline:** `--innodb-enable-unsafe-group-commit`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `1`
- **Removed:** Not needed after XtraDB 1.0.5

---

innodb_encrypt_log

- **Description:** Enables encryption of the InnoDB redo log. This also enables encryption of some temporary files created internally by InnoDB, such as those used for merge sorts and row logs.
  - See Data-at-Rest Encryption and InnoDB / XtraDB Enabling Encryption: Enabling Encryption for Redo Log for more information.
- **Commandline:** `--innodb-encrypt-log`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

innodb_encrypt_tables

- **Description:** Enables automatic encryption of all InnoDB tablespaces.
  - `OFF` - Disables table encryption for all new and existing tables that have the `ENCRYPTED` table option set to `DEFAULT`.
  - `ON` - Enables table encryption for all new and existing tables that have the `ENCRYPTED` table option set to `DEFAULT`, but allows unencrypted tables to be created.
  - `FORCE` - Enables table encryption for all new and existing tables that have the `ENCRYPTED` table option set to `DEFAULT`, and doesn't allow unencrypted tables to be created (CREATE TABLE ... ENCRYPTED=NO will fail).
  - See Data-at-Rest Encryption and InnoDB / XtraDB Enabling Encryption: Enabling Encryption for Automatically Encrypted Tablespaces for more information.
- **Commandline:** `--innodb-encrypt-tables={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Valid Values:** `ON`, `OFF`, `FORCE`

---

innodb_encrypt_temporary_tables

- **Description:** Enables automatic encryption of the InnoDB temporary tablespace.
  - See Data-at-Rest Encryption and InnoDB Enabling Encryption: Enabling Encryption for Temporary Tablespaces for more information.
- **Commandline:** `--innodb-encrypt-temporary-tables={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Valid Values:** `ON`, `OFF`
- **Introduced:** MariaDB 10.2.26, MariaDB 10.3.17, MariaDB 10.4.7

---

innodb_encryption_rotate_key_age

- **Description:** Re-encrypt in background any page having a key older than this number of key versions. When setting up encryption, this variable must be set to a non-zero value. Otherwise, when you enable encryption through `innodb_encrypt_tables` MariaDB won't be able to automatically encrypt any unencrypted tables.
  - See Data-at-Rest Encryption and InnoDB Encryption Keys: Key Rotation for more information.
- **Commandline:** `--innodb-encryption-rotate-key-age=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1`
- **Range:** `0` to `4294967295`

---

innodb_encryption_rotation_iops

- **Description:** Use this many iops for background key rotation operations performed by the background encryption threads.
  - See Data-at-Rest Encryption and InnoDB Encryption Keys: Key Rotation for more information.
- **Commandline:** `--innodb-encryption-rotation_iops=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `100`
- **Range:** `0` to `4294967295`

---

innodb_encryption_threads

- **Description:** Number of background encryption threads threads performing background key rotation and scrubbing. When setting up encryption, this variable must be set to a non-zero value. Otherwise, when you enable encryption through innodb_encrypt_tables MariaDB won't be able to automatically encrypt any unencrypted tables. Recommended never be set higher than 255.
  - See Data-at-Rest Encryption and InnoDB Background Encryption Threads for more information.
- **Commandline:** `--innodb-encryption-threads=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:**
  - `0` to `4294967295` (<= MariaDB 10.1.45, MariaDB 10.2.32, MariaDB 10.3.23, MariaDB 10.4.13, MariaDB 10.5.3)
  - `0` to `255` (>= MariaDB 10.1.46, MariaDB 10.2.33, MariaDB 10.3.24, MariaDB 10.4.14, MariaDB 10.5.4)

---

innodb_extra_rsegments

- **Description:** Removed in XtraDB 5.5 and replaced by innodb_rollback_segments. Usually there is one rollback segment protected by single mutex, a source of contention in high write environments. This option specifies a number of extra user rollback segments. Changing the default will make the data readable by XtraDB only, and is incompatible

with InnoDB. After modifying, the server must be slow-shutdown. If there is existing data, it must be dumped before changing, and re-imported after the change has taken effect.

- **Commandline:** `--innodb-extra-rsegments=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `126`
- **Removed:** XtraDB 5.5 - replaced by innodb_rollback_segments

---

innodb_extra_undoslots

- **Description:** Usually, InnoDB has 1024 undo slots in its rollback segment, so 1024 transactions can run in parallel. New transactions will fail if all slots are used. Setting this variable to `1` expands the available undo slots to 4072. Not recommended unless you get the `Warning: cannot find a free slot for an undo log` error in the error log, as it makes data files unusable for ibbackup, or MariaDB servers not run with this option. See also undo log.
- **Commandline:** `--innodb-extra-undoslots={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Removed:** XtraDB 5.5

---

innodb_fake_changes

- **Description:** From MariaDB 5.5 until MariaDB 10.1, XtraDB-only option that enables the fake changes feature. In replication, setting up or restarting a replica can cause a replication reads to perform more slowly, as MariaDB is single-threaded and needs to read the data before it can execute the queries. This can be speeded up by prefetching threads to warm the server, replaying the statements and then rolling back at commit. This however has an overhead from locking rows only then to undo changes at rollback. Fake changes attempts to reduce this overhead by reading the rows for INSERT, UPDATE and DELETE statements but not updating them. The rollback is then very fast with little or nothing to do. Added as a deprecated and ignored option in MariaDB 10.2.6 (which uses InnoDB as default instead of XtraDB) to allow for easier upgrades. Not present in MariaDB 10.3 and beyond.
- **Commandline:** `--innodb-fake-changes={0|1}`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Deprecated:** MariaDB 10.2.6
- **Removed:** MariaDB 10.3.0

---

innodb_fast_checksum

- **Description:** Implements a more CPU efficient XtraDB checksum algorithm, useful for write-heavy loads with high I/O. If set to `1` on a server with tables that have been created with it set to `0`, reads will be slower, so tables should be recreated (dumped and reloaded). XtraDB will fail to start if set to `0` and there are tables created while set to `1`. Replaced with innodb_checksum_algorithm in MariaDB 10.0/XtraDB 5.6.
- **Commandline:** `--innodb-fast-checksum={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Removed:** MariaDB 10.0/XtraDB 5.6 - replaced with innodb_checksum_algorithm

---

innodb_fast_shutdown

- **Description:** The shutdown mode.
  - `0` - InnoDB performs a slow shutdown, including full purge (before MariaDB 10.3.6, not always, due to MDEV-13603) and change buffer merge. Can be very slow, even taking hours in extreme cases.
  - `1` - the default, InnoDB performs a fast shutdown, not performing a full purge or an insert buffer merge.
  - `2`, the InnoDB redo log is flushed and a cold shutdown takes place, similar to a crash. The resulting startup

then performs crash recovery. Extremely fast, in cases of emergency, but risks corruption. Not suitable for upgrades between major versions!
    - `3` (from MariaDB 10.3.6) - active transactions will not be rolled back, but all changed pages will be written to data files. The active transactions will be rolled back by a background thread on a subsequent startup. The fastest option that will not involve InnoDB redo log apply on subsequent startup. See MDEV-15832.
- **Commandline:** `--innodb-fast-shutdown[=#]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1`
- **Range:** `0` to `3` (>= MariaDB 10.3.6), `0` to `2` (<= MariaDB 10.3.5)

---

innodb_fatal_semaphore_wait_threshold

- **Description:** In MariaDB, the fatal semaphore timeout is configurable. This variable sets the maximum number of seconds for semaphores to time out in InnoDB.
- **Commandline:** `--innodb-fatal-semaphore-wait-threshold=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `600`
- **Range:** `1` to `4294967295`

---

innodb_file_format

- **Description:** File format for new InnoDB tables. Can either be `Antelope`, the default and the original format, or `Barracuda`, which supports compression. Note that this value is also used when a table is re-created with an ALTER TABLE which requires a table copy. See XtraDB/InnoDB File Format for more on the file formats. Removed in 10.3.1 and restored as a deprecated and unused variable in 10.4.3 for compatibility purposes.
- **Commandline:** `--innodb-file-format=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:**
    - `Barracuda`
- **Valid Values:** `Antelope`, `Barracuda`
- **Deprecated:** MariaDB 10.2
- **Removed:** MariaDB 10.3.1
- **Re-introduced:** MariaDB 10.4.3 (for compatibility purposes)
- **Removed:** MariaDB 10.6.0

---

innodb_file_format_check

- **Description:** If set to `1`, the default, InnoDB checks the shared tablespace file format tag. If this is higher than the current version supported by XtraDB/InnoDB (for example Barracuda when only Antelope is supported), XtraDB/InnoDB will will not start. If it the value is not higher, XtraDB/InnoDB starts correctly and the innodb_file_format_max value is set to this value. If innodb_file_format_check is set to `0`, no checking is performed. See XtraDB/InnoDB File Format for more on the file formats.
- **Commandline:** `--innodb-file-format-check={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `ON`
- **Deprecated:** MariaDB 10.2
- **Removed:** MariaDB 10.3.1

---

innodb_file_format_max

- **Description:** The highest XtraDB/InnoDB file format. This is set to the value of the file format tag in the shared tablespace on startup (see innodb_file_format_check). If the server later creates a higher table format, innodb_file_format_max is set to that value. See XtraDB/InnoDB File Format for more on the file formats.

- **Commandline:** `--innodb-file-format-max=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `Antelope`
- **Valid Values:** `Antelope`, `Barracuda`
- **Deprecated:** MariaDB 10.2
- **Removed:** MariaDB 10.3.1

---

innodb_file_per_table

- **Description:** If set to `ON`, then new InnoDB tables are created with their own InnoDB file-per-table tablespaces. If set to `OFF`, then new tables are created in the InnoDB system tablespace instead. Page compression is only available with file-per-table tablespaces. Note that this value is also used when a table is re-created with an ALTER TABLE which requires a table copy. Deprecated in MariaDB 11.0 as there's no benefit to setting to `OFF`, the original InnoDB default.
- **Commandline:** `--innodb-file-per-table`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `ON`
- **Deprecated:** MariaDB 11.0.1

---

innodb_fill_factor

- **Description:** Percentage of B-tree page filled during bulk insert (sorted index build). Used as a hint rather than an absolute value. Setting to `70`, for example, reserves 30% of the space on each B-tree page for the index to grow in future.
- **Commandline:** `--innodb-fill-factor=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value**: `100`
- **Range:** `10` to `100`

---

innodb_flush_log_at_timeout

- **Description:** Interval in seconds to write and flush the InnoDB redo log. Before MariaDB 10, this was fixed at one second, which is still the default, but this can now be changed. It's usually increased to reduce flushing and avoid impacting performance of binary log group commit.
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1`
- **Range:** `0` to `2700`

---

innodb_flush_log_at_trx_commit

- **Description:** Set to `1`, along with sync_binlog=1 for the greatest level of fault tolerance. The value of innodb_use_global_flush_log_at_trx_commit determines whether this variable can be reset with a SET statement or not.
  - `1` The default, the log buffer is written to the InnoDB redo log file and a flush to disk performed after each transaction. This is required for full ACID compliance.
  - `0` Nothing is done on commit; rather the log buffer is written and flushed to the InnoDB redo log once a second. This gives better performance, but a server crash can erase the last second of transactions.
  - `2` The log buffer is written to the InnoDB redo log after each commit, but flushing takes place every innodb_flush_log_at_timeout seconds (by default once a second). Performance is slightly better, but a OS or power outage can cause the last second's transactions to be lost.
  - `3` Emulates MariaDB 5.5 group commit (3 syncs per group commit). See Binlog group commit and innodb_flush_log_at_trx_commit. This option has not been working correctly since 10.2 and may be removed in future, see https://github.com/MariaDB/server/pull/1873

- **Commandline:** `--innodb-flush-log-at-trx-commit[=#]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `enumeration`
- **Default Value:** `1`
- **Valid Values:** `0`, `1`, `2` or `3`

---

innodb_flush_method

- **Description:** InnoDB flushing method. Windows always uses async_unbuffered and this variable then has no effect. On Unix, before MariaDB 10.6.0, by default fsync() is used to flush data and logs. Adjusting this variable can give performance improvements, but behavior differs widely on different filesystems, and changing from the default has caused problems in some situations, so test and benchmark carefully before adjusting. In MariaDB, Windows recognises and correctly handles the Unix methods, but if none are specified it uses own default - unbuffered write (analog of O_DIRECT) + syncs (e.g FileFlushBuffers()) for all files.
  - `O_DSYNC` - O_DSYNC is used to open and flush logs, and fsync() to flush the data files.
  - `O_DIRECT` - O_DIRECT or directio(), is used to open data files, and fsync() to flush data and logs. Default on Unix from MariaDB 10.6.0.
  - `fsync` - Default on Unix until MariaDB 10.5. Can be specified directly, but if the variable is unset on Unix, fsync() will be used by default.
  - `O_DIRECT_NO_FSYNC` - introduced in MariaDB 10.0. Uses O_DIRECT during flushing I/O, but skips fsync() afterwards. Not suitable for XFS filesystems. Generally not recommended over O_DIRECT, as does not get the benefit of innodb_use_native_aio=ON.
  - `ALL_O_DIRECT` - introduced in MariaDB 5.5 and available with XtraDB only. Uses O_DIRECT for opening both data and logs and fsync() to flush data but not logs. Use with large InnoDB files only, otherwise may cause a performance degradation. Set innodb_log_block_size to 4096 on ext4 filesystems. This is the default log block size on ext4 and will avoid unaligned AIO/DIO warnings.
  - `unbuffered` - Windows-only default
  - `async_unbuffered` - Windows-only, alias for `unbuffered`
  - `normal` - Windows-only, alias for `fsync`
  - `littlesync` - for internal testing only
  - `nosync` - for internal testing only
- Deprecated in MariaDB 11.0 and replaced by four boolean dynamic variables that can be changed while the server is running: innodb_log_file_buffering (disable O_DIRECT, added by MDEV-28766 in 10.8.4, 10.9.2), innodb_data_file_buffering (disable O_DIRECT on data files), innodb_log_file_write_through (enable O_DSYNC on the log), innodb_data_file_write_through (enable O_DSYNC on persistent data files)

  From MariaDB 11.0, if set to one of the following values, then the values of the four boolean flags will be set as follows:
  - `O_DSYNC`: innodb_log_file_write_through=ON, innodb_data_file_write_through=ON, innodb_data_file_buffering=OFF, and (if supported) innodb_log_file_buffering=OFF.
  - `fsync`, `littlesync`, `nosync`, or (Microsoft Windows specific) `normal`: innodb_log_file_write_through=OFF, innodb_data_file_write_through=OFF, and innodb_data_file_buffering=ON.
- **Commandline:** `--innodb-flush-method=name`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `enumeration` (>= MariaDB 10.3.7), `string` (<= MariaDB 10.3.6)
- **Default Value:**
  - `O_DIRECT` (Unix, >= MariaDB 10.6.0)
  - `fsync` (Unix, >= MariaDB 10.3.7, <= MariaDB 10.5)
  - Not set (<= MariaDB 10.3.6)
- **Valid Values:**
  - Unix: `fsync`, `O_DSYNC`, `littlesync`, `nosync`. `O_DIRECT`, `O_DIRECT_NO_FSYNC`
  - Windows: `unbuffered`, `async_unbuffered`, `normal`
- **Deprecated:** MariaDB 11.0

---

innodb_flush_neighbor_pages

- **Description:** Determines whether, when dirty pages are flushed to the data file, neighboring pages in the data file are flushed at the same time. If set to `none`, the feature is disabled. If set to `area`, the default, the standard InnoDB behavior is used. For each page to be flushed, dirty neighboring pages are flushed too. If there's little head seek delay, such as SSD or large enough write buffer, one of the other two options may be more efficient. If set to `cont`,

for each page to be flushed, neighboring contiguous blocks are flushed at the same time. Being contiguous, a sequential I/O is used, unlike the random I/O used in `area`. Replaced by innodb_flush_neighbors in MariaDB 10.0/XtraDB 5.6.

- **Commandline:** `innodb-flush-neighbor-pages=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `enumeration`
- **Default Value:** `area`
- **Valid Values:** `none` or `0`, `area` or `1`, `cont` or `2`
- **Removed:** MariaDB 10.0/XtraDB 5.6 - replaced by innodb_flush_neighbors

---

innodb_flush_neighbors

- **Description:** Determines whether flushing a page from the buffer pool will flush other dirty pages in the same group of pages (extent). In high write environments, if flushing is not aggressive enough, it can fall behind resulting in higher memory usage, or if flushing is too aggressive, cause excess I/O activity. SSD devices, with low seek times, would be less likely to require dirty neighbor flushing to be set. Since MariaDB 10.4.4 an attempt is made under Windows and Linux to determine SSD status which was exposed in information_schema.innodb_tablespaces_scrubbing_table. This variable is ignored for table spaces that are detected as stored on SSD (and the `0` behavior applies).
  - `1`: The default, flushes contiguous dirty pages in the same extent from the buffer pool.
  - `0`: No other dirty pages are flushed.
  - `2`: Flushes dirty pages in the same extent from the buffer pool.
- **Commandline:** `--innodb-flush-neighbors=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `enumeration`
- **Default Value:** `1`
- **Valid Values:** `0`, `1`, `2`

---

innodb_flush_sync

- **Description:** If set to `ON`, the default, the innodb_io_capacity setting is ignored for I/O bursts occuring at checkpoints.
- **Commandline:** `--innodb-flush-sync={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value**: `ON`

---

innodb_flushing_avg_loops

- **Description:** Determines how quickly adaptive flushing will respond to changing workloads. The value is the number of iterations that a previously calculated flushing state snapshot is kept. Increasing the value smooths and slows the rate that the flushing operations change, while decreasing it causes flushing activity to spike quickly in response to workload changes.
- **Commandline:** `--innodb-flushing-avg-loops=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `30`
- **Range:** `1` to `1000`

---

innodb_force_load_corrupted

- **Description:** Set to `0` by default, if set to `1`, InnoDB will be permitted to load tables marked as corrupt. Only use this to recover data you can't recover any other way, or in troubleshooting. Always restore to `0` when the returning to regular use. Given that MDEV-11412 in MariaDB 10.5.4 aims to allow any metadata for a missing or corrupted table to be dropped, and given that MDEV-17567 and MDEV-25506 and related tasks made DDL operations crash-safe, the parameter no longer serves any purpose and was removed in MariaDB 10.6.6.
- **Commandline:** `--innodb-force-load-corrupted`
- **Scope:** Global

- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Removed:** MariaDB 10.6.6

---

innodb_force_primary_key

- **Description:** If set to `1` (`0` is default) CREATE TABLEs without a primary or unique key where all keyparts are NOT NULL will not be accepted, and will return an error.
- **Commandline:** `--innodb-force-primary-key`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

innodb_force_recovery

- **Description:** InnoDB crash recovery mode. `0` is the default. The other modes are for recovery purposes only, and no data can be changed while another mode is active. Some queries relying on indexes are also blocked. See InnoDB Recovery Modes for more on mode specifics.
- **Commandline:** `--innodb-force-recovery=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `enumeration`
- **Default Value:** `0`
- **Range:** `0` to `6`

---

innodb_foreground_preflush

- **Description:** Before XtraDB 5.6.13-61.0, if the checkpoint age is in the sync preflush zone while a thread is writing to the XtraDB redo log, it will try to advance the checkpoint by issuing a flush list flush batch if this is not already being done. XtraDB has enhanced page cleaner tuning, and may already be performing furious flushing, resulting in the flush simply adding unneeded mutex pressure. Instead, the thread now waits for the flushes to finish, and then has two options, controlled by this variable. XtraDB only. Added as a deprecated and ignored option in MariaDB 10.2.6 (which uses InnoDB as default instead of XtraDB) to allow for easier upgrades.
  - `exponential_backoff` - thread sleeps while it waits for the flush list flush to occur. The sleep time randomly progressively increases, periodically reset to avoid runaway sleeps.
  - `sync_preflush` - thread issues a flush list batch, and waits for it to complete. This is the same as is used when the page cleaner thread is not running.
- **Commandline:** `innodb-foreground-preflush=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `enum`
- **Default Value:**
  - `deprecated`
- **Valid Values:**
  - `deprecated`, `exponential_backoff`, `sync_preflush`
- **Deprecated:** MariaDB 10.2.6
- **Removed:** MariaDB 10.3.0

---

innodb_ft_aux_table

- **Description:** Diagnostic variable intended only to be set at runtime. It specifies the qualified name (for example `test/ft_innodb`) of an InnoDB table that has a FULLTEXT index, and after being set the INFORMATION_SCHEMA tables INNODB_FT_INDEX_TABLE, INNODB_FT_INDEX_CACHE, INNODB_FT_CONFIG, INNODB_FT_DELETED, and INNODB_FT_BEING_DELETED will contain search index information for the specified table.
- **Commandline:** `--innodb-ft-aux-table=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`

innodb_ft_cache_size

- **Description:** Cache size available for a parsed document while creating an InnoDB FULLTEXT index.
- **Commandline:** `--innodb-ft-cache-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `8000000`

innodb_ft_enable_diag_print

- **Description:** If set to `1`, additional full-text search diagnostic output is enabled.
- **Commandline:** `--innodb-ft-enable-diag-print={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

innodb_ft_enable_stopword

- **Description:** If set to `1`, the default, a set of stopwords is associated with an InnoDB FULLTEXT index when it is created. The stopword list comes from the table set by the session variable innodb_ft_user_stopword_table, if set, otherwise the global variable innodb_ft_server_stopword_table, if that is set, or the built-in list if neither variable is set.
- **Commandline:** `--innodb-ft-enable-stopword={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `ON`

innodb_ft_max_token_size

- **Description:** Maximum length of words stored in an InnoDB FULLTEXT index. A larger limit will increase the size of the index, slowing down queries, but permit longer words to be searched for. In most normal situations, longer words are unlikely search terms.
- **Commandline:** `--innodb-ft-max-token-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `84`
- **Range:** `10` to `84`

innodb_ft_min_token_size

- **Description:** Minimum length of words stored in an InnoDB FULLTEXT index. A smaller limit will increase the size of the index, slowing down queries, but permit shorter words to be searched for. For data stored in a Chinese, Japanese or Korean character set, a value of 1 should be specified to preserve functionality.
- **Commandline:** `--innodb-ft-min-token-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `3`
- **Range:** `0` to `16`

innodb_ft_num_word_optimize

- **Description:** Number of words processed during each OPTIMIZE TABLE on an InnoDB FULLTEXT index. To ensure all changes are incorporated, multiple OPTIMIZE TABLE statements could be run in case of a substantial change to the index.
- **Commandline:** `--innodb-ft-num-word-optimize=#`

- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `2000`
- **Range:** `1000` to `10000`

---

innodb_ft_result_cache_limit

- **Description:** Limit in bytes of the InnoDB FULLTEXT index query result cache per fulltext query. The latter stages of the full-text search are handled in memory, and limiting this prevents excess memory usage. If the limit is exceeded, the query returns an error.
- **Commandline:** `--innodb-ft-result-cache-limit=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `2000000000`
- **Range:** `1000000` to `18446744073709551615`

---

innodb_ft_server_stopword_table

- **Description:** Table name containing a list of stopwords to ignore when creating an InnoDB FULLTEXT index, in the format db_name/table_name. The specified table must exist before this option is set, and must be an InnoDB table with a single column, a VARCHAR named VALUE. See also innodb_ft_enable_stopword.
- **Commandline:** `--innodb-ft-server-stopword-table=db_name/table_name`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** Empty

---

innodb_ft_sort_pll_degree

- **Description:** Number of parallel threads used when building an InnoDB FULLTEXT index. See also innodb_sort_buffer_size.
- **Commandline:** `--innodb-ft-sort-pll-degree=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `2`
- **Range:** `1` to `32`

---

innodb_ft_total_cache_size

- **Description:**Total memory allocated for the cache for all InnoDB FULLTEXT index tables. A force sync is triggered if this limit is exceeded.
- **Commandline:** `--innodb-ft-total-cache-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `640000000`
- **Range:** `32000000` to `1600000000`
- **Introduced:** MariaDB 10.0.9

---

innodb_ft_user_stopword_table

- **Description:** Table name containing a list of stopwords to ignore when creating an InnoDB FULLTEXT index, in the format db_name/table_name. The specified table must exist before this option is set, and must be an InnoDB table with a single column, a VARCHAR named VALUE. See also innodb_ft_enable_stopword.
- **Commandline:** `--innodb-ft-user-stopword-table=db_name/table_name`
- **Scope:** Session
- **Dynamic:** Yes

- **Data Type:** `string`
- **Default Value:** Empty

---

innodb_ibuf_accel_rate

- **Description:** Allows the insert buffer activity to be adjusted. The following formula is used: [real activity] = [default activity] * (innodb_io_capacity/100) * (innodb_ibuf_accel_rate/100). As `innodb_ibuf_accel_rate` is increased from its default value of `100`, the lowest setting, insert buffer activity is increased. See also innodb_io_capacity. This Percona XtraDB variable has not been ported to XtraDB 5.6.
- **Commandline:** `innodb-ibuf-accel-rate=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `100`
- **Range:** `100` to `999999999`
- **Removed:** MariaDB 10.0

---

innodb_ibuf_active_contract

- **Description:** Specifies whether the insert buffer can be processed before it's full. If set to `0`, the standard InnoDB method is used, and the buffer is not processed until it's full. If set to `1`, the default, the insert buffer can be processed before it is full. This Percona XtraDB variable has not been ported to XtraDB 5.6.
- **Commandline:** `innodb-ibuf-active-contract=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1`
- **Range:** `0` to `1`
- **Removed:** MariaDB 10.0

---

innodb_ibuf_max_size

- **Description:** Maximum size in bytes of the insert buffer. Defaults to half the size of the buffer pool so you may want to reduce if you have a very large buffer pool. If set to `0`, the insert buffer is disabled, which will cause all secondary index updates to be performed synchronously, usually at a cost to performance. This Percona XtraDB variable has not been ported to XtraDB 5.6.
- **Commandline:** `innodb-ibuf-max-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** 1/2 the size of the InnoDB buffer pool
- **Range:** `0` to 1/2 the size of the InnoDB buffer pool
- **Removed:** MariaDB 10.0

---

innodb_idle_flush_pct

- **Description:** Up to what percentage of dirty pages should be flushed when innodb finds it has spare resources to do so. Has had no effect since merging InnoDB 5.7 from mysql-5.7.9 (MariaDB 10.2.2). Deprecated in MariaDB 10.2.37, MariaDB 10.3.28, MariaDB 10.4.18 and removed in MariaDB 10.5.9.
- **Commandline:** `--innodb-idle-flush-pct=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `100`
- **Range:** `0` to `100`
- **Deprecated:** MariaDB 10.2.37, MariaDB 10.3.28, MariaDB 10.4.18
- **Removed:** MariaDB 10.5.9

---

innodb_immediate_scrub_data_uncompressed

- **Description:** Enable scrubbing of data. See Data Scrubbing.
- **Commandline:** `--innodb-immediate-scrub-data-uncompressed={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

innodb_import_table_from_xtrabackup

- **Description:** If set to `1`, permits importing of .ibd files exported with the XtraBackup --export option. Previously named `innodb_expand_import`. Removed in MariaDB 10.0/XtraDB 5.6 and replaced with MySQL 5.6's transportable tablespaces.
- **Commandline:** `innodb-import-table-from-xtrabackup=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `1`
- **Removed:** MariaDB 10.0

---

innodb_instant_alter_column_allowed

- **Description:**
  - If a table is altered using ALGORITHM=INSTANT, it can force the table to use a non-canonical format: A hidden metadata record at the start of the clustered index is used to store each column's DEFAULT value. This makes it possible to add new columns that have default values without rebuilding the table. Starting with MariaDB 10.4, a BLOB in the hidden metadata record is used to store column mappings. This makes it possible to drop or reorder columns without rebuilding the table. This also makes it possible to add columns to any position or drop columns from any position in the table without rebuilding the table. If a column is dropped without rebuilding the table, old records will contain garbage in that column's former position, and new records will be written with NULL values, empty strings, or dummy values.
  - This is generally not a problem. However, there may be cases where you want to avoid putting a table into this format. For example, to ensure that future UPDATE operations after an ADD COLUMN will be performed in-place, to reduce write amplification. (Instantly added columns are essentially always variable-length.) Also avoid bugs similar to MDEV-19916, or to be able to export tables to older versions of the server.
  - This variable has been introduced as a result, with the following values:
  - `never` (0): Do not allow instant add/drop/reorder, to maintain format compatibility with MariaDB 10.x and MySQL 5.x. If the table (or partition) is not in the canonical format, then any ALTER TABLE (even one that does not involve instant column operations) will force a table rebuild.
  - `add_last` (1, default in 10.3): Store a hidden metadata record that allows columns to be appended to the table instantly (MDEV-11369). In 10.4 or later, if the table (or partition) is not in this format, then any ALTER TABLE (even one that does not involve column changes) will force a table rebuild.
  - `add_drop_reorder` (2, default): From MariaDB 10.4 only. Like 'add_last', but allow the metadata record to store a column map, to support instant add/drop/reorder of columns.
- **Commandline:** `--innodb-instant-alter-column-allowed=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `enum`
- **Valid Values:**
  - <= MariaDB 10.3: `never`, `add_last`
  - >= MariaDB 10.4: `never`, `add_last`, `add_drop_reorder`
- **Default Value:**
  - <= MariaDB 10.3: `add_last`
  - >= MariaDB 10.4: `add_drop_reorder`
- **Introduced:** MariaDB 10.3.23, MariaDB 10.4.13, MariaDB 10.5.3

---

innodb_instrument_semaphores

- **Description:** Enable semaphore request instrumentation. This could have some effect on performance but allows better information on long semaphore wait problems.
- **Commandline:** `--innodb-instrument-semaphores={0|1}`
- **Scope:** Global

- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Deprecated:** MariaDB 10.2.5 (treated as if `OFF`)
- **Removed:** MariaDB 10.3.0

---

innodb_io_capacity

- **Description:** Limit on I/O activity for InnoDB background tasks, including merging data from the insert buffer and flushing pages. Should be set to around the number of I/O operations per second that system can handle, based on the type of drive/s being used. You can also set it higher when the server starts to help with the extra workload at that time, and then reduce for normal use. Ideally, opt for a lower setting, as at higher value data is removed from the buffers too quickly, reducing the effectiveness of caching. See also innodb_flush_sync.
  - See InnoDB Page Flushing: Configuring the InnoDB I/O Capacity for more information.
- **Commandline:** `--innodb-io-capacity=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `200`
- **Range:** `100` to `18446744073709551615` (2^{64}-1)

---

innodb_io_capacity_max

- **Description:** Upper limit to which InnoDB can extend innodb_io_capacity in case of emergency. See InnoDB Page Flushing: Configuring the InnoDB I/O Capacity for more information.
- **Commandline:** `--innodb-io-capacity-max=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `2000` or twice innodb_io_capacity, whichever is higher.
- **Range :** `100` to `18446744073709551615` (2^{64}-1)

---

innodb_kill_idle_transaction

- **Description:** Time in seconds before killing an idle XtraDB transaction. If set to `0` (the default), the feature is disabled. Used to prevent accidental user locks. XtraDB only. Added as a deprecated and ignored option in MariaDB 10.2.6 (which uses InnoDB as default instead of XtraDB) to allow for easier upgrades.
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `9223372036854775807`
- **Deprecated:** MariaDB 10.2.6
- **Removed:** MariaDB 10.3.0

---

innodb_large_prefix

- **Description:** If set to `1`, tables that use specific row formats are permitted to have index key prefixes up to 3072 bytes (for 16k pages, smaller otherwise). If not set, the limit is 767 bytes.
  - This applies to the `DYNAMIC` and `COMPRESSED` row formats.
  - Removed in 10.3.1 and restored as a deprecated and unused variable in 10.4.3 for compatibility purposes.
- **Commandline:** `--innodb-large-prefix`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:**
  - `ON`
- **Deprecated:** MariaDB 10.2
- **Removed:** MariaDB 10.3.1
- **Re-introduced:** MariaDB 10.4.3 (for compatibility purposes)
- **Removed:** MariaDB 10.6.0

innodb_lazy_drop_table

- **Description:** Deprecated and removed in XtraDB 5.6. DROP TABLE processing can take a long time when innodb_file_per_table is set to 1 and there's a large buffer pool. If `innodb_lazy_drop_table` is set to `1` (`0` is default), XtraDB attempts to optimize DROP TABLE processing by deferring the dropping of related pages from the buffer pool until there is time, only initially marking them.
- **Commandline:** `innodb-lazy-drop-table={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `0`
- **Deprecated:** XtraDB 5.5.30-30.2
- **Removed:** MariaDB 10.0.0

innodb_lock_schedule_algorithm

- **Description:** Removed in MariaDB 10.6.0 due to problems with the VATS implementation (MDEV-16664). Specifies the algorithm that InnoDB uses to decide which of the waiting transactions should be granted the lock once it has been released. The possible values are: `FCFS` (First-Come-First-Served) where locks are granted in the order they appear in the lock queue and `VATS` (Variance-Aware-Transaction-Scheduling) where locks are granted based on the Eldest-Transaction-First heuristic. Note that `VATS` should not be used with Galera, and InnoDB will refuse to start if `VATS` is used with Galera. It is also not recommended to set to `VATS` even in the general case (MDEV-16664). From MariaDB 10.2.12, the value was changed to `FCFS` and a warning produced when using Galera.
- **Commandline:** `--innodb-lock-schedule-algorithm=#`
- **Scope:** Global
- **Dynamic:** No (>= MariaDB 10.2.12, MariaDB 10.1.30), Yes (<= MariaDB 10.2.11, MariaDB 10.1.29)
- **Data Type:** `enum`
- **Valid Values:** `FCFS`, `VATS`
- **Default Value:** `FCFS` (MariaDB 10.3.9, MariaDB 10.2.17), `VATS` (MariaDB 10.2.3), `FCFS` (MariaDB 10.1)
- **Deprecated:** MariaDB 10.5.7, MariaDB 10.4.16, MariaDB 10.3.26, MariaDB 10.2.35
- **Removed:** MariaDB 10.6.0

innodb_lock_wait_timeout

- **Description:** Time in seconds that an InnoDB transaction waits for an InnoDB record lock (or table lock) before giving up with the error `ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction`. When this occurs, the statement (not transaction) is rolled back. The whole transaction can be rolled back if the innodb_rollback_on_timeout option is used. Increase this for data warehousing applications or where other long-running operations are common, or decrease for OLTP and other highly interactive applications. This setting does not apply to deadlocks, which InnoDB detects immediately, rolling back a deadlocked transaction. `0` means no wait. See WAIT and NOWAIT. Setting to `100000000` or more (from MariaDB 10.6.3, `100000000` is the maximum) means the timeout is infinite.
- **Commandline:** `--innodb-lock-wait-timeout=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `INT UNSIGNED` (>= MariaDB 10.6.3), `BIGINT UNSIGNED` (<= MariaDB 10.6.2)
- **Default Value:** `50`
- **Range:**
  - `0` to `100000000` (>= MariaDB 10.6.3)
  - `0` to `1073741824` (>= MariaDB 10.3 to <= MariaDB 10.6.2)

innodb_locking_fake_changes

- **Description:** From MariaDB 5.5 to MariaDB 10.1, XtraDB-only option that if set to `OFF`, fake transactions (see innodb_fake_changes) don't take row locks. This is an experimental feature to attempt to deal with drawbacks in fake changes blocking real locks. It is not safe for use in all environments. Added as a deprecated and ignored option in MariaDB 10.2.6 (which uses InnoDB as default instead of XtraDB) to allow for easier upgrades.
- **Commandline:** `--innodb-locking-fake-changes`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`

- **Default Value:** `ON`
- **Deprecated:** MariaDB 10.2.6
- **Removed:** MariaDB 10.3.0

---

innodb_locks_unsafe_for_binlog

- **Description:** Set to `0` by default, in which case XtraDB/InnoDB uses gap locking. If set to `1`, gap locking is disabled for searches and index scans. Deprecated in MariaDB 10.0, and removed in MariaDB 10.5, use READ COMMITTED transaction isolation level instead.
- **Commandline:** `--innodb-locks-unsafe-for-binlog`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Deprecated:** MariaDB 10.0
- **Removed:** MariaDB 10.5.0

---

innodb_log_arch_dir

- **Description:** The directory for XtraDB redo log archiving. XtraDB only. Added as a deprecated and ignored option in MariaDB 10.2.6 (which uses InnoDB as default instead of XtraDB) to allow for easier upgrades.
- **Commandline:** `--innodb-log-arch-dir=name`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** `./`
- **Deprecated:** MariaDB 10.2.6
- **Removed:** MariaDB 10.3.0

---

innodb_log_arch_expire_sec

- **Description:** Time in seconds since the last change after which the archived XtraDB redo log should be deleted. XtraDB only. Added as a deprecated and ignored option in MariaDB 10.2.6 (which uses InnoDB as default instead of XtraDB) to allow for easier upgrades.
- **Commandline:** `--innodb-log-arch-expire-sec=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Deprecated:** MariaDB 10.2.6
- **Removed:** MariaDB 10.3.0

---

innodb_log_archive

- **Description:** Whether or not XtraDB redo log archiving is enabled. XtraDB only. Added as a deprecated and ignored option in MariaDB 10.2.6 (which uses InnoDB as default instead of XtraDB) to allow for easier upgrades.
- **Commandline:** `--innodb-log-archive={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Deprecated:** MariaDB 10.2.6
- **Removed:** MariaDB 10.3.0

---

innodb_log_block_size

- **Description:** Size in bytes of the XtraDB redo log records. Generally `512`, the default, or `4096`, are the only two useful values. If the server is restarted and this value is changed, all old log files need to be removed. Should be set to `4096` for SSD cards or if innodb_flush_method is set to `ALL_O_DIRECT` on ext4 filesystems. XtraDB only. Added as a deprecated and ignored option in MariaDB 10.2.6 (which uses InnoDB as default instead of XtraDB) to allow

for easier upgrades.
- **Commandline:** `innodb-log-block-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `512`
- **Deprecated:** MariaDB 10.2.6
- **Removed:** MariaDB 10.3.0

---

innodb_log_buffer_size

- **Description:** Size in bytes of the buffer for writing InnoDB redo log files to disk. Increasing this means larger transactions can run without needing to perform disk I/O before committing.
- **Commandline:** `--innodb-log-buffer-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `16777216` (16MB)
- **Range:** `262144` to `4294967295` (256KB to 4096MB)

---

innodb_log_checksum_algorithm

- **Description:** Experimental feature (as of MariaDB 10.0.9), this variable specifies how to generate and verify XtraDB redo log checksums. XtraDB only. Added as a deprecated and ignored option in MariaDB 10.2.6 (which uses InnoDB as default instead of XtraDB) to allow for easier upgrades.
  - `none` - No checksum. A constant value is instead written to logs, and no checksum validation is performed.
  - `innodb` - The default, and the original InnoDB algorithm. This is inefficient, but compatible with all MySQL, MariaDB and Percona versions that don't support other checksum algorithms.
  - `crc32` - CRC32© is used for log block checksums, which also permits recent CPUs to use hardware acceleration (on SSE4.2 x86 machines and Power8 or later) for the checksums.
  - `strict_*` - Whether or not to accept checksums from other algorithms. If strict mode is used, checksums blocks will be considered corrupt if they don't match the specified algorithm. Normally they are considered corrupt only if no other algorithm matches.
- **Commandline:** `innodb-log-checksum-algorithm=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `enum`
- **Default Value:**
  - `deprecated` (>= MariaDB 10.2.6)
  - `innodb` (<= MariaDB 10.1)
- **Valid Values:**
  - `deprecated`, `innodb`, `none`, `crc32`, `strict_none`, `strict_innodb`, `strict_crc32` (>= MariaDB 10.2.6)
  - `innodb`, `none`, `crc32`, `strict_none`, `strict_innodb`, `strict_crc32` (<= MariaDB 10.1)
- **Deprecated:** MariaDB 10.2.6
- **Removed:** MariaDB 10.3.0

---

innodb_log_checksums

- **Description:** If set to `1`, the CRC32C for Innodb or `innodb_log_checksum_algorithm` for XtraDB algorithm is used for InnoDB redo log pages. If disabled, the checksum field contents are ignored. From MariaDB 10.5.0, the variable is deprecated, and checksums are always calculated, as previously, the InnoDB redo log used the slow innodb algorithm, but with hardware or SIMD assisted CRC-32C computation being available, there is no reason to allow checksums to be disabled on the redo log.
- **Commandline:** `innodb-log-checksums={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `ON`
- **Deprecated:** MariaDB 10.5.0
- **Removed:** MariaDB 10.6.0

---

innodb_log_compressed_pages

- **Description:** Whether or not images of recompressed pages are stored in the InnoDB redo log. Deprecated and ignored from MariaDB 10.5.3.
- **Commandline:** `--innodb-log-compressed-pages={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:**
  - `ON`
- **Deprecated:** MariaDB 10.5.3
- **Removed:** MariaDB 10.6.0

---

innodb_log_file_buffering

- **Description:** Whether the file system cache for ib_logfile0 is enabled. In MariaDB 10.8.3, MariaDB disabled the file system cache on the InnoDB write-ahead log file (ib_logfile0) by default on Linux. With innodb_flush_trx_log_at_commit=2 in particular, writing to the log via the file system cache typically improves throughput, especially on slow storage or at a small number of concurrent transactions. For other values of innodb_flush_log_at_trx_commit, direct writes were observed to be mostly but not always faster. Whether it pays off to disable the file system cache on the log may depend on the type of storage, the workload, and the operating system kernel version. If the server is started up with innodb_flush_log_at_trx_commit=2, the value will be changed to `ON`. Will be set to `OFF` if innodb_flush_method is set to `O_DSYNC`. On Linux, when the physical block size cannot be determined to be a power of 2 between 64 and 4096 bytes, the file system cache cannot be disabled, and innodb_log_file_buffering=ON cannot be changed. Linux and Windows only.
- **Commandline:** `--innodb-log-file-buffering={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB 10.8.4, MariaDB 10.9.2

---

innodb_log_file_size

- **Description:** Size in bytes of each InnoDB redo log file in the log group. The combined size can be no more than 512GB. Larger values mean less disk I/O due to less flushing checkpoint activity, but also slower recovery from a crash. In MariaDB 10.5, crash recovery has been improved and shouldn't run out of memory, so the default has been increased. It can safely be set higher to reduce checkpoint flushing, even larger than innodb_buffer_pool_size. From MariaDB 10.9 the variable is dynamic, and the server no longer needs to be restarted for the resizing to take place. Unless the log is located in a persistent memory file system (PMEM), an attempt to SET GLOBAL innodb_log_file_size to less than innodb_log_buffer_size will be refused. Log resizing can be aborted by killing the connection that is executing the SET GLOBAL statement.
- **Commandline:** `--innodb-log-file-size=#`
- **Scope:** Global
- **Dynamic:** Yes (>= MariaDB 10.9), No (<= MariaDB 10.8)
- **Data Type:** `numeric`
- **Default Value:** `100663296` (96MB) (>= MariaDB 10.5), `50331648` (48MB) (<= MariaDB 10.4)
- **Range:**
  - >= MariaDB 10.8.3: `4194304` to `512GB` (4MB to 512GB)
  - <= MariaDB 10.8.2: `1048576` to `512GB` (1MB to 512GB)

---

innodb_log_file_write_through

- **Description:** Whether each write to ib_logfile0 is write through (disabling any caching, as in O_SYNC or O_DSYNC). Set to `OFF` by default, will be set to `ON` if innodb_flush_method is set to `O_DSYNC`. On systems that support FUA it may make sense to enable write-through, to avoid extra system calls.
- **Commandline:** `--innodb-log-file-write-through={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB 11.0.0

innodb_log_files_in_group

- **Description:** Number of physical files in the InnoDB redo log. Deprecated and ignored from MariaDB 10.5.2
- **Commandline:** `--innodb-log-files-in-group=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `1` (>= MariaDB 10.5), `2` (<= MariaDB 10.4)
- **Range:** `1` to `100` (>= MariaDB 10.2.4)
- **Deprecated:** MariaDB 10.5.2
- **Removed:** MariaDB 10.6.0

---

innodb_log_group_home_dir

- **Description:** Path to the InnoDB redo log files. If none is specified, innodb_log_files_in_group files named ib_logfile0 and so on, with a size of innodb_log_file_size are created in the data directory.
- **Commandline:** `--innodb-log-group-home-dir=path`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `directory name`

---

innodb_log_optimize_ddl

- **Description:** Whether InnoDB redo log activity should be reduced when natively creating indexes or rebuilding tables. Reduced logging requires additional page flushing and interferes with Mariabackup. Enabling this may slow down backup and cause delay due to page flushing. Deprecated and ignored from MariaDB 10.5.1. Deprecated (but not ignored) from MariaDB 10.4.16, MariaDB 10.3.26 and MariaDB 10.2.35.
- **Commandline:** `--innodb-log-optimize-ddl={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:**
  - `OFF` (>= MariaDB 10.5.1, MariaDB 10.4.16, MariaDB 10.3.26, MariaDB 10.2.35)
  - `ON` (<= MariaDB 10.5.0, MariaDB 10.4.15, MariaDB 10.3.25, MariaDB 10.2.34)
- **Introduced:** MariaDB 10.2.17, MariaDB 10.3.9
- **Deprecated:** MariaDB 10.5.1, MariaDB 10.4.16, MariaDB 10.3.26, MariaDB 10.2.35
- **Removed:** MariaDB 10.6.0

---

innodb_log_write_ahead_size

- **Description:** InnoDB redo log write ahead unit size to avoid read-on-write. Should match the OS cache block IO size. Removed in MariaDB 10.8, and instead on Linux and Windows, the physical block size of the underlying storage is detected and used.
- **Commandline:** `--innodb-log-write-ahead-size=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `8192`
- **Range:** `512` to innodb_page_size
- **Removed:** MariaDB 10.8

---

innodb_lru_flush_size

- **Description:** Number of pages to flush on LRU eviction.
- **Commandline:** `--innodb-lru-flush-size=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `32`
- **Range:** `1` to `18446744073709551615`
- **Introduced:** MariaDB 10.5.7

innodb_lru_scan_depth

- **Description:** Specifies how far down the buffer pool least-recently used (LRU) list the cleaning thread should look for dirty pages to flush. This process is performed once a second. In an I/O intensive-workload, can be increased if there is spare I/O capacity, or decreased if in a write-intensive workload with little spare I/O capacity.
  - See InnoDB Page Flushing for more information.
- **Commandline:** `--innodb-lru-scan-depth=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:**
  - `1536` (>= MariaDB 10.5.7)
  - `1024` (<= MariaDB 10.5.6)
- **Range - 32bit:** `100` to $2^{32}-1$
- **Range - 64bit:** `100` to $2^{64}-1$

innodb_max_bitmap_file_size

- **Description:** Limit in bytes of the changed page bitmap files. For faster incremental backup with Xtrabackup, XtraDB tracks pages with changes written to them according to the XtraDB redo log and writes the information to special changed page bitmap files. These files are rotated when the server restarts or when this limit is reached. XtraDB only. See also innodb_track_changed_pages and innodb_max_changed_pages.
  - Deprecated and ignored in MariaDB 10.2.6 (which uses InnoDB as default instead of XtraDB) to allow for easier upgrades.
- **Commandline:** `innodb-max-bitmap-file-size=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `4096` (4KB)
- **Range:** `4096` (4KB) to `18446744073709551615` (16EB)
- **Deprecated:** MariaDB 10.2.6

innodb_max_changed_pages

- **Description:** Limit to the number of changed page bitmap files (stored in the Information Schema INNODB_CHANGED_PAGES table). Zero is unlimited. See innodb_max_bitmap_file_size and innodb_track_changed_pages. Previously named `innodb_changed_pages_limit` . XtraDB only.
  - Deprecated and ignored in MariaDB 10.2.6 (which uses InnoDB as default instead of XtraDB) to allow for easier upgrades.
- **Commandline:** `innodb-max-changed-pages=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1000000`
- **Range:** `0` to `18446744073709551615`
- **Deprecated:** MariaDB 10.2.6

innodb_max_dirty_pages_pct

- **Description:** Maximum percentage of unwritten (dirty) pages in the buffer pool.
  - See InnoDB Page Flushing for more information.
- **Commandline:** `--innodb-max-dirty-pages-pct=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:**
  - `90.000000` (>= MariaDB 10.5.7)
  - `75.000000` (<= MariaDB 10.5.6)
- **Range:** `0` to `99.999`

innodb_max_dirty_pages_pct_lwm

- **Description:** Low water mark percentage of dirty pages that will enable preflushing to lower the dirty page ratio. The value 0 (default) means 'refer to innodb_max_dirty_pages_pct'.
  - See InnoDB Page Flushing for more information.
- **Commandline:** `--innodb-max-dirty-pages-pct-lwm=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `99.999`

---

innodb_max_purge_lag

- **Description:** When purge operations are lagging on a busy server, setting innodb_max_purge_lag can help. By default set to `0`, no lag, the figure is used to calculate a time lag for each INSERT, UPDATE, and DELETE when the system is lagging. InnoDB keeps a list of transactions with delete-marked index records due to UPDATE and DELETE statements. The length of this list is `purge_lag`, and the calculation, performed every ten seconds, is as follows: ((purge_lag/innodb_max_purge_lag)×10)–5 microseconds.
- **Commandline:** `--innodb-max-purge-lag=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `4294967295`

---

innodb_max_purge_lag_delay

- **Description:** Maximum delay in milliseconds imposed by the innodb_max_purge_lag setting. If set to `0`, the default, there is no maximum.
- **Commandline:** `--innodb-max-purge-lag-delay=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`

---

innodb_max_purge_lag_wait

- **Description:** Wait until History list length is below the specified limit.
- **Commandline:** `--innodb-max-purge-wait=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `4294967295`
- **Range:** `0` to `4294967295`
- **Introduced:** MariaDB 10.5.7, MariaDB 10.4.16, MariaDB 10.3.26, MariaDB 10.2.35

---

innodb_max_undo_log_size

- **Description:** If an undo tablespace is larger than this, it will be marked for truncation if innodb_undo_log_truncate is set.
- **Commandline:** `--innodb-max-undo-log-size=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:**
  - `10485760`
- **Range:** `10485760` to `18446744073709551615`

---

innodb_merge_sort_block_size

- **Description:** Size in bytes of the block used for merge sorting in fast index creation. Replaced in MariaDB 10.0/XtraDB 5.6 by innodb_sort_buffer_size.
- **Commandline:** `innodb-merge-sort-block-size=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1048576` (1M)
- **Range:** `1048576` (1M) to `1073741824` (1G)
- **Removed:** MariaDB 10.0 - replaced by innodb_sort_buffer_size

---

innodb_mirrored_log_groups

- **Description:** Unused. Restored as a deprecated and ignored option in MariaDB 10.2.6 (which uses InnoDB as default instead of XtraDB) to allow for easier upgrades.
- **Deprecated:** MariaDB 10.0
- **Removed:** MariaDB 10.2.2 - MariaDB 10.2.5

---

innodb_mtflush_threads

- **Description:** Sets the number of threads to use in Multi-Threaded Flush operations. For more information, see Fusion-io Multi-threaded Flush.
  - InnoDB's multi-thread flush feature was deprecated in MariaDB 10.2.9 and removed from MariaDB 10.3.2. In later versions of MariaDB, use `innodb_page_cleaners` system variable instead.
  - See InnoDB Page Flushing: Page Flushing with Multi-threaded Flush Threads for more information.
- **Commandline:** `--innodb-mtflush-threads=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `8`
- **Range:** `1` to `64`
- **Deprecated:** MariaDB 10.2.9
- **Removed:** MariaDB 10.3.2

---

innodb_monitor_disable

- **Description:** Disables the specified counters in the INFORMATION_SCHEMA.INNODB_METRICS table.
- **Commandline:** `--innodb-monitor-disable=string`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`

---

innodb_monitor_enable

- **Description:** Enables the specified counters in the INFORMATION_SCHEMA.INNODB_METRICS table.
- **Commandline:** `--innodb-monitor-enable=string`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`

---

innodb_monitor_reset

- **Description:** Resets the count value of the specified counters in the INFORMATION_SCHEMA.INNODB_METRICS table to zero.
- **Commandline:** `--innodb-monitor-reset=string`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`

---

innodb_monitor_reset_all

- **Description:** Resets all values for the specified counters in the INFORMATION_SCHEMA.INNODB_METRICS table.
- **Commandline:** `---innodb-monitor-reset-all=string`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`

---

innodb_numa_interleave

- **Description:** Whether or not to use the NUMA interleave memory policy to allocate the InnoDB buffer pool. Before MariaDB 10.2.4, required that MariaDB be compiled on a NUMA-enabled Linux system.
- **Commandline:** `innodb-numa-interleave={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Removed:** MariaDB 10.2.23, MariaDB 10.3.14, MariaDB 10.4.4

---

innodb_old_blocks_pct

- **Description:** Percentage of the buffer pool to use for the old block sublist.
- **Commandline:** `--innodb-old-blocks-pct=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `37`
- **Range:** `5` to `95`

---

innodb_old_blocks_time

- **Description:** Time in milliseconds an inserted block must stay in the old sublist after its first access before it can be moved to the new sublist. '0' means "no delay". Setting a non-zero value can help prevent full table scans clogging the buffer pool. See also innodb_old_blocks_pct.
- **Commandline:** `--innodb-old-blocks-time=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1000`
- **Range:** `0` to $2^{32}-1$

---

innodb_online_alter_log_max_size

- **Description:** The maximum size for temporary log files during online DDL (data and index structure changes). The temporary log file is used for each table being altered, or index being created, to store data changes to the table while the process is underway. The table is extended by innodb_sort_buffer_size up to the limit set by this variable. If this limit is exceeded, the online DDL operation fails and all uncommitted changes are rolled back. A lower value reduces the time a table could lock at the end of the operation to apply all the log's changes, but also increases the chance of the online DDL changes failing.
- **Commandline:** `--innodb-online-alter-log-max-size=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `134217728`
- **Range:** `65536` to $2^{64}-1$

---

innodb_open_files

- **Description:** Maximum .ibd files MariaDB can have open at the same time. Only applies to systems with multiple XtraDB/InnoDB tablespaces, and is separate to the table cache and open_files_limit. The default, if innodb_file_per_table is disabled, is 300 or the value of table_open_cache, whichever is higher. It will also auto-size up to the default value if it is set to a value less than `10` .

- **Commandline:** `--innodb-open-files=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `autosized`
- **Range:** `10` to `4294967295`

---

innodb_optimize_fulltext_only

- **Description:** When set to `1` (`0` is default), OPTIMIZE TABLE will only process InnoDB FULLTEXT index data. Only intended for use during fulltext index maintenance.
- **Commandline:** `--innodb-optimize-fulltext-only={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

innodb_page_cleaners

- **Description:** Number of page cleaner threads. The default is `4`, but the value will be set to the number of innodb_buffer_pool_instances if this is lower. If set to `1`, only a single cleaner thread is used, as was the case until MariaDB 10.2.1. Cleaner threads flush dirty pages from the buffer pool, performing flush list and least-recently used (LRU) flushing. Deprecated and ignored from MariaDB 10.5.1, as the original reasons for for splitting the buffer pool have mostly gone away.
  - See InnoDB Page Flushing: Page Flushing with Multiple InnoDB Page Cleaner Threads for more information.
- **Commandline:** `--innodb-page-cleaners=#`
- **Scope:** Global
- **Dynamic:** Yes (>= MariaDB 10.3.3), No (<= MariaDB 10.3.2)
- **Data Type:** `numeric`
- **Default Value:** `4` (or set to innodb_buffer_pool_instances if lower)
- **Range:** `1` to `64`
- **Deprecated:** MariaDB 10.5.1
- **Removed:** MariaDB 10.6.0

---

innodb_page_size

- **Description:** Specifies the page size in bytes for all InnoDB tablespaces. The default, `16k`, is suitable for most uses.
  - A smaller InnoDB page size might work more effectively in a situation with many small writes (OLTP), or with SSD storage, which usually has smaller block sizes.
  - A larger InnoDB page size can provide a larger maximum row size.
  - InnoDB's page size can be as large as `64k` for tables using the following row formats: DYNAMIC, COMPACT, and REDUNDANT.
  - InnoDB's page size must still be `16k` or less for tables using the COMPRESSED row format.
  - This system variable's value cannot be changed after the `datadir` has been initialized. InnoDB's page size is set when a MariaDB instance starts, and it remains constant afterwards.
- **Commandline:** `--innodb-page-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `enumeration`
- **Default Value:** `16384`
- **Valid Values:** `4k` or `4096`, `8k` or `8192`, `16k` or `16384`, `32k` and `64k`.

---

innodb_pass_corrupt_table

- **Removed:** XtraDB 5.5 - renamed innodb_corrupt_table_action.

---

innodb_prefix_index_cluster_optimization

- **Description:** Enable prefix optimization to sometimes avoid cluster index lookups. Deprecated and ignored from MariaDB 10.10, as the optimization is now always enabled.

- **Commandline:** `--innodb-prefix-index-cluster-optimization={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Deprecated:** MariaDB 10.10.1

---

innodb_print_all_deadlocks

- **Description:** If set to `1` (`0` is default), all XtraDB/InnoDB transaction deadlock information is written to the error log.
- **Commandline:** `--innodb-print-all-deadlocks={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

innodb_purge_batch_size

- **Description:** Number of InnoDB undo log pages to purge in one batch from the history list. Together with innodb_purge_threads has a small effect on tuning.
- **Commandline:** `--innodb-purge-batch-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:**
  - `1000` (>= MariaDB 10.6.16, MariaDB 10.10.7, MariaDB 10.11.6, MariaDB 11.0.4, MariaDB 11.1.3 MariaDB 11.2.2)
  - `300` (<= MariaDB 10.6.15, MariaDB 10.10.6, MariaDB 10.11.5, MariaDB 11.0.3, MariaDB 11.1.2 MariaDB 11.2.1)
- **Range:** `1` to `5000`

---

innodb_purge_rseg_truncate_frequency

- **Description:** Frequency with which undo records are purged. Set by default to every 128 times, reducing this increases the frequency at which rollback segments are freed. See also innodb_undo_log_truncate. The motivation for introducing this in MySQL seems to have been to avoid stalls due to freeing undo log pages or truncating undo log tablespaces. In MariaDB, innodb_undo_log_truncate=ON should be a much lighter operation because it will not involve any log checkpoint, hence this is deprecated and ignored from MariaDB 10.6.16, MariaDB 10.10.7, MariaDB 10.11.6, MariaDB 11.0.4, MariaDB 11.1.3 and MariaDB 11.2.2. (MDEV-32050)
- **Commandline:** `-- innodb-purge-rseg-truncate-frequency=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `128`
- **Range:** `1` to `128`
- **Deprecated:** MariaDB 10.6.16, MariaDB 10.10.7, MariaDB 10.11.6, MariaDB 11.0.4, MariaDB 11.1.3, MariaDB 11.2.2

---

innodb_purge_threads

- **Description:** Number of background threads dedicated to InnoDB purge operations. The range is `1` to `32`. At least one background thread is always used. Setting to a value greater than 1 creates that many separate purge threads. This can improve efficiency in some cases, such as when performing DML operations on many tables. See also innodb_purge_batch_size.
- **Commandline:** `--innodb-purge-threads=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `4`
- **Range:** `1` to `32`

---

innodb_random_read_ahead

- **Description:** Originally, random read-ahead was always set as an optimization technique, but was removed in MariaDB 5.5. `innodb_random_read_ahead` permits it to be re-instated if set to `1` (`0`) is default.
- **Commandline:** `--innodb-random-read-ahead={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

innodb_read_ahead

- **Description:** If set to `linear`, the default, XtraDB/InnoDB will automatically fetch remaining pages if there are enough within the same extent that can be accessed sequentially. If set to `none`, read-ahead is disabled. `random` has been removed and is now ignored, while `both` sets to both `linear` and `random`. Also see innodb_read_ahead_threshold for more control on read-aheads. Removed in MariaDB 10.0/XtraDB 5.6 and replaced by MySQL 5.6's innodb_random_read_ahead.
- **Commandline:** `innodb-read-ahead=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `enumeration`
- **Default Value:** `linear`
- **Valid Values:** `none`, `random`, `linear`, `both`
- **Removed:** MariaDB 10.0/XtraDB 5.6 - replaced by MySQL 5.6's innodb_random_read_ahead

---

innodb_read_ahead_threshold

- **Description:** Minimum number of pages InnoDB must read from an extent of 64 before initiating an asynchronous read for the following extent.
- **Commandline:** `--innodb-read-ahead-threshold=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `56`
- **Range:** `0` to `64`

---

innodb_read_io_threads

- **Description:** Number of I/O threads for InnoDB reads. You may on rare occasions need to reduce this default on Linux systems running multiple MariaDB servers to avoid exceeding system limits.
- **Commandline:** `--innodb-read-io-threads=#`
- **Scope:** Global
- **Dynamic:** Yes (>= MariaDB 10.11), No (<= MariaDB 10.10)
- **Data Type:** `numeric`
- **Default Value:** `4`
- **Range:** `1` to `64`

---

innodb_read_only

- **Description:** If set to `1` (`0` is default), the server will be read-only. For use in distributed applications, data warehouses or read-only media.
- **Commandline:** `--innodb-read-only={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

innodb_read_only_compressed

- **Description:** If set (the default before MariaDB 10.6.6), ROW_FORMAT=COMPRESSED tables will be read-only.

This was intended to be the first step towards removing write support and deprecating the feature, but this plan has been abandoned.

- **Commandline:** `--innodb-read-only-compressed`, `--skip-innodb-read-only-compressed`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF` (>= MariaDB 10.6.6), `ON` (<= MariaDB 10.6.5)
- **Introduced:** MariaDB 10.6.0

---

innodb_recovery_stats

- **Description:** If set to `1` (`0` is default) and recovery is necessary on startup, the server will write detailed recovery statistics to the error log at the end of the recovery process. This Percona XtraDB variable has not been ported to XtraDB 5.6.
- **Commandline:** No
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Removed:** MariaDB 10.0

---

innodb_recovery_update_relay_log

- **Description:** If set to `1` (`0` is default), the relay log info file will be overwritten on crash recovery if the information differs from the InnoDB record. Should not be used if multiple storage engine types are being replicated. Previously named `innodb_overwrite_relay_log_info`. Removed in MariaDB 10.0/XtraDB 5.6 and replaced by MySQL 5.6's `relay-log-recovery`
- **Commandline:** `innodb-recovery-update-relay-log={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Removed:** MariaDB 10.0 - replaced by MySQL 5.6's `relay-log-recovery`

---

innodb_replication_delay

- **Description:** Time in milliseconds for the replica server to delay the replication thread if innodb_thread_concurrency is reached. Deprecated and ignored from MariaDB 10.5.5.
- **Commandline:** `--innodb-replication-delay=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `4294967295`
- **Deprecated:** MariaDB 10.5.5
- **Removed:** MariaDB 10.6.0

---

innodb_rollback_on_timeout

- **Description:** InnoDB usually rolls back the last statement of a transaction that's been timed out (see innodb_lock_wait_timeout). If innodb_rollback_on_timeout is set to 1 (0 is default), InnoDB will roll back the entire transaction. Before MariaDB 5.5, rolling back the entire transaction was the default behavior.
- **Commandline:** `--innodb-rollback-on-timeout`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `0`

---

innodb_rollback_segments

- **Description:** Specifies the number of rollback segments that XtraDB/InnoDB will use within a transaction (see undo log). Deprecated and replaced by innodb_undo_logs in MariaDB 10.0.
- **Commandline:** `--innodb-rollback-segments=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `128`
- **Range:** `1` to `128`
- **Deprecated:** MariaDB 10.0
- **Removed:** MariaDB 10.5.0

---

innodb_safe_truncate

- **Description:** Use a backup-safe TRUNCATE TABLE implementation and crash-safe rename operations inside InnoDB. This is not compatible with hot backup tools other than Mariabackup. Users who need to use such tools may set this to `OFF`.
- **Commandline:** `--innodb-safe-truncate={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `ON`
- **Introduced:** MariaDB 10.2.19
- **Removed:** MariaDB 10.3.0

---

innodb_scrub_log

- **Description:** Enable InnoDB redo log scrubbing. See Data Scrubbing. Deprecated and ignored from MariaDB 10.5.2, as never really worked (MDEV-13019 and MDEV-18370). If old log contents should be kept secret, then enabling innodb_encrypt_log or setting a smaller innodb_log_file_size could help.
- **Commandline:** `--innodb-scrub-log`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Deprecated:** MariaDB 10.5.2
- **Removed:** MariaDB 10.6.0

---

innodb_scrub_log_interval

- **Description:** Used with Data Scrubbing in 10.1.3 only - replaced in 10.1.4 by innodb_scrub_log_speed. InnoDB redo log scrubbing interval in milliseconds.
- **Commandline:** `--innodb-scrub-log-interval=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `56`
- **Range:** `0` to `50000`
- **Introduced:** MariaDB 10.1.3
- **Removed:** MariaDB 10.1.4

---

innodb_scrub_log_speed

- **Description:** InnoDB redo log scrubbing speed in bytes/sec. See Data Scrubbing. Deprecated and ignored from MariaDB 10.5.2.
- **Commandline:** `--innodb-scrub-log-speed=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `256`
- **Range:** `1` to `50000`
- **Deprecated:** MariaDB 10.5.2

- **Removed:** MariaDB 10.6.0

---

innodb_sched_priority_cleaner

- **Description:** Set a thread scheduling priority for cleaner and least-recently used (LRU) manager threads. The range from `0` to `39` corresponds in reverse order to Linux nice values of `-20` to `19`. So `0` is the lowest priority (Linux nice value `19`) and `39` is the highest priority (Linux nice value `-20`). XtraDB only. Added as a deprecated and ignored option in MariaDB 10.2.6 (which uses InnoDB as default instead of XtraDB) to allow for easier upgrades.
- **Commandline:** `innodb-sched-priority-cleaner=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `19`
- **Range:** `0` to `39`
- **Deprecated:** MariaDB 10.2.6
- **Removed:** MariaDB 10.3.0

---

innodb_show_locks_held

- **Description:** Specifies the number of locks held for each InnoDB transaction to be displayed in SHOW ENGINE INNODB STATUS output. XtraDB only. Added as a deprecated and ignored option in MariaDB 10.2.6 (which uses InnoDB as default instead of XtraDB) to allow for easier upgrades.
- **Commandline:** `innodb-show-locks-held=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `10`
- **Range:** `0` to `1000`
- **Deprecated:** MariaDB 10.2.6
- **Removed:** MariaDB 10.3.0

---

innodb_show_verbose_locks

- **Description:** If set to `1`, and innodb_status_output_locks is also ON, the traditional InnoDB behavior is followed and locked records will be shown in SHOW ENGINE INNODB STATUS output. If set to `0`, the default, only high-level information about the lock is shown. XtraDB only. Added as a deprecated and ignored option in MariaDB 10.2.6 (which uses InnoDB as default instead of XtraDB) to allow for easier upgrades.
- **Commandline:** `innodb-show-verbose-locks=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `1`
- **Deprecated:** MariaDB 10.2.6
- **Removed:** MariaDB 10.3.0

---

innodb_simulate_comp_failures

- **Description:** Simulate compression failures. Used for testing robustness against random compression failures. XtraDB only.
- **Commandline:** None
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `99`

---

innodb_sort_buffer_size

- **Description:** Size of the sort buffers used for sorting data when an InnoDB index is created, as well as the amount by

which the temporary log file is extended during online DDL operations to record concurrent writes. The larger the setting, the fewer merge phases are required between buffers while sorting. When a CREATE TABLE or ALTER TABLE creates a new index, three buffers of this size are allocated, as well as pointers for the rows in the buffer.

- **Commandline:** `--innodb-sort-buffer-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `1048576` (1M)
- **Range:** `65536` to `67108864`

---

innodb_spin_wait_delay

- **Description:** Maximum delay (not strictly corresponding to a time unit) between spin lock polls. Default changed from `6` to `4` in MariaDB 10.3.5, as this was verified to give the best throughput by OLTP update index and read-write benchmarks on Intel Broadwell (2/20/40) and ARM (1/46/46).
- **Commandline:** `--innodb-spin-wait-delay=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `4` (>= MariaDB 10.3.5), `6` (<= MariaDB 10.3.4)
- **Range:** `0` to `4294967295`

---

innodb_stats_auto_recalc

- **Description:** If set to `1` (the default), persistent statistics are automatically recalculated when the table changes significantly (more than 10% of the rows). Affects tables created or altered with STATS_PERSISTENT=1 (see CREATE TABLE), or when innodb_stats_persistent is enabled. innodb_stats_persistent_sample_pages determines how much data to sample when recalculating. See InnoDB Persistent Statistics.
- **Commandline:** `--innodb-stats-auto-recalc={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `ON`

---

innodb_stats_auto_update

- **Description:** If set to `0` (`1` is default), index statistics will not be automatically calculated except when an ANALYZE TABLE is run, or the table is first opened. Replaced by innodb_stats_auto_recalc in MariaDB 10.0/XtraDB 5.6.
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `1`
- **Removed:** MariaDB 10.0 - replaced by innodb_stats_auto_recalc.

---

innodb_stats_include_delete_marked

- **Description:** Include delete marked records when calculating persistent statistics.
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

innodb_stats_method

- **Description:** Determines how NULLs are treated for InnoDB index statistics purposes.
  - `nulls_equal` : The default, all NULL index values are treated as a single group. This is usually fine, but if you have large numbers of NULLs the average group size is slanted higher, and the optimizer may miss using the index for ref accesses when it would be useful.
  - `nulls_unequal` : The opposite approach to `nulls_equal` is taken, with each NULL forming its own group of one. Conversely, the average group size is slanted lower, and the optimizer may use the index for ref accesses

when not suitable.
    - `nulls_ignored`: Ignore NULLs altogether from index group calculations.
    - See also Index Statistics, aria_stats_method and myisam_stats_method.
- **Commandline:** `--innodb-stats-method=name`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `enumeration`
- **Default Value:** `nulls_equal`
- **Valid Values:** `nulls_equal`, `nulls_unequal`, `nulls_ignored`

---

innodb_stats_modified_counter

- **Description:** The number of rows modified before we calculate new statistics. If set to `0`, the default, current limits are used.
- **Commandline:** `--innodb-stats-modified-counter=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `18446744073709551615`

---

innodb_stats_on_metadata

- **Description:** If set to `1`, the default, XtraDB/InnoDB updates statistics when accessing the INFORMATION_SCHEMA.TABLES or INFORMATION_SCHEMA.STATISTICS tables, and when running metadata statements such as SHOW INDEX or SHOW TABLE STATUS. If set to `0`, statistics are not updated at those times, which can reduce the access time for large schemas, as well as make execution plans more stable.
- **Commandline:** `--innodb-stats-on-metadata`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

innodb_stats_persistent

- **Description:** ANALYZE TABLE produces index statistics, and this setting determines whether they will be stored on disk, or be required to be recalculated more frequently, such as when the server restarts. This information is stored for each table, and can be set with the STATS_PERSISTENT clause when creating or altering tables (see CREATE TABLE). See InnoDB Persistent Statistics.
- **Commandline:** `--innodb-stats-persistent={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `ON`

---

innodb_stats_persistent_sample_pages

- **Description:** Number of index pages sampled when estimating cardinality and statistics for indexed columns. Increasing this value will increases index statistics accuracy, but use more I/O resources when running ANALYZE TABLE. See InnoDB Persistent Statistics.
- **Commandline:** `--innodb-stats-persistent-sample-pages=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `20`

---

innodb_stats_sample_pages

- **Description:** Gives control over the index distribution statistics by determining the number of index pages to sample. Higher values produce more disk I/O, but, especially for large tables, produce more accurate statistics and therefore

make more effective use of the query optimizer. Lower values than the default are not recommended, as the statistics can be quite inaccurate.
    - If innodb_stats_traditional is enabled, then the exact number of pages configured by this system variable will be sampled for statistics.
    - If innodb_stats_traditional is disabled, then the number of pages to sample for statistics is calculated using a logarithmic algorithm, so the exact number can change depending on the size of the table. This means that more samples may be used for larger tables.
    - If persistent statistics are enabled, then the innodb_stats_persistent_sample_pages system variable applies instead. persistent statistics are enabled with the innodb_stats_persistent system variable.
    - This system variable has been **deprecated**. The innodb_stats_transient_sample_pages system variable should be used instead.
- **Commandline:** `--innodb-stats-sample-pages=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `8`
- **Range:** `1` to `2^64-1`
- **Deprecated:** MariaDB 10.0
- **Removed:** MariaDB 10.5.0

---

innodb_stats_traditional

- **Description:** This system variable affects how the number of pages to sample for transient statistics is determined, in particular how innodb_stats_transient_sample_pages is used.
    - If innodb_stats_traditional is enabled, then the exact number of pages configured by the system variable will be sampled for statistics.
    - If innodb_stats_traditional is disabled, then the number of pages to sample for statistics is calculated using a logarithmic algorithm, so the exact number can change depending on the size of the table. This means that more samples may be used for larger tables.
    - This system variable does not affect the calculation of persistent statistics.
- **Commandline:** `--innodb-stats-traditional={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `ON`

---

innodb_stats_transient_sample_pages

- **Description:** Gives control over the index distribution statistics by determining the number of index pages to sample. Higher values produce more disk I/O, but, especially for large tables, produce more accurate statistics and therefore make more effective use of the query optimizer. Lower values than the default are not recommended, as the statistics can be quite inaccurate.
    - If `innodb_stats_traditional` is enabled, then the exact number of pages configured by this system variable will be sampled for statistics.
    - If `innodb_stats_traditional` is disabled, then the number of pages to sample for statistics is calculated using a logarithmic algorithm, so the exact number can change depending on the size of the table. This means that more samples may be used for larger tables.
    - If persistent statistics are enabled, then the `innodb_stats_persistent_sample_pages` system variable applies instead. persistent statistics are enabled with the `innodb_stats_persistent` system variable.
- **Commandline:** `--innodb-stats-transient-sample-pages=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `8`
- **Range:** `1` to `2^64-1`

---

innodb_stats_update_need_lock

- **Description:** Setting to `0` (`1` is default) may help reduce contention of the `&dict_operation_lock`, but also disables the *Data_free* option in SHOW TABLE STATUS. This Percona XtraDB variable has not been ported to XtraDB 5.6.
- **Scope:** Global
- **Dynamic:** Yes

- **Data Type:** `boolean`
- **Default Value:** `1`
- **Removed:** MariaDB 10.0/XtraDB 5.6

---

innodb_status_output

- **Description:** Enable InnoDB monitor output to the error log.
- **Commandline:** `--innodb-status-output={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

innodb_status_output_locks

- **Description:** Enable InnoDB lock monitor output to the error log and SHOW ENGINE INNODB STATUS. Also requires innodb_status_output=ON to enable output to the error log.
- **Commandline:** `--innodb-status-output-locks={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

innodb_strict_mode

- **Description:** If set to `1` (the default), InnoDB will return errors instead of warnings in certain cases, similar to strict SQL mode. See InnoDB Strict Mode for details.
- **Commandline:** `--innodb-strict-mode={0|1}`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `ON`

---

innodb_support_xa

- **Description:** If set to `1`, the default, XA transactions are supported. XA support ensures data is written to the binary log in the same order to the actual database, which is critical for replication and disaster recovery, but comes at a small performance cost. If your database is set up to only permit one thread to change data (for example, on a replication replica with only the replication thread writing), it is safe to turn this option off. Removed in MariaDB 10.3, XA transactions are always supported.
- **Commandline:** `--innodb-support-xa`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `ON`
- **Deprecated:** MariaDB 10.2
- **Removed:** MariaDB 10.3.0

---

innodb_sync_array_size

- **Description:** By default `1`, can be increased to split internal thread co-ordinating, giving higher concurrency when there are many waiting threads.
- **Commandline:** `--innodb-sync-array-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `1`
- **Range:** `1` to `1024`
- **Removed:** MariaDB 10.6.0

---

innodb_sync_spin_loops

- **Description:** The number of times a thread waits for an InnoDB mutex to be freed before the thread is suspended.
- **Commandline:** `--innodb-sync-spin-loops=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `30`
- **Range:** `0` to `4294967295`

---

innodb_table_locks

- **Description:** If autocommit is set to to `0` (`1` is default), setting innodb_table_locks to `1` , the default, will cause InnoDB to lock a table internally upon a LOCK TABLE.
- **Commandline:** `--innodb-table-locks`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `ON`

---

innodb_thread_concurrency

- **Description:** Once this number of threads is reached (excluding threads waiting for locks), XtraDB/InnoDB will place new threads in a wait state in a first-in, first-out queue for execution, in order to limit the number of threads running concurrently. A setting of `0` , the default, permits as many threads as necessary. A suggested setting is twice the number of CPU's plus the number of disks. Deprecated and ignored from MariaDB 10.5.5.
- **Commandline:** `--innodb-thread-concurrency=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `1000`
- **Deprecated:** MariaDB 10.5.5
- **Removed:** MariaDB 10.6.0

---

innodb_thread_concurrency_timer_based

- **Description:** If set to `1` , thread concurrency will be handled in a lock-free timer-based manner rather than the default mutex-based method. Depends on atomic op builtins being available. This Percona XtraDB variable has not been ported to XtraDB 5.6.
- **Commandline:** `innodb-thread-concurrency-timer-based={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Removed:** MariaDB 10.0/XtraDB 5.6

---

innodb_thread_sleep_delay

- **Description:** Time in microseconds that InnoDB threads sleep before joining the queue. Setting to `0` disables sleep. Deprecated and ignored from MariaDB 10.5.5
- **Commandline:** `--innodb-thread-sleep-delay=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:**
  - `0` (>= MariaDB 10.5.5.)
  - `10000` (<= MariaDB 10.5.4)
- **Range:** `0` to `1000000`
- **Deprecated:** MariaDB 10.5.5
- **Removed:** MariaDB 10.6.0

innodb_temp_data_file_path

- **Description:**
- **Commandline:** `--innodb-temp-data-file-path=path`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** `ibtmp1:12M:autoextend`

innodb_tmpdir

- **Description:** Allows an alternate location to be set for temporary non-tablespace files. If not set (the default), files will be created in the usual tmpdir location.
- **Commandline:** `--innodb-tmpdir=path`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** Empty

innodb_track_changed_pages

- **Description:** For faster incremental backup with Xtrabackup, XtraDB tracks pages with changes written to them according to the XtraDB redo log and writes the information to special changed page bitmap files. This read-only variable is used for controlling this feature. See also innodb_max_changed_pages and innodb_max_bitmap_file_size. XtraDB only. Added as a deprecated and ignored option in MariaDB 10.2.6 (which uses InnoDB as default instead of XtraDB) to allow for easier upgrades.
- **Commandline:** `innodb-track-changed-pages={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Deprecated:** MariaDB 10.2.6

innodb_track_redo_log_now

- **Description:** Available on debug builds only. XtraDB only. Added as a deprecated and ignored option in MariaDB 10.2.6 (which uses InnoDB as default instead of XtraDB) to allow for easier upgrades.
- **Commandline:** `innodb-track-redo-log-now={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Deprecated:** MariaDB 10.2.6

innodb_truncate_temporary_tablespace_now

- **Description:** Set to ON to shrink the temporary tablespace.
- **Commandline:** `innodb-truncate-temporary-tablespace-now={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB 11.3.0

innodb_undo_directory

- **Description:** Path to the directory (relative or absolute) that InnoDB uses to create separate tablespaces for the undo logs. . (the default value before 10.2.2) leaves the undo logs in the same directory as the other log files. From MariaDB 10.2.2, the default value is NULL, and if no path is specified, undo tablespaces will be created in the

directory defined by datadir. Use together with innodb_undo_logs and innodb_undo_tablespaces. Undo logs are most usefully placed on a separate storage device.

- **Commandline:** `--innodb-undo-directory=name`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** NULL

---

innodb_undo_log_truncate

- **Description:** When enabled, undo tablespaces that are larger than innodb_max_undo_log_size are marked for truncation. See also innodb_purge_rseg_truncate_frequency.
- **Commandline:** `--innodb-undo-log-truncate[={0|1}]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

innodb_undo_logs

- **Description:** Specifies the number of rollback segments that XtraDB/InnoDB will use within a transaction (or the number of active undo logs). By default set to the maximum, `128`, it can be reduced to avoid allocating unneeded rollback segments. See the Innodb_available_undo_logs status variable for the number of undo logs available. See also innodb_undo_directory and innodb_undo_tablespaces. Replaced innodb_rollback_segments in MariaDB 10.0. The Information Schema XTRADB_RSEG Table contains information about the XtraDB rollback segments. Deprecated and ignored in MariaDB 10.5.0, as it always makes sense to use the maximum number of rollback segments.
- **Commandline:** `--innodb-undo-logs=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `128`
- **Range:** `0` to `128`
- **Deprecated:** MariaDB 10.5.0
- **Removed:** MariaDB 10.6.0

---

innodb_undo_tablespaces

- **Description:** Number of tablespaces files used for dividing up the undo logs. Zero (the default before MariaDB 11.0) means that undo logs are all part of the system tablespace, which contains one undo tablespace more than the `innodb_undo_tablespaces` setting. A value of 1 is reset to 0 as 2 or more are needed for separate tablespaces. When the undo logs can grow large, splitting them over multiple tablespaces will reduce the size of any single tablespace. Until MariaDB 10.11.1, must be set before InnoDB is initialized, or else MariaDB will fail to start, with an error saying that `InnoDB did not find the expected number of undo tablespaces`. The files are created in the directory specified by innodb_undo_directory, and are named `undoN`, N being an integer. The default size of an undo tablespace is 10MB.
  From MariaDB 10.11, multiple undo tablespaces are enabled by default, and the default is changed to 3 so that the space occupied by possible bursts of undo log records can be reclaimed after innodb_undo_log_truncate is set. Before MariaDB 10.6, innodb_undo_logs must have a non-zero setting for `innodb_undo_tablespaces` to take effect.
- **Commandline:** `--innodb-undo-tablespaces=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `3` (>= MariaDB 11.0), `0` (<= MariaDB 10.11)
- **Range:** `0`, or `2` to `95`

---

innodb_use_atomic_writes

- **Description:** Implement atomic writes on supported SSD devices. See atomic write support for other variables affected when this is set.

- **Commandline:** `innodb-use-atomic-writes={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `ON`

---

innodb_use_fallocate

- **Description:** Preallocate files fast, using operating system functionality. On POSIX systems, posix_fallocate system call is used.
  - Automatically set to `1` when innodb_use_atomic_writes is set - see FusionIO DirectFS atomic write support.
  - See InnoDB Page Compression: Saving Storage Space with Sparse Files for more information.
- **Commandline:** `innodb-use-fallocate={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Deprecated:** MariaDB 10.2.5 (treated as if `ON`)
- **Removed:** MariaDB 10.3.0

---

innodb_use_global_flush_log_at_trx_commit

- **Description:** Determines whether a user can set the variable innodb_flush_log_at_trx_commit. If set to `1`, a user cannot reset the value with a SET command, while if set to `1`, a user can reset the value of `innodb_flush_log_at_trx_commit`. XtraDB only. Added as a deprecated and ignored option in MariaDB 10.2.6 (which uses InnoDB as default instead of XtraDB) to allow for easier upgrades.
- **Commandline:** `innodb-use-global-flush-log-at-trx_commit={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `ON`
- **Deprecated:** MariaDB 10.2.6
- **Removed:** MariaDB 10.3.0

---

innodb_use_mtflush

- **Description:** Whether to enable Multi-Threaded Flush operations. For more information, see Fusion.
  - InnoDB's multi-thread flush feature was deprecated in MariaDB 10.2.9 and removed from MariaDB 10.3.2. In later versions of MariaDB, use `innodb_page_cleaners` system variable instead.
  - See InnoDB Page Flushing: Page Flushing with Multi-threaded Flush Threads for more information.
- **Commandline:** `--innodb-use-mtflush={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Deprecated:** MariaDB 10.2.9
- **Removed:** MariaDB 10.3.2

---

innodb_use_native_aio

- **Description:** For Linux systems only, specified whether to use Linux's asynchronous I/O subsystem. Set to `ON` by default, it may be changed to `0` at startup if InnoDB detects a problem, or from MariaDB 10.6.5/MariaDB 10.7.1, if a 5.11 - 5.15 Linux kernel is detected, to avoid an io-uring bug/incompatibility (MDEV-26674). MariaDB-10.6.6/MariaDB-10.7.2 and later also consider 5.15.3+ as a fixed kernel and default to `ON`. To really benefit from the setting, the files should be opened in O_DIRECT mode (innodb_flush_method=O_DIRECT, default from MariaDB 10.6), to bypass the file system cache. In this way, the reads and writes can be submitted with DMA, using the InnoDB buffer pool directly, and no processor cycles need to be used for copying data.
- **Commandline:** `--innodb-use-native-aio={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`

- **Default Value:** `ON`

---

innodb_use_purge_thread

- **Description:** Usually with InnoDB, data changed by a transaction is written to an undo space to permit read consistency, and freed when the transaction is complete. Many, or large, transactions, can cause the main tablespace to grow dramatically, reducing performance. This option, introduced in XtraDB 5.1 and removed for 5.5, allows multiple threads to perform the purging, resulting in slower, but much more stable performance.
- **Commandline:** `--innodb-use-purge-thread=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `1`
- **Range:** `0` to `32`
- **Removed:** XtraDB 5.5

---

innodb_use_stacktrace

- **Description:** If set to `ON` (`OFF` is default), a signal handler for SIGUSR2 is installed when the InnoDB server starts. When a long semaphore wait is detected at sync/sync0array.c, a SIGUSR2 signal is sent to the waiting thread and thread that has acquired the RW-latch. For both threads a full stacktrace is produced as well as if possible. XtraDB only. Added as a deprecated and ignored option in MariaDB 10.2.6 (which uses InnoDB as default instead of XtraDB) to allow for easier upgrades.
- **Commandline:** `--innodb-use-stacktrace={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Deprecated:** MariaDB 10.2.6
- **Removed:** MariaDB 10.3.0

---

innodb_use_sys_malloc

- **Description:** If set the `1` , the default, XtraDB/InnoDB will use the operating system's memory allocator. If set to `0` it will use its own. Deprecated in MariaDB 10.0 and removed in MariaDB 10.2 along with InnoDB's internal memory allocator.
- **Commandline:** `--innodb-use-sys-malloc={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `ON`
- **Deprecated:** MariaDB 10.0
- **Removed**: MariaDB 10.2.2

---

innodb_use_sys_stats_table

- **Description:** If set to `1` (`0` is default), XtraDB will use the SYS_STATS system table for extra table index statistics. When a table is opened for the first time, statistics will then be loaded from SYS_STATS instead of sampling the index pages. Statistics are designed to be maintained only by running an ANALYZE TABLE. Replaced by MySQL 5.6's Persistent Optimizer Statistics.
- **Commandline:** `innodb-use-sys-stats-table={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `0`
- **Removed:** MariaDB 10.0/XtraDB 5.6

---

innodb_use_trim

- **Description:** Use trim to free up space of compressed blocks.

    - See InnoDB Page Compression: Saving Storage Space with Sparse Files for more information.
- **Commandline:** `--innodb-use-trim={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `ON`
- **Deprecated:** MariaDB 10.2.4
- **Removed:** MariaDB 10.3.0

`innodb_version`

- **Description:** InnoDB version number. From MariaDB 10.3.7, as the InnoDB implementation in MariaDB has diverged from MySQL, the MariaDB version is instead reported. For example, the InnoDB version reported in MariaDB 10.1 (which is based on MySQL 5.6) included encryption and variable-size page compression before MySQL 5.7 introduced them. MariaDB 10.2 (based on MySQL 5.7) introduced persistent AUTO_INCREMENT (MDEV-6076) in a GA release before MySQL 8.0. MariaDB 10.3 (based on MySQL 5.7) introduced instant ADD COLUMN (MDEV-11369) before MySQL.
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Removed:** MariaDB 10.10

`innodb_write_io_threads`

- **Description:** Number of I/O threads for InnoDB writes. You may on rare occasions need to reduce this default on Linux systems running multiple MariaDB servers to avoid exceeding system limits.
- **Commandline:** `--innodb-write-io-threads=#`
- **Scope:** Global
- **Dynamic:** Yes (>= MariaDB 10.11), No (<= MariaDB 10.10)
- **Data Type:** `numeric`
- **Default Value:** `4`
- **Range:** `1` to `64`

5.3.2.5 InnoDB Server Status Variables

Contents

27. Innodb_buffer_pool_reads
28. Innodb_buffer_pool_resize_status
29. Innodb_buffer_pool_wait_free
30. Innodb_buffer_pool_write_requests
31. Innodb_buffered_aio_submitted
32. Innodb_checkpoint_age
33. Innodb_checkpoint_max_age
34. Innodb_checkpoint_target_age
35. Innodb_current_row_locks
36. Innodb_data_fsyncs
37. Innodb_data_pending_fsyncs
38. Innodb_data_pending_reads
39. Innodb_data_pending_writes
40. Innodb_data_read
41. Innodb_data_reads
42. Innodb_data_writes
43. Innodb_data_written
44. Innodb_dblwr_pages_written
45. Innodb_dblwr_writes
46. Innodb_deadlocks
47. Innodb_defragment_compression_failures
48. Innodb_defragment_count
49. Innodb_defragment_failures
50. Innodb_dict_tables
51. Innodb_encryption_n_merge_blocks_decrypted
52. Innodb_encryption_n_merge_blocks_encrypted
53. Innodb_encryption_n_rowlog_blocks_decrypted
54. Innodb_encryption_n_rowlog_blocks_encrypted
55. Innodb_encryption_n_temp_blocks_decrypted
56. Innodb_encryption_n_temp_blocks_encrypted
57. Innodb_encryption_num_key_requests
58. Innodb_encryption_rotation_estimated_iops
59. Innodb_encryption_rotation_pages_flushed
60. Innodb_encryption_rotation_pages_modified
61. Innodb_encryption_rotation_pages_read_from_cache
62. Innodb_encryption_rotation_pages_read_from_disk
63. Innodb_have_atomic_builtins
64. Innodb_have_bzip2
65. Innodb_have_lz4
66. Innodb_have_lzma
67. Innodb_have_lzo
68. Innodb_have_punch_hole
69. Innodb_have_snappy
70. Innodb_history_list_length
71. Innodb_ibuf_discarded_delete_marks
72. Innodb_ibuf_discarded_deletes
73. Innodb_ibuf_discarded_inserts
74. Innodb_ibuf_free_list
75. Innodb_ibuf_merged_delete_marks
76. Innodb_ibuf_merged_deletes
77. Innodb_ibuf_merged_inserts
78. Innodb_ibuf_merges
79. Innodb_ibuf_segment_size
80. Innodb_ibuf_size
81. Innodb_instant_alter_column
82. Innodb_log_waits
83. Innodb_log_write_requests
84. Innodb_log_writes
85. Innodb_lsn_current
86. Innodb_lsn_flushed
87. Innodb_lsn_last_checkpoint
88. Innodb_master_thread_1_second_loops
89. Innodb_master_thread_10_second_loops
90. Innodb_master_thread_active_loops
91. Innodb_master_thread_background_loops
92. Innodb_master_thread_idle_loops
93. Innodb_master_thread_main_flush_loops

94. Innodb_master_thread_sleeps
95. Innodb_max_trx_id
96. Innodb_mem_adaptive_hash
97. Innodb_mem_dictionary
98. Innodb_mem_total
99. Innodb_mutex_os_waits
100. Innodb_mutex_spin_rounds
101. Innodb_mutex_spin_waits
102. Innodb_num_index_pages_written
103. Innodb_num_non_index_pages_written
104. Innodb_num_open_files
105. Innodb_num_page_compressed_trim_op
106. Innodb_num_page_compressed_trim_op_saved
107. Innodb_num_pages_decrypted
108. Innodb_num_pages_encrypted
109. Innodb_num_pages_page_compressed
110. Innodb_num_pages_page_compression_error
111. Innodb_num_pages_page_decompressed
112. Innodb_num_pages_page_encryption_error
113. Innodb_oldest_view_low_limit_trx_id
114. Innodb_onlineddl_pct_progress
115. Innodb_onlineddl_rowlog_pct_used
116. Innodb_onlineddl_rowlog_rows
117. Innodb_os_log_fsyncs
118. Innodb_os_log_pending_fsyncs
119. Innodb_os_log_pending_writes
120. Innodb_os_log_written
121. Innodb_page_compression_saved
122. Innodb_page_compression_trim_sect512
123. Innodb_page_compression_trim_sect1024
124. Innodb_page_compression_trim_sect2048
125. Innodb_page_compression_trim_sect4096
126. Innodb_page_compression_trim_sect8192
127. Innodb_page_compression_trim_sect16384
128. Innodb_page_compression_trim_sect32768
129. Innodb_page_size
130. Innodb_pages_created
131. Innodb_pages_read
132. Innodb_pages0_read
133. Innodb_pages_written
134. Innodb_purge_trx_id
135. Innodb_purge_undo_no
136. Innodb_read_views_memory
137. Innodb_row_lock_current_waits
138. Innodb_row_lock_numbers
139. Innodb_row_lock_time
140. Innodb_row_lock_time_avg
141. Innodb_row_lock_time_max
142. Innodb_row_lock_waits
143. Innodb_rows_deleted
144. Innodb_rows_inserted
145. Innodb_rows_read
146. Innodb_rows_updated
147. Innodb_s_lock_os_waits
148. Innodb_s_lock_spin_rounds
149. Innodb_s_lock_spin_waits
150. Innodb_scrub_background_page_reorganizations
151. Innodb_scrub_background_page_split_failures_missing_index
152. Innodb_scrub_background_page_split_failures_out_of_filespace
153. Innodb_scrub_background_page_split_failures_underflow
154. Innodb_scrub_background_page_split_failures_unknown
155. Innodb_scrub_background_page_splits
156. Innodb_scrub_log
157. Innodb_secondary_index_triggered_cluster_reads
158. Innodb_secondary_index_triggered_cluster_reads_avoided
159. Innodb_system_rows_deleted
160. Innodb_system_rows_inserted

161. Innodb_system_rows_read
162. Innodb_system_rows_updated
163. Innodb_truncated_status_writes
164. Innodb_undo_truncations
165. Innodb_x_lock_os_waits
166. Innodb_x_lock_spin_rounds
167. Innodb_x_lock_spin_waits

See Server Status Variables for a complete list of status variables that can be viewed with SHOW STATUS.

Much of the InnoDB information here can also be seen with a SHOW ENGINE INNODB STATUS statement.

See also the Full list of MariaDB options, system and status variables.

Innodb_adaptive_hash_cells

- **Description:** As shown in the INSERT BUFFER AND ADAPTIVE HASH INDEX section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5, this system variable is present in XtraDB.
  - In MariaDB 10.1 and later, this system variable is not present.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5
- **Removed:** MariaDB 10.0.0

Innodb_adaptive_hash_hash_searches

- **Description:** Hash searches as shown in the INSERT BUFFER AND ADAPTIVE HASH INDEX section of the SHOW ENGINE INNODB STATUS output.
  - Before the variable was introduced in MariaDB 10.5.0, use the `adaptive_hash_searches` counter in the information_schema.INNODB_METRICS table instead.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.5.0

Innodb_adaptive_hash_heap_buffers

- **Description:** As shown in the INSERT BUFFER AND ADAPTIVE HASH INDEX section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5, this system variable is present in XtraDB.
  - In MariaDB 10.1 and later, this system variable is not present.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5
- **Removed:** MariaDB 10.0.0

Innodb_adaptive_hash_non_hash_searches

- **Description:** Non-hash searches as shown in the INSERT BUFFER AND ADAPTIVE HASH INDEX section of the SHOW ENGINE INNODB STATUS output. From MariaDB 10.6.2, not updated if innodb_adaptive_hash_index is not enabled (the default).
  - In MariaDB 10.1, MariaDB 10.2, MariaDB 10.3, and MariaDB 10.4, this system variable is not present. Use the `adaptive_hash_searches_btree` counter in the information_schema.INNODB_METRICS table instead.
  - From MariaDB 10.5, this status variable is present.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5, MariaDB 10.5.0
- **Removed:** MariaDB 10.0.0

Innodb_available_undo_logs

- **Description:** Total number available InnoDB undo logs. Differs from the innodb_undo_logs system variable, which specifies the number of *active* undo logs.

- **Scope:** Global
- **Data Type:** `numeric`

`Innodb_background_log_sync`

- **Description:** As shown in the BACKGROUND THREAD section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5, this system variable is present in XtraDB.
  - In MariaDB 10.1, MariaDB 10.2, MariaDB 10.3, and MariaDB 10.4, this system variable is not present.
  - In MariaDB 10.5, this system variable was reintroduced.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5 (XtraDB only), MariaDB 10.5.0

`Innodb_buffer_pool_bytes_data`

- **Description:** Number of bytes contained in the InnoDB buffer pool, both dirty (modified) and clean (unmodified). See also Innodb_buffer_pool_pages_data, which can contain pages of different sizes in the case of compression.
- **Scope:** Global
- **Data Type:** `numeric`

`Innodb_buffer_pool_bytes_dirty`

- **Description:** Number of dirty (modified) bytes contained in the InnoDB buffer pool. See also Innodb_buffer_pool_pages_dirty, which can contain pages of different sizes in the case of compression.
- **Scope:** Global
- **Data Type:** `numeric`

`Innodb_buffer_pool_dump_status`

- **Description:** A text description of the progress or final status of the last Innodb buffer pool dump.
- **Scope:** Global
- **Data Type:** `string`
- **Introduced:** MariaDB 10.0.0

`Innodb_buffer_pool_load_incomplete`

- **Description:** Whether or not the loaded buffer pool is incomplete, for example after a shutdown or abort during innodb buffer pool load from file caused an incomplete save.
- **Scope:** Global
- **Data Type:** `boolean`
- **Introduced:** MariaDB 10.3.5

`Innodb_buffer_pool_load_status`

- **Description:** A text description of the progress or final status of the last Innodb buffer pool load.
- **Scope:** Global
- **Data Type:** `string`
- **Introduced:** MariaDB 10.0.0

`Innodb_buffer_pool_pages_data`

- **Description:** Number of InnoDB buffer pool pages which contain data, both dirty (modified) and clean (unmodified). See also Innodb_buffer_pool_bytes_data.
- **Scope:** Global
- **Data Type:** `numeric`

`Innodb_buffer_pool_pages_dirty`

- **Description:** Number of InnoDB buffer pool pages which contain dirty (modified) data. See also Innodb_buffer_pool_bytes_dirty.
- **Scope:** Global
- **Data Type:** `numeric`

Innodb_buffer_pool_pages_flushed

- **Description:** Number of InnoDB buffer pool pages which have been flushed.
- **Scope:** Global
- **Data Type:** `numeric`

Innodb_buffer_pool_pages_LRU_flushed

- **Description:** Flush list as shown in the INSERT BUFFER AND ADAPTIVE HASH INDEX section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.
  - In MariaDB 10.2, MariaDB 10.3, and MariaDB 10.4, this system variable is not present.
  - In MariaDB 10.5, this system variable was reintroduced.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5 (XtraDB-only), MariaDB 10.5.0

Innodb_buffer_pool_pages_LRU_freed

- **Description:** Monitor the number of pages that were freed by a buffer pool LRU eviction scan, without flushing.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.6.0

Innodb_buffer_pool_pages_free

- **Description:** Number of free InnoDB buffer pool pages.
- **Scope:** Global
- **Data Type:** `numeric`

Innodb_buffer_pool_pages_made_not_young

- **Description:** Pages not young as shown in the INSERT BUFFER AND ADAPTIVE HASH INDEX section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.
  - In MariaDB 10.2, MariaDB 10.3, and MariaDB 10.4, this system variable is not present.
  - In MariaDB 10.5, this system variable was reintroduced.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5 (XtraDB-only), MariaDB 10.5.0

Innodb_buffer_pool_pages_made_young

- **Description:** Pages made young as shown in the INSERT BUFFER AND ADAPTIVE HASH INDEX section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.
  - In MariaDB 10.2, MariaDB 10.3, and MariaDB 10.4, this system variable is not present.
  - In MariaDB 10.5, this system variable was reintroduced.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5 (XtraDB-only), MariaDB 10.5.0

Innodb_buffer_pool_pages_misc

- **Description:** Number of InnoDB buffer pool pages set aside for internal use.
- **Scope:** Global
- **Data Type:** `numeric`

`Innodb_buffer_pool_pages_old`

- **Description:** Old database page, as shown in the BUFFER POOL AND MEMORY section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5 and MariaDB 10.1, this system variable is present for XtraDB.
  - In MariaDB 10.2, MariaDB 10.3, and MariaDB 10.4, this system variable is not present.
  - In MariaDB 10.5, this system variable was reintroduced.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5 (XtraDB-only), MariaDB 10.5.0

`Innodb_buffer_pool_pages_total`

- **Description:** Total number of InnoDB buffer pool pages.
- **Scope:** Global
- **Data Type:** `numeric`

`Innodb_buffer_pool_read_ahead`

- **Description:** Number of pages read into the InnoDB buffer pool by the read-ahead background thread.
- **Scope:** Global
- **Data Type:** `numeric`

`Innodb_buffer_pool_read_ahead_evicted`

- **Description:** Number of pages read into the InnoDB buffer pool by the read-ahead background thread that were evicted without having been accessed by queries.
- **Scope:** Global
- **Data Type:** `numeric`

`Innodb_buffer_pool_read_ahead_rnd`

- **Description:** Number of random read-aheads.
- **Scope:** Global
- **Data Type:** `numeric`

`Innodb_buffer_pool_read_requests`

- **Description:** Number of requests to read from the InnoDB buffer pool.
- **Scope:** Global
- **Data Type:** `numeric`

`Innodb_buffer_pool_reads`

- **Description:** Number of reads that could not be satisfied by the InnoDB buffer pool and had to be read from disk.
- **Scope:** Global
- **Data Type:** `numeric`

`Innodb_buffer_pool_resize_status`

- **Description:** Progress of the dynamic InnoDB buffer pool resizing operation. See Setting Innodb Buffer Pool Size Dynamically.
- **Scope:** Global

- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.2

`Innodb_buffer_pool_wait_free`

- **Description:** Number of times InnoDB waited for a free page before reading or creating a page. Normally, writes to the InnoDB buffer pool happen in the background. When no clean pages are available, dirty pages are flushed first in order to free some up. This counts the numbers of wait for this operation to finish. If this value is not small, look at increasing innodb_buffer_pool_size.
- **Scope:** Global
- **Data Type:** `numeric`

`Innodb_buffer_pool_write_requests`

- **Description:** Number of requests to write to the InnoDB buffer pool.
- **Scope:** Global
- **Data Type:** `numeric`

`Innodb_buffered_aio_submitted`

- **Description:**
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.5.0

`Innodb_checkpoint_age`

- **Description:** The checkpoint age, as shown in the LOG section of the SHOW ENGINE INNODB STATUS output. (This is equivalent to subtracting "Last checkpoint at" from "Log sequence number".)
  - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.
  - In MariaDB 10.2, MariaDB 10.3, and MariaDB 10.4, this system variable is not present.
  - In MariaDB 10.5, this system variable was reintroduced.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5 (XtraDB-only), MariaDB 10.5.0

`Innodb_checkpoint_max_age`

- **Description:** Max checkpoint age, as shown in the LOG section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.
  - In MariaDB 10.2, MariaDB 10.3, and MariaDB 10.4, this system variable is not present.
  - In MariaDB 10.5, this system variable was reintroduced.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5 (XtraDB-only), MariaDB 10.5.0

`Innodb_checkpoint_target_age`

- **Description:** Checkpoint age target, as shown in the LOG section of the SHOW ENGINE INNODB STATUS output. XtraDB only. Removed in MariaDB 10.0 and replaced with MySQL 5.6's flushing implementation.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5
- **Removed:** MariaDB 10.0

`Innodb_current_row_locks`

- **Description:** Number of current row locks on InnoDB tables as shown in the TRANSACTIONS section of the SHOW

ENGINE INNODB STATUS output. Renamed from InnoDB_row_lock_numbers in XtraDB 5.5.8-20.1.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5

`Innodb_data_fsyncs`

- **Description:** Number of InnoDB fsync (sync-to-disk) calls. fsync call frequency can be influenced by the innodb_flush_method configuration option.
- **Scope:** Global
- **Data Type:** `numeric`

`Innodb_data_pending_fsyncs`

- **Description:** Number of pending InnoDB fsync (sync-to-disk) calls. fsync call frequency can be influenced by the innodb_flush_method configuration option.
- **Scope:** Global
- **Data Type:** `numeric`

`Innodb_data_pending_reads`

- **Description:** Number of pending InnoDB reads.
- **Scope:** Global
- **Data Type:** `numeric`

`Innodb_data_pending_writes`

- **Description:** Number of pending InnoDB writes.
- **Scope:** Global
- **Data Type:** `numeric`

`Innodb_data_read`

- **Description:** Number of InnoDB bytes read since server startup (not to be confused with Innodb_data_reads).
- **Scope:** Global
- **Data Type:** `numeric`

`Innodb_data_reads`

- **Description:** Number of InnoDB read operations (not to be confused with Innodb_data_read).
- **Scope:** Global
- **Data Type:** `numeric`

`Innodb_data_writes`

- **Description:** Number of InnoDB write operations.
- **Scope:** Global
- **Data Type:** `numeric`

`Innodb_data_written`

- **Description:** Number of InnoDB bytes written since server startup. From MariaDB 10.5.1, no longer includes writes to the redo log file `ib_logfile0`, which continue to be counted by Innodb_os_log_written. An error in counting was introduced in MariaDB 10.5.7 until MariaDB 10.5.20, MariaDB 10.6.13, MariaDB 10.8.8, MariaDB 10.9.6, MariaDB 10.10.4 and MariaDB 10.11.3 (MDEV-31124) in which writes via the doublewrite buffer started to be counted incorrectly, without multiplying them by innodb_page_size. A workaround for the error could be the following formulae: real_data_written = Innodb_data_written + (innodb_page_size - 1) * Innodb_dblwr_pages_written

innodb_written = real_data_written + Innodb_os_log_written
- **Scope:** Global
- **Data Type:** `numeric`

`Innodb_dblwr_pages_written`

- **Description:** Number of pages written to the InnoDB doublewrite buffer.
- **Scope:** Global
- **Data Type:** `numeric`

`Innodb_dblwr_writes`

- **Description:** Number of writes to the InnoDB doublewrite buffer.
- **Scope:** Global
- **Data Type:** `numeric`

`Innodb_deadlocks`

- **Description:** Total number of InnoDB deadlocks. Deadlocks occur when at least two transactions are waiting for the other to finish, creating a circular dependency. InnoDB usually detects these quickly, returning an error.
  - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.
  - In MariaDB 10.2, MariaDB 10.3, and MariaDB 10.4, this system variable is not present.
  - In MariaDB 10.5, this system variable was reintroduced.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5 (XtraDB-only), MariaDB 10.5.0

`Innodb_defragment_compression_failures`

- **Description:** Number of defragment re-compression failures. See Defragmenting InnoDB Tablespaces.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.1.1

`Innodb_defragment_count`

- **Description:** Number of defragment operations. See Defragmenting InnoDB Tablespaces.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.1.1

`Innodb_defragment_failures`

- **Description:** Number of defragment failures. See Defragmenting InnoDB Tablespaces.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.1.1

`Innodb_dict_tables`

- **Description:** Number of entries in the XtraDB data dictionary cache. This Percona XtraDB variable was removed in MariaDB 10/XtraDB 5.6 as it was replaced with MySQL 5.6's table_definition_cache implementation.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** XtraDB 5.0.77-b13
- **Removed:** MariaDB 10.0

Innodb_encryption_n_merge_blocks_decrypted

- **Description:**
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.1.28, MariaDB 10.2.9, MariaDB 10.3.2

Innodb_encryption_n_merge_blocks_encrypted

- **Description:**
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.1.28, MariaDB 10.2.9, MariaDB 10.3.2

Innodb_encryption_n_rowlog_blocks_decrypted

- **Description:**
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.1.28, MariaDB 10.2.9, MariaDB 10.3.2

Innodb_encryption_n_rowlog_blocks_encrypted

- **Description:**
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.1.28, MariaDB 10.2.9, MariaDB 10.3.2

Innodb_encryption_n_temp_blocks_decrypted

- **Description:**
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.26, MariaDB 10.3.17, MariaDB 10.4.7

Innodb_encryption_n_temp_blocks_encrypted

- **Description:**
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.26, MariaDB 10.3.17, MariaDB 10.4.7

Innodb_encryption_num_key_requests

- **Description:** Was not present in MariaDB 10.5.2.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.4

Innodb_encryption_rotation_estimated_iops

- **Description:** See Table and Tablespace Encryption.
- **Scope:** Global
- **Data Type:** `numeric`
- **Removed:** MariaDB 10.1.3

Innodb_encryption_rotation_pages_flushed

- **Description:** See Table and Tablespace Encryption.
- **Scope:** Global
- **Data Type:** `numeric`
- **Removed:** MariaDB 10.1.3

`Innodb_encryption_rotation_pages_modified`

- **Description:** See Table and Tablespace Encryption.
- **Scope:** Global
- **Data Type:** `numeric`
- **Removed:** MariaDB 10.1.3

`Innodb_encryption_rotation_pages_read_from_cache`

- **Description:** See Table and Tablespace Encryption.
- **Scope:** Global
- **Data Type:** `numeric`
- **Removed:** MariaDB 10.1.3

`Innodb_encryption_rotation_pages_read_from_disk`

- **Description:** See Table and Tablespace Encryption.
- **Scope:** Global
- **Data Type:** `numeric`
- **Removed:** MariaDB 10.1.3

`Innodb_have_atomic_builtins`

- **Description:** Whether the server has been built with atomic instructions, provided by the CPU ensuring that critical low-level operations can't be interrupted. XtraDB only.
- **Scope:** Global
- **Data Type:** `boolean`

`Innodb_have_bzip2`

- **Description:** Whether the server has the bzip2 compression method available. See InnoDB/XtraDB Page Compression.
- **Scope:** Global
- **Data Type:** `boolean`
- **Introduced:** MariaDB 10.1.0

`Innodb_have_lz4`

- **Description:** Whether the server has the lz4 compression method available. See InnoDB/XtraDB Page Compression.
- **Scope:** Global
- **Data Type:** `boolean`
- **Introduced:** MariaDB 10.1.0

`Innodb_have_lzma`

- **Description:** Whether the server has the lzma compression method available. See InnoDB/XtraDB Page Compression.
- **Scope:** Global
- **Data Type:** `boolean`
- **Introduced:** MariaDB 10.1.0

Innodb_have_lzo

- **Description:** Whether the server has the lzo compression method available. See InnoDB/XtraDB Page Compression.
- **Scope:** Global
- **Data Type:** `boolean`
- **Introduced:** MariaDB 10.1.0

Innodb_have_punch_hole

- **Description:**
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.4

Innodb_have_snappy

- **Description:** Whether the server has the snappy compression method available. See InnoDB/XtraDB Page Compression.
- **Scope:** Global
- **Data Type:** `boolean`
- **Introduced:** MariaDB 10.1.3

Innodb_history_list_length

- **Description:** History list length as shown in the TRANSACTIONS section of the SHOW ENGINE INNODB STATUS output. XtraDB only until introduced in MariaDB 10.5.0.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5 (XtraDB-only), MariaDB 10.5.0

Innodb_ibuf_discarded_delete_marks

- **Description:** As shown in the INSERT BUFFER AND ADAPTIVE HASH INDEX section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.
  - In MariaDB 10.2, MariaDB 10.3, and MariaDB 10.4, this system variable is not present.
  - In MariaDB 10.5, this system variable was reintroduced.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5 (XtraDB-only), MariaDB 10.5.0

Innodb_ibuf_discarded_deletes

- **Description:** As shown in the INSERT BUFFER AND ADAPTIVE HASH INDEX section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.
  - In MariaDB 10.2, MariaDB 10.3, and MariaDB 10.4, this system variable is not present.
  - In MariaDB 10.5, this system variable was reintroduced.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5 (XtraDB-only), MariaDB 10.5.0

Innodb_ibuf_discarded_inserts

- **Description:** As shown in the INSERT BUFFER AND ADAPTIVE HASH INDEX section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.
  - In MariaDB 10.2, MariaDB 10.3, and MariaDB 10.4, this system variable is not present.
  - In MariaDB 10.5, this system variable was reintroduced.

- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5 (XtraDB-only), MariaDB 10.5.0

`Innodb_ibuf_free_list`

- **Description:** As shown in the INSERT BUFFER AND ADAPTIVE HASH INDEX section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.
  - In MariaDB 10.2, MariaDB 10.3, and MariaDB 10.4, this system variable is not present.
  - In MariaDB 10.5, this system variable was reintroduced.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5 (XtraDB-only), MariaDB 10.5.0

`Innodb_ibuf_merged_delete_marks`

- **Description:** As shown in the INSERT BUFFER AND ADAPTIVE HASH INDEX section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.
  - In MariaDB 10.2, MariaDB 10.3, and MariaDB 10.4, this system variable is not present.
  - In MariaDB 10.5, this system variable was reintroduced.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5 (XtraDB-only), MariaDB 10.5.0

`Innodb_ibuf_merged_deletes`

- **Description:** As shown in the INSERT BUFFER AND ADAPTIVE HASH INDEX section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.
  - In MariaDB 10.2, MariaDB 10.3, and MariaDB 10.4, this system variable is not present.
  - In MariaDB 10.5, this system variable was reintroduced.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5 (XtraDB-only), MariaDB 10.5.0

`Innodb_ibuf_merged_inserts`

- **Description:** As shown in the INSERT BUFFER AND ADAPTIVE HASH INDEX section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.
  - In MariaDB 10.2, MariaDB 10.3, and MariaDB 10.4, this system variable is not present.
  - In MariaDB 10.5, this system variable was reintroduced.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5 (XtraDB-only), MariaDB 10.5.0

`Innodb_ibuf_merges`

- **Description:** As shown in the INSERT BUFFER AND ADAPTIVE HASH INDEX section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.
  - In MariaDB 10.2, MariaDB 10.3, and MariaDB 10.4, this system variable is not present.
  - In MariaDB 10.5, this system variable was reintroduced.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5 (XtraDB-only), MariaDB 10.5.0

Innodb_ibuf_segment_size

- **Description:** As shown in the INSERT BUFFER AND ADAPTIVE HASH INDEX section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.
  - In MariaDB 10.2, MariaDB 10.3, and MariaDB 10.4, this system variable is not present.
  - In MariaDB 10.5, this system variable was reintroduced.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5 (XtraDB-only), MariaDB 10.5.0

Innodb_ibuf_size

- **Description:** As shown in the INSERT BUFFER AND ADAPTIVE HASH INDEX section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.
  - In MariaDB 10.2, MariaDB 10.3, and MariaDB 10.4, this system variable is not present.
  - In MariaDB 10.5, this system variable was reintroduced.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5 (XtraDB-only), MariaDB 10.5.0

Innodb_instant_alter_column

- **Description:** See Instant ADD COLUMN for InnoDB.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.3.2

Innodb_log_waits

- **Description:** Number of times InnoDB was forced to wait for log writes to be flushed due to the log buffer being too small.
- **Scope:** Global
- **Data Type:** `numeric`

Innodb_log_write_requests

- **Description:** Number of requests to write to the InnoDB redo log.
- **Scope:** Global
- **Data Type:** `numeric`

Innodb_log_writes

- **Description:** Number of writes to the InnoDB redo log.
- **Scope:** Global
- **Data Type:** `numeric`

Innodb_lsn_current

- **Description:** Log sequence number as shown in the LOG section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.
  - In MariaDB 10.2, MariaDB 10.3, and MariaDB 10.4, this system variable is not present.
  - In MariaDB 10.5, this system variable was reintroduced.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5 (XtraDB-only), MariaDB 10.5.0

Innodb_lsn_flushed

- **Description:** Flushed up to log sequence number as shown in the LOG section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.
  - In MariaDB 10.2, MariaDB 10.3, and MariaDB 10.4, this system variable is not present.
  - In MariaDB 10.5, this system variable was reintroduced.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5 (XtraDB-only), MariaDB 10.5.0

Innodb_lsn_last_checkpoint

- **Description:** Log sequence number last checkpoint as shown in the LOG section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.
  - In MariaDB 10.2, MariaDB 10.3, and MariaDB 10.4, this system variable is not present.
  - In MariaDB 10.5, this system variable was reintroduced.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5 (XtraDB-only), MariaDB 10.5.0

Innodb_master_thread_1_second_loops

- **Description:** As shown in the BACKGROUND THREAD section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5, this system variable is present in XtraDB.
  - In MariaDB 10.1 and later, this system variable is not present.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5
- **Removed:** MariaDB 10.0

Innodb_master_thread_10_second_loops

- **Description:** As shown in the BACKGROUND THREAD section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5, this system variable is present in XtraDB.
  - In MariaDB 10.1 and later, this system variable is not present
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5
- **Removed:** MariaDB 10.0

Innodb_master_thread_active_loops

- **Description:** As shown in the BACKGROUND THREAD section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 10.1, this system variable is present in XtraDB.
  - In MariaDB 10.2, MariaDB 10.3, and MariaDB 10.4, this system variable is not present.
  - In MariaDB 10.5, this system variable was reintroduced.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.0.9 (XtraDB-only), MariaDB 10.5.0:

Innodb_master_thread_background_loops

- **Description:** As shown in the BACKGROUND THREAD section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5, this system variable is present in XtraDB.
  - In MariaDB 10.1 and later, this system variable is not present
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5
- **Removed:** MariaDB 10.0

Innodb_master_thread_idle_loops

- **Description:** As shown in the BACKGROUND THREAD section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 10.1, this system variable is present in XtraDB.
  - In MariaDB 10.2, MariaDB 10.3, and MariaDB 10.4, this system variable is not present.
  - In MariaDB 10.5, this system variable was reintroduced.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.0.9 (XtraDB-only), MariaDB 10.5.0:

Innodb_master_thread_main_flush_loops

- **Description:** As shown in the BACKGROUND THREAD section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5, this system variable is present in XtraDB.
  - In MariaDB 10.1 and later, this system variable is not present
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5
- **Removed:** MariaDB 10.0

Innodb_master_thread_sleeps

- **Description:** As shown in the BACKGROUND THREAD section of the SHOW ENGINE INNODB STATUS output. XtraDB only.
  - In MariaDB 5.5, this system variable is present in XtraDB.
  - In MariaDB 10.1, MariaDB 10.2, MariaDB 10.3, and MariaDB 10.4, this system variable is not present. Use the `innodb_master_thread_sleeps` counter in the `information_schema.INNODB_METRICS` table instead.
  - In MariaDB 10.5, this system variable was reintroduced.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5
- **Removed:** MariaDB 10.0

Innodb_max_trx_id

- **Description:** As shown in the TRANSACTIONS section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.
  - In MariaDB 10.2, MariaDB 10.3, and MariaDB 10.4, this system variable is not present.
  - In MariaDB 10.5, this system variable was reintroduced.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5 (XtraDB-only), MariaDB 10.5.0

Innodb_mem_adaptive_hash

- **Description:** As shown in the BUFFER POOL AND MEMORY section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.
  - In MariaDB 10.2, MariaDB 10.3, and MariaDB 10.4, this system variable is not present.
  - In MariaDB 10.5, this system variable was reintroduced.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5 (XtraDB-only), MariaDB 10.5.0

Innodb_mem_dictionary

- **Description:** As shown in the BUFFER POOL AND MEMORY section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.

    - In MariaDB 10.2, MariaDB 10.3, and MariaDB 10.4, this system variable is not present.
    - In MariaDB 10.5, this system variable was reintroduced.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5 (XtraDB-only), MariaDB 10.5.0

`Innodb_mem_total`

- **Description:** As shown in the BUFFER POOL AND MEMORY section of the SHOW ENGINE INNODB STATUS output.
    - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.
    - In MariaDB 10.2 and later, this system variable is not present.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5

`Innodb_mutex_os_waits`

- **Description:** Mutex OS waits as shown in the SEMAPHORES section of the SHOW ENGINE INNODB STATUS output.
    - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.
    - In MariaDB 10.2 and later, this system variable is not present.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5

`Innodb_mutex_spin_rounds`

- **Description:** Mutex spin rounds as shown in the SEMAPHORES section of the SHOW ENGINE INNODB STATUS output.
    - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.
    - In MariaDB 10.2 and later, this system variable is not present.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5

`Innodb_mutex_spin_waits`

- **Description:** Mutex spin waits as shown in the SEMAPHORES section of the SHOW ENGINE INNODB STATUS output.
    - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.
    - In MariaDB 10.2 and later, this system variable is not present.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5

`Innodb_num_index_pages_written`

- **Description:**
- **Scope:**
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.1.0

`Innodb_num_non_index_pages_written`

- **Description:**
- **Scope:**
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.1.0

Innodb_num_open_files

- **Description:** Number of open files held by InnoDB. InnoDB only.
- **Scope:** Global
- **Data Type:** `numeric`

Innodb_num_page_compressed_trim_op

- **Description:** Number of trim operations performed.
- **Scope:** Global
- **Data Type:** `numeric`

Innodb_num_page_compressed_trim_op_saved

- **Description:** Number of trim operations not done because of an earlier trim.
- **Scope:** Global
- **Data Type:** `numeric`

Innodb_num_pages_decrypted

- **Description:** Number of pages page decrypted. See Table and Tablespace Encryption.
- **Scope:** Global
- **Data Type:** `numeric`

Innodb_num_pages_encrypted

- **Description:** Number of pages page encrypted. See Table and Tablespace Encryption.
- **Scope:** Global
- **Data Type:** `numeric`

Innodb_num_pages_page_compressed

- **Description:** Number of pages that are page compressed.
- **Scope:** Global
- **Data Type:** `numeric`

Innodb_num_pages_page_compression_error

- **Description:** Number of compression errors.
- **Scope:** Global
- **Data Type:** `numeric`

Innodb_num_pages_page_decompressed

- **Description:** Number of pages compressed with page compression that are decompressed.
- **Scope:** Global
- **Data Type:** `numeric`

Innodb_num_pages_page_encryption_error

- **Description:** Number of page encryption errors. See Table and Tablespace Encryption.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.1.3
- **Removed:** MariaDB 10.1.4

Innodb_oldest_view_low_limit_trx_id

- **Description:** As shown in the TRANSACTIONS section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.
  - In MariaDB 10.2 and later, this system variable is not present.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5
- **Removed:** MariaDB 10.2

Innodb_onlineddl_pct_progress

- **Description:** Shows the progress of in-place alter table. It might be not so accurate because in-place alter is highly dependent on disk and buffer pool status. See Monitoring progress and temporal memory usage of Online DDL in InnoDB.
- **Scope:** Global
- **Data Type:** `numeric`

Innodb_onlineddl_rowlog_pct_used

- **Description:** Shows row log buffer usage in 5-digit integer (10000 means 100.00%). See Monitoring progress and temporal memory usage of Online DDL in InnoDB.
- **Scope:** Global
- **Data Type:** `numeric`

Innodb_onlineddl_rowlog_rows

- **Description:** Number of rows stored in the row log buffer. See Monitoring progress and temporal memory usage of Online DDL in InnoDB.
- **Scope:** Global
- **Data Type:** `numeric`

Innodb_os_log_fsyncs

- **Description:** Number of InnoDB log fsync (sync-to-disk) requests.
- **Scope:** Global
- **Data Type:** `numeric`
- **Removed:** MariaDB 10.8

Innodb_os_log_pending_fsyncs

- **Description:** Number of pending InnoDB log fsync (sync-to-disk) requests.
- **Scope:** Global
- **Data Type:** `numeric`
- **Removed:** MariaDB 10.8

Innodb_os_log_pending_writes

- **Description:** Number of pending InnoDB log writes.
- **Scope:** Global
- **Data Type:** `numeric`
- **Removed:** MariaDB 10.8

Innodb_os_log_written

- **Description:** Number of bytes written to the InnoDB log.
- **Scope:** Global
- **Data Type:** `numeric`

Innodb_page_compression_saved

- **Description:** Number of bytes saved by page compression.
- **Scope:**
- **Data Type:**

Innodb_page_compression_trim_sect512

- **Description:** Number of TRIM operations performed for the page-compression/NVM Compression workload for the 512 byte block-size.
- **Scope:**
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.1.0, MariaDB 10.0.15 Fusion-io
- **Removed:** MariaDB 10.2.4

Innodb_page_compression_trim_sect1024

- **Description:** Number of TRIM operations performed for the page-compression/NVM Compression workload for the 1K block-size.
- **Scope:**
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.1.2, MariaDB 10.0.15 Fusion-io
- **Removed:** MariaDB 10.2.4

Innodb_page_compression_trim_sect2048

- **Description:** Number of TRIM operations performed for the page-compression/NVM Compression workload for the 2K block-size.
- **Scope:**
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.1.2, MariaDB 10.0.15 Fusion-io
- **Removed:** MariaDB 10.2.4

Innodb_page_compression_trim_sect4096

- **Description:** Number of TRIM operations performed for the page-compression/NVM Compression workload for the 4K block-size.
- **Scope:**
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.1.0, MariaDB 10.0.15 Fusion-io
- **Removed:** MariaDB 10.2.4

Innodb_page_compression_trim_sect8192

- **Description:** Number of TRIM operations performed for the page-compression/NVM Compression workload for the 8K block-size.
- **Scope:**
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.1.2, MariaDB 10.0.15 Fusion-io
- **Removed:** MariaDB 10.2.4

Innodb_page_compression_trim_sect16384

- **Description:** Number of TRIM operations performed for the page-compression/NVM Compression workload for the 16K block-size.
- **Scope:**
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.1.2, MariaDB 10.0.15 Fusion-io

- **Removed:** MariaDB 10.2.4

Innodb_page_compression_trim_sect32768

- **Description:** Number of TRIM operations performed for the page-compression/NVM Compression workload for the 32K block-size.
- **Scope:**
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.1.2, MariaDB 10.0.15 Fusion-io
- **Removed:** MariaDB 10.2.4

Innodb_page_size

- **Description:** Page size used by InnoDB. Defaults to 16KB, can be compiled with a different value.
- **Scope:** Global
- **Data Type:** `numeric`

Innodb_pages_created

- **Description:** Number of InnoDB pages created.
- **Scope:** Global
- **Data Type:** `numeric`

Innodb_pages_read

- **Description:** Number of InnoDB pages read.
- **Scope:** Global
- **Data Type:** `numeric`

Innodb_pages0_read

- **Description:** Counter for keeping track of reads of the first page of InnoDB data files, because the original implementation of data-at-rest-encryption for InnoDB introduced new code paths for reading the pages. Removed in MariaDB 10.4.0 as the extra reads of the first page were removed, and the encryption subsystem will be initialized whenever we first read the first page of each data file, in fil_node_open_file().
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.4, MariaDB 10.1.21
- **Removed:** MariaDB 10.4.0

Innodb_pages_written

- **Description:** Number of InnoDB pages written.
- **Scope:** Global
- **Data Type:** `numeric`

Innodb_purge_trx_id

- **Description:** Purge transaction id as shown in the TRANSACTIONS section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.
  - In MariaDB 10.2 and later, this system variable is not present.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5
- **Removed:** MariaDB 10.2

Innodb_purge_undo_no

- **Description:** As shown in the TRANSACTIONS section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.
  - In MariaDB 10.2 and later, this system variable is not present.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5
- **Removed:** MariaDB 10.2

Innodb_read_views_memory

- **Description:** As shown in the BUFFER POOL AND MEMORY section of the SHOW ENGINE INNODB STATUS output. Shows the total of memory in bytes allocated for the InnoDB read view.
  - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.
  - In MariaDB 10.2 and later, this system variable is not present.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5.32
- **Removed:** MariaDB 10.2

Innodb_row_lock_current_waits

- **Description:** Number of pending row lock waits on InnoDB tables.
- **Scope:** Global
- **Data Type:** `numeric`

Innodb_row_lock_numbers

- **Description:** Number of current row locks on InnoDB tables as shown in the TRANSACTIONS section of the SHOW ENGINE INNODB STATUS output. Renamed to InnoDB_current_row_locks in XtraDB 5.5.10-20.1.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5 / XtraDB 5.5.8-20
- **Removed:** MariaDB 5.5 / XtraDB 5.5.10-20.1

Innodb_row_lock_time

- **Description:** Total time in milliseconds spent getting InnoDB row locks.
- **Scope:** Global
- **Data Type:** `numeric`

Innodb_row_lock_time_avg

- **Description:** Average time in milliseconds spent getting an InnoDB row lock.
- **Scope:** Global
- **Data Type:** `numeric`

Innodb_row_lock_time_max

- **Description:** Maximum time in milliseconds spent getting an InnoDB row lock.
- **Scope:** Global
- **Data Type:** `numeric`

Innodb_row_lock_waits

- **Description:** Number of times InnoDB had to wait before getting a row lock.
- **Scope:** Global

- **Data Type:** `numeric`

`Innodb_rows_deleted`

- **Description:** Number of rows deleted from InnoDB tables that where not system tables. Almost equivalent to Handler_delete which does include system tables.
- **Scope:** Global
- **Data Type:** `numeric`
- **Removed:** MariaDB 10.10

`Innodb_rows_inserted`

- **Description:** Number of rows inserted into InnoDB tables that where not system tables. No direct equivalent in Handler status variables.
- **Scope:** Global
- **Data Type:** `numeric`
- **Removed:** MariaDB 10.10

`Innodb_rows_read`

- **Description:** Number of rows read from InnoDB tables that where not system tables. Almost equivalent to the sum of Handler_read* status variables which do include system tables.
- **Scope:** Global
- **Data Type:** `numeric`
- **Removed:** MariaDB 10.10

`Innodb_rows_updated`

- **Description:** Number of rows updated in InnoDB tables that where not system tables. Almost equivalent to Handler_update which does include system tables.
- **Scope:** Global
- **Data Type:** `numeric`
- **Removed:** MariaDB 10.10

`Innodb_s_lock_os_waits`

- **Description:** As shown in the SEMAPHORES section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.
  - In MariaDB 10.2 and later, this system variable is not present.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5
- **Removed:** MariaDB 10.2

`Innodb_s_lock_spin_rounds`

- **Description:** As shown in the SEMAPHORES section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.
  - In MariaDB 10.2 and later, this system variable is not present.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5
- **Removed:** MariaDB 10.2

`Innodb_s_lock_spin_waits`

- **Description:** As shown in the SEMAPHORES section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.

    - In MariaDB 10.2 and later, this system variable is not present.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5
- **Removed:** MariaDB 10.2

`Innodb_scrub_background_page_reorganizations`

- **Description:** See Table and Tablespace Encryption.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.1.3
- **Removed:** MariaDB 10.5.2

`Innodb_scrub_background_page_split_failures_missing_index`

- **Description:** See Table and Tablespace Encryption.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.1.3
- **Removed:** MariaDB 10.5.2

`Innodb_scrub_background_page_split_failures_out_of_filespace`

- **Description:** See Table and Tablespace Encryption.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.1.3
- **Removed:** MariaDB 10.5.2

`Innodb_scrub_background_page_split_failures_underflow`

- **Description:** See Table and Tablespace Encryption.
- **Scope:** Global
- **Data Type:** `numeric`
- **Removed:** MariaDB 10.5.2

`Innodb_scrub_background_page_split_failures_unknown`

- **Description:** See Table and Tablespace Encryption.
- **Scope:** Global
- **Data Type:** `numeric`
- **Removed:** MariaDB 10.5.2

`Innodb_scrub_background_page_splits`

- **Description:** See Table and Tablespace Encryption.
- **Scope:** Global
- **Data Type:** `numeric`
- **Removed:** MariaDB 10.5.2

`Innodb_scrub_log`

- **Description:**
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.4
- **Removed:** MariaDB 10.5.2

Innodb_secondary_index_triggered_cluster_reads

- **Description:** Used to track the effectiveness of the Prefix Index Queries Optimization (MDEV-6929). Removed in MariaDB 10.10 as the optimization is now always enabled.
- **Scope:** Global
- **Data Type:** `numeric`
- **Removed:** MariaDB 10.10

Innodb_secondary_index_triggered_cluster_reads_avoided

- **Description:** Used to track the effectiveness of the Prefix Index Queries Optimization (MDEV-6929). Removed in MariaDB 10.10 as the optimization is now always enabled.
- **Scope:** Global
- **Data Type:** `numeric`
- **Removed:** MariaDB 10.10

Innodb_system_rows_deleted

- **Description:** Number of rows deleted on system tables.
- **Scope:** Global
- **Data Type:** `numeric`
- **Removed:** MariaDB 10.10

Innodb_system_rows_inserted

- **Description:** Number of rows inserted on system tables.
- **Scope:** Global
- **Data Type:** `numeric`
- **Removed:** MariaDB 10.10

Innodb_system_rows_read

- **Description:** Number of rows read on system tables.
- **Scope:** Global
- **Data Type:** `numeric`
- **Removed:** MariaDB 10.10.0

Innodb_system_rows_updated

- **Description:** Number of rows updated on system tables.
- **Scope:** Global
- **Data Type:** `numeric`
- **Removed:** MariaDB 10.10.0

Innodb_truncated_status_writes

- **Description:** Number of times output from SHOW ENGINE INNODB STATUS has been truncated.
- **Scope:** Global
- **Data Type:** `numeric`

Innodb_undo_truncations

- **Description:** Number of undo tablespace truncation operations.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.3.10

`Innodb_x_lock_os_waits`

- **Description:** As shown in the SEMAPHORES section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.
  - In MariaDB 10.2 and later, this system variable is not present.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5
- **Removed:** MariaDB 10.2

`Innodb_x_lock_spin_rounds`

- **Description:** As shown in the SEMAPHORES section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.
  - In MariaDB 10.2 and later, this system variable is not present.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5
- **Removed:** MariaDB 10.2

`Innodb_x_lock_spin_waits`

- **Description:** As shown in the SEMAPHORES section of the SHOW ENGINE INNODB STATUS output.
  - In MariaDB 5.5 and MariaDB 10.1, this system variable is present in XtraDB.
  - In MariaDB 10.2 and later, this system variable is not present.
- **Scope:** Global
- **Data Type:** `numeric`
- **Introduced:** MariaDB 5.5
- **Removed:** MariaDB 10.2

5.3.2.6 AUTO_INCREMENT Handling in InnoDB

Contents

AUTO_INCREMENT Lock Modes

The innodb_autoinc_lock_mode system variable determines the lock mode when generating AUTO_INCREMENT values for InnoDB tables. These modes allow InnoDB to make significant performance optimizations in certain circumstances.

The innodb_autoinc_lock_mode system variable may be removed in a future release. See MDEV-19577 for more information.

Traditional Lock Mode

When innodb_autoinc_lock_mode is set to `0`, InnoDB uses the traditional lock mode.

In this mode, InnoDB holds a table-level lock for all INSERT statements until the statement completes.

Consecutive Lock Mode

When innodb_autoinc_lock_mode is set to `1`, InnoDB uses the consecutive lock mode.

In this mode, InnoDB holds a table-level lock for all bulk INSERT statements (such as LOAD DATA or INSERT ... SELECT) until the end of the statement. For simple INSERT statements, no table-level lock is held. Instead, a lightweight mutex is used which scales significantly better. This is the default setting.

Interleaved Lock Mode

When innodb_autoinc_lock_mode is set to `2`, InnoDB uses the interleaved lock mode.

In this mode, InnoDB does not hold any table-level locks at all. This is the fastest and most scalable mode, but is not safe for statement-based replication.

Setting AUTO_INCREMENT Values

The AUTO_INCREMENT value for an InnoDB table can be set for a table by executing the ALTER TABLE statement and specifying the AUTO_INCREMENT table option. For example:

```
ALTER TABLE tab AUTO_INCREMENT=100;
```

However, in MariaDB 10.2.3 and before, InnoDB stores the table's AUTO_INCREMENT counter in memory. In these versions, when the server restarts, the counter is re-initialized to the highest value found in the table. This means that the above operation can be undone if the server is restarted before any rows are written to the table.

In MariaDB 10.2.4 and later, the AUTO_INCREMENT counter is persistent, so this restriction is no longer present. Persistent, however, does not mean transactional. Gaps may still occur in some cases, such as if a INSERT IGNORE statement fails, or if a user executes ROLLBACK or ROLLBACK TO SAVEPOINT.

For example:

```
CREATE TABLE t1 (pk INT AUTO_INCREMENT PRIMARY KEY, i INT, UNIQUE (i)) ENGINE=InnoDB;

INSERT INTO t1 (i) VALUES (1),(2),(3);
INSERT IGNORE INTO t1 (pk, i) VALUES (100,1);
Query OK, 0 rows affected, 1 warning (0.099 sec)

SELECT * FROM t1;
+----+------+
| pk | i    |
+----+------+
|  1 |    1 |
|  2 |    2 |
|  3 |    3 |
+----+------+

SHOW CREATE TABLE t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE `t1` (
  `pk` int(11) NOT NULL AUTO_INCREMENT,
  `i` int(11) DEFAULT NULL,
  PRIMARY KEY (`pk`),
  UNIQUE KEY `i` (`i`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=latin1
```

If the server is restarted at this point, then the AUTO_INCREMENT counter will revert to `101`, which is the persistent value set as part of the failed INSERT IGNORE.

```
# Restart server
SHOW CREATE TABLE t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE `t1` (
  `pk` int(11) NOT NULL AUTO_INCREMENT,
  `i` int(11) DEFAULT NULL,
  PRIMARY KEY (`pk`),
  UNIQUE KEY `i` (`i`)
) ENGINE=InnoDB AUTO_INCREMENT=101 DEFAULT CHARSET=latin1
```

5.3.2.7 InnoDB Buffer Pool

Contents

The InnoDB buffer pool is a key component for optimizing MariaDB. It stores data and indexes, and you usually want it as large as possible so as to keep as much of the data and indexes in memory, reducing disk IO, as main bottleneck.

How the Buffer Pool Works

The buffer pool attempts to keep frequently-used blocks in the buffer, and so essentially works as two sublists, a *new* sublist of recently-used information, and an *old* sublist of older information. By default, 37% of the list is reserved for the old list.

When new information is accessed that doesn't appear in the list, it is placed at the top of the old list, the oldest item in the old list is removed, and everything else bumps back one position in the list.

When information is accessed that appears in the *old* list, it is moved to the top the new list, and everything above moves back one position.

innodb_buffer_pool_size

The most important server system variable is innodb_buffer_pool_size. This size should contain most of the active data set of your server so that SQL request can work directly with information in the buffer pool cache. Starting at several gigabytes of memory is a good starting point if you have that RAM available. Once warmed up to its normal load there should be very few innodb_buffer_pool_reads compared to innodb_buffer_pool_read_requests. Look how these values change over a minute. If the change in innodb_buffer_pool_reads is less than 1% of the change in innodb_buffer_pool_read_requests then you have a good amount of usage. If you are getting the status variable innodb_buffer_pool_wait_free increasing then you don't have enough buffer pool (or your flushing isn't occurring frequently enough).

Be aware that before MariaDB 10.4.4 the total memory allocated is about 10% more than the specified size as extra space is also reserved for control structures and buffers.

The larger the size, the longer it will take to initialize. On a modern 64-bit server with a 10GB memory pool, this can take five seconds or more. Increasing innodb_buffer_pool_chunk_size by several factors will reduce this significantly. MariaDB 10.6 could start up with a 96GB buffer pool in less than 1 second.

Make sure that the size is not too large, causing swapping. The benefit of a larger buffer pool size is more than undone if your operating system is regularly swapping.

Since MariaDB 10.2.2, the buffer pool can be set dynamically, and new variables are introduced that may affect the size and performance. See Setting Innodb Buffer Pool Size Dynamically.

innodb_buffer_pool_instances

The functionality described below was disabled in MariaDB 10.5, and removed in MariaDB 10.6, as the original reasons for for splitting the buffer pool have mostly gone away.

If innodb_buffer_pool_size is set to more than 1GB, innodb_buffer_pool_instances divides the InnoDB buffer pool into a specific number of instances. The default was 1 in MariaDB 5.5, but for large systems with buffer pools of many gigabytes, many instances can help reduce contention concurrency. The default is 8 in MariaDB 10.0, with the exception of 32-bit Windows, where it depends on the value of innodb_buffer_pool_size. Each instance manages its own data structures and takes an equal portion of the total buffer pool size, so for example if innodb_buffer_pool_size is 4GB and innodb_buffer_pool_instances is set to 4, each instance will be 1GB. Each instance should ideally be at least 1GB in size.

innodb_old_blocks_pct and innodb_old_blocks_time

The default 37% reserved for the old list can be adjusted by changing the value of innodb_old_blocks_pct. It can accept anything between between 5% and 95%.

The innodb_old_blocks_time variable specifies the delay before a block can be moved from the old to the new sublist. `0` means no delay, while the default has been set to `1000`.

Before changing either of these values from their defaults, make sure you understand the impact and how your system currently uses the buffer. Their main reason for existence is to reduce the impact of full table scans, which are usually infrequent, but large, and previously could clear everything from the buffer. Setting a non-zero delay could help in situations where full table scans are performed in quick succession.

Temporarily changing these values can also be useful to avoid the negative impact of a full table scan, as explained in InnoDB logical backups.

Dumping and Restoring the Buffer Pool

When the server starts, the buffer pool is empty. As it starts to access data, the buffer pool will slowly be populated. As more data will be accessed, the most frequently accessed data will be put into the buffer pool, and old data may be evicted. This means that a certain period of time is necessary before the buffer pool is really useful. This period of time is called the warmup.

Since MariaDB 10.0, InnoDB can dump the buffer pool before the server shuts down, and restore it when it starts again. If this feature is used (default since MariaDB 10.2), no warmup is necessary. Use the innodb_buffer_pool_dump_at_shutdown and innodb_buffer_pool_load_at_startup system variables to enable or disable the buffer pool dump at shutdown and the restore at startup respectively.

It is also possible to dump the InnoDB buffer pool at any moment while the server is running, and it is possible to restore the last buffer pool dump at any moment. To do this, the special innodb_buffer_pool_dump_now and innodb_buffer_pool_load_now system variables can be set to ON. When selected, their value is always OFF.

A buffer pool restore, both at startup or at any other moment, can be aborted by setting innodb_buffer_pool_load_abort to ON.

The file which contains the buffer pool dump is specified via the innodb_buffer_pool_filename system variable.

5.3.2.8 InnoDB Change Buffering

The change buffer has been disabled by default from MariaDB 10.5.15, MariaDB 10.6.7, MariaDB 10.7.3 and MariaDB 10.8.2 (MDEV-27734), was deprecated and ignored from MariaDB 10.9.0 (MDEV-27735), and was removed in MariaDB 11.0.0 (MDEV-29694).

Benchmarks attached to MDEV-19514 show that the change buffer sometimes reduces performance, and in the best case seem to bring a few per cent improvement to throughput. However, such improvement could come with a price: If the buffered changes are never merged (MDEV-19514, motivated by the reduction of random crashes and the removal of an innodb_force_recovery option that can inflict further corruption), then the InnoDB system tablespace can grow out of control (MDEV-21952).

Contents

INSERT, UPDATE and DELETE statements can be particularly heavy operations to perform, as all indexes need to be updated after each change. For this reason these changes are often buffered.

Pages are modified in the buffer pool, and not immediately on disk. After all the records that cover the changes to a data page have been written to the InnoDB redo log, the changed page may be written ("flushed") to a data file. Pages that have been modified in memory and not yet flushed are called dirty pages.

The Change Buffer is an optimization that allows some data to be modified even though the data page does not exist in the buffer pool. Instead of modifying the data in its final destination, we would insert a record into a special Change Buffer that resides in the system tablespace. When the page is read into the buffer pool for any reason, the buffered changes will be applied to it.

The Change Buffer only contains changes to secondary index leaf pages.

Before MariaDB 5.5, only inserted rows could be buffered, so this buffer was called Insert Buffer. The old name still appears in several places, for example in the output of SHOW ENGINE INNODB STATUS.

Inserts to UNIQUE secondary indexes cannot be buffered unless unique_checks=0 is used. This may sometimes allow duplicates to be inserted into the UNIQUE secondary index. Much of the time, the UNIQUE constraint would be checked because the change buffer could only be used if the index page is not located in the buffer pool.

When rows are deleted, a flag is set, thus rows are not immediately deleted. Delete-marked records may be purged after the transaction has been committed and any read views that were created before the commit have been closed. Delete-mark and purge buffering of any secondary indexes is allowed.

ROLLBACK never makes use of the change buffer; it would force a merge of any changes that were buffered during the execution of the transaction.

The Change Buffer is an optimization because:

- Some random-access page reads will be transformed into modifications of change buffer pages.
- A change buffer page can be modified several times in memory and be flushed to disk only once.
- Dirty pages are flushed together, so the number of IO operations is lower.

If the server crashes or is shut down, the Change Buffer might not be empty. The Change Buffer resides in the InnoDB system tablespace, covered by the write-ahead log, so they can be applied at server restart. A shutdown with innodb_fast_shutdown=0 will merge all buffered changes.

Starting with MariaDB 10.5, there no longer is a background task that would merge the change buffer to the secondary index pages. The changes would only be merged on demand.

The Change Buffer was removed in MariaDB 11.0 because it has been a prominent source of corruption bugs that have been extremely hard to reproduce.

The main server system variable here is innodb_change_buffering, which determines which form of change buffering, if any, to use.

The following settings are available:

- inserts
  - Only buffer insert operations
- deletes
  - Only buffer delete operations
- changes
  - Buffer both insert and delete operations
- purges
  - Buffer the actual physical deletes that occur in the background
- all
  - Buffer inserts, deletes and purges. Default setting from MariaDB 5.5 until MariaDB 10.5.14, MariaDB 10.6.6, MariaDB 10.7.2 and MariaDB 10.8.1.
- none
  - Don't buffer any operations. Default from MariaDB 10.5.15, MariaDB 10.6.7, MariaDB 10.7.3 and MariaDB 10.8.2.

Modifying the value of this variable only affects the buffering of new operations. The merging of already buffered changes is not affected.

The innodb_change_buffer_max_size system variable determines the maximum size of the change buffer, expressed as a percentage of the buffer pool.

5.3.2.9 InnoDB Doublewrite Buffer

The InnoDB doublewrite buffer was implemented to recover from half-written pages. This can happen when there's a power failure while InnoDB is writing a page to disk. On reading that page, InnoDB can discover the corruption from the mismatch of the page checksum. However, in order to recover, an intact copy of the page would be needed.

The double write buffer provides such a copy.

Whenever InnoDB flushes a page to disk, it is first written to the double write buffer. Only when the buffer is safely flushed to disk will InnoDB write the page to the final destination. When recovering, InnoDB scans the double write buffer and for each valid page in the buffer checks if the page in the data file is valid too.

Doublewrite Buffer Settings

To turn off the doublewrite buffer, set the innodb_doublewrite system variable to `0`. This is safe on filesystems that write pages atomically - that is, a page write fully succeeds or fails. But with other filesystems, it is not recommended for production systems. An alternative option is atomic writes. See atomic write support for more details.

5.3.2.10 InnoDB Tablespaces

Tables that use the InnoDB storage engine are written to disk in data files called tablespaces. An individual tablespace can contain data from one or more InnoDB tables as well as the associated indexes.

InnoDB System Tablespaces

The system tablespace, how to change its size, and the use of raw disk partitions.

InnoDB File-Per-Table Tablespaces

InnoDB file-per-table tablespaces: what they are, where they're located, ho...

InnoDB Temporary Tablespaces

Information on tablespaces for user-created temporary tables.

There are 1 related questions.

5.3.2.10.1 InnoDB System Tablespaces

Contents

When InnoDB needs to store general information relating to the system as a whole, rather than a specific table, the specific file it writes to is the system tablespace. By default, this is the `ibdata1` file located in the data directory, (as defined by either the `datadir` or `innodb_data_home_dir` system variables). InnoDB uses the system tablespace to store the data dictionary, change buffer, and undo logs.

You can define the system tablespace filename or filenames, size and other options by setting the `innodb_data_file_path` system variable. This system variable can be specified as a command-line argument to `mysqld` or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
innodb_data_file_path=ibdata1:50M:autoextend
```

This system variable defaults to the file `ibdata1`, and it defaults to a minimum size of `12M`, and it defaults with the `autoextend` attribute enabled.

Changing Sizes

InnoDB defaults to allocating 12M to the `ibdata1` file for the system tablespace. While this is sufficient for most use cases, it may not be for all. You may find after using MariaDB for a while that the allocation is too small for the system tablespace or it grows too large for your disk. Fortunately, you can adjust this size as need later.

Increasing the Size

When setting the `innodb_data_file_path` system variable, you can define a size for each file given. In cases where you need a larger system tablespace, add the `autoextend` option to the last value.

```
[mariadb]
...
innodb_data_file_path=ibdata1:12M;ibdata2:50M:autoextend
```

Under this configuration, when the last system tablespace grows beyond the size allocation, InnoDB increases the size of the file by increments. To control the allocation increment, set the `innodb_autoextend_increment` system variable.

Decreasing the Size

MariaDB starting with 11.2.0

From MariaDB 11.2.0, when MariaDB starts up, unused InnoDB tablespace is reclaimed, reducing the file size (MDEV-14795).

Technically, how this works is:

1. Find the last used extent in the system tablespace by iterating through the BITMAP in the extent descriptor page.
2. Check whether the tablespace is being used within fixed size, and if the last used extent is less than the fixed size, then set the desired target size to fixed size.
3. Flush all pages belonging to the system tablespace in flush list.
4. Truncate the truncated pages from FSP_FREE and FSP_FREE_FRAG list.
5. Reset the bitmap in descriptor pages for the truncated pages.
6. Update the FSP_SIZE and FSP_FREE_LIMIT in header page.
7. In case of multiple files, calculate the truncated last file size and do the truncation in last file.

MariaDB until 11.2.0

In cases where the InnoDB system tablespace has grown too large, before MariaDB 11.2, the process to reduce it in size is a little more complicated than increasing the size. MariaDB does not allow you to remove data from the tablespace file itself. Instead you need to delete the tablespace files themselves, then restore the database from backups.

The backup utility mariadb-dump produces backup files containing the SQL statements needed to recreate the database. As a result, it restores a database with the bare minimum data rather than any additional information that might have built up in the tablespace file.

Use mariadb-dump to backup all of your InnoDB database tables, including the system tables in the `mysql` database that use InnoDB. You can find out what they are using the Information Schema.

```
SELECT TABLE_NAME FROM information_schema.TABLES
WHERE TABLE_SCHEMA = 'mysql' AND ENGINE = 'InnoDB';
```

If you only use InnoDB, you may find it easier to back up all databases and tables.

```
$ mariadb-dump -u root -p --all-databases > full-backup.sql
```

Then stop the MariaDB Server and remove the InnoDB tablespace files. In the data directory or the InnoDB data home directory, delete all the `ibdata` and `ib_log` files as well as any file with an `.ibd` or `.frm` extension.

Once this is done, restart the server and import the dump file:

```
$ mysql -u root -p < full-backup.sql
```

Using Raw Disk Partitions

Instead of having InnoDB write to the file system, you can set it to use raw disk partitions. On Windows and some Linux distributions, this allows you to perform non-buffered I/O without the file system overhead. Note that in many use cases this may not actually improve performance. Run tests to verify if there are any real gains for your application usage.

To enable a raw disk partition, first start MariaDB with the `newraw` option set on the tablespace. For example:

```
[mariadb]
...
innodb_data_file_path=/dev/sdc:10Gnewraw
```

When the MariaDB Server starts, it initializes the partition. Don't create or change any data, (any data written to InnoDB at this stage will be lost on restart). Once the server has successful started, stop it then edit the configuration file again, changing the `newraw` keyword to `raw`.

```
[mariadb]
...
innodb_data_file_path=/dev/sdc:10Graw
```

When you start MariaDB again, it'll read and write InnoDB data to the given disk partition instead of the file system.

Raw Disk Partitions on Windows

When defining a raw disk partition for InnoDB on the Windows operating system, use the same procedure as defined above, but when defining the path for the `innodb_data_file_path` system variable, use `./` at the start. For example:

```
[mariadb]
...
innodb_data_file_path=//./E::10Graw
```

The given path is synonymous with the Windows syntax for accessing the physical drive.

System Tables within the InnoDB System Tablespace

InnoDB creates some system tables within the InnoDB System Tablespace:

- `SYS_DATAFILES`

- SYS_FOREIGN
- SYS_FOREIGN_COLS
- SYS_TABLESPACES
- SYS_VIRTUAL
- SYS_ZIP_DICT
- SYS_ZIP_DICT_COLS

These tables cannot be queried. However, you might see references to them in some places, such as in the INNODB_SYS_TABLES table in the information_schema database.

5.3.2.10.2 InnoDB File-Per-Table Tablespaces

Contents

When you create a table using the InnoDB storage engine, data written to that table is stored on the file system in a data file called a tablespace. Tablespace files contain both the data and indexes.

When innodb_file_per_table=ON is set, InnoDB uses one tablespace file per InnoDB table. These tablespace files have the `.ibd` extension. When innodb_file_per_table=OFF is set, InnoDB stores all tables in the InnoDB system tablespace.

InnoDB versions in MySQL 5.7 and above also support an additional type of tablespace called general tablespaces that are created with CREATE TABLESPACE. However, InnoDB versions in MariaDB Server do not support general tablespaces or CREATE TABLESPACE.

File-Per-Table Tablespace Locations

By default, InnoDB's file-per-table tablespaces are created in the system's data directory, which is defined by the datadir system variable. The system variable innodb_data_home_dir will not change the location of file-per-table tablespaces.

In the event that you have a specific tablespace that you need stored in a dedicated path, you can set the location using the DATA DIRECTORY table option when you create the table.

For instance,

```
CREATE TABLE test.t1 (
   id INT PRIMARY KEY AUTO_INCREMENT,
   name VARCHAR(50)
) ENGINE=InnoDB
DATA DIRECTORY = "/data/contact";
```

MariaDB then creates a database directory on the configured path and the file-per-table tablespace will be created inside that directory. On Unix-like operating systems, you can see the file using the ls command:

```
# ls -al /data/contact/test
drwxrwx--- 2 mysql mysql  4096 Dec 8 18:46 .
drwxr-xr-x 3 mysql mysql  4096 Dec 8 18:46 ..
-rw-rw---- 1 mysql mysql 98304 Dec 8 20:41 t1.ibd
```

Note, the system user that runs the MariaDB Server process (which is usually `mysql`) must have write permissions on the given path.

Copying Transportable Tablespaces

InnoDB's file-per-table tablespaces are transportable, which means that you can copy a file-per-table tablespace from one MariaDB Server to another server. You may find this useful in cases where you need to transport full tables between servers and don't want to use backup tools like mariabackup or mariadb-dump. In fact, this process can even be used with mariabackup in some cases, such as when restoring partial backups or when restoring individual tables or partitions from a backup.

Copying Transportable Tablespaces for Non-partitioned Tables

You can copy the transportable tablespace of a non-partitioned table from one server to another by exporting the tablespace file from the original server, and then importing the tablespace file into the new server.

Exporting Transportable Tablespaces for Non-partitioned Tables

You can export a non-partitioned table by locking the table and copying the table's `.ibd` and `.cfg` files from the relevant tablespace location for the table to a backup location. For example, the process would go like this:

- First, use the FLUSH TABLES ... FOR EXPORT statement on the target table:

```
FLUSH TABLES test.t1 FOR EXPORT;
```

This forces the server to close the table and provides your connection with a read lock on the table.

- Then, while your connection still holds the lock on the table, copy the tablespace file and the metadata file to a safe directory:

```
# cp /data/contacts/test/t1.ibd /data/saved-tablespaces/
# cp /data/contacts/test/t1.cfg /data/saved-tablespaces/
```

- Then, once you've copied the files, you can release the lock with UNLOCK TABLES:

```
UNLOCK TABLES;
```

Importing Transportable Tablespaces for Non-partitioned Tables

You can import a non-partitioned table by discarding the table's original tablespace, copying the table's `.ibd` and `.cfg` files from the backup location to the relevant tablespace location for the table, and then telling the server to import the tablespace. For example, the process would go like this:

- First, on the destination server, you need to create a copy of the table. Use the same CREATE TABLE statement that was used to create the table on the original server:

```
CREATE TABLE test.t1 (
   id INT PRIMARY KEY AUTO_INCREMENT,
   name VARCHAR(50)
) ENGINE=InnoDB;
```

- Then, use ALTER TABLE ... DISCARD TABLESPACE to discard the new table's tablespace:

```
ALTER TABLE test.t1 DISCARD TABLESPACE;
```

- Then, copy the `.ibd` and `.cfg` files from the original server to the relevant directory on the target MariaDB Server:

```
# scp /data/tablespaces/t1.ibd target-server.com:/var/lib/mysql/test/
# scp /data/tablespaces/t1.cfg target-server.com:/var/lib/mysql/test/
```

File-per-table tablespaces can be imported with just the `.ibd` file in many cases. If you do not have the tablespace's `.cfg` file for whatever reason, then it is usually worth trying to import the tablespace with just the `.ibd` file.

- Then, once the files are in the proper directory on the target server, use ALTER TABLE ... IMPORT TABLESPACE to import the new table's tablespace:

```
ALTER TABLE test.t1 IMPORT TABLESPACE;
```

Copying Transportable Tablespaces for Partitioned Tables

Currently, MariaDB does not directly support the transport of tablespaces from partitioned tables. See MDEV-10568 for more information about that. It is still possible to transport partitioned tables if we use a workaround. You can copy the transportable tablespaces of a partitioned table from one server to another by exporting the tablespace file of each partition from the original server, and then importing the tablespace file of each partition into the new server.

Exporting Transportable Tablespaces for Partitioned Tables

You can export a partitioned table by locking the table and copying the `.ibd` and `.cfg` files of each partition from the relevant tablespace location for the partition to a backup location. For example, the process would go like this:

- First, let's create a test table with some data on the original server:

```
CREATE TABLE test.t2 (
   employee_id INT,
   name VARCHAR(50)
) ENGINE=InnoDB
PARTITION BY RANGE (employee_id) (
   PARTITION p0 VALUES LESS THAN (6),
   PARTITION p1 VALUES LESS THAN (11),
   PARTITION p2 VALUES LESS THAN (16),
   PARTITION p3 VALUES LESS THAN MAXVALUE
);

INSERT INTO test.t2 (name, employee_id) VALUES
   ('Geoff Montee', 1),
   ('Chris Calendar', 6),
   ('Kyle Joiner', 11),
   ('Will Fong', 16);
```

- Then, we need to export the partitioned tablespace from the original server, which follows the same process as exporting non-partitioned tablespaces. That means that we need to use the FLUSH TABLES ... FOR EXPORT statement on the target table:

```
FLUSH TABLES test.t2 FOR EXPORT;
```

This forces the server to close the table and provides your connection with a read lock on the table.

- Then, if we grep the database directory in the data directory for the newly created `t2` table, we can see a number of `.ibd` and `.cfg` files for the table:

```
# ls -l /var/lib/mysql/test/ | grep t2
total 428
-rw-rw---- 1 mysql mysql 827 Dec 5 16:08 t2.frm
-rw-rw---- 1 mysql mysql 48 Dec 5 16:08 t2.par
-rw-rw---- 1 mysql mysql 579 Dec 5 18:47 t2#P#p0.cfg
-rw-r----- 1 mysql mysql 98304 Dec 5 16:43 t2#P#p0.ibd
-rw-rw---- 1 mysql mysql 579 Dec 5 18:47 t2#P#p1.cfg
-rw-rw---- 1 mysql mysql 98304 Dec 5 16:08 t2#P#p1.ibd
-rw-rw---- 1 mysql mysql 579 Dec 5 18:47 t2#P#p2.cfg
-rw-rw---- 1 mysql mysql 98304 Dec 5 16:08 t2#P#p2.ibd
-rw-rw---- 1 mysql mysql 579 Dec 5 18:47 t2#P#p3.cfg
-rw-rw---- 1 mysql mysql 98304 Dec 5 16:08 t2#P#p3.ibd
```

- Then, while our connection still holds the lock on the table, we need to copy the tablespace files and the metadata files to a safe directory:

```
$ mkdir /tmp/backup
$ sudo cp /var/lib/mysql/test/t2*.ibd /tmp/backup
$ sudo cp /var/lib/mysql/test/t2*.cfg /tmp/backup
```

- Then, once we've copied the files, we can release the lock with UNLOCK TABLES:

```
UNLOCK TABLES;
```

Importing Transportable Tablespaces for Partitioned Tables

You can import a partitioned table by creating a placeholder table, discarding the placeholder table's original tablespace, copying the partition's `.ibd` and `.cfg` files from the backup location to the relevant tablespace location for the placeholder table, and then telling the server to import the tablespace. At that point, the server can exchange the tablespace for the placeholder table with the one for the partition. For example, the process would go like this:

- First, we need to copy the saved tablespace files from the original server to the target server:

```
$ scp /tmp/backup/t2* user@target-host:/tmp/backup
```

- Then, we need to import the partitioned tablespaces onto the target server. The import process for partitioned tables is more complicated than the import process for non-partitioned tables. To start with, if it doesn't already exist, then we need to create a partitioned table on the target server that matches the partitioned table on the original server:

```
CREATE TABLE test.t2 (
   employee_id INT,
   name VARCHAR(50)
) ENGINE=InnoDB
PARTITION BY RANGE (employee_id) (
   PARTITION p0 VALUES LESS THAN (6),
   PARTITION p1 VALUES LESS THAN (11),
   PARTITION p2 VALUES LESS THAN (16),
   PARTITION p3 VALUES LESS THAN MAXVALUE
);
```

- Then, using this table as a model, we need to create a placeholder of this table with the same structure that does not use partitioning. This can be done with a CREATE TABLE... AS SELECT statement:

```
CREATE TABLE test.t2_placeholder LIKE test.t2;
ALTER TABLE test.t2_placeholder REMOVE PARTITIONING;
```

This statement will create a new table called `t2_placeholder` that has the same schema structure as `t2`, but it does not use partitioning and it contains no rows.

For Each Partition

From this point forward, the rest of our steps need to happen for each individual partition. For each partition, we need to do the following process:

- First, we need to use ALTER TABLE ... DISCARD TABLESPACE to discard the placeholder table's tablespace:

```
ALTER TABLE test.t2_placeholder DISCARD TABLESPACE;
```

- Then, copy the `.ibd` and `.cfg` files for the next partition to the relevant directory for the `t2_placeholder` table on the target MariaDB Server:

```
# cp /tmp/backup/t2#P#p0.cfg /var/lib/mysql/test/t2_placeholder.cfg
# cp /tmp/backup/t2#P#p0.ibd /var/lib/mysql/test/t2_placeholder.ibd
# chown mysql:mysql /var/lib/mysql/test/t2_placeholder*
```

> File-per-table tablespaces can be imported with just the `.ibd` file in many cases. If you do not have the tablepace's `.cfg` file for whatever reason, then it is usually worth trying to import the tablespace with just the `.ibd` file.

- Then, once the files are in the proper directory on the target server, we need to use ALTER TABLE ... IMPORT TABLESPACE to import the new table's tablespace:

```
ALTER TABLE test.t2_placeholder IMPORT TABLESPACE;
```

The placeholder table now contains data from the `p0` partition on the source server.

```
SELECT * FROM test.t2_placeholder;

+-------------+--------------+
| employee_id | name         |
+-------------+--------------+
|           1 | Geoff Montee |
+-------------+--------------+
```

- Then, it's time to transfer the partition from the placeholder to the target table. This can be done with an ALTER TABLE... EXCHANGE PARTITION statement:

```
ALTER TABLE test.t2 EXCHANGE PARTITION p0 WITH TABLE test.t2_placeholder;
```

The target table now contains the first partition from the source table.

```
SELECT * FROM test.t2;

+-------------+--------------+
| employee_id | name         |
+-------------+--------------+
|           1 | Geoff Montee |
+-------------+--------------+
```

- Repeat this procedure for each partition you want to import. For each partition, we need to discard the placeholder table's tablespace, and then import the partitioned table's tablespace into the placeholder table, and then exchange the tablespaces between the placeholder table and the partition of our target table.

When this process is complete for all partitions, the target table will contain the imported data:

```
SELECT * FROM test.t2;

+-------------+----------------+
| employee_id | name           |
+-------------+----------------+
|           1 | Geoff Montee   |
|           6 | Chris Calendar |
|          11 | Kyle Joiner    |
|          16 | Will Fong      |
+-------------+----------------+
```

- Then, we can remove the placeholder table from the database:

```
DROP TABLE test.t2_placeholder;
```

Known Problems with Copying Transportable Tablespaces

Differing Storage Formats for Temporal Columns

MariaDB 10.1.2 added the mysql56_temporal_format system variable, which enables a new MySQL 5.6-compatible storage format for the TIME, DATETIME and TIMESTAMP data types.

If a file-per-tablespace file contains columns that use one or more of these temporal data types and if the tablespace file's original table was created with a certain storage format for these columns, then the tablespace file can only be imported into tables that were also created with the same storage format for these columns as the original table. Otherwise, you will see errors like the following:

```
ALTER TABLE dt_test IMPORT TABLESPACE;
ERROR 1808 (HY000): Schema mismatch (Column dt precise type mismatch.)
```

See MDEV-15225 for more information.

See the pages for the TIME, DATETIME and TIMESTAMP data types to determine how to update the storage format for temporal columns in tables that were created before MariaDB 10.1.2 or that were created with mysql56_temporal_format=OFF.

Differing ROW_FORMAT Values

InnoDB file-per-table tablespaces can use different row formats. A specific row format can be specified when creating a table either by setting the ROW_FORMAT table option or by the setting the innodb_default_row_format system variable. See Setting a Table's Row Format for more information on how to set an InnoDB table's row format.

If a file-per-tablespace file was created with a certain row format, then the tablespace file can only be imported into tables that were created with the same row format as the original table. Otherwise, you will see errors like the following:

```
ALTER TABLE t0 IMPORT TABLESPACE;
ERROR 1808 (HY000): Schema mismatch (Expected FSP_SPACE_FLAGS=0x21, .ibd file contains 0x0.)
```

The error message is a bit more descriptive in MariaDB 10.2.17 and later:

```
ALTER TABLE t0 IMPORT TABLESPACE;
ERROR 1808 (HY000): Schema mismatch (Table flags don't match, server table has 0x1 and the
meta-data file has 0x0; .cfg file uses ROW_FORMAT=REDUNDANT)
```

Be sure to check a tablespace's row format before moving it from one server to another. Keep in mind that the default row format can change between major versions of MySQL or MariaDB. See Checking a Table's Row Format for information on how to check an InnoDB table's row format.

See MDEV-15049 and MDEV-16851 for more information.

Foreign Key Constraints

DISCARD on a table with foreign key constraints is only possible after disabling foreign_key_checks:

```
SET SESSION foreign_key_checks=0;
ALTER TABLE t0 DISCARD TABLESPACE;
```

IMPORT on the other hand does not enforce foreign key constraints. So when importing tablespaces, referential integrity can only be guaranteed to import all tables bound by foreign key constraint at the same time, from an EXPORT of those tables taken with the same transactional state.

Tablespace Encryption

MariaDB supports data-at-rest encryption for the InnoDB storage engine. When enabled, the Server encrypts data before writing it to the tablespace and decrypts reads from the tablespace before returning result-sets. This means that a malicious user attempting to exfiltrate sensitive data won't be able to import the tablespace onto a different server as shown above without the encryption key.

For more information on data encryption, see Encrypting Data for InnoDB.

5.3.2.10.3 InnoDB Temporary Tablespaces

When the user creates a temporary table using the CREATE TEMPORARY TABLE statement and the engine is set as InnoDB, MariaDB creates a temporary tablespace file. When the table is not compressed, MariaDB writes to a shared temporary tablespace as defined by the innodb_temp_data_file_path system variable. MariaDB does not allow the creation of ROW_FORMAT=COMPRESSED temporary tables. All temporary tables will be uncompressed. MariaDB deletes temporary tablespaces when the server shuts down gracefully and is recreated when it starts again. It cannot be placed on a raw device.

Internal temporary tablespaces, (that is, temporary tables that cannot be kept in memory) use either Aria or MyISAM, depending on the aria_used_for_temp_tables system variable. You can set the default storage engine for user-created temporary tables using the default_tmp_storage_engine system variable.

Prior to MariaDB 10.2, temporary tablespaces existed as part of the InnoDB system tablespace or were file-per-table depending on the configuration of the innodb_file_per_table system variable.

Sizing Temporary Tablespaces

In order to size temporary tablespaces, use the innodb_temp_data_file_path system variable. This system variable can be specified as a command-line argument to mysqld or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
innodb_temp_data_file_path=ibtmp1:32M:autoextend
```

This system variable's syntax is the same as the innodb_data_file_path system variable. That is, a file name, size and option. By default, it writes a 12MB autoextending file to `ibtmp1` in the data directory.

To increase the size of the temporary tablespace, you can add a path to an additional tablespace file to the value of the the innodb_temp_data_file_path system variable. Providing additional paths allows you to spread the temporary tablespace between multiple tablespace files. The last file can have the `autoextend` attribute, which ensures that you won't run out of space. For example:

```
[mariadb]
...
innodb_temp_data_file_path=ibtmp1:32M;ibtmp2:32M:autoextend
```

Unlike normal tablespaces, temporary tablespaces are deleted when you stop MariaDB. To shrink temporary tablespaces to their minimum sizes, restart the server.

Shrinking the Tablespace

MariaDB starting with 11.3

From MariaDB 11.3.0, the temporary tablespace can be shrunk by setting innodb_truncate_temporary_tablespace_now to ON:

```
SET GLOBAL innodb_truncate_temporary_tablespace_now=1;
```

5.3.2.11 InnoDB File Format

Contents

Prior to MariaDB 10.3, the InnoDB storage engine supports two different file formats.

Setting a Table's File Format

In MariaDB 10.2 and before, the default file format for InnoDB tables can be chosen by setting the innodb_file_format.

In MariaDB 10.2.1 and before, the default file format is `Antelope`. In MariaDB 10.2.2 and later, the default file format is `Barracuda` and `Antelope` is deprecated.

A table's tablespace is tagged with the lowest InnoDB file format that supports the table's row format. So, even if the `Barracuda` file format is enabled, tables that use the `COMPACT` or `REDUNDANT` row formats will be tagged with the `Antelope` file format in the information_schema.INNODB_SYS_TABLES table.

Supported File Formats

The InnoDB storage engine supports two different file formats:

- `Antelope`
- `Barracuda`

Antelope

In MariaDB 10.2.1 and before, the default file format is `Antelope`. In MariaDB 10.2.2 and later, the `Antelope` file format is deprecated.

`Antelope` is the original InnoDB file format. It supports the `COMPACT` and `REDUNDANT` row formats, but not the `DYNAMIC` or `COMPRESSED` row formats.

Barracuda

In MariaDB 10.1 and before, the `Barracuda` file format is only supported if the innodb_file_per_table system variable is set to `ON`. In MariaDB 10.2.2 and later, the default file format is `Barracuda` and `Antelope` is deprecated.

`Barracuda` is a newer InnoDB file format. It supports the `COMPACT`, `REDUNDANT`, `DYNAMIC` and `COMPRESSED` row formats. Tables with large BLOB or TEXT columns in particular could benefit from the dynamic row format.

Future Formats

InnoDB might use new file formats in the future. Each format will have an identifier from 0 to 25, and a name. The names have already been decided, and are animal names listed in an alphabetical order: Antelope, Barracuda, Cheetah, Dragon, Elk, Fox, Gazelle, Hornet, Impala, Jaguar, Kangaroo, Leopard, Moose, Nautilus, Ocelot, Porpoise, Quail, Rabbit, Shark, Tiger, Urchin, Viper, Whale, Xenops, Yak and Zebra.

Checking a Table's File Format.

The information_schema.INNODB_SYS_TABLES table can be queried to see the file format used by a table.

> A table's tablespace is tagged with the lowest InnoDB file format that supports the table's row format. So, even if the `Barracuda` file format is enabled, tables that use the `COMPACT` or `REDUNDANT` row formats will be tagged with the `Antelope` file format in the information_schema.INNODB_SYS_TABLES table.

Compatibility

Each tablespace is tagged with the id of the most recent file format used by one of its tables. All versions of InnoDB can read tables that use an older file format. However, it can not read from more recent formats. For this reason, each time InnoDB opens a table it checks the tablespace's format, and returns an error if a newer format is used.

This check can be skipped via the innodb_file_format_check variable. Beware that, is InnoDB tries to repair a table in an unknown format, the table will be corrupted! This happens on restart if innodb_file_format_check is disabled and the server crashed, or it was closed with fast shutdown.

To downgrade a table from the Barracuda format to Antelope, the table's `ROW_FORMAT` can be set to a value supported by Antelope, via an ALTER TABLE statement. This recreates the indexes.

The Antelope format can be used to make sure that tables work on MariaDB and MySQL servers which are older than 5.5.

5.3.2.12 InnoDB Row Formats

InnoDB Row Formats Overview

InnoDB's row formats are REDUNDANT, COMPACT, DYNAMIC, and COMPRESSED.

InnoDB REDUNDANT Row Format

The REDUNDANT row format is the original non-compacted row format.

InnoDB COMPACT Row Format

Similar to the REDUNDANT row format, but stores data in a more compact manner.

InnoDB DYNAMIC Row Format

Similar to the COMPACT row format, but can store even more data on overflow pages.

InnoDB COMPRESSED Row Format

Similar to the COMPACT row format, but can store even more data on overflow pages.

Troubleshooting Row Size Too Large Errors with InnoDB

Fixing "Row size too large (> 8126). Changing some columns to TEXT or BLOB may help."

5.3.2.12.1 InnoDB Row Formats Overview

Contents

The InnoDB storage engine supports four different row formats:

- REDUNDANT
- COMPACT
- DYNAMIC
- COMPRESSED

In MariaDB 10.1 and before, the latter two row formats are only supported if the InnoDB file format is `Barracuda`. Therefore, the innodb_file_format system variable must be set to `Barracuda` to use these row formats in those versions.

In MariaDB 10.1 and before, the latter two row formats are also only supported if the table is in a file per-table tablespace. Therefore, the innodb_file_per_table system variable must be set to `ON` to use these row formats in those versions.

Default Row Format

The innodb_default_row_format system variable can be used to set the default row format for InnoDB tables. The possible values are:

- `redundant`
- `compact`
- `dynamic`

This system variable's default value is `dynamic`, which means that the default row format is `DYNAMIC`.

This system variable cannot be set to `compressed`, which means that the default row format cannot be `COMPRESSED`.

For example, the following statements would create a table with the `DYNAMIC` row format:

```
SET SESSION innodb_strict_mode=ON;

SET GLOBAL innodb_default_row_format='dynamic';

CREATE TABLE tab (
   id int,
   str varchar(50)
) ENGINE=InnoDB;
```

Setting a Table's Row Format

One way to specify an InnoDB table's row format is by setting the ROW_FORMAT table option to the relevant row format in a CREATE TABLE or ALTER TABLE statement. For example:

```
SET SESSION innodb_strict_mode=ON;

SET GLOBAL innodb_file_per_table=ON;

SET GLOBAL innodb_file_format='Barracuda';

CREATE TABLE tab (
   id int,
   str varchar(50)
) ENGINE=InnoDB ROW_FORMAT=DYNAMIC;
```

In MariaDB 10.1 and before, InnoDB can silently ignore and override some row format choices if you do not have the innodb_file_format system variable set to `Barracuda` and the innodb_file_per_table system variable set to `ON`.

Checking a Table's Row Format

The SHOW TABLE STATUS statement can be used to see the row format used by a table. For example:

```
SHOW TABLE STATUS FROM db1 WHERE Name='tab'\G
*************************** 1. row ***************************
           Name: tab
         Engine: InnoDB
        Version: 10
     Row_format: Dynamic
           Rows: 0
 Avg_row_length: 0
    Data_length: 16384
Max_data_length: 0
   Index_length: 0
      Data_free: 0
 Auto_increment: NULL
    Create_time: 2019-04-18 20:24:04
    Update_time: NULL
     Check_time: NULL
      Collation: latin1_swedish_ci
       Checksum: NULL
 Create_options: row_format=DYNAMIC
        Comment:
```

The information_schema.INNODB_SYS_TABLES table can also be queried to see the row format used by a table. For example:

```
SELECT * FROM information_schema.INNODB_SYS_TABLES WHERE name='db1/tab'\G
*************************** 1. row ***************************
     TABLE_ID: 42
         NAME: db1/tab
         FLAG: 33
       N_COLS: 4
        SPACE: 27
  FILE_FORMAT: Barracuda
   ROW_FORMAT: Dynamic
ZIP_PAGE_SIZE: 0
   SPACE_TYPE: Single
```

A table's tablespace is tagged with the lowest InnoDB file format that supports the table's row format. So, even if the `Barracuda` file format is enabled, tables that use the `COMPACT` or `REDUNDANT` row formats will be tagged with the `Antelope` file format in the information_schema.INNODB_SYS_TABLES table.

Row Formats

REDUNDANT Row Format

The `REDUNDANT` row format is the original non-compacted row format.

The `REDUNDANT` row format was the only available row format before MySQL 5.0.3. In that release, this row format was retroactively named the `REDUNDANT` row format. In the same release, the `COMPACT` row format was introduced as the new default row format.

See InnoDB REDUNDANT Row Format for more information.

COMPACT Row Format

Default row format in MariaDB 10.2.1 and earlier `COMPACT`.

The `COMPACT` row format is similar to the `REDUNDANT` row format, but it stores data in a more compact manner that requires about 20% less storage.

See InnoDB COMPACT Row Format for more information.

DYNAMIC Row Format

`DYNAMIC` is the default row format.

The `DYNAMIC` row format is similar to the `COMPACT` row format, but tables using the `DYNAMIC` row format can store even more data on overflow pages than tables using the `COMPACT` row format. This results in more efficient data storage than tables using the `COMPACT` row format, especially for tables containing columns using the VARBINARY, VARCHAR, BLOB and TEXT data types. However, InnoDB tables using the `COMPRESSED` row format are more efficient.

See InnoDB DYNAMIC Row Format for more information.

COMPRESSED Row Format

An alternative way to compress InnoDB tables is by using InnoDB Page Compression.

The `COMPRESSED` row format is similar to the `COMPACT` row format, but tables using the `COMPRESSED` row format can store even more data on overflow pages than tables using the `COMPACT` row format. This results in more efficient data storage than tables using the `COMPACT` row format, especially for tables containing columns using the VARBINARY, VARCHAR, BLOB and TEXT data types.

The `COMPRESSED` row format also supports compression of all data and index pages.

See InnoDB COMPRESSED Row Format for more information.

Maximum Row Size

Several factors help determine the maximum row size of an InnoDB table.

First, MariaDB enforces a 65,535 byte limit on a table's maximum row size. The total size of a table's BLOB and TEXT columns do not count towards this limit. Only the pointers for a table's BLOB and TEXT columns count towards this limit. MariaDB enforces this limit for all storage engines, so this limit also applies to InnoDB tables. Therefore, this limit is the absolute maximum row size for an InnoDB table.

If you try to create a table that exceeds MariaDB's global limit on a table's maximum row size, then you will see an error like this:

```
ERROR 1118 (42000): Row size too large. The maximum row size for the used table type,
not counting BLOBs, is 65535. This includes storage overhead, check the manual. You
have to change some columns to TEXT or BLOBs
```

However, InnoDB also has its own limits on the maximum row size, so an InnoDB table's maximum row size could be smaller than MariaDB's global limit.

Second, the maximum amount of data that an InnoDB table can store in a row's main data page depends on the value of the innodb_page_size system variable. At most, the data that a single row can consume on the row's main data page is half of the value of the innodb_page_size system variable. With the default value of `16k`, that would mean that a single row can consume at most around 8 KB on the row's main data page. However, the limit on the row's main data page is not the absolute limit on the row's size.

Third, all InnoDB row formats can store certain kinds of data in overflow pages, so the maximum row size of an InnoDB table can be larger than the maximum amount of data that can be stored in the row's main data page.

Some row formats can store more data in overflow pages than others. For example, the `DYNAMIC` and `COMPRESSED` row formats can store the most data in overflow pages. To see how to determine the how the various InnoDB row formats can use overflow pages, see the following sections:

- InnoDB REDUNDANT Row Format: Overflow Pages with the REDUNDANT Row Format
- InnoDB COMPACT Row Format: Overflow Pages with the COMPACT Row Format
- InnoDB DYNAMIC Row Format: Overflow Pages with the DYNAMIC Row Format
- InnoDB COMPRESSED Row Format: Overflow Pages with the COMPRESSED Row Format

If a table's definition can allow rows that the table's InnoDB row format can't actually store, then InnoDB will raise errors or warnings in certain scenarios.

If the table were using the `REDUNDANT` or `COMPACT` row formats, then the error or warning would be the following:

```
ERROR 1118 (42000): Row size too large (> 8126). Changing some columns to
TEXT or BLOB or using ROW_FORMAT=DYNAMIC or ROW_FORMAT=COMPRESSED
may help. In current row format, BLOB prefix of 768 bytes is stored inline.
```

And if the table were using the `DYNAMIC` or `COMPRESSED` row formats, then the error or warning would be the following:

```
ERROR 1118 (42000): Row size too large (> 8126). Changing some columns to
TEXT or BLOB may help. In current row format, BLOB prefix of 0 bytes is stored inline.
```

These messages are raised in the following cases:

- If InnoDB strict mode is **enabled** and if a DDL statement is executed that touches the table, such as CREATE TABLE or ALTER TABLE, then InnoDB will raise an **error** with this message
- If InnoDB strict mode is **disabled** and if a DDL statement is executed that touches the table, such as CREATE TABLE `or ALTER TABLE`, then InnoDB will raise a **warning** with this message.
- Regardless of whether InnoDB strict mode is enabled, if a DML statement is executed that attempts to write a row that the table's InnoDB row format can't store, then InnoDB will raise an **error** with this message.

For information on how to solve the problem, see Troubleshooting Row Size Too Large Errors with InnoDB.

Known Issues

Upgrading Causes Row Size Too Large Errors

Prior to MariaDB 10.2.26, MariaDB 10.3.17, and MariaDB 10.4.7, MariaDB doesn't properly calculate the row sizes while executing DDL. In these versions, *unsafe* tables can be created, even if InnoDB strict mode is enabled. The calculations were fixed by MDEV-19292 in MariaDB 10.2.26, MariaDB 10.3.17, and MariaDB 10.4.7.

As a side effect, some tables that could be created or altered in previous versions may get rejected with the following error in these releases and any later releases.

```
ERROR 1118 (42000): Row size too large (> 8126). Changing some columns to
TEXT or BLOB may help. In current row format, BLOB prefix of 0 bytes is stored inline.
```

And users could also see the following message as an error or warning in the error log:

```
[Warning] InnoDB: Cannot add field col in table db1.tab because after adding it, the row size is
8478 which is greater than maximum allowed size (8126) for a record on index leaf page.
```

InnoDB used the wrong calculations to determine row sizes for quite a long time, so a lot of users may unknowingly have *unsafe* tables that the InnoDB row format can't actually store.

InnoDB does not currently have an easy way to check which existing tables have this problem. See MDEV-20400 for more information.

For information on how to solve the problem, see Troubleshooting Row Size Too Large Errors with InnoDB.

5.3.2.12.2 InnoDB REDUNDANT Row Format

Contents

The `REDUNDANT` row format is the original non-compacted row format.

The `REDUNDANT` row format was the only available row format before MySQL 5.0.3. In that release, this row format was retroactively named the `REDUNDANT` row format. In the same release, the `COMPACT` row format was introduced as the new default row format.

Using the `REDUNDANT` Row Format

The easiest way to create an InnoDB table that uses the `REDUNDANT` row format is by setting the ROW_FORMAT table option to `REDUNDANT` in a CREATE TABLE or ALTER TABLE statement.

It is recommended to set the innodb_strict_mode system variable to `ON` when using this format.

The `REDUNDANT` row format is supported by both the `Antelope` and the `Barracuda` file formats, so tables with this row format can be created regardless of the value of the innodb_file_format system variable.

For example:

```
SET SESSION innodb_strict_mode=ON;

CREATE TABLE tab (
   id int,
   str varchar(50)
) ENGINE=InnoDB ROW_FORMAT=REDUNDANT;
```

Index Prefixes with the `REDUNDANT` Row Format

The `REDUNDANT` row format supports index prefixes up to 767 bytes.

Overflow Pages with the `REDUNDANT` Row Format

All InnoDB row formats can store certain kinds of data in overflow pages. This allows for the maximum row size of an InnoDB table to be larger than the maximum amount of data that can be stored in the row's main data page. See Maximum Row Size for more information about the other factors that can contribute to the maximum row size for InnoDB tables.

In the `REDUNDANT` row format variable-length columns, such as columns using the VARBINARY, VARCHAR, BLOB and TEXT data types, can be partially stored in overflow pages.

InnoDB only considers using overflow pages if the table's row size is greater than half of innodb_page_size. If the row size is greater than this, then InnoDB chooses variable-length columns to be stored on overflow pages until the row size is less than half of innodb_page_size.

For VARBINARY, VARCHAR, BLOB and TEXT columns, only values longer than 767 bytes are considered for for storage on overflow pages. Bytes that are stored to track a value's length do not count towards this limit. This limit is only based on the length of the actual column's data.

Fixed-length columns greater than 767 bytes are encoded as variable-length columns, so they can also be stored in overflow pages if the table's row size is greater than half of innodb_page_size. Even though a column using the CHAR data type can hold at most 255 characters, a CHAR column can still exceed 767 bytes in some cases. For example, a `char(255)` column can exceed 767 bytes if the character set is `utf8mb4`.

If a column is chosen to be stored on overflow pages, then the first 767 bytes of the column's value and a 20-byte pointer to the column's first overflow page are stored on the main page. Each overflow page is the size of [innodb-system-variables#innodb_page_size|innodb_page_size]]. If a column is too large to be stored on a single overflow page, then it is stored on multiple overflow pages. Each overflow page contains part of the data and a 20-byte pointer to the next overflow page, if a next page exists.

5.3.2.12.3 InnoDB COMPACT Row Format

MariaDB until 10.2.1

In MariaDB 10.2.1 and before, the default row format is `COMPACT`.

Contents

The `COMPACT` row format is similar to the `REDUNDANT` row format, but it stores data in a more compact manner that requires about 20% less storage.

Using the `COMPACT` Row Format

MariaDB starting with 10.2.2

In MariaDB 10.2.2 and later, the easiest way to create an InnoDB table that uses the `COMPACT` row format is by setting the ROW_FORMAT table option to to `COMPACT` in a CREATE TABLE or ALTER TABLE statement.

It is recommended to set the innodb_strict_mode system variable to `ON` when using this row format.

The `COMPACT` row format is supported by both the `Antelope` and the `Barracuda` file formats, so tables with this row format can be created regardless of the value of the innodb_file_format system variable.

For example:

```
SET SESSION innodb_strict_mode=ON;

CREATE TABLE tab (
   id int,
   str varchar(50)
) ENGINE=InnoDB ROW_FORMAT=COMPACT;
```

MariaDB until 10.2.1

In MariaDB 10.2.1 and before, the default row format is `COMPACT` . Therefore, in these versions, the easiest way to create an InnoDB table that uses the `COMPACT` row format is by **not** setting the ROW_FORMAT table option at all in the CREATE TABLE or ALTER TABLE statement.

It is recommended to set the innodb_strict_mode system variable to `ON` when using this row format.

The `COMPACT` row format is supported by both the `Antelope` and the `Barracuda` file formats, so tables with this row format can be created regardless of the value of the innodb_file_format system variable.

For example:

```
SET SESSION innodb_strict_mode=ON;

CREATE TABLE tab (
   id int,
   str varchar(50)
) ENGINE=InnoDB;
```

Index Prefixes with the `COMPACT` Row Format

The `COMPACT` row format supports index prefixes up to 767 bytes.

Overflow Pages with the `COMPACT` Row Format

All InnoDB row formats can store certain kinds of data in overflow pages. This allows for the maximum row size of an InnoDB table to be larger than the maximum amount of data that can be stored in the row's main data page. See Maximum Row Size for more information about the other factors that can contribute to the maximum row size for InnoDB tables.

In the `COMPACT` row format variable-length columns, such as columns using the VARBINARY, VARCHAR, BLOB and TEXT data types, can be partially stored in overflow pages.

InnoDB only considers using overflow pages if the table's row size is greater than half of innodb_page_size. If the row size is greater than this, then InnoDB chooses variable-length columns to be stored on overflow pages until the row size is less than half of innodb_page_size.

For VARBINARY, VARCHAR, BLOB and TEXT columns, only values longer than 767 bytes are considered for storage on overflow pages. Bytes that are stored to track a value's length do not count towards this limit. This limit is only based on the length of the actual column's data.

Fixed-length columns greater than 767 bytes are encoded as variable-length columns, so they can also be stored in overflow pages if the table's row size is greater than half of innodb_page_size. Even though a column using the CHAR data type can hold at most 255 characters, a CHAR column can still exceed 767 bytes in some cases. For example, a `char(255)` column can exceed 767 bytes if the character set is `utf8mb4` .

If a column is chosen to be stored on overflow pages, then the first 767 bytes of the column's value and a 20-byte pointer to the column's first overflow page are stored on the main page. Each overflow page is the size of innodb_page_size. If a column is too large to be stored on a single overflow page, then it is stored on multiple overflow pages. Each overflow page contains part of the data and a 20-byte pointer to the next overflow page, if a next page exists.

5.3.2.12.4 InnoDB DYNAMIC Row Format

`DYNAMIC` is the default InnoDB row format.

Contents

The `DYNAMIC` row format is similar to the `COMPACT` row format, but tables using the `DYNAMIC` row format can store even more data on overflow pages than tables using the `COMPACT` row format. This results in more efficient data storage than tables using the `COMPACT` row format, especially for tables containing columns using the VARBINARY, VARCHAR, BLOB and TEXT data types. However, InnoDB tables using the `COMPRESSED` row format are more efficient.

The `DYNAMIC` row format was originally introduced in MariaDB 5.5.

Using the DYNAMIC Row Format

MariaDB starting with 10.2.2

In MariaDB 10.2.2 and later, the default row format is `DYNAMIC`, as long as the innodb_default_row_format system variable has not been modified. Therefore, in these versions, the easiest way to create an InnoDB table that uses the `DYNAMIC` row format is by **not** setting the ROW_FORMAT table option at all in a CREATE TABLE or ALTER TABLE statement.

It is recommended to set the innodb_strict_mode system variable to `ON` when using this row format.

For example:

```
SET SESSION innodb_strict_mode=ON;

SET GLOBAL innodb_default_row_format='dynamic';

CREATE TABLE tab (
   id int,
   str varchar(50)
) ENGINE=InnoDB;
```

MariaDB until 10.2.1

In MariaDB 10.2.1 and before, the easiest way to create an InnoDB table that uses the `DYNAMIC` row format is by setting the ROW_FORMAT table option to to `DYNAMIC` in a CREATE TABLE or ALTER TABLE statement.

It is recommended to set the innodb_strict_mode system variable to `ON` when using this row format.

The `DYNAMIC` row format is only supported by the `Barracuda` file format. As a side effect, in MariaDB 10.1 and before, the `DYNAMIC` row format is only supported if the InnoDB file format is `Barracuda`. Therefore, the innodb_file_format system variable must be set to `Barracuda` to use these row formats in those versions.

In MariaDB 10.1 and before, the `DYNAMIC` row format is also only supported if the table is in a file per-table tablespace. Therefore, the innodb_file_per_table system variable must be set to `ON` to use this row format in those versions.

For example:

```
SET SESSION innodb_strict_mode=ON;

SET GLOBAL innodb_file_per_table=ON;

SET GLOBAL innodb_file_format='Barracuda';

CREATE TABLE tab (
   id int,
   str varchar(50)
) ENGINE=InnoDB ROW_FORMAT=DYNAMIC;
```

Index Prefixes with the DYNAMIC Row Format

The `DYNAMIC` row format supports index prefixes up to 3072 bytes. In MariaDB 10.2 and before, the innodb_large_prefix system variable is used to configure the maximum index prefix length. In these versions, if innodb_large_prefix is set to `ON`, then the maximum prefix length is 3072 bytes, and if it is set to `OFF`, then the maximum prefix length is 767 bytes.

Overflow Pages with the DYNAMIC Row Format

All InnoDB row formats can store certain kinds of data in overflow pages. This allows for the maximum row size of an InnoDB table to be larger than the maximum amount of data that can be stored in the row's main data page. See Maximum Row Size for more information about the other factors that can contribute to the maximum row size for InnoDB tables.

In the `DYNAMIC` row format variable-length columns, such as columns using the VARBINARY, VARCHAR, BLOB and TEXT data types, can be completely stored in overflow pages.

InnoDB only considers using overflow pages if the table's row size is greater than half of innodb_page_size. If the row size is greater than this, then InnoDB chooses variable-length columns to be stored on overflow pages until the row size is less than half of innodb_page_size.

For BLOB and TEXT columns, only values longer than 40 bytes are considered for storage on overflow pages. For VARBINARY and VARCHAR columns, only values longer than 255 bytes are considered for storage on overflow pages. Bytes that are stored to track a value's length do not count towards these limits. These limits are only based on the length of the actual column's data.

These limits differ from the limits for the `COMPACT` row format, where the limit is 767 bytes for all types.

Fixed-length columns greater than 767 bytes are encoded as variable-length columns, so they can also be stored in overflow pages if the table's row size is greater than half of innodb_page_size. Even though a column using the CHAR data type can hold at most 255 characters, a CHAR column can still exceed 767 bytes in some cases. For example, a `char(255)` column can exceed 767 bytes if the character set is `utf8mb4`.

If a column is chosen to be stored on overflow pages, then the entire value of the column is stored on overflow pages, and only a 20-byte pointer to the column's first overflow page is stored on the main page. Each overflow page is the size of innodb_page_size. If a column is too large to be stored on a single overflow page, then it is stored on multiple overflow pages. Each overflow page contains part of the data and a 20-byte pointer to the next overflow page, if a next page exists.

This behavior differs from the behavior of the `COMPACT` row format, which always stores the column prefix on the main page. This allows tables using the `DYNAMIC` row format to contain a high number of columns using the VARBINARY, VARCHAR, BLOB and TEXT data types.

5.3.2.12.5 InnoDB COMPRESSED Row Format

Contents

In MariaDB 10.1 and later, an alternative (and usually superior) way to compress InnoDB tables is by using InnoDB Page Compression. See Comparison with the COMPRESSED Row Format.

The `COMPRESSED` row format is similar to the `COMPACT` row format, but tables using the `COMPRESSED` row format can store even more data on overflow pages than tables using the `COMPACT` row format. This results in more efficient data storage than tables using the `COMPACT` row format, especially for tables containing columns using the VARBINARY, VARCHAR, BLOB and TEXT data types.

The `COMPRESSED` row format also supports compression of all data and index pages.

Using the `COMPRESSED` Row Format

An InnoDB table that uses the `COMPRESSED` row format can be created by setting the ROW_FORMAT table option to

`COMPRESSED` and by setting the KEY_BLOCK_SIZE table option to one of the following values in a CREATE TABLE or ALTER TABLE statement, where the units are in `KB`:

- `1`
- `2`
- `4`
- `8`
- `16`

`16k` is the default value of the innodb_page_size system variable, so using `16` will usually result in minimal compression unless one of the following is true:

- The table has many columns that can be stored in overflow pages, such as columns that use the VARBINARY, VARCHAR, BLOB and TEXT data types.
- The server is using a non-default innodb_page_size value that is greater than `16k`.

In MariaDB 10.1 and later, the value of the innodb_page_size system variable can be set to `32k` and `64k`. This is especially useful because the larger page size permits more columns using the VARBINARY, VARCHAR, BLOB and TEXT data types. Regardless, even when the value of the innodb_page_size system variable is set to some value higher than `16k`, `16` is still the maximum value for the KEY_BLOCK_SIZE table option for InnoDB tables using the `COMPRESSED` row format.

The `COMPRESSED` row format cannot be set as the default row format with the innodb_default_row_format system variable.

The `COMPRESSED` row format is only supported by the `Barracuda` file format. As a side effect, in MariaDB 10.1 and before, the `COMPRESSED` row format is only supported if the InnoDB file format is `Barracuda`. Therefore, the innodb_file_format system variable must be set to `Barracuda` to use these row formats in those versions.

In MariaDB 10.1 and before, the `COMPRESSED` row format is also only supported if the table is in a file per-table tablespace. Therefore, the innodb_file_per_table system variable must be set to `ON` to use this row format in those versions.

It is also recommended to set the innodb_strict_mode system variable to `ON` when using this row format.

InnoDB automatically uses the `COMPRESSED` row format for a table if the KEY_BLOCK_SIZE table option is set to some value in a CREATE TABLE or ALTER TABLE statement. For example:

```
SET SESSION innodb_strict_mode=ON;

SET GLOBAL innodb_file_per_table=ON;

SET GLOBAL innodb_file_format='Barracuda';

CREATE TABLE tab (
   id int,
   str varchar(50)
) ENGINE=InnoDB KEY_BLOCK_SIZE=4;
```

If the KEY_BLOCK_SIZE table option is **not** set to some value, but the ROW_FORMAT table option is set to `COMPRESSED` in a CREATE TABLE or ALTER TABLE statement, then InnoDB uses a default value of `8` for the KEY_BLOCK_SIZE table option. For example:

```
SET SESSION innodb_strict_mode=ON;

SET GLOBAL innodb_file_per_table=ON;

SET GLOBAL innodb_file_format='Barracuda';

CREATE TABLE tab (
   id int,
   str varchar(50)
) ENGINE=InnoDB ROW_FORMAT=COMPRESSED;
```

Compression with the `COMPRESSED` Row Format

The `COMPRESSED` row format supports compression of all data and index pages.

To avoid compressing and uncompressing pages too many times, InnoDB tries to keep both compressed and uncompressed pages in the buffer pool when there is enough room. This results in a bigger cache. When there is not enough room, an adaptive LRU algorithm is used to decide whether compressed or uncompressed pages should be evicted from the buffer: for CPU-bound workloads, the compressed pages are evicted first; for I/O-bound workloads, the

uncompressed pages are evicted first. Of course, when necessary, both the compressed and uncompressed version of the same data can be evicted from the buffer.

Each compressed page has an uncompressed *modification log*, stored within the page itself. InnoDB writes small changes into it. When the space in the modification log runs out, the page is uncompressed, changes are applied, and the page is recompressed again. This is done to avoid some unnecessary decompression and compression operations.

Sometimes a *compression failure* might happen, because the data has grown too much to fit the page. When this happens, the page (and the index node) is split into two different pages. This process can be repeated recursively until the data fit the pages. This can be CPU-consuming on some busy servers which perform many write operations.

Before writing a compressed page into a data file, InnoDB writes it into the redo log. This ensures that the redo log can always be used to recover tables after a crash, even if the compression library is updated and some incompatibilities are introduced. But this also means that the redo log will grow faster and might need more space, or the frequency of checkpoints might need to increase.

Monitoring Performance of the `COMPRESSED` Row Format

The following `INFORMATION_SCHEMA` tables can be used to monitor the performances of InnoDB compressed tables:

- INNODB_CMP and INNODB_CMP_RESET
- INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET
- INNODB_CMPMEM and INNODB_CMPMEM_RESET

Index Prefixes with the `COMPRESSED` Row Format

The `COMPRESSED` row format supports index prefixes up to 3072 bytes. In MariaDB 10.2 and before, the innodb_large_prefix system variable is used to configure the maximum index prefix length. In these versions, if innodb_large_prefix is set to `ON`, then the maximum prefix length is 3072 bytes, and if it is set to `OFF`, then the maximum prefix length is 767 bytes.

Overflow Pages with the `COMPRESSED` Row Format

All InnoDB row formats can store certain kinds of data in overflow pages. This allows for the maximum row size of an InnoDB table to be larger than the maximum amount of data that can be stored in the row's main data page. See Maximum Row Size for more information about the other factors that can contribute to the maximum row size for InnoDB tables.

In the `COMPRESSED` row format variable-length columns, such as columns using the VARBINARY, VARCHAR, BLOB and TEXT data types, can be completely stored in overflow pages.

InnoDB only considers using overflow pages if the table's row size is greater than half of innodb_page_size. If the row size is greater than this, then InnoDB chooses variable-length columns to be stored on overflow pages until the row size is less than half of innodb_page_size.

For BLOB and TEXT columns, only values longer than 40 bytes are considered for storage on overflow pages. For VARBINARY and VARCHAR columns, only values longer than 255 bytes are considered for storage on overflow pages. Bytes that are stored to track a value's length do not count towards these limits. These limits are only based on the length of the actual column's data.

These limits differ from the limits for the `COMPACT` row format, where the limit is 767 bytes for all types.

Fixed-length columns greater than 767 bytes are encoded as variable-length columns, so they can also be stored in overflow pages if the table's row size is greater than half of innodb_page_size. Even though a column using the CHAR data type can hold at most 255 characters, a CHAR column can still exceed 767 bytes in some cases. For example, a `char(255)` column can exceed 767 bytes if the character set is `utf8mb4`.

If a column is chosen to be stored on overflow pages, then the entire value of the column is stored on overflow pages, and only a 20-byte pointer to the column's first overflow page is stored on the main page. Each overflow page is the size of innodb_page_size. If a column is too large to be stored on a single overflow page, then it is stored on multiple overflow pages. Each overflow page contains part of the data and a 20-byte pointer to the next overflow page, if a next page exists.

This behavior differs from the behavior of the `COMPACT` row format, which always stores the column prefix on the main page. This allows tables using the `COMPRESSED` row format to contain a high number of columns using the VARBINARY, VARCHAR, BLOB and TEXT data types.

Read-Only

MariaDB starting with 10.6

From MariaDB 10.6.0 until MariaDB 10.6.5, tables that are of the `COMPRESSED` row format are read-only by default.

This was intended to be the first step towards removing write support and deprecating the feature.

This plan has been scrapped, and from MariaDB 10.6.6, `COMPRESSED` tables are no longer read-only by default.

From MariaDB 10.6.0 to MariaDB 10.6.5, set the innodb_read_only_compressed variable to `OFF` to make the tables writable.

5.3.2.3.4 Troubleshooting Row Size Too Large Errors with InnoDB

5.3.2.13 InnoDB Strict Mode

Contents

InnoDB strict mode is similar to SQL strict mode. When it is enabled, certain InnoDB warnings become errors instead.

Configuring InnoDB Strict Mode

InnoDB strict mode is enabled by default.

InnoDB strict mode can be enabled or disabled by configuring the innodb_strict_mode server system variable.

Its global value can be changed dynamically with SET GLOBAL. For example:

```
SET GLOBAL innodb_strict_mode=ON;
```

Its value for the current session can also be changed dynamically with SET SESSION. For example:

```
SET SESSION innodb_strict_mode=ON;
```

It can also be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
innodb_strict_mode=ON
```

InnoDB Strict Mode Errors

Wrong Create Options

If InnoDB strict mode is enabled, and if a DDL statement is executed and invalid or conflicting table options are specified, then an error is raised. The error will only be a generic error that says the following:

```
ERROR 1005 (HY000): Can't create table `db1`.`tab` (errno: 140 "Wrong create options")
```

However, more details about the error can be found by executing SHOW WARNINGS.

For example, the error is raised in the following cases:

- The KEY_BLOCK_SIZE table option is set to a non-zero value, but the ROW_FORMAT table option is set to some row format other than the COMPRESSED row format. For example:

```
SET SESSION innodb_strict_mode=ON;

CREATE OR REPLACE TABLE tab (
   id int PRIMARY KEY,
   str varchar(50)
)
KEY_BLOCK_SIZE=4
ROW_FORMAT=DYNAMIC;
ERROR 1005 (HY000): Can't create table `db1`.`tab` (errno: 140 "Wrong create options")

SHOW WARNINGS;
+---------+------+-----------------------------------------------------------------------+
| Level   | Code | Message                                                               |
+---------+------+-----------------------------------------------------------------------+
| Warning | 1478 | InnoDB: cannot specify ROW_FORMAT = DYNAMIC with KEY_BLOCK_SIZE.      |
| Error   | 1005 | Can't create table `db1`.`tab` (errno: 140 "Wrong create options")    |
| Warning | 1030 | Got error 140 "Wrong create options" from storage engine InnoDB       |
+---------+------+-----------------------------------------------------------------------+
3 rows in set (0.000 sec)
```

- The KEY_BLOCK_SIZE table option is set to a non-zero value, but the configured value is larger than either `16` or the value of the innodb_page_size system variable, whichever is smaller.

```
SET SESSION innodb_strict_mode=ON;

CREATE OR REPLACE TABLE tab (
   id int PRIMARY KEY,
   str varchar(50)
)
KEY_BLOCK_SIZE=16;
ERROR 1005 (HY000): Can't create table `db1`.`tab` (errno: 140 "Wrong create options")

SHOW WARNINGS;
+---------+------+-----------------------------------------------------------------------+
| Level   | Code | Message                                                               |
+---------+------+-----------------------------------------------------------------------+
| Warning | 1478 | InnoDB: KEY_BLOCK_SIZE=16 cannot be larger than 8.                    |
| Error   | 1005 | Can't create table `db1`.`tab` (errno: 140 "Wrong create options")    |
| Warning | 1030 | Got error 140 "Wrong create options" from storage engine InnoDB       |
+---------+------+-----------------------------------------------------------------------+
3 rows in set (0.000 sec)
```

- The KEY_BLOCK_SIZE table option is set to a non-zero value, but the innodb_file_per_table system variable is not set to `ON`.

```
SET GLOBAL innodb_file_per_table=OFF;
SET SESSION innodb_strict_mode=ON;

CREATE OR REPLACE TABLE tab (
   id int PRIMARY KEY,
   str varchar(50)
)
KEY_BLOCK_SIZE=4;
ERROR 1005 (HY000): Can't create table `db1`.`tab` (errno: 140 "Wrong create options")

SHOW WARNINGS;
+---------+------+-----------------------------------------------------------------------+
| Level   | Code | Message                                                               |
+---------+------+-----------------------------------------------------------------------+
| Warning | 1478 | InnoDB: KEY_BLOCK_SIZE requires innodb_file_per_table.                |
| Error   | 1005 | Can't create table `db1`.`tab` (errno: 140 "Wrong create options")    |
| Warning | 1030 | Got error 140 "Wrong create options" from storage engine InnoDB       |
+---------+------+-----------------------------------------------------------------------+
3 rows in set (0.000 sec)
```

- The KEY_BLOCK_SIZE table option is set to a non-zero value, but it is not set to one of the supported values: [1, 2, 4, 8, 16].

```
SET SESSION innodb_strict_mode=ON;

CREATE OR REPLACE TABLE tab (
   id int PRIMARY KEY,
   str varchar(50)
)
KEY_BLOCK_SIZE=5;
ERROR 1005 (HY000): Can't create table `db1`.`tab` (errno: 140 "Wrong create options")

SHOW WARNINGS;
+---------+------+-----------------------------------------------------------------------+
| Level   | Code | Message                                                               |
+---------+------+-----------------------------------------------------------------------+
| Warning | 1478 | InnoDB: invalid KEY_BLOCK_SIZE = 5. Valid values are [1, 2, 4, 8, 16] |
| Error   | 1005 | Can't create table `db1`.`tab` (errno: 140 "Wrong create options")    |
| Warning | 1030 | Got error 140 "Wrong create options" from storage engine InnoDB       |
+---------+------+-----------------------------------------------------------------------+
3 rows in set (0.000 sec)
```

- The ROW_FORMAT table option is set to the COMPRESSED row format, but the innodb_file_per_table system variable is not set to `ON`.

```
SET GLOBAL innodb_file_per_table=OFF;
SET SESSION innodb_strict_mode=ON;

CREATE OR REPLACE TABLE tab (
   id int PRIMARY KEY,
   str varchar(50)
)
ROW_FORMAT=COMPRESSED;
ERROR 1005 (HY000): Can't create table `db1`.`tab` (errno: 140 "Wrong create options")

SHOW WARNINGS;
+---------+------+-----------------------------------------------------------------------+
| Level   | Code | Message                                                               |
+---------+------+-----------------------------------------------------------------------+
| Warning | 1478 | InnoDB: ROW_FORMAT=COMPRESSED requires innodb_file_per_table.         |
| Error   | 1005 | Can't create table `db1`.`tab` (errno: 140 "Wrong create options")    |
| Warning | 1030 | Got error 140 "Wrong create options" from storage engine InnoDB       |
+---------+------+-----------------------------------------------------------------------+
3 rows in set (0.000 sec)
```

- The ROW_FORMAT table option is set to a value, but it is not set to one of the values supported by InnoDB: REDUNDANT, COMPACT, DYNAMIC, and COMPRESSED.

```
SET SESSION innodb_strict_mode=ON;

CREATE OR REPLACE TABLE tab (
   id int PRIMARY KEY,
   str varchar(50)
)
ROW_FORMAT=PAGE;
ERROR 1005 (HY000): Can't create table `db1`.`tab` (errno: 140 "Wrong create options")

SHOW WARNINGS;
+---------+------+-----------------------------------------------------------------------+
| Level   | Code | Message                                                               |
+---------+------+-----------------------------------------------------------------------+
| Warning | 1478 | InnoDB: invalid ROW_FORMAT specifier.                                 |
| Error   | 1005 | Can't create table `db1`.`tab` (errno: 140 "Wrong create options")    |
| Warning | 1030 | Got error 140 "Wrong create options" from storage engine InnoDB       |
+---------+------+-----------------------------------------------------------------------+
3 rows in set (0.000 sec)
```

- Either the KEY_BLOCK_SIZE table option is set to a non-zero value or the ROW_FORMAT table option is set to the COMPRESSED row format, but the innodb_page_size system variable is set to a value greater than `16k`.

```
SET SESSION innodb_strict_mode=ON;

CREATE OR REPLACE TABLE tab (
   id int PRIMARY KEY,
   str varchar(50)
)
ROW_FORMAT=COMPRESSED;
ERROR 1005 (HY000): Can't create table `db1`.`tab` (errno: 140 "Wrong create options")

SHOW WARNINGS;
+---------+------+-----------------------------------------------------------------------+
| Level   | Code | Message                                                               |
+---------+------+-----------------------------------------------------------------------+
| Warning | 1478 | InnoDB: Cannot create a COMPRESSED table when innodb_page_size > 16k. |
| Error   | 1005 | Can't create table `db1`.`tab` (errno: 140 "Wrong create options")    |
| Warning | 1030 | Got error 140 "Wrong create options" from storage engine InnoDB       |
+---------+------+-----------------------------------------------------------------------+
3 rows in set (0.00 sec)
```

- The DATA DIRECTORY table option is set, but the innodb_file_per_table system variable is not set to `ON`.

```
SET GLOBAL innodb_file_per_table=OFF;
SET SESSION innodb_strict_mode=ON;

CREATE OR REPLACE TABLE tab (
   id int PRIMARY KEY,
   str varchar(50)
)
DATA DIRECTORY='/mariadb';
ERROR 1005 (HY000): Can't create table `db1`.`tab` (errno: 140 "Wrong create options")

SHOW WARNINGS;
+---------+------+-----------------------------------------------------------------------+
| Level   | Code | Message                                                               |
+---------+------+-----------------------------------------------------------------------+
| Warning | 1478 | InnoDB: DATA DIRECTORY requires innodb_file_per_table.                |
| Error   | 1005 | Can't create table `db1`.`tab` (errno: 140 "Wrong create options")    |
| Warning | 1030 | Got error 140 "Wrong create options" from storage engine InnoDB       |
+---------+------+-----------------------------------------------------------------------+
3 rows in set (0.000 sec)
```

- The DATA DIRECTORY table option is set, but the table is a temporary table.

```
SET SESSION innodb_strict_mode=ON;

CREATE OR REPLACE TEMPORARY TABLE tab (
   id int PRIMARY KEY,
   str varchar(50)
)
DATA DIRECTORY='/mariadb';
ERROR 1005 (HY000): Can't create table `db1`.`tab` (errno: 140 "Wrong create options")

SHOW WARNINGS;
+---------+------+-----------------------------------------------------------------------+
| Level   | Code | Message                                                               |
+---------+------+-----------------------------------------------------------------------+
| Warning | 1478 | InnoDB: DATA DIRECTORY cannot be used for TEMPORARY tables.           |
| Error   | 1005 | Can't create table `db1`.`tab` (errno: 140 "Wrong create options")    |
| Warning | 1030 | Got error 140 "Wrong create options" from storage engine InnoDB       |
+---------+------+-----------------------------------------------------------------------+
3 rows in set (0.000 sec)
```

- The INDEX DIRECTORY table option is set.

```
SET SESSION innodb_strict_mode=ON;

CREATE OR REPLACE TABLE tab (
   id int PRIMARY KEY,
   str varchar(50)
)
INDEX DIRECTORY='/mariadb';
ERROR 1005 (HY000): Can't create table `db1`.`tab` (errno: 140 "Wrong create options")

SHOW WARNINGS;
+---------+------+-----------------------------------------------------------------------+
| Level   | Code | Message                                                               |
+---------+------+-----------------------------------------------------------------------+
| Warning | 1478 | InnoDB: INDEX DIRECTORY is not supported                              |
| Error   | 1005 | Can't create table `db1`.`tab` (errno: 140 "Wrong create options") |
| Warning | 1030 | Got error 140 "Wrong create options" from storage engine InnoDB    |
+---------+------+-----------------------------------------------------------------------+
3 rows in set (0.000 sec)
```

- The PAGE_COMPRESSED table option is set to `1`, so InnoDB page compression is enabled, but the ROW_FORMAT table option is set to some row format other than the COMPACT or DYNAMIC row formats.

```
SET SESSION innodb_strict_mode=ON;

CREATE OR REPLACE TABLE tab (
   id int PRIMARY KEY,
   str varchar(50)
)
PAGE_COMPRESSED=1
ROW_FORMAT=COMPRESSED;
ERROR 1005 (HY000): Can't create table `db1`.`tab` (errno: 140 "Wrong create options")

SHOW WARNINGS;
+---------+------+-----------------------------------------------------------------------+
| Level   | Code | Message                                                               |
+---------+------+-----------------------------------------------------------------------+
| Warning |  140 | InnoDB: PAGE_COMPRESSED table can't have ROW_TYPE=COMPRESSED       |
| Error   | 1005 | Can't create table `db1`.`tab` (errno: 140 "Wrong create options") |
| Warning | 1030 | Got error 140 "Wrong create options" from storage engine InnoDB    |
+---------+------+-----------------------------------------------------------------------+
3 rows in set (0.000 sec)
```

- The PAGE_COMPRESSED table option is set to `1`, so InnoDB page compression is enabled, but the innodb_file_per_table system variable is not set to `ON`.

```
SET GLOBAL innodb_file_per_table=OFF;
SET SESSION innodb_strict_mode=ON;

CREATE OR REPLACE TABLE tab (
   id int PRIMARY KEY,
   str varchar(50)
)
PAGE_COMPRESSED=1;
ERROR 1005 (HY000): Can't create table `db1`.`tab` (errno: 140 "Wrong create options")

SHOW WARNINGS;
+---------+------+-----------------------------------------------------------------------+
| Level   | Code | Message                                                               |
+---------+------+-----------------------------------------------------------------------+
| Warning |  140 | InnoDB: PAGE_COMPRESSED requires innodb_file_per_table.            |
| Error   | 1005 | Can't create table `db1`.`tab` (errno: 140 "Wrong create options") |
| Warning | 1030 | Got error 140 "Wrong create options" from storage engine InnoDB    |
+---------+------+-----------------------------------------------------------------------+
3 rows in set (0.000 sec)
```

- The PAGE_COMPRESSED table option is set to `1`, so InnoDB page compression is enabled, but the KEY_BLOCK_SIZE table option is also specified.

```
SET SESSION innodb_strict_mode=ON;

CREATE OR REPLACE TABLE tab (
   id int PRIMARY KEY,
   str varchar(50)
)
PAGE_COMPRESSED=1
KEY_BLOCK_SIZE=4;
ERROR 1005 (HY000): Can't create table `db1`.`tab` (errno: 140 "Wrong create options")

SHOW WARNINGS;
+---------+------+-----------------------------------------------------------------------+
| Level   | Code | Message                                                               |
+---------+------+-----------------------------------------------------------------------+
| Warning |  140 | InnoDB: PAGE_COMPRESSED table can't have key_block_size               |
| Error   | 1005 | Can't create table `db1`.`tab` (errno: 140 "Wrong create options") |
| Warning | 1030 | Got error 140 "Wrong create options" from storage engine InnoDB    |
+---------+------+-----------------------------------------------------------------------+
3 rows in set (0.000 sec)
```

- The PAGE_COMPRESSION_LEVEL table option is set, but the PAGE_COMPRESSED table option is set to `0`, so InnoDB page compression is disabled.

```
SET SESSION innodb_strict_mode=ON;

CREATE OR REPLACE TABLE tab (
   id int PRIMARY KEY,
   str varchar(50)
)
PAGE_COMPRESSED=0
PAGE_COMPRESSION_LEVEL=9;
ERROR 1005 (HY000): Can't create table `db1`.`tab` (errno: 140 "Wrong create options")

SHOW WARNINGS;
+---------+------+-----------------------------------------------------------------------+
| Level   | Code | Message                                                               |
+---------+------+-----------------------------------------------------------------------+
| Warning |  140 | InnoDB: PAGE_COMPRESSION_LEVEL requires PAGE_COMPRESSED               |
| Error   | 1005 | Can't create table `db1`.`tab` (errno: 140 "Wrong create options") |
| Warning | 1030 | Got error 140 "Wrong create options" from storage engine InnoDB    |
+---------+------+-----------------------------------------------------------------------+
3 rows in set (0.000 sec)
```

MariaDB until 10.2

In MariaDB 10.2 and before, the error is raised in the following additional cases:

- The KEY_BLOCK_SIZE table option is set to a non-zero value, but the innodb_file_format system variable is not set to `Barracuda`.

```
SET GLOBAL innodb_file_format='Antelope';
SET SESSION innodb_strict_mode=ON;

CREATE OR REPLACE TABLE tab (
   id int PRIMARY KEY,
   str varchar(50)
)
KEY_BLOCK_SIZE=4;
ERROR 1005 (HY000): Can't create table `db1`.`tab` (errno: 140 "Wrong create options")

SHOW WARNINGS;
+---------+------+-----------------------------------------------------------------------+
| Level   | Code | Message                                                               |
+---------+------+-----------------------------------------------------------------------+
| Warning | 1478 | InnoDB: KEY_BLOCK_SIZE requires innodb_file_format > Antelope.     |
| Error   | 1005 | Can't create table `db1`.`tab` (errno: 140 "Wrong create options") |
| Warning | 1030 | Got error 140 "Wrong create options" from storage engine InnoDB    |
+---------+------+-----------------------------------------------------------------------+
3 rows in set (0.00 sec)
```

- The ROW_FORMAT table option is set to either the COMPRESSED or the DYNAMIC row format, but the innodb_file_format system variable is not set to `Barracuda`.

```
SET GLOBAL innodb_file_format='Antelope';
SET SESSION innodb_strict_mode=ON;

CREATE OR REPLACE TABLE tab (
   id int PRIMARY KEY,
   str varchar(50)
)
ROW_FORMAT=COMPRESSED;
ERROR 1005 (HY000): Can't create table `db1`.`tab` (errno: 140 "Wrong create options")

SHOW WARNINGS;
+---------+------+-----------------------------------------------------------------------+
| Level   | Code | Message                                                               |
+---------+------+-----------------------------------------------------------------------+
| Warning | 1478 | InnoDB: ROW_FORMAT=COMPRESSED requires innodb_file_format > Antelope. |
| Error   | 1005 | Can't create table `db1`.`tab` (errno: 140 "Wrong create options")    |
| Warning | 1030 | Got error 140 "Wrong create options" from storage engine InnoDB       |
+---------+------+-----------------------------------------------------------------------+
3 rows in set (0.00 sec)
```

- The PAGE_COMPRESSED table option is set to `1`, so InnoDB page compression is enabled, but the innodb_file_format system variable is not set to `Barracuda`.

```
SET GLOBAL innodb_file_format='Antelope';
SET SESSION innodb_strict_mode=ON;

CREATE OR REPLACE TABLE tab (
   id int PRIMARY KEY,
   str varchar(50)
)
PAGE_COMPRESSED=1;
SHOW WARNINGS;
ERROR 1005 (HY000): Can't create table `db1`.`tab` (errno: 140 "Wrong create options")

SHOW WARNINGS;
+---------+------+-----------------------------------------------------------------------+
| Level   | Code | Message                                                               |
+---------+------+-----------------------------------------------------------------------+
| Warning |  140 | InnoDB: PAGE_COMPRESSED requires innodb_file_format > Antelope.       |
| Error   | 1005 | Can't create table `db1`.`tab` (errno: 140 "Wrong create options")    |
| Warning | 1030 | Got error 140 "Wrong create options" from storage engine InnoDB       |
+---------+------+-----------------------------------------------------------------------+
3 rows in set (0.00 sec)
```

COMPRESSED Row Format

If InnoDB strict mode is enabled, and if a table uses the COMPRESSED row format, and if the table's KEY_BLOCK_SIZE is too small to contain a row, then an error is returned by the statement.

Row Size Too Large

If InnoDB strict mode is enabled, and if a table exceeds its row format's maximum row size, then InnoDB will return an error.

```
ERROR 1118 (42000): Row size too large (> 8126). Changing some columns to
TEXT or BLOB may help. In current row format, BLOB prefix of 0 bytes is stored inline.
```

See Troubleshooting Row Size Too Large Errors with InnoDB for more information.

5.3.2.14 InnoDB Redo Log

Contents

Directly editing or moving the redo logs can cause corruption, and should never normally be attempted.

Overview

The redo log is used by InnoDB during crash recovery. The redo log files have names like `ib_logfileN`, where `N` is an integer. From MariaDB 10.5, there is only one redo log, so the file will always be named `ib_logfile0`. If the innodb_log_group_home_dir system variable is configured, then the redo log files will be created in that directory. Otherwise, they will be created in the directory defined by the datadir system variable.

Flushing Effects on Performance and Consistency

The innodb_flush_log_at_trx_commit system variable determines how often the transactions are flushed to the redo log, and it is important to achieve a good balance between speed and reliability.

Binary Log Group Commit and Redo Log Flushing

In MariaDB 10.0 and above, when both innodb_flush_log_at_trx_commit=1 (the default) is set and the binary log is enabled, there is now one less sync to disk inside InnoDB during commit (2 syncs shared between a group of transactions instead of 3). See Binary Log Group Commit and InnoDB Flushing Performance for more information.

Redo Log Group Capacity

The redo log group capacity is the total combined size of all InnoDB redo logs. The relevant factors are:

- From MariaDB 10.5, there is 1 redo log. For MariaDB 10.4 and before, the number of redo log files is configured by the innodb_log_files_in_group system variable.
- The size of each redo log file is configured by the innodb_log_file_size system variable. This can safely be set to a much higher value from MariaDB 10.5. Before MariaDB 10.9, resizing required the server to be restarted. From MariaDB 10.9 the variable can be set dynamically.

The redo log group capacity is determined by the following calculation:

```
innodb_log_group_capacity = innodb_log_file_size * innodb_log_files_in_group
```

For example, if innodb_log_file_size is set to `2G` and innodb_log_files_in_group is set to `2`, then we would have the following:

- `innodb_log_group_capacity = innodb_log_file_size * innodb_log_files_in_group`
- `= 2G * 2`
- `= 4G`

Changing the Redo Log Group Capacity

The number (until MariaDB 10.4 only - from MariaDB 10.5 there is only 1 redo log) or size of redo log files can be changed with the following process:

- Stop the server.
- To change the log file size, configure innodb_log_file_size. To increase the number of log files (until MariaDB 10.4 only), configure innodb_log_files_in_group.
- Start the server.

Log Sequence Number (LSN)

Records within the InnoDB redo log are identified via a log sequence number (LSN).

Checkpoints

When InnoDB performs a checkpoint, it writes the LSN of the oldest dirty page in the InnoDB buffer pool to the InnoDB redo log. If a page is the oldest dirty page in the InnoDB buffer pool, then that means that all pages with lower LSNs have been flushed to the physical InnoDB tablespace files. If the server were to crash, then InnoDB would perform crash recovery by only applying log records with LSNs that are greater than or equal to the LSN of the oldest dirty page written in the last checkpoint.

Checkpoints are one of the tasks performed by the InnoDB master background thread. This thread schedules checkpoints 7 seconds apart when the server is very active, but checkpoints can happen more frequently when the server is less active.

Dirty pages are not actually flushed from the buffer pool to the physical InnoDB tablespace files during a checkpoint. That process happens asynchronously on a continuous basis by InnoDB's write I/O background threads configured by the innodb_write_io_threads system variable. If you want to make this process more aggressive, then you can decrease the value of the innodb_max_dirty_pages_pct system variable. You may also need to better tune InnoDB's I/O capacity on your system by setting the innodb_io_capacity system variable.

Determining the Checkpoint Age

The checkpoint age is the amount of data written to the InnoDB redo log since the last checkpoint.

Determining the Checkpoint Age in InnoDB

> **MariaDB starting with 10.5**
> MariaDB 10.5 reintroduced the Innodb_checkpoint_age status variable (available in XtraDB until MariaDB 10.1) for determining the checkpoint age.

The checkpoint age can also be determined by the process shown below.

To determine the InnoDB checkpoint age, do the following:

- Query SHOW ENGINE INNODB STATUS.
- Find the `LOG` section. For example:

```
---
LOG
---
Log sequence number 252794398789379
Log flushed up to 252794398789379
Pages flushed up to 252792767756840
Last checkpoint at 252792767756840
0 pending log flushes, 0 pending chkp writes
23930412 log i/o's done, 2.03 log i/o's/second
```

- Perform the following calculation:

`innodb_checkpoint_age = Log sequence number - Last checkpoint at`

In the example above, that would be:

- `innodb_checkpoint_age = Log sequence number - Last checkpoint at`
- = 252794398789379 - 252792767756840
- = 1631032539 bytes
- = 1631032539 byes / (1024 * 1024 * 1024) (GB/bytes)
- = 1.5 GB of redo log written since last checkpoint

Determining the Redo Log Occupancy

The redo log occupancy is the percentage of the InnoDB redo log capacity that is taken up by dirty pages that have not yet been flushed to the physical InnoDB tablespace files in a checkpoint. Therefore, it's determined by the following calculation:

`innodb_log_occupancy = innodb_checkpoint_age / innodb_log_group_capacity`

For example, if `innodb_checkpoint_age` is `1.5G` and `innodb_log_group_capacity` is `4G`, then we would have the

following:

- `innodb_log_occupancy = innodb_checkpoint_age / innodb_log_group_capacity`
- `= 1.5G / 4G`
- `= 0.375`

If the calculated value for redo log occupancy is too close to `1.0`, then the InnoDB redo log capacity may be too small for the current workload.

MariaDB 10.8 Updates

A number of redo log improvements were made in MariaDB 10.8:

- Autosize innodb_buffer_pool_chunk_size (MDEV-25342).
- Improve the redo log for concurrency (MDEV-14425).
- Remove FIL_PAGE_FILE_FLUSH_LSN (MDEV-27199).

Before MariaDB 10.8.1, mariadb-backup --prepare created a zero-length ib_logfile0 as a dummy placeholder. From MariaDB 10.8.1 (MDEV-14425), the size of that dummy file was increased to 12304 (0x3010) bytes, and all updates of FIL_PAGE_FILE_FLUSH_LSN in the first page of the system tablespace are removed.

From MariaDB 10.8.1, if the server is started up with a zero-sized ib_logfile0, it is assumed that an upgrade is being performed after a backup had been prepared. The start LSN will then be read from FIL_PAGE_FILE_FLUSH_LSN, and a new log file will be created starting from exactly that LSN.

Manually creating a zero-sized ib_logfile0 without manually updating the FIL_PAGE_FILE_FLUSH_LSN in the system tablespace to a recent enough LSN may result in error messages such as "page LSN is in the future". If a log was discarded while some changes had already been written to data pages, all sort of corruption may occur.

If the database was initialized with a server that never updates the FIL_PAGE_FILE_FLUSH_LSN field, then any server startup attempts with a zero-size ib_logfile0 will be refused because of an invalid LSN. If that field was ever updated with a valid LSN by an older server, this safety mechanism cannot work, and the server may "rewind" to an earlier LSN.

5.3.2.15 InnoDB Undo Log

Contents

Overview

When a transaction writes data, it always inserts them in the table indexes or data (in the buffer pool or in physical files). No private copies are created. The old versions of data being modified by active InnoDB transactions are stored in the undo log. The original data can then be restored, or viewed by a consistent read.

Implementation Details

Before a row is modified, a diff is copied into the undo log. Each normal row contains a pointer to the most recent version of the same row in the undo log. Each row in the undo log contains a pointer to previous version, if any. So, each modified row has a history chain.

Rows are never physically deleted until a transaction ends. If they were deleted, the restore in ROLLBACK would be impossible. Thus, rows are simply marked for deletion.

Each transaction uses a *view* of the records. The transaction isolation level determines how this view is created. For example, READ UNCOMMITTED usually uses the current version of rows, even if they are not committed (*dirty reads*). Other isolation levels require that the most recent committed version of rows is searched in the undo log. READ COMMITTED uses a different view for each table, while REPEATABLE READ and SERIALIZABLE use the same view for all tables.

There is also a global history list of the data. When a transaction is committed, its history is added to this history list. The order of the list is the chronological order of the commits.

The purge thread deletes the rows in the undo log which are not needed by any existing view. The rows for which a most recent version exists are deleted, as well as the delete-marked rows.

If InnoDB needs to restore an old version, it will simply replace the newer version with the older one. When a transaction inserts a new row, there is no older version. However, in that case, the restore can be done by deleting the inserted rows.

Effects of Long-Running Transactions

Understanding how the undo log works helps with understanding the negative effects long transactions.

- Long transactions generate several old versions of the rows in the undo log. Those rows will probably be needed for a longer time, because other long transactions will need them. Since those transactions will generate more modified rows, a sort of combinatorial explosion can be observed. Thus, the undo log requires more space.
- Transaction may need to read very old versions of the rows in the history list, thus their performance will degrade.

Of course read-only transactions do not write more entries in the undo log; however, they delay the purging of existing entries.

Also, long transactions can more likely result in deadlocks, but this problem is not related to the undo log.

Configuration

System variables affecting undo logs include:

- innodb_max_undo_log_size
- innodb_undo_directory
- innodb_undo_log_truncate
- innodb_undo_logs
- innodb_undo_tablespaces
- innodb_purge_batch_size
- innodb_purge_rseg_truncate_frequency

The undo log is not a log file that can be viewed on disk in the usual sense, such as the error log or slow query log, but rather an area of storage.

Before MariaDB 11.0, the undo log is usually part of the physical system tablespace, but from MariaDB 10.0, the innodb_undo_directory and innodb_undo_tablespaces system variables can be used to split into different tablespaces and store in a different location (perhaps on a different storage device). From MariaDB 11.0, multiple undo tablespaces are enabled by default, and the innodb_undo_tablespaces default is changed to 3 so that the space occupied by possible bursts of undo log records can be reclaimed after innodb_undo_log_truncate is set.

Each insert or update portion of the undo log is known as a rollback segment. The innodb_undo_logs system variable allowed to reduce the number of rollback segments from the usual 128, to limit the number of concurrently active write transactions. innodb_undo_logs was deprecated and ignored in MariaDB 10.5 and removed in MariaDB 10.6, as it always makes sense to use the maximum number of rollback segments.

The related innodb_available_undo_logs status variable stores the total number of available InnoDB undo logs.

5.3.2.16 InnoDB Page Flushing

Contents

Page Flushing with InnoDB Page Cleaner Threads

InnoDB page cleaner threads flush dirty pages from the InnoDB buffer pool. These dirty pages are flushed using a least-recently used (LRU) algorithm.

innodb_max_dirty_pages_pct

The innodb_max_dirty_pages_pct variable specifies the maximum percentage of unwritten (dirty) pages in the buffer pool. If this percentage is exceeded, flushing will take place.

innodb_max_dirty_pages_pct_lwm

The innodb_max_dirty_pages_pct_lwm variable determines the low-water mark percentage of dirty pages that will enable preflushing to lower the dirty page ratio. The value 0 (the default) means that there will be no separate background flushing

so long as:

- the share of dirty pages does not exceed innodb_max_dirty_pages_pct
- the last checkpoint age (LSN difference since the latest checkpoint) does not exceed innodb_log_file_size (minus some safety margin)
- the buffer pool is not running out of space, which could trigger eviction flushing

Note that in MariaDB 10.5.7 and MariaDB 10.5.8 only, flushing was more aggressive, and the page cleaner thread would always run in the background, as long as dirty pages exist in the buffer pool. To make flushing more eager, set to a higher value, for example `SET GLOBAL innodb_max_dirty_pages_pct_lwm=0.001;` (the default until MariaDB 10.2.1).

Page Flushing with Multiple InnoDB Page Cleaner Threads

MariaDB 10.2.2 - 10.5.1

The innodb_page_cleaners system variable was added in MariaDB 10.2.2, and makes it possible to use multiple InnoDB page cleaner threads. It is deprecated and ignored from MariaDB 10.5.1, as the original reasons for for splitting the buffer pool have mostly gone away.

The number of InnoDB page cleaner threads can be configured by setting the innodb_page_cleaners system variable. This system variable can be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
innodb_page_cleaners=8
```

In MariaDB 10.3.3 and later, this system variable can also be changed dynamically with SET GLOBAL. For example:

```
SET GLOBAL innodb_page_cleaners=8;
```

This system variable's default value is either `4` or the configured value of the innodb_buffer_pool_instances system variable, whichever is lower.

Page Flushing with a Single InnoDB Page Cleaner Thread

In MariaDB 10.2.1 and before, and from MariaDB 10.5.1, when the original reasons for splitting the buffer pool have mostly gone away, only a single InnoDB page cleaner thread is supported.

Page Flushing with Multi-threaded Flush Threads

MariaDB 10.1.0 - 10.3.2

InnoDB's multi-thread flush feature was first added in MariaDB 10.1.0. It was deprecated in MariaDB 10.2.9 and removed in MariaDB 10.3.2.

In MariaDB 10.3.1 and before, InnoDB's multi-thread flush feature can be used. This is especially useful in MariaDB 10.1, which only supports a single page cleaner thread.

InnoDB's multi-thread flush feature can be enabled by setting the innodb_use_mtflush system variable. The number of threads cane be configured by setting the innodb_mtflush_threads system variable. This system variable can be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
innodb_use_mtflush = ON
innodb_mtflush_threads = 8
```

The innodb_mtflush_threads system variable's default value is `8`. The maximum value is `64`. In multi-core systems, it is recommended to set its value close to the configured value of the innodb_buffer_pool_instances system variable. However, it is also recommended to use your own benchmarks to find a suitable value for your particular application.

InnoDB's multi-thread flush feature was deprecated in MariaDB 10.2.9 and removed from MariaDB 10.3.2. In later versions of MariaDB, use multiple InnoDB page cleaner threads instead.

Configuring the InnoDB I/O Capacity

Increasing the amount of I/O capacity available to InnoDB can also help increase the performance of page flushing.

The amount of I/O capacity available to InnoDB can be configured by setting the innodb_io_capacity system variable. This system variable can be changed dynamically with SET GLOBAL. For example:

```
SET GLOBAL innodb_io_capacity=20000;
```

This system variable can also be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
innodb_io_capacity=20000
```

The maximum amount of I/O capacity available to InnoDB in an emergency defaults to either 2000 or twice innodb_io_capacity, whichever is higher, or can be directly configured by setting the innodb_io_capacity_max system variable. This system variable can be changed dynamically with SET GLOBAL. For example:

```
SET GLOBAL innodb_io_capacity_max=20000;
```

This system variable can also be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
innodb_io_capacity_max=20000
```

5.3.2.17 InnoDB Purge

Contents

When a transaction updates a row in an InnoDB table, InnoDB's MVCC implementation keeps old versions of the row in the InnoDB undo log. The old versions are kept at least until all transactions older than the transaction that updated the row are no longer open. At that point, the old versions can be deleted. InnoDB has purge process that is used to delete these old versions.

Optimizing Purge Performance

Configuring the Purge Threads

The number of purge threads can be set by configuring the innodb_purge_threads system variable. This system variable can be specified as a command-line argument to mysqld or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
innodb_purge_threads = 6
```

Configuring the Purge Batch Size

The purge batch size is defined as the number of InnoDB redo log records that must be written before triggering purge. The purge batch size can be set by configuring the innodb_purge_batch_size system variable. This system variable can be specified as a command-line argument to mysqld or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
innodb_purge_batch_size = 50
```

Configuring the Max Purge Lag

If purge operations are lagging on a busy server, then this can be a tough situation to recover from. As a solution, InnoDB allows you to set the max purge lag. The max purge lag is defined as the maximum number of InnoDB undo log that can be waiting to be purged from the history until InnoDB begins delaying DML statements.

The max purge lag can be set by configuring the innodb_max_purge_lag system variable. This system variable can be changed dynamically with SET GLOBAL. For example:

```
SET GLOBAL innodb_max_purge_lag=1000;
```

This system variable can also be specified as a command-line argument to mysqld or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
innodb_max_purge_lag = 1000
```

The maximum delay can be set by configuring the innodb_max_purge_lag_delay system variable. This system variable can be changed dynamically with SET GLOBAL. For example:

```
SET GLOBAL innodb_max_purge_lag_delay=100;
```

This system variable can also be specified as a command-line argument to mysqld or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
innodb_max_purge_lag_delay = 100
```

Configuring the Purge Rollback Segment Truncation Frequency

The purge rollback segment truncation frequency is defined as the number of purge loops that are run before unnecessary rollback segments are truncated. The purge rollback segment truncation frequency can be set by configuring the innodb_purge_rseg_truncate_frequency system variable. This system variable can be changed dynamically with SET GLOBAL. For example:

```
SET GLOBAL innodb_purge_rseg_truncate_frequency=64;
```

This system variable can also be specified as a command-line argument to mysqld or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
innodb_purge_rseg_truncate_frequency = 64
```

Configuring the Purge Undo Log Truncation

Purge undo log truncation occurs when InnoDB truncates an entire InnoDB undo log tablespace, rather than deleting individual InnoDB undo log records.

Purge undo log truncation can be enabled by configuring the innodb_undo_log_truncate system variable. This system variable can be changed dynamically with SET GLOBAL. For example:

```
SET GLOBAL innodb_undo_log_truncate=ON;
```

This system variable can also be specified as a command-line argument to mysqld or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
innodb_undo_log_truncate = ON
```

An InnoDB undo log tablespace is truncated when it exceeds the maximum size that is configured for InnoDB undo log tablespaces. The maximum size can be set by configuring the innodb_max_undo_log_size system variable. This system

variable can be changed dynamically with SET GLOBAL. For example:

```
SET GLOBAL innodb_max_undo_log_size='64M';
```

This system variable can also be specified as a command-line argument to mysqld or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
innodb_max_undo_log_size = 64M
```

Purge's Effect on Row Metadata

An InnoDB table's clustered index has three hidden system columns that are automatically generated. These hidden system columns are:

- `DB_ROW_ID` - If the table has no other `PRIMARY KEY` or no other `UNIQUE KEY` defined as `NOT NULL` that can be promoted to the table's `PRIMARY KEY`, then InnoDB will use a hidden system column called `DB_ROW_ID`. InnoDB will automatically generated the value for the column from a global InnoDB-wide 48-bit sequence (instead of being table-local).
- `DB_TRX_ID` - The transaction ID of either the transaction that last changed the row or the transaction that currently has the row locked.
- `DB_ROLL_PTR` - A pointer to the InnoDB undo log that contains the row's previous record. The value of `DB_ROLL_PTR` is only valid if `DB_TRX_ID` belongs to the current read view. The oldest valid read view is the purge view.

If a row's last InnoDB undo log record is purged, this can obviously effect the value of the row's `DB_ROLL_PTR` column, because there would no longer be any InnoDB undo log record for the pointer to reference.

In MariaDB 10.2 and before, the purge process wouldn't touch the value of the row's `DB_TRX_ID` column.

However, in MariaDB 10.3 and later, the purge process will set a row's `DB_TRX_ID` column to `0` after all of the row's associated InnoDB undo log records have been deleted. This change allows InnoDB to perform an optimization: if a query wants to read a row, and if the row's `DB_TRX_ID` column is set to `0`, then it knows that no other transaction has the row locked. Usually, InnoDB needs to lock the transaction system's mutex in order to safely check whether a row is locked, but this optimization allows InnoDB to confirm that the row can be safely read without any heavy internal locking.

This optimization can speed up reads, but it come at a noticeable cost at other times. For example, it can cause the purge process to use more I/O after inserting a lot of rows, since the value of each row's `DB_TRX_ID` column will have to be reset.

1.1.1.2.9.1.1.1 Information Schema InnoDB Tables

5.3.2.19 InnoDB Online DDL

InnoDB Online DDL Overview
All about online DDL operations with InnoDB.

InnoDB Online DDL Operations with the INPLACE Alter Algorithm
These DDL operations can be done in-place with InnoDB.

InnoDB Online DDL Operations with the NOCOPY Alter Algorithm
These DDL operations can be done without copying the table with InnoDB.

InnoDB Online DDL Operations with the INSTANT Alter Algorithm
These DDL operations can be done instantly with InnoDB.

Instant ADD COLUMN for InnoDB
Instantly add a new column to a table

5.3.2.19.1 InnoDB Online DDL Overview

Contents

InnoDB tables support online DDL, which permits concurrent DML and uses optimizations to avoid unnecessary table copying.

The ALTER TABLE statement supports two clauses that are used to implement online DDL:

- ALGORITHM - This clause controls how the DDL operation is performed.
- LOCK - This clause controls how much concurrency is allowed while the DDL operation is being performed.

Alter Algorithms

InnoDB supports multiple algorithms for performing DDL operations. This offers a significant performance improvement over previous versions. The supported algorithms are:

- `DEFAULT` - This implies the default behavior for the specific operation.
- `COPY`
- `INPLACE`
- `NOCOPY` - This was added in MariaDB 10.3.7.
- `INSTANT` - This was added in MariaDB 10.3.7.

Specifying an Alter Algorithm

The set of alter algorithms can be considered as a hierarchy. The hierarchy is ranked in the following order, with *least efficient* algorithm at the top, and *most efficient* algorithm at the bottom:

- `COPY`
- `INPLACE`
- `NOCOPY`
- `INSTANT`

When a user specifies an alter algorithm for a DDL operation, MariaDB does not necessarily use that specific algorithm for the operation. It interprets the choice in the following way:

- If the user specifies `COPY`, then InnoDB uses the `COPY` algorithm.
- If the user specifies any other algorithm, then InnoDB interprets that choice as the *least efficient* algorithm that the user is willing to accept. This means that if the user specifies `INPLACE`, then InnoDB will use the *most efficient* algorithm supported by the specific operation from the set (`INPLACE`, `NOCOPY`, `INSTANT`). Likewise, if the user specifies `NOCOPY`, then InnoDB will use the *most efficient* algorithm supported by the specific operation from the set (`NOCOPY`, `INSTANT`).

There is also a special value that can be specified:

- If the user specifies `DEFAULT`, then InnoDB uses its default choice for the operation. The default choice is to use the most efficient algorithm supported by the operation. The default choice will also be used if no algorithm is specified.

Therefore, if you want InnoDB to use the most efficient algorithm supported by an operation, then you usually do not have to explicitly specify any algorithm at all.

Specifying an Alter Algorithm Using the ALGORITHM Clause

InnoDB supports the ALGORITHM clause.

The ALGORITHM clause can be used to specify the *least efficient* algorithm that the user is willing to accept. It is supported by the ALTER TABLE and CREATE INDEX statements.

For example, if a user wanted to add a column to a table, but only if the operation used an algorithm that is at least as efficient as the `INPLACE`, then they could execute the following:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50)
);

ALTER TABLE tab ADD COLUMN c varchar(50), ALGORITHM=INPLACE;
```

In MariaDB 10.3 and later, the above operation would actually use the `INSTANT` algorithm, because the `ADD COLUMN` operation supports the `INSTANT` algorithm, and the `INSTANT` algorithm is more efficient than the `INPLACE` algorithm.

Specifying an Alter Algorithm Using System Variables

MariaDB starting with 10.3

In MariaDB 10.3 and later, the alter_algorithm system variable can be used to pick the *least efficient* algorithm that the user is willing to accept.

For example, if a user wanted to add a column to a table, but only if the operation used an algorithm that is at least as efficient as the `INPLACE`, then they could execute the following:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50)
);

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab ADD COLUMN c varchar(50);
```

In MariaDB 10.3 and later, the above operation would actually use the `INSTANT` algorithm, because the `ADD COLUMN` operation supports the `INSTANT` algorithm, and the `INSTANT` algorithm is more efficient than the `INPLACE` algorithm.

MariaDB until 10.2

In MariaDB 10.2 and before, the old_alter_table system variable can be used to specify whether the `COPY` algorithm should be used.

For example, if a user wanted to add a column to a table, but they wanted to use the `COPY` algorithm instead of the default algorithm for the operation, then they could execute the following:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50)
);

SET SESSION old_alter_table=1;
ALTER TABLE tab ADD COLUMN c varchar(50);
```

Supported Alter Algorithms

The supported algorithms are described in more details below.

DEFAULT Algorithm

The default behavior, which occurs if `ALGORITHM=DEFAULT` is specified, or if `ALGORITHM` is not specified at all, usually

only makes a copy if the operation doesn't support being done in-place at all. In this case, the *most efficient* available algorithm will usually be used.

This means that, if an operation supports the `INSTANT` algorithm, then it will use that algorithm by default. If an operation does not support the `INSTANT` algorithm, but it does support the `NOCOPY` algorithm, then it will use that algorithm by default. If an operation does not support the `NOCOPY` algorithm, but it does support the `INPLACE` algorithm, then it will use that algorithm by default.

COPY Algorithm

The `COPY` algorithm refers to the original ALTER TABLE algorithm.

When the `COPY` algorithm is used, MariaDB essentially does the following operations:

```
-- Create a temporary table with the new definition
CREATE TEMPORARY TABLE tmp_tab (
...
);

-- Copy the data from the original table
INSERT INTO tmp_tab
   SELECT * FROM original_tab;

-- Drop the original table
DROP TABLE original_tab;

-- Rename the temporary table, so that it replaces the original one
RENAME TABLE tmp_tab TO original_tab;
```

This algorithm is very inefficient, but it is generic, so it works for all storage engines.

If the `COPY` algorithm is specified with the ALGORITHM clause or with the alter_algorithm system variable, then the `COPY` algorithm will be used even if it is not necessary. This can result in a lengthy table copy. If multiple ALTER TABLE operations are required that each require the table to be rebuilt, then it is best to specify all operations in a single ALTER TABLE statement, so that the table is only rebuilt once.

Using the COPY Algorithm with InnoDB

If the `COPY` algorithm is used with an InnoDB table, then the following statements apply:

- The table will be rebuilt using the current values of the innodb_file_per_table, innodb_file_format, and innodb_default_row_format system variables.
- The operation will have to create a temporary table to perform the the table copy. This temporary table will be in the same directory as the original table, and it's file name will be in the format `#sql${PID}_${THREAD_ID}_${TMP_TABLE_COUNT}`, where `${PID}` is the process ID of `mysqld`, `${THREAD_ID}` is the connection ID, and `${TMP_TABLE_COUNT}` is the number of temporary tables that the connection has open. Therefore, the datadir may contain files with file names like `#sql1234_12_1.ibd`.
- The operation inserts one record at a time into each index, which is very inefficient.
- InnoDB does not use a sort buffer.
- In MariaDB 10.2.13, MariaDB 10.3.5 and later, the table copy operation creates a lot fewer InnoDB undo log writes. See MDEV-11415 for more information.
- The table copy operation creates a lot of InnoDB redo log writes.

INPLACE Algorithm

The `COPY` algorithm can be incredibly slow, because the whole table has to be copied and rebuilt. The `INPLACE` algorithm was introduced as a way to avoid this by performing operations in-place and avoiding the table copy and rebuild, when possible.

When the `INPLACE` algorithm is used, the underlying storage engine uses optimizations to perform the operation while avoiding the table copy and rebuild. However, `INPLACE` is a bit of a misnomer, since some operations may still require the table to be rebuilt for some storage engines. Regardless, several operations can be performed without a full copy of the table for some storage engines.

A more accurate name for the algorithm would have been the `ENGINE` algorithm, since the storage engine decides how to implement the algorithm.

If an ALTER TABLE operation supports the INPLACE algorithm, then it can be performed using optimizations by the underlying storage engine, but it may rebuilt.

If the INPLACE algorithm is specified with the ALGORITHM clause or with the alter_algorithm system variable and if the ALTER TABLE operation does not support the INPLACE algorithm, then an error will be raised. For example:

```
SET SESSION alter_algorithm='INPLACE';

ALTER TABLE tab MODIFY COLUMN c int;
ERROR 1846 (0A000): ALGORITHM=INPLACE is not supported. Reason: Cannot change column type
INPLACE. Try ALGORITHM=COPY
```

In this case, raising an error is preferable, if the alternative is for the operation to make a copy of the table, and perform unexpectedly slowly.

Using the INPLACE Algorithm with InnoDB

If the INPLACE algorithm is used with an InnoDB table, then the following statements apply:

- The operation might have to write sort files in the directory defined by the innodb_tmpdir system variable.
- The operation might also have to write a temporary log file to track data changes by DML queries executed during the operation. The maximum size for this log file is configured by the innodb_online_alter_log_max_size system variable.
- Some operations require the table to be rebuilt, even though the algorithm is inaccurately called "in-place". This includes operations such as adding or dropping columns, adding a primary key, changing a column to NULL, etc.
- If the operation requires the table to be rebuilt, then the operation might have to create temporary tables.
  - It may have to create a temporary intermediate table for the actual table rebuild operation.
    - In MariaDB 10.2.19 and later, this temporary table will be in the same directory as the original table, and it's file name will be in the format `#sql${PID}_${THREAD_ID}_${TMP_TABLE_COUNT}`, where `${PID}` is the process ID of `mysqld`, `${THREAD_ID}` is the connection ID, and `${TMP_TABLE_COUNT}` is the number of temporary tables that the connection has open. Therefore, the datadir may contain files with file names like `#sql1234_12_1.ibd`.
    - In MariaDB 10.2.18 and before, this temporary table will be in the same directory as the original table, and it's file name will be in the format `#sql-ib${TABLESPACE_ID}-${RAND}`, where `${TABLESPACE_ID}` is the table's tablespace ID within InnoDB and `${RAND}` is a randomly initialized number. Therefore, the datadir may contain files with file names like `#sql-ib230291-1363966925.ibd`.
  - When it replaces the original table with the rebuilt table, it may also have to rename the original table using a temporary table name.
    - If the server is MariaDB 10.3 or later or if it is running MariaDB 10.2 and the innodb_safe_truncate system variable is set to `OFF`, then the format will actually be `#sql-ib${TABLESPACE_ID}-${RAND}`, where `${TABLESPACE_ID}` is the table's tablespace ID within InnoDB and `${RAND}` is a randomly initialized number. Therefore, the datadir may contain files with file names like `#sql-ib230291-1363966925.ibd`.
    - If the server is running MariaDB 10.1 or before or if it is running MariaDB 10.2 and the innodb_safe_truncate system variable is set to `ON`, then the renamed table will have a temporary table name in the format `#sql-ib${TABLESPACE_ID}`, where `${TABLESPACE_ID}` is the table's tablespace ID within InnoDB. Therefore, the datadir may contain files with file names like `#sql-ib230291.ibd`.
- The storage needed for the above items can add up to the size of the original table, or more in some cases.
- Some operations are instantaneous, if they only require the table's metadata to be changed. This includes operations such as renaming a column, changing a column's DEFAULT value, etc.

Operations Supported by InnoDB with the INPLACE Algorithm

With respect to the allowed operations, the INPLACE algorithm supports a subset of the operations supported by the COPY algorithm, and it supports a superset of the operations supported by the NOCOPY algorithm.

See InnoDB Online DDL Operations with ALGORITHM=INPLACE for more information.

NOCOPY Algorithm

MariaDB starting with 10.3

In MariaDB 10.3 and later, the NOCOPY algorithm is supported.

The `INPLACE` algorithm can sometimes be surprisingly slow in instances where it has to rebuild the clustered index, because when the clustered index has to be rebuilt, the whole table has to be rebuilt. The `NOCOPY` algorithm was introduced as a way to avoid this.

If an ALTER TABLE operation supports the `NOCOPY` algorithm, then it can be performed without rebuilding the clustered index.

If the `NOCOPY` algorithm is specified with the ALGORITHM clause or with the alter_algorithm system variable and if the ALTER TABLE operation does not support the `NOCOPY` algorithm, then an error will be raised. For example:

```
SET SESSION alter_algorithm='NOCOPY';

ALTER TABLE tab MODIFY COLUMN c int;
ERROR 1846 (0A000): ALGORITHM=NOCOPY is not supported. Reason: Cannot change column type INPLACE.
```

In this case, raising an error is preferable, if the alternative is for the operation to rebuild the clustered index, and perform unexpectedly slowly.

Operations Supported by InnoDB with the `NOCOPY` Algorithm

With respect to the allowed operations, the `NOCOPY` algorithm supports a subset of the operations supported by the `INPLACE` algorithm, and it supports a superset of the operations supported by the `INSTANT` algorithm.

See InnoDB Online DDL Operations with ALGORITHM=NOCOPY for more information.

INSTANT Algorithm

MariaDB starting with 10.3

In MariaDB 10.3 and later, the `INSTANT` algorithm is supported.

The `INPLACE` algorithm can sometimes be surprisingly slow in instances where it has to modify data files. The `INSTANT` algorithm was introduced as a way to avoid this.

If an ALTER TABLE operation supports the `INSTANT` algorithm, then it can be performed without modifying any data files.

If the `INSTANT` algorithm is specified with the ALGORITHM clause or with the alter_algorithm system variable and if the ALTER TABLE operation does not support the `INSTANT` algorithm, then an error will be raised. For example:

```
SET SESSION alter_algorithm='INSTANT';

ALTER TABLE tab MODIFY COLUMN c int;
ERROR 1846 (0A000): ALGORITHM=INSTANT is not supported. Reason: Cannot change column type
INPLACE. Try ALGORITHM=COPY
```

In this case, raising an error is preferable, if the alternative is for the operation to modify data files, and perform unexpectedly slowly.

Operations Supported by InnoDB with the INSTANT Algorithm

With respect to the allowed operations, the `INSTANT` algorithm supports a subset of the operations supported by the `NOCOPY` algorithm.

See InnoDB Online DDL Operations with ALGORITHM=INSTANT for more information.

Alter Locking Strategies

InnoDB supports multiple locking strategies for performing DDL operations. This offers a significant performance improvement over previous versions. The supported locking strategies are:

- `DEFAULT` - This implies the default behavior for the specific operation.
- `NONE`
- `SHARED`
- `EXCLUSIVE`

Regardless of which locking strategy is used to perform a DDL operation, InnoDB will have to exclusively lock the table for a

short time at the start and end of the operation's execution. This means that any active transactions that may have accessed the table must be committed or aborted for the operation to continue. This applies to most DDL statements, such as ALTER TABLE, CREATE INDEX, DROP INDEX, OPTIMIZE TABLE, RENAME TABLE, etc.

Specifying an Alter Locking Strategy

Specifying an Alter Locking Strategy Using the `LOCK` Clause

The ALTER TABLE statement supports the LOCK clause.

The LOCK clause can be used to specify the locking strategy that the user is willing to accept. It is supported by the ALTER TABLE and CREATE INDEX statements.

For example, if a user wanted to add a column to a table, but only if the operation is non-locking, then they could execute the following:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50)
);

ALTER TABLE tab ADD COLUMN c varchar(50), ALGORITHM=INPLACE, LOCK=NONE;
```

If the LOCK clause is not explicitly set, then the operation uses `LOCK=DEFAULT`.

Specifying an Alter Locking Strategy Using `ALTER ONLINE TABLE`

ALTER ONLINE TABLE is equivalent to `LOCK=NONE`. Therefore, the ALTER ONLINE TABLE statement can be used to ensure that your ALTER TABLE operation allows all concurrent DML.

Supported Alter Locking Strategies

The supported algorithms are described in more details below.

To see which locking strategies InnoDB supports for each operation, see the pages that describe which operations are supported for each algorithm:

- InnoDB Online DDL Operations with ALGORITHM=INPLACE
- InnoDB Online DDL Operations with ALGORITHM=NOCOPY
- InnoDB Online DDL Operations with ALGORITHM=INSTANT

DEFAULT Locking Strategy

The default behavior, which occurs if `LOCK=DEFAULT` is specified, or if `LOCK` is not specified at all, acquire the least restrictive lock on the table that is supported for the specific operation. This permits the maximum amount of concurrency that is supported for the specific operation.

NONE Locking Strategy

The `NONE` locking strategy performs the operation without acquiring any lock on the table. This permits **all** concurrent DML.

If this locking strategy is not permitted for an operation, then an error is raised.

SHARED Locking Strategy

The `SHARED` locking strategy performs the operation after acquiring a read lock on the table. This permit **read-only** concurrent DML.

If this locking strategy is not permitted for an operation, then an error is raised.

EXCLUSIVE Locking Strategy

The `EXCLUSIVE` locking strategy performs the operation after acquiring a write lock on the table. This does **not** permit concurrent DML.

5.3.2.19.2 InnoDB Online DDL Operations with the INPLACE Alter Algorithm

Contents

Supported Operations by Inheritance

When the ALGORITHM clause is set to `INPLACE`, the supported operations are a superset of the operations that are supported when the ALGORITHM clause is set to `NOCOPY`. Similarly, when the ALGORITHM clause is set to `NOCOPY`, the supported operations are a superset of the operations that are supported when the ALGORITHM clause is set to `INSTANT`.

Therefore, when the ALGORITHM clause is set to `INPLACE`, some operations are supported by inheritance. See the following additional pages for more information about these supported operations:

- InnoDB Online DDL Operations with ALGORITHM=NOCOPY
- InnoDB Online DDL Operations with ALGORITHM=INSTANT

Column Operations

`ALTER TABLE ... ADD COLUMN`

InnoDB supports adding columns to a table with ALGORITHM set to `INPLACE`.

The table is rebuilt, which means that all of the data is reorganized substantially, and the indexes are rebuilt. As a result, the

operation is quite expensive.

With the exception of adding an auto-increment column, this operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50)
);

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab ADD COLUMN c varchar(50);
Query OK, 0 rows affected (0.006 sec)
```

This applies to ALTER TABLE ... ADD COLUMN for InnoDB tables.

ALTER TABLE ... DROP COLUMN

InnoDB supports dropping columns from a table with ALGORITHM set to `INPLACE`.

The table is rebuilt, which means that all of the data is reorganized substantially, and the indexes are rebuilt. As a result, the operation is quite expensive.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab DROP COLUMN c;
Query OK, 0 rows affected (0.021 sec)
```

This applies to ALTER TABLE ... DROP COLUMN for InnoDB tables.

ALTER TABLE ... MODIFY COLUMN

This applies to ALTER TABLE ... MODIFY COLUMN for InnoDB tables.

Reordering Columns

InnoDB supports reordering columns within a table with ALGORITHM set to `INPLACE`.

The table is rebuilt, which means that all of the data is reorganized substantially, and the indexes are rebuilt. As a result, the operation is quite expensive.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab MODIFY COLUMN c varchar(50) AFTER a;
Query OK, 0 rows affected (0.022 sec)
```

Changing the Data Type of a Column

InnoDB does **not** support modifying a column's data type with ALGORITHM set to `INPLACE` in most cases. There are some exceptions:

- In MariaDB 10.2.2 and later, InnoDB supports increasing the length of `VARCHAR` columns with ALGORITHM set to `INPLACE`, unless it would require changing the number of bytes requires to represent the column's length. A `VARCHAR` column that is between 0 and 255 bytes in size requires 1 byte to represent its length, while a `VARCHAR` column that is 256 bytes or longer requires 2 bytes to represent its length. This means that the length of a column cannot be increased with ALGORITHM set to `INPLACE` if the original length was less than 256 bytes, and the new length is 256 bytes or more.
- In MariaDB 10.4.3 and later, InnoDB supports increasing the length of `VARCHAR` columns with ALGORITHM set to `INPLACE` in the cases where the operation supports having the ALGORITHM clause set to `INSTANT`.

See InnoDB Online DDL Operations with ALGORITHM=INSTANT: Changing the Data Type of a Column for more information.

For example, this fails:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab MODIFY COLUMN c int;
ERROR 1846 (0A000): ALGORITHM=INPLACE is not supported. Reason: Cannot change column type
INPLACE. Try ALGORITHM=COPY
```

But this succeeds in MariaDB 10.2.2 and later, because the original length of the column is less than 256 bytes, and the new length is still less than 256 bytes:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
) CHARACTER SET=latin1;

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab MODIFY COLUMN c varchar(100);
Query OK, 0 rows affected (0.005 sec)
```

But this fails in MariaDB 10.2.2 and later, because the original length of the column is less than 256 bytes, and the new length is greater than 256 bytes:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(255)
) CHARACTER SET=latin1;

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab MODIFY COLUMN c varchar(256);
ERROR 1846 (0A000): ALGORITHM=INPLACE is not supported. Reason: Cannot change column type
INPLACE. Try ALGORITHM=COPY
```

Changing a Column to NULL

InnoDB supports modifying a column to allow NULL values with ALGORITHM set to `INPLACE`.

The table is rebuilt, which means that all of the data is reorganized substantially, and the indexes are rebuilt. As a result, the operation is quite expensive.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50) NOT NULL
);

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab MODIFY COLUMN c varchar(50) NULL;
Query OK, 0 rows affected (0.021 sec)
```

Changing a Column to NOT NULL

InnoDB supports modifying a column to **not** allow NULL values with ALGORITHM set to `INPLACE`. It is required for strict mode to be enabled in SQL_MODE. The operation will fail if the column contains any `NULL` values. Changes that would interfere with referential integrity are also not permitted.

The table is rebuilt, which means that all of the data is reorganized substantially, and the indexes are rebuilt. As a result, the operation is quite expensive.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab MODIFY COLUMN c varchar(50) NOT NULL;
Query OK, 0 rows affected (0.021 sec)
```

Adding a New `ENUM` Option

InnoDB supports adding a new ENUM option to a column with ALGORITHM set to `INPLACE`. In order to add a new ENUM option with ALGORITHM set to `INPLACE`, the following requirements must be met:

- It must be added to the end of the list.
- The storage requirements must not change.

This operation only changes the table's metadata, so the table does not have to be rebuilt..

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example, this succeeds:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c ENUM('red', 'green')
);

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab MODIFY COLUMN c ENUM('red', 'green', 'blue');
Query OK, 0 rows affected (0.004 sec)
```

But this fails:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c ENUM('red', 'green')
);

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab MODIFY COLUMN c ENUM('red', 'blue', 'green');
ERROR 1846 (0A000): ALGORITHM=INPLACE is not supported. Reason: Cannot change column type
INPLACE. Try ALGORITHM=COPY
```

Adding a New `SET` Option

InnoDB supports adding a new SET option to a column with ALGORITHM set to `INPLACE`. In order to add a new SET option with ALGORITHM set to `INPLACE`, the following requirements must be met:

- It must be added to the end of the list.
- The storage requirements must not change.

This operation only changes the table's metadata, so the table does not have to be rebuilt..

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example, this succeeds:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c SET('red', 'green')
);

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab MODIFY COLUMN c SET('red', 'green', 'blue');
Query OK, 0 rows affected (0.004 sec)
```

But this fails:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c SET('red', 'green')
);

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab MODIFY COLUMN c SET('red', 'blue', 'green');
ERROR 1846 (0A000): ALGORITHM=INPLACE is not supported. Reason: Cannot change column type
INPLACE. Try ALGORITHM=COPY
```

Removing System Versioning from a Column

In MariaDB 10.3.8 and later, InnoDB supports removing system versioning from a column with ALGORITHM set to `INPLACE`. In order for this to work, the system_versioning_alter_history system variable must be set to `KEEP`. See MDEV-16330 for more information.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50) WITH SYSTEM VERSIONING
);

SET SESSION system_versioning_alter_history='KEEP';
SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab MODIFY COLUMN c varchar(50) WITHOUT SYSTEM VERSIONING;
Query OK, 0 rows affected (0.005 sec)
```

`ALTER TABLE ... ALTER COLUMN`

This applies to ALTER TABLE ... ALTER COLUMN for InnoDB tables.

Setting a Column's Default Value

InnoDB supports modifying a column's DEFAULT value with ALGORITHM set to `INPLACE`.

This operation only changes the table's metadata, so the table does not have to be rebuilt.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to

`NONE` . When this strategy is used, all concurrent DML is permitted. For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab ALTER COLUMN c SET DEFAULT 'No value explicitly provided.';
Query OK, 0 rows affected (0.005 sec)
```

Removing a Column's Default Value

InnoDB supports removing a column's DEFAULT value with ALGORITHM set to `INPLACE` .

This operation only changes the table's metadata, so the table does not have to be rebuilt.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE` . When this strategy is used, all concurrent DML is permitted.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50) DEFAULT 'No value explicitly provided.'
);

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab ALTER COLUMN c DROP DEFAULT;
Query OK, 0 rows affected (0.005 sec)
```

ALTER TABLE ... CHANGE COLUMN

InnoDB supports renaming a column with ALGORITHM set to `INPLACE` , unless the column's data type or attributes changed in addition to the name.

This operation only changes the table's metadata, so the table does not have to be rebuilt.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE` . When this strategy is used, all concurrent DML is permitted.

For example, this succeeds:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab CHANGE COLUMN c str varchar(50);
Query OK, 0 rows affected (0.006 sec)
```

But this fails:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab CHANGE COLUMN c num int;
ERROR 1846 (0A000): ALGORITHM=INPLACE is not supported. Reason: Cannot change column type
INPLACE. Try ALGORITHM=COPY
```

This applies to ALTER TABLE ... CHANGE COLUMN for InnoDB tables.

Index Operations

ALTER TABLE ... ADD PRIMARY KEY

InnoDB supports adding a primary key to a table with ALGORITHM set to `INPLACE`.

If the new primary key column is not defined as NOT NULL, then it is highly recommended for strict mode to be enabled in SQL_MODE. Otherwise, `NULL` values will be silently converted to the default value for the given data type, which is probably not the desired behavior in this scenario.

The table is rebuilt, which means that all of the data is reorganized substantially, and the indexes are rebuilt. As a result, the operation is quite expensive.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example, this succeeds:

```
CREATE OR REPLACE TABLE tab (
   a int,
   b varchar(50),
   c varchar(50)
);

SET SESSION sql_mode='STRICT_TRANS_TABLES';
SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab ADD PRIMARY KEY (a);
Query OK, 0 rows affected (0.021 sec)
```

But this fails:

```
CREATE OR REPLACE TABLE tab (
   a int,
   b varchar(50),
   c varchar(50)
);

INSERT INTO tab VALUES (NULL, NULL, NULL);

SET SESSION sql_mode='STRICT_TRANS_TABLES';
SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab ADD PRIMARY KEY (a);
ERROR 1265 (01000): Data truncated for column 'a' at row 1
```

And this fails:

```
CREATE OR REPLACE TABLE tab (
   a int,
   b varchar(50),
   c varchar(50)
);

INSERT INTO tab VALUES (1, NULL, NULL);
INSERT INTO tab VALUES (1, NULL, NULL);

SET SESSION sql_mode='STRICT_TRANS_TABLES';
SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab ADD PRIMARY KEY (a);
ERROR 1062 (23000): Duplicate entry '1' for key 'PRIMARY'
```

This applies to ALTER TABLE ... ADD PRIMARY KEY for InnoDB tables.

ALTER TABLE ... DROP PRIMARY KEY

InnoDB does **not** support dropping a primary key with ALGORITHM set to `INPLACE` in most cases.

If you try to do so, then you will see an error. InnoDB only supports this operation with ALGORITHM set to `COPY`. Concurrent DML is *not* permitted.

However, there is an exception. If you are dropping a primary key, and adding a new one at the same time, then that

operation can be performed with ALGORITHM set to `INPLACE`. This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example, this fails:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab DROP PRIMARY KEY;
ERROR 1846 (0A000): ALGORITHM=INPLACE is not supported. Reason: Dropping a primary key is not
allowed without also adding a new primary key. Try ALGORITHM=COPY
```

But this succeeds:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION sql_mode='STRICT_TRANS_TABLES';
SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab DROP PRIMARY KEY, ADD PRIMARY KEY (b);
Query OK, 0 rows affected (0.020 sec)
```

This applies to ALTER TABLE ... DROP PRIMARY KEY for InnoDB tables.

`ALTER TABLE ... ADD INDEX` and `CREATE INDEX`

This applies to ALTER TABLE ... ADD INDEX and CREATE INDEX for InnoDB tables.

Adding a Plain Index

InnoDB supports adding a plain index to a table with ALGORITHM set to `INPLACE`. The table is not rebuilt.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example, this succeeds:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab ADD INDEX b_index (b);
Query OK, 0 rows affected (0.010 sec)
```

And this succeeds:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INPLACE';
CREATE INDEX b_index ON tab (b);
Query OK, 0 rows affected (0.011 sec)
```

Adding a Fulltext Index

InnoDB supports adding a FULLTEXT index to a table with ALGORITHM set to `INPLACE`. The table is not rebuilt in some cases.

However, there are some limitations, such as:

- Adding a FULLTEXT index to a table that does not have a user-defined `FTS_DOC_ID` column will require the table to be rebuilt once. When the table is rebuilt, the system adds a hidden `FTS_DOC_ID` column. From that point forward, adding additional FULLTEXT indexes to the same table will not require the table to be rebuilt when ALGORITHM is set to `INPLACE`.
- Only one FULLTEXT index may be added at a time when ALGORITHM is set to `INPLACE`.
- If a table has more than one FULLTEXT index, then it cannot be rebuilt by any ALTER TABLE operations when ALGORITHM is set to `INPLACE`.
- If a table has a FULLTEXT index, then it cannot be rebuilt by any ALTER TABLE operations when the LOCK clause is set to `NONE`.

This operation supports a read-only locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `SHARED`. When this strategy is used, read-only concurrent DML is permitted.

For example, this succeeds, but requires the table to be rebuilt, so that the hidden `FTS_DOC_ID` column can be added:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab ADD FULLTEXT INDEX b_index (b);
Query OK, 0 rows affected (0.055 sec)
```

And this succeeds in the same way as above:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INPLACE';
CREATE FULLTEXT INDEX b_index ON tab (b);
Query OK, 0 rows affected (0.041 sec)
```

And this succeeds, and the second command does not require the table to be rebuilt:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab ADD FULLTEXT INDEX b_index (b);
Query OK, 0 rows affected (0.043 sec)

ALTER TABLE tab ADD FULLTEXT INDEX c_index (c);
Query OK, 0 rows affected (0.017 sec)
```

But this second command fails, because only one FULLTEXT index can be added at a time:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50),
   d varchar(50)
);

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab ADD FULLTEXT INDEX b_index (b);
Query OK, 0 rows affected (0.041 sec)

ALTER TABLE tab ADD FULLTEXT INDEX c_index (c), ADD FULLTEXT INDEX d_index (d);
ERROR 1846 (0A000): ALGORITHM=INPLACE is not supported. Reason: InnoDB presently supports one
FULLTEXT index creation at a time. Try ALGORITHM=COPY
```

And this third command fails, because a table cannot be rebuilt when it has more than one FULLTEXT index:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab ADD FULLTEXT INDEX b_index (b);
Query OK, 0 rows affected (0.040 sec)

ALTER TABLE tab ADD FULLTEXT INDEX c_index (c);
Query OK, 0 rows affected (0.015 sec)

ALTER TABLE tab FORCE;
ERROR 1846 (0A000): ALGORITHM=INPLACE is not supported. Reason: InnoDB presently supports one
FULLTEXT index creation at a time. Try ALGORITHM=COPY
```

Adding a Spatial Index

InnoDB supports adding a SPATIAL index to a table with ALGORITHM set to `INPLACE`.

However, there are some limitations, such as:

- If a table has a SPATIAL index, then it cannot be rebuilt by any ALTER TABLE operations when the LOCK clause is set to `NONE`.

This operation supports a read-only locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `SHARED`. When this strategy is used, read-only concurrent DML is permitted.

For example, this succeeds:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c GEOMETRY NOT NULL
);

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab ADD SPATIAL INDEX c_index (c);
Query OK, 0 rows affected (0.006 sec)
```

And this succeeds in the same way as above:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c GEOMETRY NOT NULL
);

SET SESSION alter_algorithm='INPLACE';
CREATE SPATIAL INDEX c_index ON tab (c);
Query OK, 0 rows affected (0.006 sec)
```

ALTER TABLE ... DROP INDEX and DROP INDEX

InnoDB supports dropping indexes from a table with ALGORITHM set to `INPLACE`.

This operation only changes the table's metadata, so the table does not have to be rebuilt.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example, this succeeds:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50),
   INDEX b_index (b)
);

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab DROP INDEX b_index;
```

And this succeeds:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50),
   INDEX b_index (b)
);

SET SESSION alter_algorithm='INPLACE';
DROP INDEX b_index ON tab;
```

This applies to ALTER TABLE ... DROP INDEX and DROP INDEX for InnoDB tables.

ALTER TABLE ... ADD FOREIGN KEY

InnoDB supports adding foreign key constraints to a table with ALGORITHM set to `INPLACE`. In order to add a new foreign key constraint to a table with ALGORITHM set to `INPLACE`, the foreign_key_checks system variable needs to be set to `OFF`. If it is set to `ON`, then `ALGORITHM=COPY` is required.

This operation only changes the table's metadata, so the table does not have to be rebuilt.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example, this fails:

```
CREATE OR REPLACE TABLE tab1 (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50),
   d int
);

CREATE OR REPLACE TABLE tab2 (
   a int PRIMARY KEY,
   b varchar(50)
);

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab1 ADD FOREIGN KEY tab2_fk (d) REFERENCES tab2 (a);
ERROR 1846 (0A000): ALGORITHM=INPLACE is not supported. Reason: Adding foreign keys needs
foreign_key_checks=OFF. Try ALGORITHM=COPY
```

But this succeeds:

```
CREATE OR REPLACE TABLE tab1 (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50),
   d int
);

CREATE OR REPLACE TABLE tab2 (
   a int PRIMARY KEY,
   b varchar(50)
);

SET SESSION foreign_key_checks=OFF;
SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab1 ADD FOREIGN KEY tab2_fk (d) REFERENCES tab2 (a);
Query OK, 0 rows affected (0.011 sec)
```

This applies to ALTER TABLE ... ADD FOREIGN KEY for InnoDB tables.

ALTER TABLE ... DROP FOREIGN KEY

InnoDB supports dropping foreign key constraints from a table with ALGORITHM set to `INPLACE`.

This operation only changes the table's metadata, so the table does not have to be rebuilt.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example:

```
CREATE OR REPLACE TABLE tab2 (
   a int PRIMARY KEY,
   b varchar(50)
);

CREATE OR REPLACE TABLE tab1 (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50),
   d int,
   FOREIGN KEY tab2_fk (d) REFERENCES tab2 (a)
);

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab1 DROP FOREIGN KEY tab2_fk;
Query OK, 0 rows affected (0.005 sec)
```

This applies to ALTER TABLE ... DROP FOREIGN KEY for InnoDB tables.

Table Operations

ALTER TABLE ... AUTO_INCREMENT=...

InnoDB supports changing a table's AUTO_INCREMENT value with ALGORITHM set to `INPLACE`. This operation should finish instantly. The table is not rebuilt.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab AUTO_INCREMENT=100;
Query OK, 0 rows affected (0.004 sec)
```

This applies to ALTER TABLE ... AUTO_INCREMENT=... for InnoDB tables.

ALTER TABLE ... ROW_FORMAT=...

InnoDB supports changing a table's row format with ALGORITHM set to `INPLACE`.

The table is rebuilt, which means that all of the data is reorganized substantially, and the indexes are rebuilt. As a result, the operation is quite expensive.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
) ROW_FORMAT=DYNAMIC;

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab ROW_FORMAT=COMPRESSED;
Query OK, 0 rows affected (0.025 sec)
```

This applies to ALTER TABLE ... ROW_FORMAT=... for InnoDB tables.

ALTER TABLE ... KEY_BLOCK_SIZE=...

InnoDB supports changing a table's KEY_BLOCK_SIZE with ALGORITHM set to `INPLACE`.

The table is rebuilt, which means that all of the data is reorganized substantially, and the indexes are rebuilt. As a result, the operation is quite expensive.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
) ROW_FORMAT=COMPRESSED
  KEY_BLOCK_SIZE=4;

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab KEY_BLOCK_SIZE=2;
Query OK, 0 rows affected (0.021 sec)
```

This applies to KEY_BLOCK_SIZE=... for InnoDB tables.

ALTER TABLE ... PAGE_COMPRESSED=... and ALTER TABLE ... PAGE_COMPRESSION_LEVEL=...

In MariaDB 10.3.10 and later, InnoDB supports setting a table's PAGE_COMPRESSED value to `1` with ALGORITHM set to `INPLACE`. InnoDB also supports changing a table's PAGE_COMPRESSED value from `1` to `0` with ALGORITHM set to `INPLACE`.

In these versions, InnoDB also supports changing a table's PAGE_COMPRESSION_LEVEL value with ALGORITHM set to

`INPLACE`.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the `LOCK` clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

See MDEV-16328 for more information.

For example, this succeeds:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab PAGE_COMPRESSED=1;
Query OK, 0 rows affected (0.006 sec)
```

And this succeeds:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
) PAGE_COMPRESSED=1;

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab PAGE_COMPRESSED=0;
Query OK, 0 rows affected (0.020 sec)
```

And this succeeds:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
) PAGE_COMPRESSED=1
  PAGE_COMPRESSION_LEVEL=5;

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab PAGE_COMPRESSION_LEVEL=4;
Query OK, 0 rows affected (0.006 sec)
```

This applies to PAGE_COMPRESSED=... and PAGE_COMPRESSION_LEVEL=... for InnoDB tables.

ALTER TABLE ... DROP SYSTEM VERSIONING

InnoDB supports dropping system versioning from a table with ALGORITHM set to `INPLACE`.

This operation supports the read-only locking strategy. This strategy can be explicitly chosen by setting the `LOCK` clause to `SHARED`. When this strategy is used, read-only concurrent DML is permitted.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
) WITH SYSTEM VERSIONING;

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab DROP SYSTEM VERSIONING;
```

This applies to ALTER TABLE ... DROP SYSTEM VERSIONING for InnoDB tables.

ALTER TABLE ... DROP CONSTRAINT

In MariaDB 10.3.6 and later, InnoDB supports dropping a CHECK constraint from a table with ALGORITHM set to `INPLACE`. See MDEV-16331 for more information.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50),
   CONSTRAINT b_not_empty CHECK (b != '')
);

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab DROP CONSTRAINT b_not_empty;
Query OK, 0 rows affected (0.004 sec)
```

This applies to ALTER TABLE ... DROP CONSTRAINT for InnoDB tables.

ALTER TABLE ... FORCE

InnoDB supports forcing a table rebuild with ALGORITHM set to `INPLACE`.

The table is rebuilt, which means that all of the data is reorganized substantially, and the indexes are rebuilt. As a result, the operation is quite expensive.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab FORCE;
Query OK, 0 rows affected (0.022 sec)
```

This applies to ALTER TABLE ... FORCE for InnoDB tables.

ALTER TABLE ... ENGINE=InnoDB

InnoDB supports forcing a table rebuild with ALGORITHM set to `INPLACE`.

The table is rebuilt, which means that all of the data is reorganized substantially, and the indexes are rebuilt. As a result, the operation is quite expensive.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab ENGINE=InnoDB;
Query OK, 0 rows affected (0.022 sec)
```

This applies to ALTER TABLE ... ENGINE=InnoDB for InnoDB tables.

OPTIMIZE TABLE ...

InnoDB supports optimizing a table with ALGORITHM set to `INPLACE`.

If the innodb_defragment system variable is set to `OFF`, and if the innodb_optimize_fulltext_only system variable is also set to `OFF`, then `OPTIMIZE TABLE` will be equivalent to `ALTER TABLE … FORCE`.

The table is rebuilt, which means that all of the data is reorganized substantially, and the indexes are rebuilt. As a result, the operation is quite expensive.

If either of the previously mentioned system variables is set to `ON`, then `OPTIMIZE TABLE` will optimize some data without rebuilding the table. However, the file size will not be reduced.

For example, this succeeds:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SHOW GLOBAL VARIABLES WHERE Variable_name IN('innodb_defragment', 'innodb_optimize_fulltext_only'
+--------------------------------+-------+
| Variable_name                  | Value |
+--------------------------------+-------+
| innodb_defragment              | OFF   |
| innodb_optimize_fulltext_only | OFF   |
+--------------------------------+-------+

SET SESSION alter_algorithm='INPLACE';
OPTIMIZE TABLE tab;
+---------+----------+----------+-------------------------------------------------------------------
| Table   | Op       | Msg_type | Msg_text
+---------+----------+----------+-------------------------------------------------------------------
| db1.tab | optimize | note     | Table does not support optimize, doing recreate + analyze inste
| db1.tab | optimize | status   | OK
+---------+----------+----------+-------------------------------------------------------------------
2 rows in set (0.026 sec)
```

And this succeeds, but the table is not rebuilt:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET GLOBAL innodb_defragment=ON;
SHOW GLOBAL VARIABLES WHERE Variable_name IN('innodb_defragment', 'innodb_optimize_fulltext_only'
+--------------------------------+-------+
| Variable_name                  | Value |
+--------------------------------+-------+
| innodb_defragment              | ON    |
| innodb_optimize_fulltext_only | OFF   |
+--------------------------------+-------+

SET SESSION alter_algorithm='INPLACE';
OPTIMIZE TABLE tab;
+---------+----------+----------+----------+
| Table   | Op       | Msg_type | Msg_text |
+---------+----------+----------+----------+
| db1.tab | optimize | status   | OK       |
+---------+----------+----------+----------+
1 row in set (0.004 sec)
```

This applies to OPTIMIZE TABLE for InnoDB tables.

ALTER TABLE ... RENAME TO and RENAME TABLE ...

InnoDB supports renaming a table with ALGORITHM set to `INPLACE`.

This operation only changes the table's metadata, so the table does not have to be rebuilt.

This operation supports the exclusive locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to

`EXCLUSIVE`. When this strategy is used, concurrent DML is **not** permitted.

For example, this succeeds:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab RENAME TO old_tab;
Query OK, 0 rows affected (0.011 sec)
```

And this succeeds:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INPLACE';
RENAME TABLE tab TO old_tab;
```

This applies to ALTER TABLE ... RENAME TO and RENAME TABLE for InnoDB tables.

Limitations

Limitations Related to Fulltext Indexes

- If a table has more than one FULLTEXT index, then it cannot be rebuilt by any ALTER TABLE operations when ALGORITHM is set to `INPLACE`.
- If a table has a FULLTEXT index, then it cannot be rebuilt by any ALTER TABLE operations when the LOCK clause is set to `NONE`.

Limitations Related to Spatial Indexes

- If a table has a SPATIAL index, then it cannot be rebuilt by any ALTER TABLE operations when the LOCK clause is set to `NONE`.

Limitations Related to Generated (Virtual and Persistent/Stored) Columns

Generated columns do not currently support online DDL for all of the same operations that are supported for "real" columns.

See Generated (Virtual and Persistent/Stored) Columns: Statement Support for more information on the limitations.

5.3.2.19.3 InnoDB Online DDL Operations with the NOCOPY Alter Algorithm

Contents

Supported Operations by Inheritance

When the ALGORITHM clause is set to `NOCOPY`, the supported operations are a superset of the operations that are supported when the ALGORITHM clause is set to `INSTANT`.

Therefore, when the ALGORITHM clause is set to `NOCOPY`, some operations are supported by inheritance. See the following additional pages for more information about these supported operations:

- InnoDB Online DDL Operations with ALGORITHM=INSTANT

Column Operations

`ALTER TABLE ... ADD COLUMN`

In MariaDB 10.3.2 and later, InnoDB supports adding columns to a table with ALGORITHM set to `NOCOPY` in the cases where the operation supports having the ALGORITHM clause set to `INSTANT`.

See InnoDB Online DDL Operations with ALGORITHM=INSTANT: ALTER TABLE ... ADD COLUMN for more information.

This applies to ALTER TABLE ... ADD COLUMN for InnoDB tables.

`ALTER TABLE ... DROP COLUMN`

In MariaDB 10.4 and later, InnoDB supports dropping columns from a table with ALGORITHM set to `NOCOPY` in the cases where the operation supports having the ALGORITHM clause set to `INSTANT`.

See InnoDB Online DDL Operations with ALGORITHM=INSTANT: ALTER TABLE ... DROP COLUMN for more information.

This applies to ALTER TABLE ... DROP COLUMN for InnoDB tables.

`ALTER TABLE ... MODIFY COLUMN`

This applies to ALTER TABLE ... MODIFY COLUMN for InnoDB tables.

Reordering Columns

In MariaDB 10.4 and later, InnoDB supports reordering columns within a table with ALGORITHM set to `NOCOPY` in the cases where the operation supports having the ALGORITHM clause set to `INSTANT`.

See InnoDB Online DDL Operations with ALGORITHM=INSTANT: Reordering Columns for more information.

Changing the Data Type of a Column

InnoDB does **not** support modifying a column's data type with ALGORITHM set to `NOCOPY` in most cases. There are a few exceptions in the cases where the operation supports having the ALGORITHM clause set to `INSTANT`.

See InnoDB Online DDL Operations with ALGORITHM=INSTANT: Changing the Data Type of a Column for more information.

Changing a Column to NULL

In MariaDB 10.4.3 and later, InnoDB supports modifying a column to allow NULL values with ALGORITHM set to `NOCOPY` in the cases where the operation supports having the ALGORITHM clause set to `INSTANT`.

See InnoDB Online DDL Operations with ALGORITHM=INSTANT: Changing a Column to NULL for more information.

Changing a Column to NOT NULL

InnoDB does **not** support modifying a column to **not** allow NULL values with ALGORITHM set to `NOCOPY`.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
) ROW_FORMAT=REDUNDANT;

SET SESSION alter_algorithm='NOCOPY';
ALTER TABLE tab MODIFY COLUMN c varchar(50) NOT NULL;
ERROR 1845 (0A000): ALGORITHM=NOCOPY is not supported for this operation. Try ALGORITHM=INPLACE
```

Adding a New `ENUM` Option

InnoDB supports adding a new ENUM option to a column with ALGORITHM set to `NOCOPY` in the cases where the operation supports having the ALGORITHM clause set to `INSTANT`.

See InnoDB Online DDL Operations with ALGORITHM=INSTANT: Adding a New ENUM Option for more information.

Adding a New `SET` Option

InnoDB supports adding a new SET option to a column with ALGORITHM set to `NOCOPY` in the cases where the operation supports having the ALGORITHM clause set to `INSTANT`.

See InnoDB Online DDL Operations with ALGORITHM=INSTANT: Adding a New SET Option for more information.

Removing System Versioning from a Column

In MariaDB 10.3.8 and later, InnoDB supports removing system versioning from a column with ALGORITHM set to `NOCOPY` in the cases where the operation supports having the ALGORITHM clause set to `INSTANT`.

See InnoDB Online DDL Operations with ALGORITHM=INSTANT: Removing System Versioning from a Column for more information.

ALTER TABLE ... ALTER COLUMN

This applies to ALTER TABLE ... ALTER COLUMN for InnoDB tables.

Setting a Column's Default Value

InnoDB supports modifying a column's DEFAULT value with ALGORITHM set to `NOCOPY` in the cases where the operation supports having the ALGORITHM clause set to `INSTANT`.

See InnoDB Online DDL Operations with ALGORITHM=INSTANT: Setting a Column's Default Value for more information.

Removing a Column's Default Value

InnoDB supports removing a column's DEFAULT value with ALGORITHM set to `NOCOPY` in the cases where the operation supports having the ALGORITHM clause set to `INSTANT`.

See InnoDB Online DDL Operations with ALGORITHM=INSTANT: Removing a Column's Default Value for more information.

ALTER TABLE ... CHANGE COLUMN

InnoDB supports renaming a column with ALGORITHM set to `NOCOPY` in the cases where the operation supports having the ALGORITHM clause set to `INSTANT`.

See InnoDB Online DDL Operations with ALGORITHM=INSTANT: ALTER TABLE ... CHANGE COLUMN for more information.

This applies to ALTER TABLE ... CHANGE COLUMN for InnoDB tables.

Index Operations

ALTER TABLE ... ADD PRIMARY KEY

InnoDB does **not** support adding a primary key to a table with ALGORITHM set to `NOCOPY`.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int,
   b varchar(50),
   c varchar(50)
);

SET SESSION sql_mode='STRICT_TRANS_TABLES';
SET SESSION alter_algorithm='NOCOPY';
ALTER TABLE tab ADD PRIMARY KEY (a);
ERROR 1845 (0A000): ALGORITHM=NOCOPY is not supported for this operation. Try ALGORITHM=INPLACE
```

This applies to ALTER TABLE ... ADD PRIMARY KEY for InnoDB tables.

ALTER TABLE ... DROP PRIMARY KEY

InnoDB does **not** support dropping a primary key with ALGORITHM set to `NOCOPY`.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='NOCOPY';
ALTER TABLE tab DROP PRIMARY KEY;
ERROR 1846 (0A000): ALGORITHM=NOCOPY is not supported. Reason: Dropping a primary key is not
allowed without also adding a new primary key. Try ALGORITHM=COPY
```

This applies to ALTER TABLE ... DROP PRIMARY KEY for InnoDB tables.

`ALTER TABLE ... ADD INDEX` and `CREATE INDEX`

This applies to ALTER TABLE ... ADD INDEX and CREATE INDEX for InnoDB tables.

Adding a Plain Index

InnoDB supports adding a plain index to a table with ALGORITHM set to `NOCOPY`.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example, this succeeds:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='NOCOPY';
ALTER TABLE tab ADD INDEX b_index (b);
Query OK, 0 rows affected (0.009 sec)
```

And this succeeds:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='NOCOPY';
CREATE INDEX b_index ON tab (b);
Query OK, 0 rows affected (0.009 sec)
```

Adding a Fulltext Index

InnoDB supports adding a FULLTEXT index to a table with ALGORITHM set to `NOCOPY`.

However, there are some limitations, such as:

- Adding a FULLTEXT index to a table that does not have a user-defined `FTS_DOC_ID` column will require the table to be rebuilt once. When the table is rebuilt, the system adds a hidden `FTS_DOC_ID` column. This initial operation will have to be performed with ALGORITHM set to `INPLACE` .From that point forward, adding additional FULLTEXT indexes to the same table will not require the table to be rebuilt, and ALGORITHM can be set to `NOCOPY`.
- Only one FULLTEXT index may be added at a time when ALGORITHM is set to `NOCOPY`.

This operation supports a read-only locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `SHARED`. When this strategy is used, read-only concurrent DML is permitted.

For example, this succeeds, but the first operation requires the table to be rebuilt ALGORITHM set to `INPLACE`, so that the hidden `FTS_DOC_ID` column can be added:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab ADD FULLTEXT INDEX b_index (b);
Query OK, 0 rows affected (0.043 sec)

SET SESSION alter_algorithm='NOCOPY';
ALTER TABLE tab ADD FULLTEXT INDEX c_index (c);
Query OK, 0 rows affected (0.017 sec)
```

And this succeeds in the same way as above:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INPLACE';
CREATE FULLTEXT INDEX b_index ON tab (b);
Query OK, 0 rows affected (0.048 sec)

SET SESSION alter_algorithm='NOCOPY';
CREATE FULLTEXT INDEX c_index ON tab (c);
Query OK, 0 rows affected (0.016 sec)
```

But this second command fails, because only one FULLTEXT index can be added at a time:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50),
   d varchar(50)
);

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab ADD FULLTEXT INDEX b_index (b);
Query OK, 0 rows affected (0.041 sec)

SET SESSION alter_algorithm='NOCOPY';
ALTER TABLE tab ADD FULLTEXT INDEX c_index (c), ADD FULLTEXT INDEX d_index (d);
ERROR 1846 (0A000): ALGORITHM=NOCOPY is not supported. Reason: InnoDB presently supports one
FULLTEXT index creation at a time. Try ALGORITHM=COPY
```

Adding a Spatial Index

InnoDB supports adding a SPATIAL index to a table with ALGORITHM set to `NOCOPY`.

This operation supports a read-only locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `SHARED`. When this strategy is used, read-only concurrent DML is permitted.

For example, this succeeds:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c GEOMETRY NOT NULL
);

SET SESSION alter_algorithm='NOCOPY';
ALTER TABLE tab ADD SPATIAL INDEX c_index (c);
Query OK, 0 rows affected (0.005 sec)
```

And this succeeds in the same way as above:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c GEOMETRY NOT NULL
);

SET SESSION alter_algorithm='NOCOPY';
CREATE SPATIAL INDEX c_index ON tab (c);
Query OK, 0 rows affected (0.005 sec)
```

`ALTER TABLE ... DROP INDEX` and `DROP INDEX`

InnoDB supports dropping indexes from a table with ALGORITHM set to `NOCOPY` in the cases where the operation supports having the ALGORITHM clause set to `INSTANT`.

See InnoDB Online DDL Operations with ALGORITHM=INSTANT: ALTER TABLE ... DROP INDEX and DROP INDEX for more information.

This applies to ALTER TABLE ... DROP INDEX and DROP INDEX for InnoDB tables.

ALTER TABLE ... ADD FOREIGN KEY

InnoDB does supports adding foreign key constraints to a table with ALGORITHM set to `NOCOPY`. In order to add a new foreign key constraint to a table with ALGORITHM set to `NOCOPY`, the foreign_key_checks system variable needs to be set to `OFF`. If it is set to `ON`, then `ALGORITHM=COPY` is required.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example, this fails:

```
CREATE OR REPLACE TABLE tab1 (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50),
   d int
);

CREATE OR REPLACE TABLE tab2 (
   a int PRIMARY KEY,
   b varchar(50)
);

SET SESSION alter_algorithm='NOCOPY';
ALTER TABLE tab1 ADD FOREIGN KEY tab2_fk (d) REFERENCES tab2 (a);
ERROR 1846 (0A000): ALGORITHM=NOCOPY is not supported. Reason: Adding foreign keys needs
foreign_key_checks=OFF. Try ALGORITHM=COPY
```

But this succeeds:

```
CREATE OR REPLACE TABLE tab1 (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50),
   d int
);

CREATE OR REPLACE TABLE tab2 (
   a int PRIMARY KEY,
   b varchar(50)
);

SET SESSION foreign_key_checks=OFF;
SET SESSION alter_algorithm='NOCOPY';
ALTER TABLE tab1 ADD FOREIGN KEY tab2_fk (d) REFERENCES tab2 (a);
Query OK, 0 rows affected (0.011 sec)
```

This applies to ALTER TABLE ... ADD FOREIGN KEY for InnoDB tables.

ALTER TABLE ... DROP FOREIGN KEY

InnoDB supports dropping foreign key constraints from a table with ALGORITHM set to `NOCOPY` in the cases where the operation supports having the ALGORITHM clause set to `INSTANT`.

See InnoDB Online DDL Operations with ALGORITHM=INSTANT: ALTER TABLE ... DROP FOREIGN KEY for more information.

This applies to ALTER TABLE ... DROP FOREIGN KEY for InnoDB tables.

Table Operations

ALTER TABLE ... AUTO_INCREMENT=...

InnoDB supports changing a table's AUTO_INCREMENT value with ALGORITHM set to `NOCOPY` in the cases where the operation supports having the ALGORITHM clause set to `INSTANT`.

See InnoDB Online DDL Operations with ALGORITHM=INSTANT: ALTER TABLE ... AUTO_INCREMENT=... for more

information.

This applies to ALTER TABLE ... AUTO_INCREMENT=... for InnoDB tables.

ALTER TABLE ... ROW_FORMAT=...

InnoDB does **not** support changing a table's row format with ALGORITHM set to `NOCOPY`.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
) ROW_FORMAT=DYNAMIC;

SET SESSION alter_algorithm='NOCOPY';
ALTER TABLE tab ROW_FORMAT=COMPRESSED;
ERROR 1846 (0A000): ALGORITHM=NOCOPY is not supported. Reason: Changing table options requires
the table to be rebuilt. Try ALGORITHM=INPLACE
```

This applies to ALTER TABLE ... ROW_FORMAT=... for InnoDB tables.

ALTER TABLE ... KEY_BLOCK_SIZE=...

InnoDB does **not** support changing a table's KEY_BLOCK_SIZE with ALGORITHM set to `NOCOPY`.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
) ROW_FORMAT=COMPRESSED
  KEY_BLOCK_SIZE=4;

SET SESSION alter_algorithm='NOCOPY';
ALTER TABLE tab KEY_BLOCK_SIZE=2;
ERROR 1846 (0A000): ALGORITHM=NOCOPY is not supported. Reason: Changing table options requires
the table to be rebuilt. Try ALGORITHM=INPLACE
```

This applies to KEY_BLOCK_SIZE=... for InnoDB tables.

ALTER TABLE ... PAGE_COMPRESSED=1 and ALTER TABLE ... PAGE_COMPRESSION_LEVEL=...

In MariaDB 10.3.10 and later, InnoDB supports setting a table's PAGE_COMPRESSED value to `1` with ALGORITHM set to `NOCOPY` in the cases where the operation supports having the ALGORITHM clause set to `INSTANT`.

InnoDB does **not** support changing a table's PAGE_COMPRESSED value from `1` to `0` with ALGORITHM set to `NOCOPY`.

In these versions, InnoDB also supports changing a table's PAGE_COMPRESSION_LEVEL value with ALGORITHM set to `NOCOPY` in the cases where the operation supports having the ALGORITHM clause is set to `INSTANT`.

See InnoDB Online DDL Operations with ALGORITHM=INSTANT: ALTER TABLE ... PAGE_COMPRESSED=1 and ALTER TABLE ... PAGE_COMPRESSION_LEVEL=... for more information.

This applies to ALTER TABLE ... PAGE_COMPRESSED=... and ALTER TABLE ... PAGE_COMPRESSION_LEVEL=... for InnoDB tables.

ALTER TABLE ... DROP SYSTEM VERSIONING

InnoDB does **not** support dropping system versioning from a table with ALGORITHM set to `NOCOPY`.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
) WITH SYSTEM VERSIONING;

SET SESSION alter_algorithm='NOCOPY';
ALTER TABLE tab DROP SYSTEM VERSIONING;
ERROR 1845 (0A000): ALGORITHM=NOCOPY is not supported for this operation. Try ALGORITHM=INPLACE
```

This applies to ALTER TABLE ... DROP SYSTEM VERSIONING for InnoDB tables.

ALTER TABLE ... DROP CONSTRAINT

In MariaDB 10.3.6 and later, InnoDB supports dropping a CHECK constraint from a table with ALGORITHM set to `NOCOPY` in the cases where the operation supports having the ALGORITHM clause set to `INSTANT`.

See InnoDB Online DDL Operations with ALGORITHM=INSTANT: ALTER TABLE ... DROP CONSTRAINT for more information.

This applies to ALTER TABLE ... DROP CONSTRAINT for InnoDB tables.

ALTER TABLE ... FORCE

InnoDB does **not** support forcing a table rebuild with ALGORITHM set to `NOCOPY`.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='NOCOPY';
ALTER TABLE tab FORCE;
ERROR 1845 (0A000): ALGORITHM=NOCOPY is not supported for this operation. Try ALGORITHM=INPLACE
```

This applies to ALTER TABLE ... FORCE for InnoDB tables.

ALTER TABLE ... ENGINE=InnoDB

InnoDB does **not** support forcing a table rebuild with ALGORITHM set to `NOCOPY`.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='NOCOPY';
ALTER TABLE tab ENGINE=InnoDB;
ERROR 1845 (0A000): ALGORITHM=NOCOPY is not supported for this operation. Try ALGORITHM=INPLACE
```

This applies to ALTER TABLE ... ENGINE=InnoDB for InnoDB tables.

OPTIMIZE TABLE ...

InnoDB does **not** support optimizing a table with with ALGORITHM set to `NOCOPY`.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SHOW GLOBAL VARIABLES WHERE Variable_name IN('innodb_defragment',
'innodb_optimize_fulltext_only');
+-------------------------------+-------+
| Variable_name                 | Value |
+-------------------------------+-------+
| innodb_defragment             | OFF   |
| innodb_optimize_fulltext_only | OFF   |
+-------------------------------+-------+
2 rows in set (0.001 sec)

SET SESSION alter_algorithm='NOCOPY';
OPTIMIZE TABLE tab;
+---------+----------+----------+-------------------------------------------------------------------
-----------------+
| Table   | Op       | Msg_type | Msg_text
|
+---------+----------+----------+-------------------------------------------------------------------
-----------------+
| db1.tab | optimize | note     | Table does not support optimize, doing recreate + analyze
instead           |
| db1.tab | optimize | error    | ALGORITHM=NOCOPY is not supported for this operation. Try
ALGORITHM=INPLACE |
| db1.tab | optimize | status   | Operation failed
|
+---------+----------+----------+-------------------------------------------------------------------
-----------------+
3 rows in set, 1 warning (0.002 sec)
```

This applies to OPTIMIZE TABLE for InnoDB tables.

`ALTER TABLE ... RENAME TO` and `RENAME TABLE ...`

InnoDB supports renaming a table with ALGORITHM set to `NOCOPY` in the cases where the operation supports having the ALGORITHM clause set to `INSTANT`.

See InnoDB Online DDL Operations with ALGORITHM=INSTANT: ALTER TABLE ... RENAME TO and RENAME TABLE ... for more information.

This applies to ALTER TABLE ... RENAME TO and RENAME TABLE for InnoDB tables.

Limitations

Limitations Related to Generated (Virtual and Persistent/Stored) Columns

Generated columns do not currently support online DDL for all of the same operations that are supported for "real" columns.

See Generated (Virtual and Persistent/Stored) Columns: Statement Support for more information on the limitations.

5.3.2.19.4 InnoDB Online DDL Operations with the INSTANT Alter Algorithm

Contents

Column Operations

`ALTER TABLE ... ADD COLUMN`

In MariaDB 10.3.2 and later, InnoDB supports adding columns to a table with ALGORITHM set to `INSTANT` if the new column is the last column in the table. See MDEV-11369 for more information. If the table has a hidden `FTS_DOC_ID` column is present, then this is not supported.

In MariaDB 10.4 and later, InnoDB supports adding columns to a table with ALGORITHM set to `INSTANT`, regardless of where in the column list the new column is added.

When this operation is performed with ALGORITHM set to `INSTANT`, the tablespace file will have a non-canonical storage format. See Non-canonical Storage Format Caused by Some Operations for more information.

With the exception of adding an auto-increment column, this operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example, this succeeds:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50)
);

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab ADD COLUMN c varchar(50);
Query OK, 0 rows affected (0.004 sec)
```

And this succeeds in MariaDB 10.4 and later:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50)
);

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab ADD COLUMN c varchar(50) AFTER a;
Query OK, 0 rows affected (0.004 sec)
```

This applies to ALTER TABLE ... ADD COLUMN for InnoDB tables.

See Instant ADD COLUMN for InnoDB for more information.

ALTER TABLE ... DROP COLUMN

In MariaDB 10.4 and later, InnoDB supports dropping columns from a table with ALGORITHM set to `INSTANT`. See MDEV-15562 for more information.

When this operation is performed with ALGORITHM set to `INSTANT`, the tablespace file will have a non-canonical storage format. See Non-canonical Storage Format Caused by Some Operations for more information.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab DROP COLUMN c;
Query OK, 0 rows affected (0.004 sec)
```

This applies to ALTER TABLE ... DROP COLUMN for InnoDB tables.

ALTER TABLE ... MODIFY COLUMN

This applies to ALTER TABLE ... MODIFY COLUMN for InnoDB tables.

Reordering Columns

In MariaDB 10.4 and later, InnoDB supports reordering columns within a table with ALGORITHM set to `INSTANT`. See MDEV-15562 for more information.

When this operation is performed with ALGORITHM set to `INSTANT`, the tablespace file will have a non-canonical storage format. See Non-canonical Storage Format Caused by Some Operations for more information.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab MODIFY COLUMN c varchar(50) AFTER a;
Query OK, 0 rows affected (0.004 sec)
```

Changing the Data Type of a Column

InnoDB does **not** support modifying a column's data type with ALGORITHM set to `INSTANT` in most cases. There are some exceptions:

- InnoDB supports increasing the length of `VARCHAR` columns with ALGORITHM set to `INSTANT`, unless it would require changing the number of bytes requires to represent the column's length. A `VARCHAR` column that is between 0 and 255 bytes in size requires 1 byte to represent its length, while a `VARCHAR` column that is 256 bytes or longer requires 2 bytes to represent its length. This means that the length of a column cannot be increased with ALGORITHM set to `INSTANT` if the original length was less than 256 bytes, and the new length is 256 bytes or more.
- In MariaDB 10.4.3 and later, InnoDB supports increasing the length of `VARCHAR` columns with ALGORITHM set to `INSTANT` with no restrictions if the ROW_FORMAT table option is set to REDUNDANT. See MDEV-15563 for more information.
- In MariaDB 10.4.3 and later, InnoDB also supports increasing the length of `VARCHAR` columns with ALGORITHM set to `INSTANT` in a more limited manner if the ROW_FORMAT table option is set to COMPACT, DYNAMIC, or COMPRESSED. In this scenario, the following limitations apply:
  - The length can be increased with ALGORITHM set to `INSTANT` if the original length of the column is 127 bytes or less, and the new length of the column is 256 bytes or more.
  - The length can be increased with ALGORITHM set to `INSTANT` if the original length of the column is 255 bytes or less, and the new length of the column is still 255 bytes or less.
  - The length can be increased with ALGORITHM set to `INSTANT` if the original length of the column is 256 bytes or more, and the new length of the column is still 256 bytes or more.
  - The length can **not** be increased with ALGORITHM set to `INSTANT` if the original length was between 128 bytes and 255 bytes, and the new length is 256 bytes or more.
  - See MDEV-15563 for more information.

The supported operations in this category support the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example, this fails:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab MODIFY COLUMN c int;
ERROR 1846 (0A000): ALGORITHM=INSTANT is not supported. Reason: Cannot change column type
INPLACE. Try ALGORITHM=COPY
```

But this succeeds because the original length of the column is less than 256 bytes, and the new length is still less than 256 bytes:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
) CHARACTER SET=latin1;

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab MODIFY COLUMN c varchar(100);
Query OK, 0 rows affected (0.005 sec)
```

But this fails because the original length of the column is between 128 bytes and 255 bytes, and the new length is greater than 256 bytes:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(255)
) CHARACTER SET=latin1;

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab MODIFY COLUMN c varchar(256);
ERROR 1846 (0A000): ALGORITHM=INSTANT is not supported. Reason: Cannot change column type
INPLACE. Try ALGORITHM=COPY
```

But this succeeds in MariaDB 10.4.3 and later because the table has `ROW_FORMAT=REDUNDANT`:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(200)
) ROW_FORMAT=REDUNDANT;

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab MODIFY COLUMN c varchar(300);
Query OK, 0 rows affected (0.004 sec)
```

And this succeeds in MariaDB 10.4.3 and later because the table has `ROW_FORMAT=DYNAMIC` and the column's original length is 127 bytes or less:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(127)
) ROW_FORMAT=DYNAMIC
  CHARACTER SET=latin1;

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab MODIFY COLUMN c varchar(300);
Query OK, 0 rows affected (0.003 sec)
```

And this succeeds in MariaDB 10.4.3 and later because the table has `ROW_FORMAT=COMPRESSED` and the column's original length is 127 bytes or less:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(127)
) ROW_FORMAT=COMPRESSED
  CHARACTER SET=latin1;

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab MODIFY COLUMN c varchar(300);
Query OK, 0 rows affected (0.003 sec)
```

But this fails even in MariaDB 10.4.3 and later because the table has `ROW_FORMAT=DYNAMIC` and the column's original length is between 128 bytes and 255 bytes:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(128)
) ROW_FORMAT=DYNAMIC
  CHARACTER SET=latin1;

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab MODIFY COLUMN c varchar(300);
ERROR 1846 (0A000): ALGORITHM=INSTANT is not supported. Reason: Cannot change column type
INPLACE. Try ALGORITHM=COPY
```

Changing a Column to NULL

In MariaDB 10.4.3 and later, InnoDB supports modifying a column to allow NULL values with ALGORITHM set to `INSTANT` if the ROW_FORMAT table option is set to REDUNDANT. See MDEV-15563 for more information.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50) NOT NULL
) ROW_FORMAT=REDUNDANT;

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab MODIFY COLUMN c varchar(50) NULL;
Query OK, 0 rows affected (0.004 sec)
```

Changing a Column to NOT NULL

InnoDB does **not** support modifying a column to **not** allow NULL values with ALGORITHM set to `INSTANT`.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
) ROW_FORMAT=REDUNDANT;

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab MODIFY COLUMN c varchar(50) NOT NULL;
ERROR 1845 (0A000): ALGORITHM=INSTANT is not supported for this operation. Try
ALGORITHM=INPLACE
```

Adding a New `ENUM` Option

InnoDB supports adding a new ENUM option to a column with ALGORITHM set to `INSTANT`. In order to add a new ENUM option with ALGORITHM set to `INSTANT`, the following requirements must be met:

- It must be added to the end of the list.
- The storage requirements must not change.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example, this succeeds:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c ENUM('red', 'green')
);

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab MODIFY COLUMN c ENUM('red', 'green', 'blue');
Query OK, 0 rows affected (0.002 sec)
```

But this fails:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c ENUM('red', 'green')
);

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab MODIFY COLUMN c ENUM('red', 'blue', 'green');
ERROR 1846 (0A000): ALGORITHM=INSTANT is not supported. Reason: Cannot change column type
INPLACE. Try ALGORITHM=COPY
```

Adding a New `SET` Option

InnoDB supports adding a new SET option to a column with ALGORITHM set to `INSTANT`. In order to add a new SET option with ALGORITHM set to `INSTANT`, the following requirements must be met:

- It must be added to the end of the list.
- The storage requirements must not change.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example, this succeeds:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c SET('red', 'green')
);

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab MODIFY COLUMN c SET('red', 'green', 'blue');
Query OK, 0 rows affected (0.002 sec)
```

But this fails:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c SET('red', 'green')
);

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab MODIFY COLUMN c SET('red', 'blue', 'green');
ERROR 1846 (0A000): ALGORITHM=INSTANT is not supported. Reason: Cannot change column type
INPLACE. Try ALGORITHM=COPY
```

Removing System Versioning from a Column

In MariaDB 10.3.8 and later, InnoDB supports removing system versioning from a column with ALGORITHM set to `INSTANT`. In order for this to work, the system_versioning_alter_history system variable must be set to `KEEP`. See MDEV-16330 for more information.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50) WITH SYSTEM VERSIONING
);

SET SESSION system_versioning_alter_history='KEEP';
SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab MODIFY COLUMN c varchar(50) WITHOUT SYSTEM VERSIONING;
Query OK, 0 rows affected (0.004 sec)
```

`ALTER TABLE ... ALTER COLUMN`

This applies to ALTER TABLE ... ALTER COLUMN for InnoDB tables.

Setting a Column's Default Value

InnoDB supports modifying a column's DEFAULT value with ALGORITHM set to `INSTANT`.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab ALTER COLUMN c SET DEFAULT 'No value explicitly provided.';
Query OK, 0 rows affected (0.003 sec)
```

Removing a Column's Default Value

InnoDB supports removing a column's DEFAULT value with ALGORITHM set to `INSTANT`.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50) DEFAULT 'No value explicitly provided.'
);

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab ALTER COLUMN c DROP DEFAULT;
Query OK, 0 rows affected (0.002 sec)
```

ALTER TABLE ... CHANGE COLUMN

InnoDB supports renaming a column with ALGORITHM set to `INSTANT`, unless the column's data type or attributes changed in addition to the name.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example, this succeeds:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab CHANGE COLUMN c str varchar(50);
Query OK, 0 rows affected (0.004 sec)
```

But this fails:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab CHANGE COLUMN c num int;
ERROR 1846 (0A000): ALGORITHM=INSTANT is not supported. Reason: Cannot change column type
INPLACE. Try ALGORITHM=COPY
```

This applies to ALTER TABLE ... CHANGE COLUMN for InnoDB tables.

Index Operations

ALTER TABLE ... ADD PRIMARY KEY

InnoDB does **not** support adding a primary key to a table with ALGORITHM set to `INSTANT`.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int,
   b varchar(50),
   c varchar(50)
);

SET SESSION sql_mode='STRICT_TRANS_TABLES';
SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab ADD PRIMARY KEY (a);
ERROR 1845 (0A000): ALGORITHM=INSTANT is not supported for this operation. Try
ALGORITHM=INPLACE
```

This applies to ALTER TABLE ... ADD PRIMARY KEY for InnoDB tables.

ALTER TABLE ... DROP PRIMARY KEY

InnoDB does **not** support dropping a primary key with ALGORITHM set to `INSTANT`.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab DROP PRIMARY KEY;
ERROR 1846 (0A000): ALGORITHM=INSTANT is not supported. Reason: Dropping a primary key is not
allowed without also adding a new primary key. Try ALGORITHM=COPY
```

This applies to ALTER TABLE ... DROP PRIMARY KEY for InnoDB tables.

ALTER TABLE ... ADD INDEX and CREATE INDEX

This applies to ALTER TABLE ... ADD INDEX and CREATE INDEX for InnoDB tables.

Adding a Plain Index

InnoDB does **not** support adding a plain index to a table with ALGORITHM set to `INSTANT`.

For example, this fails:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab ADD INDEX b_index (b);
ERROR 1846 (0A000): ALGORITHM=INSTANT is not supported. Reason: ADD INDEX. Try ALGORITHM=NOCOPY
```

And this fails:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INSTANT';
CREATE INDEX b_index ON tab (b);
ERROR 1846 (0A000): ALGORITHM=INSTANT is not supported. Reason: ADD INDEX. Try ALGORITHM=NOCOPY
```

Adding a Fulltext Index

InnoDB does **not** support adding a FULLTEXT index to a table with ALGORITHM set to `INSTANT`.

For example, this fails:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab ADD FULLTEXT INDEX b_index (b);
Query OK, 0 rows affected (0.042 sec)

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab ADD FULLTEXT INDEX c_index (c);
ERROR 1846 (0A000): ALGORITHM=INSTANT is not supported. Reason: ADD INDEX. Try ALGORITHM=NOCOPY
```

And this fails:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INPLACE';
CREATE FULLTEXT INDEX b_index ON tab (b);
Query OK, 0 rows affected (0.040 sec)

SET SESSION alter_algorithm='INSTANT';
CREATE FULLTEXT INDEX c_index ON tab (c);
ERROR 1846 (0A000): ALGORITHM=INSTANT is not supported. Reason: ADD INDEX. Try ALGORITHM=NOCOPY
```

Adding a Spatial Index

InnoDB does **not** support adding a SPATIAL index to a table with ALGORITHM set to `INSTANT`.

For example, this fails:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c GEOMETRY NOT NULL
);

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab ADD SPATIAL INDEX c_index (c);
ERROR 1846 (0A000): ALGORITHM=INSTANT is not supported. Reason: ADD INDEX. Try ALGORITHM=NOCOPY
```

And this fails:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c GEOMETRY NOT NULL
);

SET SESSION alter_algorithm='INSTANT';
CREATE SPATIAL INDEX c_index ON tab (c);
ERROR 1846 (0A000): ALGORITHM=INSTANT is not supported. Reason: ADD INDEX. Try ALGORITHM=NOCOPY
```

ALTER TABLE ... ADD FOREIGN KEY

InnoDB does **not** support adding foreign key constraints to a table with ALGORITHM set to `INSTANT`.

For example:

```
CREATE OR REPLACE TABLE tab1 (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50),
   d int
);

CREATE OR REPLACE TABLE tab2 (
   a int PRIMARY KEY,
   b varchar(50)
);

SET SESSION foreign_key_checks=OFF;
SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab1 ADD FOREIGN KEY tab2_fk (d) REFERENCES tab2 (a);
ERROR 1846 (0A000): ALGORITHM=INSTANT is not supported. Reason: ADD INDEX. Try ALGORITHM=NOCOPY
```

This applies to ALTER TABLE ... ADD FOREIGN KEY for InnoDB tables.

ALTER TABLE ... DROP FOREIGN KEY

InnoDB supports dropping foreign key constraints from a table with ALGORITHM set to `INSTANT`.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example:

```
CREATE OR REPLACE TABLE tab2 (
   a int PRIMARY KEY,
   b varchar(50)
);

CREATE OR REPLACE TABLE tab1 (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50),
   d int,
   FOREIGN KEY tab2_fk (d) REFERENCES tab2 (a)
);

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab1 DROP FOREIGN KEY tab2_fk;
Query OK, 0 rows affected (0.004 sec)
```

This applies to ALTER TABLE ... DROP FOREIGN KEY for InnoDB tables.

Table Operations

ALTER TABLE ... AUTO_INCREMENT=...

InnoDB supports changing a table's AUTO_INCREMENT value with ALGORITHM set to `INSTANT`.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab AUTO_INCREMENT=100;
Query OK, 0 rows affected (0.002 sec)
```

This applies to ALTER TABLE ... AUTO_INCREMENT=... for InnoDB tables.

ALTER TABLE ... ROW_FORMAT=...

InnoDB does **not** support changing a table's row format with ALGORITHM set to `INSTANT`.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
) ROW_FORMAT=DYNAMIC;

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab ROW_FORMAT=COMPRESSED;
ERROR 1846 (0A000): ALGORITHM=INSTANT is not supported. Reason: Changing table options requires
the table to be rebuilt. Try ALGORITHM=INPLACE
```

This applies to ALTER TABLE ... ROW_FORMAT=... for InnoDB tables.

ALTER TABLE ... KEY_BLOCK_SIZE=...

InnoDB does **not** support changing a table's KEY_BLOCK_SIZE with ALGORITHM set to `INSTANT`.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
) ROW_FORMAT=COMPRESSED
  KEY_BLOCK_SIZE=4;

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab KEY_BLOCK_SIZE=2;
ERROR 1846 (0A000): ALGORITHM=INSTANT is not supported. Reason: Changing table options requires
the table to be rebuilt. Try ALGORITHM=INPLACE
```

This applies to KEY_BLOCK_SIZE=... for InnoDB tables.

ALTER TABLE ... PAGE_COMPRESSED=1 and ALTER TABLE ... PAGE_COMPRESSION_LEVEL=...

In MariaDB 10.3.10 and later, InnoDB supports setting a table's PAGE_COMPRESSED value to `1` with ALGORITHM set to `INSTANT`. InnoDB does **not** support changing a table's PAGE_COMPRESSED value from `1` to `0` with ALGORITHM set to `INSTANT`.

In these versions, InnoDB also supports changing a table's PAGE_COMPRESSION_LEVEL value with ALGORITHM set to `INSTANT`.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

See MDEV-16328 for more information.

For example, this succeeds:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab PAGE_COMPRESSED=1;
Query OK, 0 rows affected (0.004 sec)
```

And this succeeds:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
) PAGE_COMPRESSED=1
  PAGE_COMPRESSION_LEVEL=5;

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab PAGE_COMPRESSION_LEVEL=4;
Query OK, 0 rows affected (0.004 sec)
```

But this fails:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
) PAGE_COMPRESSED=1;

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab PAGE_COMPRESSED=0;
ERROR 1846 (0A000): ALGORITHM=INSTANT is not supported. Reason: Changing table options requires
the table to be rebuilt. Try ALGORITHM=INPLACE
```

This applies to ALTER TABLE ... PAGE_COMPRESSED=... and ALTER TABLE ... PAGE_COMPRESSION_LEVEL=... for InnoDB tables.

ALTER TABLE ... DROP SYSTEM VERSIONING

InnoDB does **not** support dropping system versioning from a table with ALGORITHM set to `INSTANT`.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
) WITH SYSTEM VERSIONING;

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab DROP SYSTEM VERSIONING;
ERROR 1845 (0A000): ALGORITHM=INSTANT is not supported for this operation. Try
ALGORITHM=INPLACE
```

This applies to ALTER TABLE ... DROP SYSTEM VERSIONING for InnoDB tables.

ALTER TABLE ... DROP CONSTRAINT

In MariaDB 10.3.6 and later, InnoDB supports dropping a CHECK constraint from a table with ALGORITHM set to `INSTANT`. See MDEV-16331 for more information.

This operation supports the non-locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `NONE`. When this strategy is used, all concurrent DML is permitted.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50),
   CONSTRAINT b_not_empty CHECK (b != '')
);

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab DROP CONSTRAINT b_not_empty;
Query OK, 0 rows affected (0.002 sec)
```

This applies to ALTER TABLE ... DROP CONSTRAINT for InnoDB tables.

ALTER TABLE ... FORCE

InnoDB does **not** support forcing a table rebuild with ALGORITHM set to `INSTANT`.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab FORCE;
ERROR 1845 (0A000): ALGORITHM=INSTANT is not supported for this operation. Try
ALGORITHM=INPLACE
```

This applies to ALTER TABLE ... FORCE for InnoDB tables.

ALTER TABLE ... ENGINE=InnoDB

InnoDB does **not** support forcing a table rebuild with ALGORITHM set to `INSTANT`.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab ENGINE=InnoDB;
ERROR 1845 (0A000): ALGORITHM=INSTANT is not supported for this operation. Try
ALGORITHM=INPLACE
```

This applies to ALTER TABLE ... ENGINE=InnoDB for InnoDB tables.

OPTIMIZE TABLE ...

InnoDB does **not** support optimizing a table with with ALGORITHM set to `INSTANT`.

For example:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SHOW GLOBAL VARIABLES WHERE Variable_name IN('innodb_defragment',
'innodb_optimize_fulltext_only');
+------------------------------+-------+
| Variable_name                | Value |
+------------------------------+-------+
| innodb_defragment            | OFF   |
| innodb_optimize_fulltext_only | OFF   |
+------------------------------+-------+
2 rows in set (0.001 sec)

SET SESSION alter_algorithm='INSTANT';
OPTIMIZE TABLE tab;
+---------+----------+----------+-------------------------------------------------------------------
-----------------+
| Table   | Op       | Msg_type | Msg_text
|
+---------+----------+----------+-------------------------------------------------------------------
-----------------+
| db1.tab | optimize | note     | Table does not support optimize, doing recreate + analyze
instead            |
| db1.tab | optimize | error    | ALGORITHM=INSTANT is not supported for this operation. Try
ALGORITHM=INPLACE |
| db1.tab | optimize | status   | Operation failed
|
+---------+----------+----------+-------------------------------------------------------------------
-----------------+
3 rows in set, 1 warning (0.002 sec)
```

This applies to OPTIMIZE TABLE for InnoDB tables.

ALTER TABLE ... RENAME TO and RENAME TABLE ...

InnoDB supports renaming a table with ALGORITHM set to `INSTANT`.

This operation supports the exclusive locking strategy. This strategy can be explicitly chosen by setting the LOCK clause to `EXCLUSIVE`. When this strategy is used, concurrent DML is **not** permitted.

For example, this succeeds:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INSTANT';
ALTER TABLE tab RENAME TO old_tab;
Query OK, 0 rows affected (0.008 sec)
```

And this succeeds:

```
CREATE OR REPLACE TABLE tab (
   a int PRIMARY KEY,
   b varchar(50),
   c varchar(50)
);

SET SESSION alter_algorithm='INSTANT';
RENAME TABLE tab TO old_tab;
Query OK, 0 rows affected (0.008 sec)
```

This applies to ALTER TABLE ... RENAME TO and RENAME TABLE for InnoDB tables.

Limitations

Limitations Related to Generated (Virtual and Persistent/Stored) Columns

Generated columns do not currently support online DDL for all of the same operations that are supported for "real" columns.

See Generated (Virtual and Persistent/Stored) Columns: Statement Support for more information on the limitations.

Non-canonical Storage Format Caused by Some Operations

Some operations cause a table's tablespace file to use a non-canonical storage format when the `INSTANT` algorithm is used. The affected operations include:

- Adding a column.
- Dropping a column.
- Reordering columns.

These operations require the following non-canonical changes to the storage format:

- A hidden metadata record at the start of the clustered index is used to store each column's DEFAULT value. This makes it possible to add new columns that have default values without rebuilding the table.
- A BLOB in the hidden metadata record is used to store column mappings. This makes it possible to drop or reorder columns without rebuilding the table. This also makes it possible to add columns to any position or drop columns from any position in the table without rebuilding the table.
- If a column is dropped, old records will contain garbage in that column's former position, and new records will be written with NULL values, empty strings, or dummy values.

This non-canonical storage format has the potential to incur some performance or storage overhead for all subsequent DML operations. If you notice some issues like this and you want to normalize a table's storage format to avoid this problem, then you can do so by forcing a table rebuild by executing ALTER TABLE ... FORCE with ALGORITHM set to `INPLACE`. For example:

```
SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab FORCE;
Query OK, 0 rows affected (0.008 sec)
```

However, keep in mind that there are certain scenarios where you may not be able to rebuild the table with ALGORITHM set to `INPLACE`. See InnoDB Online DDL Operations with ALGORITHM=INPLACE: Limitations for more information on those cases. If you hit one of those scenarios, but you still want to rebuild the table, then you would have to do so with ALGORITHM set to `COPY`.

Known Bugs

There are some known bugs that could lead to issues when an InnoDB DDL operation is performed using the INSTANT algorithm. This algorithm will usually be chosen by default if the operation supports the algorithm.

The effect of many of these bugs is that the table seems to *forget* that its tablespace file is in the non-canonical storage format.

If you are concerned that a table may be affected by one of these bugs, then your best option would be to normalize the table structure. This can be done by rebuilding the table. For example:

```
SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab FORCE;
Query OK, 0 rows affected (0.008 sec)
```

If you are concerned about these bugs, and you want to perform an operation that supports the INSTANT algorithm, but you want to avoid using that algorithm, then you can set the algorithm to INPLACE and add the `FORCE` keyword to the ALTER TABLE statement:

```
SET SESSION alter_algorithm='INPLACE';
ALTER TABLE tab ADD COLUMN c varchar(50), FORCE;
```

Closed Bugs

- MDEV-20066: This bug could cause a table to become corrupt if a column was added instantly. It is fixed in

MariaDB 10.3.18 and MariaDB 10.4.8.
- MDEV-20117: This bug could cause a table to become corrupt if a column was dropped instantly. It is fixed in MariaDB 10.4.9.
- MDEV-19743: This bug could cause a table to become corrupt during page reorganization if a column was added instantly. It is fixed in MariaDB 10.3.17 and MariaDB 10.4.7.
- MDEV-19783: This bug could cause a table to become corrupt if a column was added instantly. It is fixed in MariaDB 10.3.17 and MariaDB 10.4.7
- MDEV-20090: This bug could cause a table to become corrupt if columns were added, dropped, or reordered instantly. It is fixed in MariaDB 10.4.9.
- MDEV-18519: This bug could cause a table to become corrupt if a column was added instantly. It is fixed in MariaDB 10.6.9, MariaDB 10.7.5, MariaDB 10.8.4 and MariaDB 10.9.2.
- MDEV-18519: This bug could cause a table to become corrupt if a column was added instantly. This isn't and won't be fixed in versions less than MariaDB 10.6.

5.3.2.19.5 Instant ADD COLUMN for InnoDB

Contents

Normally, adding a column to a table requires the full table to be rebuilt. The complexity of the operation is proportional to the size of the table, or O(n·m) where n is the number of rows in the table and m is the number of indexes.

In MariaDB 10.0 and later, the ALTER TABLE statement supports online DDL for storage engines that have implemented the relevant online DDL algorithms and locking strategies.

The InnoDB storage engine has implemented online DDL for many operations. These online DDL optimizations allow concurrent DML to the table in many cases, even if the table needs to be rebuilt.

See InnoDB Online DDL Overview for more information about online DDL with InnoDB.

Allowing concurrent DML during the operation does not solve all problems. When a column was added to a table with the older in-place optimization, the resulting table rebuild could still significantly increase the I/O and memory consumption and cause replication lag.

In contrast, with the new instant ALTER TABLE ... ADD COLUMN, all that is needed is an O(log n) operation to insert a special hidden record into the table, and an update of the data dictionary. For a large table, instead of taking several hours, the operation would be completed in the blink of an eye. The ALTER TABLE ... ADD COLUMN operation is only slightly more expensive than a regular INSERT, due to locking constraints.

In the past, some developers may have implemented a kind of "instant add column" in the application by encoding multiple columns in a single TEXT or BLOB column. MariaDB Dynamic Columns was an early example of that. A more recent example is JSON and related string manipulation functions.

Adding real columns has the following advantages over encoding columns into a single "expandable" column:

- Efficient storage in a native binary format
- Data type safety
- Indexes can be built natively
- Constraints are available: UNIQUE, CHECK, FOREIGN KEY
- DEFAULT values can be specified
- Triggers can be written more easily

With instant ALTER TABLE ... ADD COLUMN, you can enjoy all the benefits of structured storage without the drawback of having to rebuild the table.

Instant ALTER TABLE ... ADD COLUMN is available for both old and new InnoDB tables. Basically you can just upgrade from MySQL 5.x or MariaDB and start adding columns instantly.

Columns instantly added to a table exist in a separate data structure from the main table definition, similar to how InnoDB separates `BLOB` columns. If the table ever becomes empty, (such as from TRUNCATE or DELETE statements), InnoDB incorporates the instantly added columns into the main table definition. See InnoDB Online DDL Operations with ALGORITHM=INSTANT: Non-canonical Storage Format Caused by Some Operations for more information.

The operation is also crash safe. If the server is killed while executing an instant ALTER TABLE ... ADD COLUMN, when the table is restored InnoDB integrates the new column, flattening the table definition.

Limitations

- In MariaDB 10.3, instant ALTER TABLE ... ADD COLUMN only applies when the added columns appear last in the table. The place specifier `LAST` is the default. If `AFTER` col is specified, then col must be the last column, or the

operation will require the table to be rebuilt. In MariaDB 10.4, this restriction was lifted.

- If the table contains a hidden `FTS_DOC_ID` column due to a FULLTEXT INDEX, then instant ALTER TABLE ... ADD COLUMN will not be possible.
- InnoDB data files after instant ALTER TABLE ... ADD COLUMN cannot be imported to older versions of MariaDB or MySQL without first being rebuilt.
- After using Instant ALTER TABLE ... ADD COLUMN, any table-rebuilding operation such as ALTER TABLE … FORCE will incorporate instantaneously added columns into the main table body.
- Instant ALTER TABLE ... ADD COLUMN is not available for ROW_FORMAT=COMPRESSED.
- In MariaDB 10.3, ALTER TABLE … DROP COLUMN requires the table to be rebuilt. In MariaDB 10.4, this restriction was lifted.

Example

```
CREATE TABLE t(id INT PRIMARY KEY, u INT UNSIGNED NOT NULL UNIQUE)
ENGINE=InnoDB;

INSERT INTO t(id,u) VALUES(1,1),(2,2),(3,3);

ALTER TABLE t ADD COLUMN
(d DATETIME DEFAULT current_timestamp(),
 p POINT NOT NULL DEFAULT ST_GeomFromText('POINT(0 0)'),
 t TEXT CHARSET utf8 DEFAULT 'The quick brown fox jumps over the lazy dog');

UPDATE t SET t=NULL WHERE id=3;

SELECT id,u,d,ST_AsText(p),t FROM t;

SELECT variable_value FROM information_schema.global_status
WHERE variable_name = 'innodb_instant_alter_column';
```

The above example illustrates that when the added columns are declared NOT NULL, a DEFAULT value must be available, either implied by the data type or set explicitly by the user. The expression need not be constant, but it must not refer to the columns of the table, such as DEFAULT u+1 (a MariaDB extension). The DEFAULT current_timestamp() would be evaluated at the time of the ALTER TABLE and apply to each row, like it does for non-instant ALTER TABLE. If a subsequent ALTER TABLE changes the DEFAULT value for subsequent INSERT, the values of the columns in existing records will naturally be unaffected.

The design was brainstormed in April by engineers from MariaDB Corporation, Alibaba and Tencent. A prototype was developed by Vin Chen (陈福荣) from the Tencent Game DBA Team.

5.3.2.20 Binary Log Group Commit and InnoDB Flushing Performance

Contents

MariaDB 10.0 introduced a performance improvement related to group commit that affects the performance of flushing InnoDB transactions when the binary log is enabled.

Overview

In MariaDB 10.0 and above, when both innodb_flush_log_at_trx_commit=1 (the default) is set and the binary log is enabled, there is now one less sync to disk inside InnoDB during commit (2 syncs shared between a group of transactions instead of 3).

Durability of commits is not decreased — this is because even if the server crashes before the commit is written to disk by InnoDB, it will be recovered from the binary log at next server startup (and it is guaranteed that sufficient information is synced to disk so that such a recovery is always possible).

Switching to Old Flushing Behavior

The old behavior, with 3 syncs to disk per (group) commit (and consequently lower performance), can be selected with the new innodb_flush_log_at_trx_commit=3 option. There is normally no benefit to doing this, however there are a couple of edge cases to be aware of.

Non-durable Binary Log Settings

If innodb_flush_log_at_trx_commit=1 is set and the binary log is enabled, but sync_binlog=0 is set, then commits are not guaranteed durable inside InnoDB after commit. This is because if sync_binlog=0 is set and if the server crashes, then transactions that were not flushed to the binary log prior to the crash will be missing from the binary log.

In this specific scenario, innodb_flush_log_at_trx_commit=3 can be set to ensure that transactions will be durable in InnoDB, even if they are not necessarily durable from the perspective of the binary log.

One should be aware that if sync_binlog=0 is set, then a crash is nevertheless likely to cause transactions to be missing from the binary log. This will cause the binary log and InnoDB to be inconsistent with each other. This is also likely to cause any replication slaves to become inconsistent, since transactions are replicated through the binary log. Thus it is recommended to set sync_binlog=1. With the group commit improvements introduced in MariaDB 5.3, this setting has much less penalty in recent versions compared to older versions of MariaDB and MySQL.

Recent Transactions Missing from Backups

Mariabackup and Percona XtraBackup only see transactions that have been flushed to the redo log. With the group commit improvements, there may be a small delay (defined by the binlog_commit_wait_usec system variable) between when a commit happens and when the commit will be included in a backup.

Note that the backup will still be fully consistent with itself and the binary log. This problem is normally not an issue in practice. A backup usually takes a long time to complete (relative to the 1 second or so that binlog_commit_wait_usec is normally set to), and a backup usually includes a lot of transactions that were committed during the backup. With this in mind, it is not generally noticeable if the backup does not include transactions that were committed during the last 1 second or so of the backup process. It is just mentioned here for completeness.

5.3.2.21 InnoDB Page Compression

Contents

Overview

InnoDB page compression provides a way to compress InnoDB tables.

Use Cases

- InnoDB page compression can be used on any storage device and any file system.
- InnoDB page compression is most efficient on file systems that support sparse files. See Saving Storage Space with Sparse Files for more information.
- InnoDB page compression is most beneficial on solid state drives (SSDs) and other flash storage. See Optimized for Flash Storage for more information.
- InnoDB page compression performs best when your storage device and file system support atomic writes, since that allows the InnoDB doublewrite buffer to be disabled. See Atomic Write Support for more information.

Comparison with the `COMPRESSED` Row Format

InnoDB page compression is a modern way to compress your InnoDB tables. It is similar to InnoDB's COMPRESSED row format, but it has many advantages. Some of the differences are:

- With InnoDB page compression, compressed pages are immediately decompressed after being read from the tablespace file, and only uncompressed pages are stored in the buffer pool. In contrast, with InnoDB's COMPRESSED row format, compressed pages are decompressed immediately after they are read from the tablespace file, and both the uncompressed and compressed pages are stored in the buffer pool. This means that the COMPRESSED row format uses more space in the buffer pool than InnoDB page compression does.
- With InnoDB page compression, pages are compressed just before being written to the tablespace file. In contrast, with InnoDB's COMPRESSED row format, pages are re-compressed immediately after any changes, and the compressed pages are stored in the buffer pool alongside the uncompressed pages. These changes are then occasionally flushed to disk. This means that the COMPRESSED row format re-compresses data more frequently than InnoDB page compression does.
- With InnoDB page compression, multiple compression algorithms are supported. In contrast, with InnoDB's COMPRESSED row format, zlib is the only supported compression algorithm. This means that the COMPRESSED row format has less compression options than InnoDB page compression does.

In general, InnoDB page compression is superior to the COMPRESSED row format.

Comparison with Storage Engine-Independent Column Compression

- See Storage Engine-Independent Column Compression - Comparison with InnoDB Page Compression.

Configuring the InnoDB Page Compression Algorithm

There is not currently a table option to set different InnoDB page compression algorithms for individual tables.

However, the server-wide InnoDB page compression algorithm can be configured by setting the innodb_compression_algorithm system variable.

When this system variable is changed, the InnoDB page compression algorithm does not change for existing pages that were already compressed with a different InnoDB page compression algorithm. InnoDB is able to handle this situation without issues, because every page in an InnoDB tablespace contains metadata about the InnoDB page compression algorithm in the page header. This means that InnoDB supports having uncompressed pages and pages compressed with different InnoDB page compression algorithms in the same InnoDB tablespace at the same time.

This system variable can be set to one of the following values:

System Variable Value	Description
`none`	Pages are not compressed. This is the default value in MariaDB 10.2.3 and before, and MariaDB 10.1.21 and before.
`zlib`	Pages are compressed using the bundled zlib compression algorithm. This is the default value in MariaDB 10.2.4 and later, and MariaDB 10.1.22 and later.
`lz4`	Pages are compressed using the lz4 compression algorithm.
`lzo`	Pages are compressed using the lzo compression algorithm.
`lzma`	Pages are compressed using the lzma compression algorithm.

bzip2	Pages are compressed using the bzip2 compression algorithm.
snappy	Pages are compressed using the snappy algorithm.

However, on many distributions, the standard MariaDB builds do not support all InnoDB page compression algorithms by default. From MariaDB 10.7, algorithms can be installed as a plugin.

This system variable can be changed dynamically with SET GLOBAL. For example:

```
SET GLOBAL innodb_compression_algorithm='lzma';
```

This system variable can also be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
innodb_compression_algorithm=lzma
```

Checking Supported InnoDB Page Compression Algorithms

On many distributions, the standard MariaDB builds do not support all InnoDB page compression algorithms by default. Therefore, if you want to use a specific InnoDB page compression algorithm, then you should check whether your MariaDB build supports it.

The zlib compression algorithm is always supported. From MariaDB 10.7, algorithms can be installed as a plugin.

A MariaDB build's support for other InnoDB page compression algorithms can be checked by querying the following status variables with SHOW GLOBAL STATUS:

Status Variable	Description
Innodb_have_lz4	Whether InnoDB supports the lz4 compression algorithm.
Innodb_have_lzo	Whether InnoDB supports the lzo compression algorithm.
Innodb_have_lzma	Whether InnoDB supports the lzma compression algorithm.
Innodb_have_bzip2	Whether InnoDB supports the bzip2 compression algorithm.
Innodb_have_snappy	Whether InnoDB supports the snappy compression algorithm.

For example:

```
SHOW GLOBAL STATUS WHERE Variable_name IN (
   'Innodb_have_lz4',
   'Innodb_have_lzo',
   'Innodb_have_lzma',
   'Innodb_have_bzip2',
   'Innodb_have_snappy'
);
+--------------------+-------+
| Variable_name      | Value |
+--------------------+-------+
| Innodb_have_lz4    | OFF   |
| Innodb_have_lzo    | OFF   |
| Innodb_have_lzma   | ON    |
| Innodb_have_bzip2  | OFF   |
| Innodb_have_snappy | OFF   |
+--------------------+-------+
```

Adding Support for an InnoDB Page Compression Algorithm

On many distributions, the standard MariaDB builds do not support all InnoDB page compression algorithms by default. From MariaDB 10.7, algorithms can be installed as a plugin, but in earlier versions, if you want to use certain InnoDB page compression algorithms, then you may need to do the following:

- Download the package for the desired compression library from the above links.
- Install the package for the desired compression library.
- Compile MariaDB from the source distribution.

The general steps for compiling MariaDB are:

- Download and unpack the source code distribution:

```
wget https://downloads.mariadb.com/MariaDB/mariadb-10.4.8/source/mariadb-10.4.8.tar.gz
tar -xvzf mariadb-10.4.8.tar.gz
cd mariadb-10.4.8/
```

- Configure the build using cmake:

```
cmake .
```

- Check CMakeCache.txt to confirm that it has found the desired compression library on your system.
- Compile the build:

```
make
```

- Either install the build:

```
make install
```

Or make a package to install:

```
make package
```

See Compiling MariaDB From Source for more information.

Enabling InnoDB Page Compression

InnoDB page compression is not enabled by default. However, InnoDB page compression can be enabled for just individual InnoDB tables or it can be enabled for all new InnoDB tables by default.

InnoDB page compression is also only supported if the InnoDB table is in a file per-table tablespace. Therefore, the innodb_file_per_table system variable must be set to `ON` to use InnoDB page compression.

InnoDB page compression is only supported if the InnoDB table uses the `Barracuda` file format.Therefore, in MariaDB 10.1 and before, the innodb_file_format system variable must be set to `Barracuda` to use InnoDB page compression.

InnoDB page compression is also only supported if the InnoDB table's row format is COMPACT or DYNAMIC.

Enabling InnoDB Page Compression by Default

In MariaDB 10.2.3 and later, InnoDB page compression can be enabled for all new InnoDB tables by default by setting the innodb_compression_default system variable to `ON`.

This system variable can be set to one of the following values:

System Variable Value	Description
`OFF`	New InnoDB tables do not use InnoDB page compression. This is the default value.
`ON`	New InnoDB tables use InnoDB page compression.

This system variable can be changed dynamically with SET GLOBAL. For example:

```
SET GLOBAL innodb_compression_default=ON;
```

This system variable's session value can be changed dynamically with SET SESSION. For example:

```
SET GLOBAL innodb_file_per_table=ON;

SET GLOBAL innodb_file_format='Barracuda';

SET GLOBAL innodb_default_row_format='dynamic';

SET GLOBAL innodb_compression_algorithm='lzma';

SET SESSION  innodb_compression_default=ON;

CREATE TABLE users (
   user_id int not null,
   b varchar(200),
   primary key(user_id)
)
   ENGINE=InnoDB;
```

This system variable can also be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
innodb_compression_default=ON
```

Enabling InnoDB Page Compression for Individual Tables

InnoDB page compression can be enabled for individual tables by setting the PAGE_COMPRESSED table option to `1`. For example:

```
SET GLOBAL innodb_file_per_table=ON;

SET GLOBAL innodb_file_format='Barracuda';

SET GLOBAL innodb_default_row_format='dynamic';

SET GLOBAL innodb_compression_algorithm='lzma';

CREATE TABLE users (
   user_id int not null,
   b varchar(200),
   primary key(user_id)
)
   ENGINE=InnoDB
   PAGE_COMPRESSED=1;
```

Configuring the Compression Level

Some InnoDB page compression algorithms support a compression level option, which configures how the InnoDB page compression algorithm will balance speed and compression.

The compression level's supported values range from `1` to `9`. The range goes from the fastest to the most compact, which means that `1` is the fastest and `9` is the most compact.

Only the following InnoDB page compression algorithms currently support compression levels:

- zlib
- lzma

If an InnoDB page compression algorithm does not support compression levels, then it ignores any provided compression level value.

Configuring the Default Compression Level

The default compression level can be configured by setting the innodb_compression_level system variable.

This system variable's default value is `6`.

This system variable can be changed dynamically with SET GLOBAL. For example:

```
SET GLOBAL innodb_compression_level=9;
```

This system variable can also be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
innodb_compression_level=9
```

Configuring the Compression Level for Individual Tables

The compression level for individual tables can also be configured by setting the PAGE_COMPRESSION_LEVEL table option for the table. For example:

```
SET GLOBAL innodb_file_per_table=ON;

SET GLOBAL innodb_file_format='Barracuda';

SET GLOBAL innodb_default_row_format='dynamic';

SET GLOBAL innodb_compression_algorithm='lzma';

CREATE TABLE users (
   user_id int not null,
   b varchar(200),
   primary key(user_id)
)
   ENGINE=InnoDB
   PAGE_COMPRESSED=1
   PAGE_COMPRESSION_LEVEL=9;
```

Configuring the Failure Threshold and Maximum Padding

InnoDB page compression can encounter compression failures.

InnoDB page compression's failure threshold can be configured. If InnoDB encounters more compression failures than the failure threshold, then it pads pages with zeroed out bytes before attempting to compress them as a way to reduce failures. If the failure rate stays above the failure threshold, then InnoDB pads pages with more zeroed out bytes in 128 byte increments.

InnoDB page compression's maximum padding can also be configured.

Configuring the Failure Threshold

The failure threshold can be configured by setting the innodb_compression_failure_threshold_pct system variable.

This system variable's supported values range from `0` to `100`.

This system variable's default value is `5`.

This system variable can be changed dynamically with SET GLOBAL. For example:

```
SET GLOBAL innodb_compression_failure_threshold_pct=10;
```

This system variable can also be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
innodb_compression_failure_threshold_pct=10
```

Configuring the Maximum Padding

The maximum padding can be configured by setting the innodb_compression_pad_pct_max system variable.

This system variable's supported values range from `0` to `75`.

This system variable's default value is `50`.

This system variable can be changed dynamically with SET GLOBAL. For example:

```
SET GLOBAL innodb_compression_pad_pct_max=75;
```

This system variable can also be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
innodb_compression_pad_pct_max=75
```

Saving Storage Space with Sparse Files

When InnoDB page compression is used, InnoDB may still write the compressed page to the tablespace file with the original size of the uncompressed page, which would be equivalent to the value of the innodb_page_size system variable. This is done by design, because when InnoDB's I/O code needs to read the page from disk, it can only read the full page size. However, this is obviously not optimal.

On file systems that support sparse files, this problem is solved by writing the tablespace file as a sparse file using the *punch hole* technique. With the *punch hole* technique, InnoDB will only write the actual compressed page size to the tablespace file, aligned to sector size. The rest of the page is trimmed.

This *punch hole* technique allows InnoDB to read the compressed page from disk as the full page size, even though the compressed page really takes up less space on the file system.

There are some potential disadvantages to using sparse files:

- Some utilities may require special options in order to handle sparse files in an efficient manner.
- Most existing file systems are slow to unlink() sparse files. As a consequence, if a tablespace file is a sparse file, then dropping the table can be very slow.

Sparse File Support on Linux

On Linux, the following file systems support sparse files:

- `ext3`
- `ext4`
- `xfs`
- `btrfs`
- `nvmfs`

On Linux, file systems need to support the fallocate() system call with the `FALLOC_FL_PUNCH_HOLE` and `FALLOC_FL_KEEP_SIZE` flags. For example:

```
fallocate(file_handle, FALLOC_FL_PUNCH_HOLE | FALLOC_FL_KEEP_SIZE, file_offset, remainder_len);
```

Some Linux utilities may require special options in order to work with sparse files efficiently. For example:

- The ls utility will report the non-sparse size of the tablespace file when executed with default behavior, but `ls -s` will report the actual amount of storage allocated for the tablespace file.
- The cp utility is pretty good at auto-detecting sparse files, but it also provides the cp --sparse=always and cp --sparse=never options, if the auto-detection is not desired.
- The tar utility will archive sparse files with their non-sparse size when executed with default behavior, but tar --sparse will auto-detect sparse files, and archive them with their sparse size.

Sparse File Support on Windows

On Windows, the following file systems support sparse files:

- `NTFS`

On Windows, file systems need to support the DeviceIoControl() function with the FSCTL_SET_SPARSE and FSCTL_SET_ZERO_DATA control codes. For example:

```
DeviceIoControl(file_handle, FSCTL_SET_SPARSE, inbuf, inbuf_size,
   outbuf, outbuf_size,  NULL, &overlapped)
...
DeviceIoControl(file_handle, FSCTL_SET_ZERO_DATA, inbuf, inbuf_size,
   outbuf, outbuf_size,  NULL, &overlapped)
```

Configuring InnoDB to use Sparse Files

In MariaDB 10.3 and later, InnoDB uses the *punch hole* technique to create sparse files used automatically when the

underlying file system supports sparse files.

In MariaDB 10.2 and before, InnoDB can be configured to use the *punch hole* technique to create sparse files by configuring the innodb_use_trim and innodb_use_fallocate system variables. These system variables can be set in a server option group in an option file prior to starting up the server. For example:

```
[mariadb]
...
innodb_use_trim=ON
innodb_use_fallocate=ON
```

Optimized for Flash Storage

InnoDB page compression was designed to be optimized on solid state drives (SSDs) and other flash storage.

InnoDB page compression was originally developed by collaborating with Fusion-io. As a consequence, it was originally designed to work best on FusionIO devices using NVMFS. Fusion-io has since been acquired by Western Digital, and they have decided not to continue supporting NVMFS.

However, InnoDB page compression is still likely to be most optimized on solid state drives (SSDs) and other flash storage.

InnoDB page compression works without any issues on hard disk drives (HDDs). However, since its compression relies on the use of sparse files, the data may be somewhat fragmented on disk. This fragmentation may hurt performance on HDDs, since they handle random reads and writes much more slowly than flash storage.

Configuring InnoDB Page Flushing

With InnoDB page compression, pages are compressed when they are flushed to disk. Therefore, it can be helpful to optimize the configuration of InnoDB's page flushing. See InnoDB Page Flushing for more information.

Monitoring InnoDB Page Compression

InnoDB page compression can be monitored by querying the following status variables with SHOW GLOBAL STATUS:

Status Variable	Description
Innodb_page_compression_saved	Bytes saved by compression
Innodb_page_compression_trim_sect512	Number of 512 sectors trimmed
Innodb_page_compression_trim_sect1024	Number of 1024 sectors trimmed
Innodb_page_compression_trim_sect2048	Number of 2048 sectors trimmed
Innodb_page_compression_trim_sect4096	Number of 4096 sectors trimmed
Innodb_page_compression_trim_sect8192	Number of 8192 sectors trimmed
Innodb_page_compression_trim_sect16384	Number of 16384 sectors trimmed
Innodb_page_compression_trim_sect32768	Number of 32768 sectors trimmed
Innodb_num_pages_page_compressed	Number of pages compressed
Innodb_num_page_compressed_trim_op	Number of trim operations
Innodb_num_page_compressed_trim_op_saved	Number of trim operations saved
Innodb_num_pages_page_decompressed	Number of pages decompressed
Innodb_num_pages_page_compression_error	Number of compression errors

With InnoDB page compression, a page is only compressed when it is flushed to disk. This means that if you are monitoring InnoDB page compression via these status variables, then the status variables values will only get incremented when the dirty pages are flushed to disk, which does not necessarily happen immediately. For example:

```
CREATE TABLE `tab` (
     `id` int(11) NOT NULL,
     `str` varchar(50) DEFAULT NULL,
     PRIMARY KEY (`id`)
   ) ENGINE=InnoDB;

INSERT INTO tab VALUES (1, 'str1');

SHOW GLOBAL STATUS LIKE 'Innodb_num_pages_page_compressed';
+-----------------------------------+-------+
| Variable_name                     | Value |
+-----------------------------------+-------+
| Innodb_num_pages_page_compressed | 0     |
+-----------------------------------+-------+

SET GLOBAL innodb_file_per_table=ON;

SET GLOBAL innodb_file_format='Barracuda';

SET GLOBAL innodb_default_row_format='dynamic';

SET GLOBAL innodb_compression_algorithm='lzma';

ALTER TABLE tab PAGE_COMPRESSED=1;

SHOW GLOBAL STATUS LIKE 'Innodb_num_pages_page_compressed';
+-----------------------------------+-------+
| Variable_name                     | Value |
+-----------------------------------+-------+
| Innodb_num_pages_page_compressed | 0     |
+-----------------------------------+-------+

SELECT SLEEP(10);
+-----------+
| SLEEP(10) |
+-----------+
|         0 |
+-----------+

SHOW GLOBAL STATUS LIKE 'Innodb_num_pages_page_compressed';
+-----------------------------------+-------+
| Variable_name                     | Value |
+-----------------------------------+-------+
| Innodb_num_pages_page_compressed | 3     |
+-----------------------------------+-------+
```

Compatibility with Backup Tools

Mariabackup supports InnoDB page compression.

Percona XtraBackup does not support InnoDB page compression.

Acknowledgements

- InnoDB page compression was developed by collaborating with Fusion-io. Special thanks especially to Dhananjoy Das and Torben Mathiasen.

5.3.2.22 InnoDB Data Scrubbing

> Note that most of the background and redo log scrubbing code has been removed in MariaDB 10.5.2. See MDEV-15528 and MDEV-21870.

Sometimes there is a requirement that when some data is deleted, it is really gone. This might be the case when one stores user's personal information or some other sensitive data. Normally though, when a row is deleted, the space is only marked as free on the page. It may eventually be overwritten, but there is no guarantee when that will happen. A copy of the deleted rows may also be present in the log files.

MariaDB 10.1.3 introduced support for InnoDB data scrubbing. Background threads periodically scan tablespaces and logs and remove all data that should be deleted. The number of background threads for tablespace scans is set by innodb-encryption-threads. Log scrubbing happens in a separate thread.

To configure scrubbing one can use the following variables:

innodb-background-scrub-data-check-interval	Seconds	Check at this intervall if tablespaces needs scrubbing. Deprecated and ignored from MariaDB 10.5.2.
innodb-background-scrub-data-compressed	Boolean	Enable scrubbing of compressed data by background threads. Deprecated and ignored from MariaDB 10.5.2.
innodb-background-scrub-data-interval	Seconds	Scrub spaces that were last scrubbed longer than this many seconds ago. Deprecated and ignored from MariaDB 10.5.2.
innodb-background-scrub-data-uncompressed	Boolean	Enable scrubbing of uncompressed data by background threads. Deprecated and ignored from MariaDB 10.5.2.
innodb-immediate-scrub-data-uncompressed	Boolean	Enable scrubbing of uncompressed data
innodb-scrub-log	Boolean	Enable redo log scrubbing. Deprecated and ignored from MariaDB 10.5.2.
innodb-scrub-log-speed	Bytes/sec	Redo log scrubbing speed in bytes/sec. Deprecated and ignored from MariaDB 10.5.2.

Redo log scrubbing did not fully work as intended, and was deprecated and ignored in MariaDB 10.5.2 (MDEV-21870). If old log contents should be kept secret, then enabling innodb_encrypt_log or setting a smaller innodb_log_file_size could help.

The Information Schema INNODB_TABLESPACES_SCRUBBING table contains scrubbing information.

Thanks

- Scrubbing was donated to the MariaDB project by Google.

5.3.2.23 InnoDB Lock Modes

Contents

Locks are acquired by a transaction to prevent concurrent transactions from modifying, or even reading, some rows or ranges of rows. This is done to make sure that concurrent write operations never collide.

InnoDB supports a number of lock modes.

Shared and Exclusive Locks

The two standard row-level locks are *share locks*(S) and *exclusive locks*(X).

A shared lock is obtained to read a row, and allows other transactions to read the locked row, but not to write to the locked row. Other transactions may also acquire their own shared locks.

An exclusive lock is obtained to write to a row, and stops other transactions from locking the same row. It's specific behavior depends on the isolation level; the default (REPEATABLE READ), allows other transactions to read from the exclusively locked row.

Intention Locks

InnoDB also permits table locking, and to allow locking at both table and row level to co-exist gracefully, a series of locks called *intention locks* exist.

An *intention shared lock*(IS) indicates that a transaction intends to set a shared lock.

An *intention exclusive lock*(IX) indicates that a transaction intends to set an exclusive lock.

Whether a lock is granted or not can be summarised as follows:

- An X lock is not granted if any other lock (X, S, IX, IS) is held.
- An S lock is not granted if an X or IX lock is held. It is granted if an S or IS lock is held.
- An IX lock is not granted if in X or S lock is held. It is granted if an IX or IS lock is held.
- An IS lock is not granted if an X lock is held. It is granted if an S, IX or IS lock is held.

AUTO_INCREMENT Locks

Locks are also required for auto-increments - see AUTO_INCREMENT handling in InnoDB.

Gap Locks

With the default isolation level, `REPEATABLE READ`, and, until MariaDB 10.4, the default setting of the innodb_locks_unsafe_for_binlog variable, a method called gap locking is used. When InnoDB sets a shared or exclusive lock on a record, it's actually on the index record. Records will have an internal InnoDB index even if they don't have a unique index defined. At the same time, a lock is held on the gap before the index record, so that another transaction cannot insert a new index record in the gap between the record and the preceding record.

The gap can be a single index value, multiple index values, or not exist at all depending on the contents of the index.

If a statement uses all the columns of a unique index to search for unique row, gap locking is not used.

Similar to the shared and exclusive intention locks described above, there can be a number of types of gap locks. These include the shared gap lock, exclusive gap lock, intention shared gap lock and intention exclusive gap lock.

Gap locks are disabled if the innodb_locks_unsafe_for_binlog system variable is set (until MariaDB 10.4), or the isolation level is set to `READ COMMITTED`.

5.3.2.24 InnoDB Monitors

Contents

The InnoDB Monitor refers to particular kinds of monitors included in MariaDB and since the early versions of MySQL.

There are four types: the standard InnoDB monitor, the InnoDB Lock Monitor, InnoDB Tablespace Monitor and the InnoDB Table Monitor.

Standard InnoDB Monitor

The standard InnoDB Monitor returns extensive InnoDB information, particularly lock, semaphore, I/O and buffer activity:

To enable the standard InnoDB Monitor, from MariaDB 10.0.14, set the innodb_status_output system variable to 1. Before MariaDB 10.0.14, running the following statement was the method used:

```
CREATE TABLE innodb_monitor (a INT) ENGINE=INNODB;
```

To disable the standard InnoDB monitor, either set the system variable to zero, or, before MariaDB 10.0.14, drop the table

```
DROP TABLE innodb_monitor;
```

> The CREATE TABLE and DROP TABLE method of enabling and disabling the InnoDB Monitor has been deprecated, and may be removed in a future version of MariaDB.

For a description of the output, see SHOW ENGINE INNODB STATUS.

InnoDB Lock Monitor

The InnoDB Lock Monitor displays additional lock information.

To enable the InnoDB Lock Monitor, the standard InnoDB monitor must be enabled. Then, from MariaDB 10.0.14, set the

innodb_status_output_locks system variable to 1. Before MariaDB 10.0.14, running the following statement was the method used:

```
CREATE TABLE innodb_lock_monitor (a INT) ENGINE=INNODB;
```

To disable the standard InnoDB monitor, either set the system variable to zero, or, before MariaDB 10.0.14, drop the table

```
DROP TABLE innodb_lock_monitor;
```

> The CREATE TABLE and DROP TABLE method of enabling and disabling the InnoDB Lock Monitor has been deprecated, and may be removed in a future version of MariaDB.

InnoDB Tablespace Monitor

> The InnoDB Tablespace Monitor is deprecated, and may be removed in a future version of MariaDB.

Enabling the Tablespace Monitor outputs a list of file segments in the shared tablespace to the error log, and validates the tablespace allocation data structures.

To enable the Tablespace Monitor, run the following statement:

```
CREATE TABLE innodb_tablespace_monitor (a INT) ENGINE=INNODB;
```

To disable it, drop the table:

```
DROP TABLE innodb_tablespace_monitor;
```

InnoDB Table Monitor

> The InnoDB Table Monitor is deprecated, and may be removed in a future version of MariaDB.

Enabling the Table Monitor outputs the contents of the InnoDB internal data dictionary to the error log every fifteen seconds.

To enable the Table Monitor, run the following statement:

```
CREATE TABLE innodb_table_monitor (a INT) ENGINE=INNODB;
```

To disable it, drop the table:

```
DROP TABLE innodb_table_monitor;
```

SHOW ENGINE INNODB STATUS

The SHOW ENGINE INNODB STATUS statement can be used to obtain the standard InnoDB Monitor output when required, rather than sending it to the error log. It will also display the InnoDB Lock Monitor information if the innodb_status_output_locks system variable is set to `1`.

5.3.2.25 InnoDB Encryption Overview

Contents

MariaDB supports data-at-rest encryption for tables using the InnoDB storage engines. When enabled, the server encrypts data when it writes it to and decrypts data when it reads it from the file system. You can configure InnoDB encryption to

automatically have all new InnoDB tables automatically encrypted, or specify encrypt per table.

For encrypting data with the Aria storage engine, see Encrypting Data for Aria.

Basic Configuration

Using data-at-rest encryption requires that you first configure an Encryption Key Management plugin, such as the `file_key_management` or `aws_key_management` plugins. MariaDB uses this plugin to store, retrieve and manage the various keys it uses when encrypting data to and decrypting data from the file system.

Once you have the plugin configured, you need to set a few additional system variables to enable encryption on InnoDB tables, including `innodb_encrypt_tables`, `innodb_encrypt_logs`, `innodb_encryption_threads`, and `innodb_encryption_rotate_key_age`.

```
[mariadb]
...

# File Key Management
plugin_load_add = file_key_management
file_key_management_filename = /etc/mysql/encryption/keyfile.enc
file_key_management_filekey = FILE:/etc/mysql/encryption/keyfile.key
file_key_management_encryption_algorithm = AES_CTR

# InnoDB Encryption
innodb_encrypt_tables = ON
innodb_encrypt_temporary_tables = ON
innodb_encrypt_log = ON
innodb_encryption_threads = 4
innodb_encryption_rotate_key_age = 1
```

For more information on system variables for encryption and other features, see the InnoDB system variables page.

Creating Encrypted Tables

To create encrypted tables, specify the table options `ENCRYPTED=YES` and `ENCRYPTION_KEY_ID=` with a corresponding key id;

```
CREATE TABLE t (i int primary key) ENGINE=InnoDB ENCRYPTED=YES ENCRYPTION_KEY_ID=2;
```

Finding Encrypted Tables

When using data-at-rest encryption with the InnoDB storage engine, it is not necessary that you encrypt every table in your database. You can check which tables are encrypted and which are not by querying the `INNODB_TABLESPACES_ENCRYPTION` table in the Information Schema. This table provides information on which tablespaces are encrypted, which encryption key each tablespace is encrypted with, and whether the background encryption threads are currently working on the tablespace. Since the system tablespace can also contain tables, it can be helpful to join the `INNODB_TABLESPACES_ENCRYPTION` table with the `INNODB_SYS_TABLES` table to find out the encryption status of each specific table, rather than each tablespace. For example:

```
SELECT st.SPACE, st.NAME, te.ENCRYPTION_SCHEME, te.ROTATING_OR_FLUSHING
FROM information_schema.INNODB_TABLESPACES_ENCRYPTION te
JOIN information_schema.INNODB_SYS_TABLES st
   ON te.SPACE = st.SPACE \G
*************************** 1. row ***************************
               SPACE: 0
                NAME: SYS_DATAFILES
   ENCRYPTION_SCHEME: 1
ROTATING_OR_FLUSHING: 0
*************************** 2. row ***************************
               SPACE: 0
                NAME: SYS_FOREIGN
   ENCRYPTION_SCHEME: 1
ROTATING_OR_FLUSHING: 0
*************************** 3. row ***************************
               SPACE: 0
                NAME: SYS_FOREIGN_COLS
   ENCRYPTION_SCHEME: 1
ROTATING_OR_FLUSHING: 0
```

```
*************************** 4. row ***************************
               SPACE: 0
                NAME: SYS_TABLESPACES
   ENCRYPTION_SCHEME: 1
ROTATING_OR_FLUSHING: 0
*************************** 5. row ***************************
               SPACE: 0
                NAME: SYS_VIRTUAL
   ENCRYPTION_SCHEME: 1
ROTATING_OR_FLUSHING: 0
*************************** 6. row ***************************
               SPACE: 0
                NAME: db1/default_encrypted_tab1
   ENCRYPTION_SCHEME: 1
ROTATING_OR_FLUSHING: 0
*************************** 7. row ***************************
               SPACE: 416
                NAME: db1/default_encrypted_tab2
   ENCRYPTION_SCHEME: 1
ROTATING_OR_FLUSHING: 0
*************************** 8. row ***************************
               SPACE: 402
                NAME: db1/tab
   ENCRYPTION_SCHEME: 1
ROTATING_OR_FLUSHING: 0
*************************** 9. row ***************************
               SPACE: 185
                NAME: db1/tab1
   ENCRYPTION_SCHEME: 1
ROTATING_OR_FLUSHING: 0
*************************** 10. row ***************************
               SPACE: 184
                NAME: db1/tab2
   ENCRYPTION_SCHEME: 1
ROTATING_OR_FLUSHING: 0
*************************** 11. row ***************************
               SPACE: 414
                NAME: db1/testgb2
   ENCRYPTION_SCHEME: 1
ROTATING_OR_FLUSHING: 0
*************************** 12. row ***************************
               SPACE: 4
                NAME: mysql/gtid_slave_pos
   ENCRYPTION_SCHEME: 1
ROTATING_OR_FLUSHING: 0
*************************** 13. row ***************************
               SPACE: 2
                NAME: mysql/innodb_index_stats
   ENCRYPTION_SCHEME: 1
ROTATING_OR_FLUSHING: 0
*************************** 14. row ***************************
               SPACE: 1
                NAME: mysql/innodb_table_stats
   ENCRYPTION_SCHEME: 1
ROTATING_OR_FLUSHING: 0
*************************** 15. row ***************************
               SPACE: 3
                NAME: mysql/transaction_registry
   ENCRYPTION_SCHEME: 1
ROTATING_OR_FLUSHING: 0
15 rows in set (0.000 sec)
```

Redo Logs

Using data-at-rest encryption with InnoDB, the `innodb_encrypt_tables` system variable only encrypts the InnoDB tablespaces. In order to also encrypt the InnoDB Redo Logs, you also need to set the `innodb_encrypt_logs` system variable.

Beginning in MariaDB 10.4, where the encryption key management plugin supports key rotation the InnoDB Redo Log can also rotate encryption keys. In previous releases, the Redo Log can only use the first encryption key.

5.3.3 MariaDB ColumnStore

MariaDB ColumnStore is a columnar storage engine that utilizes a massively parallel distributed data architecture. It's a columnar storage system built by porting InfiniDB 4.6.7 to MariaDB, and released under the GPL license.

From MariaDB 10.5.4, it is available as a storage engine for MariaDB Server. Before then, it is only available as a separate download.

MariaDB ColumnStore is designed for big data scaling to process petabytes of data, linear scalability and exceptional performance with real-time response to analytical queries. It leverages the I/O benefits of columnar storage, compression, just-in-time projection, and horizontal and vertical partitioning to deliver tremendous performance when analyzing large data sets.

Documentation for the latest release of Columnstore is not available on the Knowledge Base. Instead, see:

- Release Notes
- Deployment Instructions

About MariaDB ColumnStore

About MariaDB ColumnStore.

MariaDB ColumnStore Release Notes

MariaDB ColumnStore Release Notes

ColumnStore Getting Started

Quick summary of steps needed to install MariaDB ColumnStore

ColumnStore Upgrade Guides

Documentation on upgrading from prior versions and InfiniDB migration.

ColumnStore Architecture

MariaDB ColumnStore Architecture

Managing ColumnStore

Managing MariaDB ColumnStore System Environment and Database

ColumnStore Data Ingestion

How to load and manipulate data into MariaDB ColumnStore

ColumnStore SQL Structure and Commands

SQL syntax supported by MariaDB ColumnStore

ColumnStore Performance Tuning

Information relating to configuring and analyzing the ColumnStore system for optimal performance.

ColumnStore System Variables

ColumnStore System Variables

ColumnStore Security Vulnerabilities

Security vulnerabilities affecting MariaDB ColumnStore

ColumnStore Troubleshooting

Articles on troubleshooting tips and techniques

StorageManager

Articles on StorageManager and S3 configuration

Using MariaDB ColumnStore

Provides details on using third party products and tools with MariaDB ColumnStore

Building ColumnStore in MariaDB

This is a description of how to build and start a local ColumnStore install...

Can't create a table starting with a capital letter. All tables are lower case-

Hi, I was playing around with my MariaDB ColumnStore and I noticed the I am...

There are 60 related questions.

5.3.4 Aria

Information about the Aria storage engine. From MariaDB 10.4, system tables use the Aria storage engine.

Aria Storage Engine

The Aria storage engine is compiled-in by default and is considered as an upgrade to MyISAM.

Aria Clients and Utilities

Clients and utilities for working with Aria tables

Aria FAQ

Frequently-asked questions about the Aria storage engine.

Aria Storage Formats

Table storage formats in Aria - PAGE, FIXED and DYNAMIC

Aria Status Variables

Aria-related server status variables.

Aria System Variables

Aria-related system variables.

Aria Group Commit

Aria group commits for speeding up multi-user inserts

Benchmarking Aria

Aria benchmarks

Aria Two-step Deadlock Detection

How Aria detects and deals with deadlocks

Aria Encryption Overview

Data-at-rest encryption for user-created tables and internal on-disk tempor...

The Aria Name

How Aria got its name.

There are 7 related questions.

5.3.4.1 Aria Storage Engine

Contents

The Aria storage engine is compiled in by default from MariaDB 5.1 and it is required to be 'in use' when MariaDB is started.

From MariaDB 10.4, all system tables are Aria.

Additionally, internal on-disk tables are in the Aria table format instead of the MyISAM table format. This should speed up some GROUP BY and DISTINCT queries because Aria has better caching than MyISAM.

Note: The ***Aria*** storage engine was previously called ***Maria*** (see The Aria Name for details on the rename) and in previous versions of MariaDB the engine was still called Maria.

The following table options to Aria tables in CREATE TABLE and ALTER TABLE:

- **`TRANSACTIONAL= 0 | 1`** : If the `TRANSACTIONAL` table option is set for an Aria table, then the table will be crash-safe. This is implemented by logging any changes to the table to Aria's transaction log, and syncing those writes at the end of the statement. This will marginally slow down writes and updates. However, the benefit is that if the server dies before the statement ends, all non-durable changes will roll back to the state at the beginning of the statement. This also needs up to 6 bytes more for each row and key to store the transaction id (to allow concurrent insert's and selects).
  - `TRANSACTIONAL=1` is not supported for partitioned tables.
  - An Aria table's default value for the `TRANSACTIONAL` table option depends on the table's value for the `ROW_FORMAT` table option. See below for more details.
  - If the `TRANSACTIONAL` table option is set for an Aria table, the table does not actually support transactions. See MDEV-21364 for more information. In this context, *transactional* just means *crash-safe*.
- **`PAGE_CHECKSUM= 0 | 1`** : If index and data should use page checksums for extra safety.
- **`TABLE_CHECKSUM= 0 | 1`** : Same as `CHECKSUM` in MySQL 5.1
- **`ROW_FORMAT=PAGE | FIXED | DYNAMIC`** : The table's row format.
  - The default value is `PAGE` .
  - To emulate MyISAM, set `ROW_FORMAT=FIXED` or `ROW_FORMAT=DYNAMIC`

The `TRANSACTIONAL` and `ROW_FORMAT` table options interact as follows:

- If `TRANSACTIONAL=1` is set, then the only supported row format is `PAGE` . If `ROW_FORMAT` is set to some other value, then Aria issues a warning, but still forces the row format to be `PAGE` .
- If `TRANSACTIONAL=0` is set, then the table will be not be crash-safe, and any row format is supported.
- If `TRANSACTIONAL` is not set to any value, then any row format is supported. If `ROW_FORMAT` is set, then the table will use that row format. Otherwise, the table will use the default `PAGE` row format. In this case, if the table uses the `PAGE` row format, then it will be crash-safe. If it uses some other row format, then it will not be crash-safe.

Some other improvements are:

- CHECKSUM TABLE now ignores values in NULL fields. This makes `CHECKSUM TABLE` faster and fixes some cases where same table definition could give different checksum values depending on row format. The disadvantage is that the value is now different compared to other MySQL installations. The new checksum calculation is fixed for all table engines that uses the default way to calculate and MyISAM which does the calculation internally. Note: Old MyISAM tables with internal checksum will return the same checksum as before. To fix them to calculate according to new rules you have to do an `ALTER TABLE` . You can use the old ways to calculate checksums by using the option `--old` to mariadbd*mysqld or set the system variable '`@@old`' to `1` when you do `CHECKSUM TABLE ... EXTENDED;`*
- At startup Aria will check the Aria logs and automatically recover the tables from the last checkpoint if the server was not taken down correctly. See Aria Log Files

Startup Options for Aria

For a full list, see Aria System Variables.

In normal operations, the only variables you have to consider are:

- aria-pagecache-buffer-size
  - This is where all index and data pages are cached. The bigger this is, the faster Aria will work.
- aria-block-size
  - The default value 8192, should be ok for most cases. The only problem with a higher value is that it takes longer to find a packed key in the block as one has to search roughly 8192/2 to find each key. We plan to fix this by adding a dictionary at the end of the page to be able to do a binary search within the block before starting a scan. Until this is done and key lookups takes too long time even if you are not hitting disk, then you should consider making this smaller.
  - Possible values to try are `2048` , `4096` or `8192`
  - Note that you can't change this without dumping, deleting old tables and deleting all log files and then restoring your Aria tables. (This is the only option that requires a dump and load)
- aria-log-purge-type
  - Set this to " `at_flush` " if you want to keep a copy of the transaction logs (good as an extra backup). The logs will stay around until you execute FLUSH ENGINE LOGS.

Aria Log Files

`aria_log_control` file is a very short log file (52 bytes) that contains the current state of all Aria tables related to logging and checkpoints. In particular, it contains the following information:

```
Aria file version: 1
Block size: 8192
maria_uuid: ee948482-6cb7-11ed-accb-3c7c3ff16468
last_checkpoint_lsn: (1,0x235a)
last_log_number: 1
trid: 28
recovery_failures: 0
```

- The `uuid` is a unique identifier per system. All Aria files created will have a copy of this in their .MAI headers. This is mainly used to check if someone has copied an Aria file between MariaDB servers.
- `last_checkpoint_lsn` and `last_log_number` are information about the current aria_log files.
- `trid` is the highest transaction number seen so far. Used by recovery.

`aria_log.*` files contain the log of all operations that change Aria files (including create table, drop table, insert etc..) This is a 'normal' WAL (Write Ahead Log), similar to the InnoDB log file, except that aria_logs contain both redo and undo. Old aria_log files are automatically deleted when they are not needed anymore (Neither the last checkpoint or any running transaction need to refer to the old data anymore).

Missing valid id

The error `Missing valid id at start of file. File is not a valid aria control file` means that something overwrote at least the first 4 bytes in the file. This can happen due to a problem with the file system (hardware or software), or a bug in which a thread inside MariaDB wrote on the wrong file descriptor (in which case you should report the bug, attaching a copy of the control file to assist).

In the case of a corrupted log file, with the server shut down, one should be able to fix that by deleting all aria_log files. If the control_file is corrupted, then one has to delete the aria_control_file and all aria_log.* files. The effect of this is that on table open of an Aria table, the server will think that it has been moved from another system and do an automatic check and repair of it. If there was no issues, the table will be opened and can be used as normal. See also When is it safe to remove old log files.

1.3.5 Aria Clients and Utilities

5.3.4.3 Aria FAQ

This FAQ provides information on the Aria storage engine.

> The ***Aria*** storage engine was previously known as ***Maria***, (see, the Aria Name). In current releases of MariaDB, you can refer to the engine as Maria or Aria. As this will change in future releases, please update references in your scripts and automation to use the correct name.

Contents

What is Aria?

Aria is a storage engine for MySQL® and MariaDB. It was originally developed with the goal of becoming the default transactional **and** non-transactional storage engine for MariaDB and MySQL.

It has been in development since 2007 and was first announced on Monty's blog. The same core MySQL engineers who developed the MySQL server and the MyISAM, MERGE, and MEMORY storage engines are also working on Aria.

Why is the engine called Aria?

Originally, the storage engine was called **Maria**, after Monty's younger daughter. Monty named MySQL after his first child, **My** and his second child **Max** gave his name to MaxDB and the MySQL-Max distributions.

In practice, having both *MariaDB* the database server and *Maria* the storage engine with such similar names proved confusing. To mitigate this, the decision was made to change the name. A Rename Maria contest was held during the first half of 2010 and names were submitted from around the world. Monty picked the name ***Aria*** from a short list of finalist. Chris Tooley, who suggested it, received the prize of a Linux-powered System 76 Meerkat NetTop from Monty Program.

For more information, see the Aria Name.

What's the goal for the current version?

The current version of Aria is 1.5. The goal of this release is to develop a crash-safe alternative to MyISAM. That is, when MariaDB restarts after a crash, Aria recovers all tables to the state as of the start of a statement or at the start of the last `LOCK TABLES` statement.

The current goal is to keep the code stable and fix all bugs.

What's the goal for the next version?

The next version of Aria is 2.0. The goal for this release is to develop a fully transactional storage engine with at least all the major features of InnoDB.

Currently, Aria 2.0 is on hold as its developers are focusing on improving MariaDB. However, they are interested in working with interested customers and partners to add more features to Aria and eventually release 2.0.

These are some of the goals for Aria 2.0:

- ACID compliant
- Commit/Rollback
- Concurrent updates/deletes
- Row locking

- Group commit (Already in MariaDB 5.2)
- Faster lookup in index pages (Page directory)

Beginning in Aria 2.5, the plan is to focus on improving performance.

What is the ultimate goal of Aria?

Long term, we have the following goals for Aria:

- To create a new, ACID and Multi-Version Concurrency Control (MVCC), transactional storage engine that can function as both the default non-transactional and transactional storage engine for MariaDB and MySQL®.
- To be a MyISAM replacement. This is possible because Aria can also be run in non-transactional mode, supports the same row formats as MyISAM, and supports or will support all major features of MyISAM.
- To be the default non-transactional engine in MariaDB (instead of MyISAM).

What are the design goals in Aria?

- Multi-Version Concurrency Control (MVCC) and ACID storage engine.
- Optionally non-transactional tables that should be 'as fast and as compact' as MyISAM tables.
- Be able to use Aria for internal temporary tables in MariaDB (instead of MyISAM).
- All indexes should have equal speed (clustered index is not on our current road map for Aria. If you need clustered index, you should use XtraDB).
- Allow 'any' length transactions to work (Having long running transactions will cause more log space to be used).
- Allow log shipping; that is, you can do incremental backups of Aria tables just by copying the Aria logs.
- Allow copying of Aria tables between different Aria servers (under some well-defined constraints).
- Better blob handling (than is currently offered in MyISAM, at a minimum).
- No memory copying or extra memory used for blobs on insert/update.
- Blobs allocated in big sequential blocks - Less fragmentation over time.
- Blobs are stored so that Aria can easily be extended to have access to any part of a blob with a single fetch in the future.
- Efficient storage on disk (that is, low row data overhead, low page data overhead and little lost space on pages). Note: There is still some more work to succeed with this goal. The disk layout is fine, but we need more in-memory caches to ensure that we get a higher fill factor on the pages.
- Small footprint, to make MariaDB + Aria suitable for desktop and embedded applications.
- Flexible memory allocation and scalable algorithms to utilize large amounts of memory efficiently, when it is available.

Where can I find documentation and help about Aria?

Documentation is available at Aria and related topics. The project is maintained on GitHub.

If you want to know what happens or be part of developing Aria, you can subscribe to the developers, docs, or discuss mailing lists.

To report and check bugs in Aria, see Reporting Bugs.

You can usually find some of the Maria developers on our Zulip instance at https://mariadb.zulipchat.com or on the IRC channel #maria at https://libera.chat/.

Who develops Aria?

The Core Team who develop Aria are:

Technical lead

- Michael "Monty" Widenius - Creator of MySQL and MyISAM

Core Developers (in alphabetical order)

- Guilhem Bichot - Replication expert, on line backup for MyISAM, etc.
- Kristian Nielsen - MySQL build tools, NDB, MySQL server
- Oleksandr Byelkin - Query cache, sub-queries, views.
- Sergei Golubchik - Server Architect, Full text search, keys for MyISAM-Merge, Plugin architecture, etc.

All except Guilhem Bichot are working for MariaDB Corporation Ab.

What is the release policy/schedule of Aria?

Aria follows the same release criteria as for MariaDB. Some clarifications, unique for the Aria storage engine:

- Aria index and data file formats should be backwards and forwards compatible to ensure easy upgrades and downgrades.
- The log file format should also be compatible, but we don't make any guarantees yet. In some cases when upgrading, you must remove the old `aria_log.%` and `maria_log.%` files before restarting MariaDB. (So far, this has only occurred in the upgrade from MariaDB 5.1 and MariaDB 5.2).

Extended commitment for Beta 1.5

- Aria is now feature complete according to specification.

How does Aria 1.5 Compare to MyISAM?

Aria 1.0 was basically a crash-safe non-transactional version of MyISAM. Aria 1.5 added more concurrency (multiple inserter) and some optimizations.

Aria supports all aspects of MyISAM, except as noted below. This includes external and internal check/repair/compressing of rows, different row formats, different index compress formats, `aria_chk` etc. After a normal shutdown you can copy Aria files between servers.

Advantages of Aria compared to MyISAM

- Data and indexes are crash safe.
- On a crash, changes will be rolled back to state of the start of a statement or a last `LOCK TABLES` statement.
- Aria can replay almost everything from the log. (Including `CREATE`, `DROP`, `RENAME`, `TRUNCATE` tables). Therefore, you make a backup of Aria by just copying the log. The things that can't be replayed (yet) are:
  - Batch `INSERT` into an empty table (This includes `LOAD DATA INFILE`, `SELECT... INSERT` and `INSERT` (many rows)).
  - `ALTER TABLE`. Note that `.frm` tables are NOT recreated!
- `LOAD INDEX` can skip index blocks for unwanted indexes.
- Supports all MyISAM `ROW` formats and new `PAGE` format where data is stored in pages. (default size is 8K).
- Multiple concurrent inserters into the same table.
- When using `PAGE` format (default) row data is cached by page cache.
- Aria has unit tests of most parts.
- Supports both crash-safe (soon to be transactional) and not transactional tables. (Non-transactional tables are not logged and rows uses less space): `CREATE TABLE foo (...) TRANSACTIONAL=0|1 ENGINE=Aria`.
- `PAGE` is the only crash-safe/transactional row format.
- `PAGE` format should give a notable speed improvement on systems which have bad data caching. (For example Windows).
- From MariaDB 10.5, max key length is 2000 bytes, compared to 1000 bytes in MyISAM.

Differences between Aria and MyISAM

- Aria uses BIG (1G by default) log files.
- Aria has a log control file (`aria_log_control`) and log files (`aria_log.%`). The log files can be automatically purged when not needed or purged on demand (after backup).
- Aria uses 8K pages by default (MyISAM uses 1K). This makes Aria a bit faster when using keys of fixed size, but slower when using variable-length packed keys (until we add a directory to index pages).

Disadvantages of Aria compared to MyISAM

- Aria doesn't support `INSERT DELAYED`.
- Aria does not support multiple key caches.
- Storage of very small rows (< 25 bytes) are not efficient for `PAGE` format.
- `MERGE` tables don't support Aria (should be very easy to add later).
- Aria data pages in block format have an overhead of 10 bytes/page and 5 bytes/row. Transaction and multiple concurrent-writer support will use an extra overhead of 7 bytes for new rows, 14 bytes for deleted rows and 0 bytes for old compacted rows.
- No external locking (MyISAM has external locking, but this is a rarely used feature).
- Aria has one page size for both index and data (defined when Aria is used the first time). MyISAM supports different page sizes per index.
- Small overhead (15 bytes) per index page.
- Aria doesn't support MySQL internal RAID (disabled in MyISAM too, it's a deprecated feature).
- Minimum data file size for PAGE format is 16K (with 8K pages).

- Aria doesn't support indexes on virtual fields.

Differences between MariaDB 5.1 release and the normal MySQL-5.1 release?

See:

- Aria storage engine
- MariaDB versus MySQL

Why do you use the `TRANSACTIONAL` keyword now when Aria is not yet transactional?

In the current development phase Aria tables created with `TRANSACTIONAL=1` are crash safe and atomic but not transactional because changes in Aria tables can't be rolled back with the `ROLLBACK` command. As we planned to make Aria tables fully transactional, we decided it was better to use the `TRANSACTIONAL` keyword from the start so so that applications don't need to be changed later.

What are the known problems with the MySQL-5.1-Maria release?

- See `KNOWN_BUGS.txt` for open/design bugs.
- See jira.mariadb.org for newly reported bugs. Please report anything you can't find here!
- If there is a bug in the Aria recovery code or in the code that generates the logs, or if the logs become corrupted, then mysqld may fail to start because Aria can't execute the logs at start up.
- Query cache and concurrent insert using page row format have a bug, please disable query cache while using page row format and MDEV-6817 isn't complete

If Aria doesn't start or you have an unrecoverable table (shouldn't happen):

- Remove the `aria_log.%` files from the data directory.
- Restart `mysqld` and run CHECK TABLE, REPAIR TABLE or mariadb-check on your Aria tables.

Alternatively,

- Remove logs and run aria_chk on your `*.MAI` files.

What is going to change in later Aria main releases?

The `LOCK TABLES` statement will not start a crash-safe segment. You should use BEGIN and COMMIT instead.

To make things future safe, you could do this:

```
BEGIN;
LOCK TABLES ....
UNLOCK TABLES;
COMMIT;
```

And later you can just remove the `LOCK TABLES` and `UNLOCK TABLES` statements.

How can I create a MyISAM-like (non-transactional) table in Aria?

Example:

```
CREATE TABLE t1 (a int) ROW_FORMAT=FIXED TRANSACTIONAL=0 PAGE_CHECKSUM=0;
CREATE TABLE t2 (a int) ROW_FORMAT=DYNAMIC TRANSACTIONAL=0 PAGE_CHECKSUM=0;
SHOW CREATE TABLE t1;
SHOW CREATE TABLE t2;
```

Note that the rows are not cached in the page cache for `FIXED` or `DYNAMIC` format. If you want to have the data cached (something MyISAM doesn't support) you should use `ROW_FORMAT=PAGE`:

```
CREATE TABLE t3 (a int) ROW_FORMAT=PAGE TRANSACTIONAL=0 PAGE_CHECKSUM=0;
SHOW CREATE TABLE t3;
```

You can use `PAGE_CHECKSUM=1` also for non-transactional tables; This puts a page checksums on all index pages. It also puts a checksum on data pages if you use `ROW_FORMAT=PAGE`.

You may still have a speed difference (may be slightly positive or negative) between MyISAM and Aria because of different page sizes. You can change the page size for MariaDB with `--aria-block-size=\#`, where `\#` is 1024, 2048, 4096, 8192, 16384 or 32768.

Note that if you change the page size you have to dump all your old tables into text (with mariadb-dump) and remove the old Aria log and files:

```
# rm datadir/aria_log*
```

What are the advantages/disadvantages of the new `PAGE` format compared to the old MyISAM-like row formats (`DYNAMIC` and `FIXED`)

The MyISAM-like `DYNAMIC` and `FIXED` format are extremely simple and have very little space overhead, so it's hard to beat them for when it comes to simple scanning of unmodified data. The `DYNAMIC` format does however get notably worse over time if you update the row a lot in a manner that increases the size of the row.

The advantages of the `PAGE` format (compared to `DYNAMIC` or `FIXED`) for non-transactional tables are:

- It's cached by the Page Cache, which gives better random performance (as it uses less system calls).
- Does not fragment as easily easily as the `DYNAMIC` format during `UPDATE` statements. The maximum number of fragments are very low.
- Code can easily be extended to only read the accessed columns (for example to skip reading blobs).
- Faster updates (compared to `DYNAMIC`).

The disadvantages are:

- Slight storage overhead (should only be notable for very small row sizes)
- Slower full table scan time.
- When using `row_format=PAGE`, (the default), Aria first writes the row, then the keys, at which point the check for duplicate keys happens. This makes `PAGE` format slower than `DYNAMIC` (or MyISAM) if there is a lot of duplicated keys because of the overhead of writing and removing the row. If this is a problem, you can use `row_format=DYNAMIC` to get same behavior as MyISAM.

What's the proper way to copy a Aria table from one place to another?

An Aria table consists of 3 files:

```
XXX.frm : The definition for the table, used by MySQL.
XXX.MYI : Aria internal information about the structure of the data and index and data for all in
XXX.MAD : The data.
```

It's safe to copy all the Aria files to another directory or MariaDB instance if any of the following holds:

- If you shutdown the MariaDB Server properly with mariadb-admin shutdown, so that there is nothing for Aria to recover when it starts.

or

- If you have run a FLUSH TABLES statement and not accessed the table using SQL from that time until the tables have been copied.

In addition, you must adhere the following rule for transactional tables:

You can't copy the table to a location within the same MariaDB server if the new table has existed before and the new table is still active in the Aria recovery log (that is, Aria may need to access the old data during recovery). If you are unsure whether the old name existed, run aria_chk --zerofill on the table before you use it.

After copying a transactional table and before you use the table, we recommend that you run the command:

```
$ aria_chk --zerofill table_name
```

This will overwrite all references to the logs (LSN), all transactional references (TRN) and all unused space with 0. It also marks the table as 'movable'. An additional benefit of zerofill is that the Aria files will compress better. No real data is ever removed as part of zerofill.

Aria will automatically notice if you have copied a table from another system and do 'zerofill' for the first access of the table if it was not marked as 'movable'. The reason for using aria_chk --zerofill is that you avoid a delay in the MariaDB server for the first access of the table.

Note that this automatic detection doesn't work if you copy tables within the same MariaDB server!

When is it safe to remove old log files?

If you want to remove the Aria log files (`aria_log.%`) with `rm` or delete, then you must first shut down MariaDB cleanly (for example, with mariadb-admin shutdown) before deleting the old files.

The same rules apply when upgrading MariaDB; When upgrading, first take down MariaDB in a clean way and then upgrade. This will allow you to remove the old log files if there are incompatible problems between releases.

Don't remove the `aria_log_control` file! This is not a log file, but a file that contains information about the Aria setup (current transaction id, unique id, next log file number etc.).

If you do, Aria will generate a new `aria_log_control` file at startup and will regard all old Aria files as files moved from another system. This means that they have to be 'zerofilled' before they can be used. This will happen automatically at next access of the Aria files, which can take some time if the files are big.

If this happens, you will see things like this in your mysqld.err file:

```
[Note] Zerofilling moved table: '.\database\xxxx'
```

As part of zerofilling no vital data is removed.

How does one solve the Missing valid id error?

See Aria Log Files for details.

5.3.4.4 Aria Storage Formats

Contents

The Aria storage engine supports three different table storage formats.

These are FIXED, DYNAMIC and PAGE, and they can be set with the ROW FORMAT option in the CREATE TABLE statement. PAGE is the default format, while FIXED and DYNAMIC are essentially the same as the MyISAM formats.

The SHOW TABLE STATUS statement can be used to see the storage format used by a table.

Fixed-length

Fixed-length (or static) tables contain records of a fixed-length. Each column is the same length for all records, regardless of the actual contents. It is the default format if a table has no BLOB, TEXT, VARCHAR or VARBINARY fields, and no ROW FORMAT is provided. You can also specify a fixed table with ROW_FORMAT=FIXED in the table definition.

> Tables containing BLOB or TEXT fields cannot be FIXED, as by design these are both dynamic fields.

Fixed-length tables have a number of characteristics

- fast, since MariaDB will always know where a record begins
- easy to cache
- take up more space than dynamic tables, as the maximum amount of storage space will be allocated to each record.

- reconstructing after a crash is uncomplicated due to the fixed positions
- no fragmentation or need to re-organize, unless records have been deleted and you want to free the space up.

Dynamic

Dynamic tables contain records of a variable length. It is the default format if a table has any BLOB, TEXT, VARCHAR or VARBINARY fields, and no ROW FORMAT is provided. You can also specify a DYNAMIC table with ROW_FORMAT=DYNAMIC in the table definition.

Dynamic tables have a number of characteristics

- Each row contains a header indicating the length of the row.
- Rows tend to become fragmented easily. UPDATING a record to be longer will likely ensure it is stored in different places on the disk.
- All string columns with a length of four or more are dynamic.
- They require much less space than fixed-length tables.
- Restoring after a crash is more complicated than with FIXED tables.

Page

Page format is the default format for Aria tables, and is the only format that can be used if TRANSACTIONAL=1.

Page tables have a number of characteristics:

- It's cached by the page cache, which gives better random performance as it uses fewer system calls.
- Does not fragment as easily easily as the DYNAMIC format during UPDATES. The maximum number of fragments are very low.
- Updates more quickly than dynamic tables.
- Has a slight storage overhead, mainly notable on very small rows
- Slower to perform a full table scan
- Slower if there are multiple duplicated keys, as Aria will first write a row, then keys, and only then check for duplicates

Transactional

See Aria Storage Engine for the impact of the TRANSACTIONAL option on the row format.

5.3.4.5 Aria Status Variables

Contents

This page documents status variables related to the Aria storage engine. See Server Status Variables for a complete list of status variables that can be viewed with SHOW STATUS.

See also the Full list of MariaDB options, system and status variables.

`Aria_pagecache_blocks_not_flushed`

- **Description:** The number of dirty blocks in the Aria page cache. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

`Aria_pagecache_blocks_unused`

- **Description:** Free blocks in the Aria page cache. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

`Aria_pagecache_blocks_used`

- **Description:** Blocks used in the Aria page cache. The global value can be flushed by `FLUSH STATUS`.
- **Scope:** Global
- **Data Type:** `numeric`

`Aria_pagecache_read_requests`

- **Description:** The number of requests to read something from the Aria page cache.
- **Scope:** Global
- **Data Type:** `numeric`

`Aria_pagecache_reads`

- **Description:** The number of Aria page cache read requests that caused a block to be read from the disk.
- **Scope:** Global
- **Data Type:** `numeric`

`Aria_pagecache_write_requests`

- **Description:** The number of requests to write a block to the Aria page cache.
- **Scope:** Global
- **Data Type:** `numeric`

`Aria_pagecache_writes`

- **Description:** The number of blocks written to disk from the Aria page cache.
- **Scope:** Global
- **Data Type:** `numeric`

`Aria_transaction_log_syncs`

- **Description:** The number of Aria log fsyncs.
- **Scope:** Global
- **Data Type:** `numeric`

5.3.4.6 Aria System Variables

Contents

This page documents system variables related to the Aria storage engine. For options that are not system variables, see Aria Options.

See Server System Variables for a complete list of system variables and instructions on setting system variables.

Also see the Full list of MariaDB options, system and status variables.

aria_block_size

- **Description:** Block size to be used for Aria index pages. Changing this requires dumping, deleting old tables and deleting all log files, and then restoring your Aria tables. If key lookups take too long (and one has to search roughly 8192/2 by default to find each key), can be made smaller, e.g. `4096`.
- **Commandline:** `--aria-block-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `8192`
- **Range:**
  - >= MariaDB 10.4.7: `4096` to `32768` in increments of `1024`
  - <= MariaDB 10.4.6: `1024` to `32768` in increments of `1024`

---

aria_checkpoint_interval

- **Description:** Interval in seconds between automatic checkpoints. 0 means 'no automatic checkpoints' which makes sense only for testing.
- **Commandline:** `--aria-checkpoint-interval=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `30`
- **Range:** `0` to `4294967295`

---

aria_checkpoint_log_activity

- **Description:** Number of bytes that the transaction log has to grow between checkpoints before a new checkpoint is written to the log.
- **Commandline:** `aria-checkpoint-log-activity=#`

- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1048576`
- **Range** `0` to `4294967295`

---

aria_encrypt_tables

- **Description:** Enables automatic encryption of all user-created Aria tables that have the `ROW_FORMAT` table option set to `PAGE` . See Data at Rest Encryption and Enabling Encryption for User-created Tables.
- **Commandline:** `aria-encrypt-tables={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

aria_force_start_after_recovery_failures

- **Description:** Number of consecutive log recovery failures after which logs will be automatically deleted to cure the problem; 0 (the default) disables the feature.
- **Commandline:** `--aria-force-start-after-recovery-failures=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `0`

---

aria_group_commit

- **Description:** Specifies Aria group commit mode.
- **Commandline:** `--aria_group_commit="value"`
- **Alias:** `maria_group_commit`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Valid values:**
  - `none` - *Group commit is disabled.*
  - `hard` - *Wait the number of microseconds specified by aria_group_commit_interval before actually doing the commit. If the interval is 0 then just check if any other threads have requested a commit during the time this commit was preparing (just before sync() file) and send their data to disk also before sync().*
  - `soft` - *The service thread will wait the specified time and then sync() to the log. If the interval is 0 then it won't wait for any commits (this is dangerous and should generally not be used in production)*
- **Default Value:** `none`

---

aria_group_commit_interval

- **Description:** Interval between Aria group commits in microseconds (1/1000000 second) for other threads to come and do a commit in "hard" mode and sync()/commit at all in "soft" mode. Option only has effect if aria_group_commit is used.
- **Commandline:** `--aria_group_commit_interval=#`
- **Alias:** `maria_group_commit_interval`
- **Scope:** Global
- **Dynamic:** No
- **Type:** numeric
- **Valid Values:**
  - **Default Value:** `0` *(no waiting)*
  - **Range:** `0-4294967295`

---

aria_log_file_size

- **Description:** Limit for Aria transaction log size

- **Commandline:** `--aria-log-file-size=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1073741824`

---

aria_log_purge_type

- **Description:** Specifies how the Aria transactional log will be purged. Set to `at_flush` to keep a copy of the transaction logs (good as an extra backup). The logs will stay until the next FLUSH LOGS;
- **Commandline:** `--aria-log-purge-type=name`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `enumeration`
- **Default Value:** `immediate`
- **Valid Values:** `immediate`, `external`, `at_flush`

---

aria_max_sort_file_size

- **Description:** Don't use the fast sort index method to created index if the temporary file would get bigger than this.
- **Commandline:** `--aria-max-sort-file-size=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `9223372036853727232`
- **Range:** `0` to `9223372036854775807`

---

aria_page_checksum

- **Description:** Determines whether index and data should use page checksums for extra safety. Can be overridden per table with PAGE_CHECKSUM clause in CREATE TABLE.
- **Commandline:** `--aria-page-checksum=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `ON`

---

aria_pagecache_age_threshold

- **Description:** This characterizes the number of hits a hot block has to be untouched until it is considered aged enough to be downgraded to a warm block. This specifies the percentage ratio of that number of hits to the total number of blocks in the page cache.
- **Commandline:** `--aria-pagecache-age-threshold=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `300`
- **Range:** `100` to `9999900`

---

aria_pagecache_buffer_size

- **Description:** The size of the buffer used for index and data blocks for Aria tables. This can include explicit Aria tables, system tables, and temporary tables. Increase this to get better handling and measure by looking at aria-status-variables/#aria_pagecache_reads (should be small) vs aria-status-variables/#aria_pagecache_read_requests.
- **Commandline:** `--aria-pagecache-buffer-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `134217720` (128MB)
- **Range:** `131072` (128KB) upwards

aria_pagecache_division_limit

- **Description:** The minimum percentage of warm blocks in the key cache.
- **Commandline:** `--aria-pagecache-division-limit=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `100`
- **Range:** `1` to `100`

aria_pagecache_file_hash_size

- **Description:** Number of hash buckets for open and changed files. If you have many Aria files open you should increase this for faster flushing of changes. A good value is probably 1/10th of the number of possible open Aria files.
- **Commandline:** `--aria-pagecache-file-hash-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `512`
- **Range:** `128` to `16384`

aria_recover

- **Description:** `aria_recover` has been renamed to `aria_recover_options` in MariaDB 10.2.0. See aria_recover_options for the description.

aria_recover_options

- **Description:** Specifies how corrupted tables should be automatically repaired. More than one option can be specified, for example `FORCE,BACKUP`.
  - `NORMAL`: Normal automatic repair, the default until MariaDB 10.2.3
  - `OFF`: Autorecovery is disabled, the equivalent of not using the option
  - `QUICK`: Does not check rows in the table if there are no delete blocks.
  - `FORCE`: Runs the recovery even if it determines that more than one row from the data file will be lost.
  - `BACKUP`: Keeps a backup of the data files.
- **Commandline:** `--aria-recover-options[=#]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `enumeration`
- **Default Value:**
  - `BACKUP,QUICK` (>= MariaDB 10.2.4)
  - `NORMAL` (<= MariaDB 10.2.3)
- **Valid Values:** `NORMAL`, `BACKUP`, `FORCE`, `QUICK`, `OFF`
- **Introduced:** MariaDB 10.2.0

aria_repair_threads

- **Description:** Number of threads to use when repairing Aria tables. The value of 1 disables parallel repair. Increasing from the default will usually result in faster repair, but will use more CPU and memory.
- **Commandline:** `--aria-repair-threads=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1`

aria_sort_buffer_size

- **Description:** The buffer that is allocated when sorting the index when doing a REPAIR or when creating indexes with CREATE INDEX or ALTER TABLE.

- **Commandline:** `--aria-sort-buffer-size=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `268434432`

---

aria_stats_method

- **Description:** Determines how NULLs are treated for Aria index statistics purposes. If set to `nulls_equal`, all NULL index values are treated as a single group. This is usually fine, but if you have large numbers of NULLs the average group size is slanted higher, and the optimizer may miss using the index for ref accesses when it would be useful. If set to `nulls_unequal`, the default, the opposite approach is taken, with each NULL forming its own group of one. Conversely, the average group size is slanted lower, and the optimizer may use the index for ref accesses when not suitable. Setting to `nulls_ignored` ignores NULLs altogether from index group calculations. Statistics need to be recalculated after this method is changed. See also Index Statistics, myisam_stats_method and innodb_stats_method.
- **Commandline:** `--aria-stats-method=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `nulls_unequal`
- **Valid Values:** `nulls_equal`, `nulls_unequal`, `nulls_ignored`

---

aria_sync_log_dir

- **Description:** Controls syncing directory after log file growth and new file creation.
- **Commandline:** `--aria-sync-log-dir=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `enumeration`
- **Default Value:** `NEWFILE`
- **Valid Values:** `NEWFILE`, `NEVER`, `ALWAYS`

---

aria_used_for_temp_tables

- **Description:** Readonly variable indicating whether the Aria storage engine is used for temporary tables. If set to `ON`, the default, the Aria storage engine is used. If set to `OFF`, MariaDB reverts to using MyISAM for on-disk temporary tables. The MEMORY storage engine is used for temporary tables regardless of this variable's setting where appropriate. The default can be changed by not using the `--with-aria-tmp-tables` option when building MariaDB.
- **Commandline:** No
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `ON`

---

deadlock_search_depth_long

- **Description:** Long search depth for the two-step deadlock detection. Only used by the Aria storage engine.
- **Commandline:** `--deadlock-search-depth-long=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `15`
- **Range:** `0` to `33`

---

deadlock_search_depth_short

- **Description:** Short search depth for the two-step deadlock detection. Only used by the Aria storage engine.
- **Commandline:** `--deadlock-search-depth-short=#`

- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `4`
- **Range:** `0` to `32`

`deadlock_timeout_long`

- **Description:** Long timeout in microseconds for the two-step deadlock detection. Only used by the Aria storage engine.
- **Commandline:** `--deadlock-timeout-long=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `50000000`
- **Range:** `0` to `4294967295`

`deadlock_timeout_short`

- **Description:** Short timeout in microseconds for the two-step deadlock detection. Only used by the Aria storage engine.
- **Commandline:** `--deadlock-timeout-short=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `10000`
- **Range:** `0` to `4294967295`

5.3.4.7 Aria Group Commit

Since MariaDB 5.2, the Aria storage engine has included a feature to group commits to speed up concurrent threads doing many inserts into the same or different Aria tables.

By default, group commit for Aria is turned off. It is controlled by the aria_group_commit and aria_group_commit_interval system variables.

Information on setting server variables can be found on the Server System Variables page.

Terminology

- A `commit` is `flush of logs` followed by a sync.
- `sent to disk` means written to disk but not sync()ed,
- `flushed` mean sent to disk and synced().
- `LSN` means log serial number. It's refers to the position in the transaction log.

Non Group commit logic (aria_group_commit="none")

The thread which first started the `commit` is performing the actual flush of logs. Other threads set the new goal (LSN) of the next pass (if it is maximum) and wait for the pass end or just wait for the pass end.

The effect of this is that a flush (write of logs + sync) will save all data for all threads/transactions that have been waiting since the last flush.

If hard group commit is enabled (aria_group_commit="hard")

If hard commit and aria_group_commit_interval=0

The first thread sends all changed buffers to disk. This is repeated as long as there are new LSNs added. The process can

not loop forever because we have a limited number of threads and they will wait for the data to be synced.

Pseudo code:

```
do
   send changed buffers to disk
 while new_goal
sync
```

If hard commit and aria_group_commit_interval > 0

If less than rate microseconds has passed since the last sync, then after buffers have been sent to disk, wait until rate microseconds has passed since last sync, do sync and return. This ensures that if we call sync infrequently we don't do any waits.

If soft group commit is enabled (aria_group_commit="soft")

Note that soft group commit should only be used if you can afford to lose a few rows if your machine shuts down hard (as in the case of a power failure).

Works like in `non group commit'` but the thread doesn't do any real sync(). If aria_group_commit_interval is not zero, the sync() will be performed by a service thread with the given rate when needed (new LSN appears). If aria_group_commit_interval is zero, there will be no sync() calls.

Code

The code for this can be found in storage/maria/ma_loghandler.c::translog_flush()

5.3.4.8 Benchmarking Aria

We have not yet had time to benchmark Aria properly. Here follows some things that have been discussed on the maria-discuss email list.

Aria used for internal temporary tables

By default Aria (instead of MyISAM) is used for the internal temporary tables when MEMORY tables overflows to disk or MEMORY tables can't be used (for example when you are using temporary results with BLOB's). In most cases Aria should give you better performance than using MyISAM, but this is not always the case.

```
CREATE TABLE `t1` (`id` int(11) DEFAULT NULL, `tea` text)
  ENGINE=MyISAM DEFAULT CHARSET=latin1;
insert t1 select rand()*2e8, repeat(rand(), rand()*64) from t1;
```

Repeat the last row until you get 2097152 rows.

The queries tested

```
Q1: SELECT id, tea from t1 group by left(id,1) order by null;
Q2: SELECT id, count(*), tea from t1 group by left(id,1) order by null;
Q3: SELECT id, tea from t1 group by left(id,2) order by null;
Q4: SELECT id, count(*), tea from t1 group by left(id,2) order by null;
Q5: SELECT id, tea from t1 group by id % 100 order by null;
Q6: SELECT id, count(*), tea from t1 group by id % 100 order by null;
```

Results (times in seconds, lower is better):

Test	Aria 8K page size	Aria 2K page size	MyISAM
Q1	3.08	2.41	2.17
Q2	6.24	5.21	12.89
Q3	4.87	4.05	4.04
Q4	8.20	7.04	15.14

Q5	7.10	6.37	6.28
Q6	10.38	9.09	17.00

The good news is that for common group by queries that is using summary functions there is a close to 50 % speedup of using Aria for internal temporary tables.

Note that queries Q1,Q3 and Q5 are not typical queries as there is no sum functions involved. In this case rows are just written to the tmp tables and there is no updates. As soon as there are summary functions and updates the new row format in Aria gives a close to 50 % speedup.

The above table also shows how the page size (determined by the aria_block_size system variable) affects the performance. The reason for the difference is that there is more data to move back/from the page cache for inserting of keys. (When reading data we are normally not copying pages). The bigger page size however allows longer keys and fewer index levels so for bigger data sets the different should be smaller. It's possible to in the future optimize Aria to not copy pages from the page cache also for index writes and then this difference should disappear.

The default page size for Aria is 8K.

If you want to run MariaDB with MyISAM for temporary tables, don't use the configure option '--with-aria-tmp-tables' when building MariaDB.

5.3.4.9 Aria Two-step Deadlock Detection

Description

The Aria storage engine can automatically detect and deal with deadlocks (see the Wikipedia deadlocks article).

This feature is controlled by four configuration variables, two that control the search depth and two that control the timeout.

- deadlock_search_depth_long
- deadlock_search_depth_short
- deadlock_timeout_long
- deadlock_timeout_short

How it Works

If Aria is ever unable to obtain a lock, we might have a deadlock. There are two primary ways for detecting if a deadlock has actually occurred. First is to search a wait-for graph (see the wait-for graph on Wikipedia) and the second is to just wait and let the deadlock exhibit itself. Aria Two-step Deadlock Detection does a combination of both.

First, if the lock request cannot be granted immediately, we do a short search of the wait-for graph with a small search depth as configured by the `deadlock_search_depth_short` variable. We have a depth limit because the graph can (theoretically) be arbitrarily big and we don't want to recursively search the graph arbitrarily deep. This initial, short search is very fast and most deadlocks will be detected right away. If no deadlock cycles are found with the short search the system waits for the amount of time configured in `deadlock_timeout_short` to see if the lock conflicts will be removed and the lock can be granted. Assuming this did not happen and the lock request still waits, the system then moves on to step two, which is a repeat of the process but this time searching deeper using the `deadlock_search_depth_long`. If no deadlock has been detected, it waits `deadlock_timeout_long` and times out.

When a deadlock is detected the system uses a weighting algorithm to determine which thread in the deadlock should be killed and then kills it.

5.3.4.10 Aria Encryption Overview

Contents

MariaDB can encrypt data in tables that use the Aria storage engine. This includes both user-created tables and internal on-disk temporary tables that use the Aria storage engine. This ensures that your Aria data is only accessible through MariaDB.

For encryption with the InnoDB and XtraDB storage engines, see Encrypting Data for InnoDB/XtraDB.

Basic Configuration

In order to enable encryption for tables using the Aria storage engine, there are a couple server system variables that you need to set and configure. Most users will want to set `aria_encrypt_tables` and `encrypt_tmp_disk_tables`.

Users of data-at-rest encryption will also need to have a key management and encryption plugin configured. Some examples are File Key Management Plugin and AWS Key Management Plugin.

```
[mariadb]
...

# File Key Management
plugin_load_add = file_key_management
file_key_management_filename = /etc/mysql/encryption/keyfile.enc
file_key_management_filekey = FILE:/etc/mysql/encryption/keyfile.key
file_key_management_encryption_algorithm = AES_CTR

# Aria Encryption
aria_encrypt_tables=ON
encrypt_tmp_disk_tables=ON
```

Determining Whether a Table is Encrypted

The InnoDB storage engine has the information_schema.INNODB_TABLESPACES_ENCRYPTION table that can be used to get information about which tables are encrypted. Aria does not currently have anything like that (see MDEV-17324 about that).

To determine whether an Aria table is encrypted, you currently have to search the data file for some plain text that you know is in the data.

For example, let's say that we have the following table:

```
SELECT * FROM db1.aria_tab LIMIT 1;
+----+------+
| id | str  |
+----+------+
|  1 | str1 |
+----+------+
1 row in set (0.00 sec
```

Then, we could search the data file that belongs to `db1.aria_tab` for `str1` using a command-line tool, such as strings:

```
$ sudo strings /var/lib/mysql/db1/aria_tab.MAD | grep "str1"
str1
```

If you can find the plain text of the string, then you know that the table is not encrypted.

Encryption and the Aria Log

Only Aria tables are currently encrypted. The Aria log is not yet encrypted. See MDEV-8587 about that.

5.3.4.11 The Aria Name

Contents

The Aria storage engine used to be called Maria. This page gives the history and background of how and why this name was changed to Aria.

Backstory

When starting what became the MariaDB project, Monty and the initial developers only planned to work on a next generation MyISAM storage engine replacement. This storage engine would be crash safe and eventually support transactions. Monty named the storage engine, and the project, after his daughter, Maria.

Work began in earnest on the Maria storage engine but the plans quickly expanded and morphed and soon the developers

were not just working on a storage engine, but on a complete branch of the MySQL database. Since the project was already called Maria, it made sense to call the whole database server MariaDB.

Renaming Maria (the engine)

So now there was the database, MariaDB, and the storage engine, Maria. To end the confusion this caused, the decision was made to rename the storage engine.

Monty's first suggestion was to name it *Lucy*, after his dog, but few who heard it liked that idea. So the decision was made that the next best thing was for the community to suggest and vote on names.

This was done by running a contest in 2009 through the end of May 2010. After that the best names were voted on by the community and Monty picked and announced the winner (Aria) at OSCon 2010 in Portland.

The winning entry was submitted by Chris Tooley. He received a Linux-powered System 76 Meerkat NetTop as his prize for suggesting the winning name from Monty Program.

5.3.5 Archive

The `ARCHIVE` storage engine is a storage engine that uses gzip to compress rows. It is mainly used for storing large amounts of data, without indexes, with only a very small footprint.

A table using the `ARCHIVE` storage engine is stored in two files on disk. There's a table definition file with an extension of .frm, and a data file with the extension .ARZ. At times during optimization, a .ARN file will appear.

New rows are inserted into a compression buffer and are flushed to disk when needed. SELECTs cause a flush. Sometimes, rows created by multi-row inserts are not visible until the statement is complete.

`ARCHIVE` allows a maximum of one key. The key must be on an `AUTO_INCREMENT` column, and can be a `PRIMARY KEY` or a non-unique key. However, it has a limitation: it is not possible to insert a value which is lower than the next `AUTO_INCREMENT` value.

Contents

Installing the Plugin

Although the plugin's shared library is distributed with MariaDB by default, the plugin is not actually installed by MariaDB by default. There are two methods that can be used to install the plugin with MariaDB.

The first method can be used to install the plugin without restarting the server. You can install the plugin dynamically by executing `INSTALL SONAME` or `INSTALL PLUGIN`. For example:

```
INSTALL SONAME 'ha_archive';
```

The second method can be used to tell the server to load the plugin when it starts up. The plugin can be installed this way by providing the `--plugin-load` or the `--plugin-load-add` options. This can be specified as a command-line argument to `mysqld` or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
plugin_load_add = ha_archive
```

Uninstalling the Plugin

You can uninstall the plugin dynamically by executing `UNINSTALL SONAME` or `UNINSTALL PLUGIN`. For example:

```
UNINSTALL SONAME 'ha_archive';
```

If you installed the plugin by providing the `--plugin-load` or the `--plugin-load-add` options in a relevant server option group in an option file, then those options should be removed to prevent the plugin from being loaded the next time the server is restarted.

Characteristics

- Supports INSERT and SELECT, but not DELETE, UPDATE or REPLACE.
- Data is compressed with zlib as it is inserted, making it very small.
- Data is slow the select, as it needs to be uncompressed, and, besides the query cache, there is no cache.
- Supports AUTO_INCREMENT (since MariaDB/MySQL 5.1.6), which can be a unique or a non-unique index.
- Since MariaDB/MySQL 5.1.6, selects scan past BLOB columns unless they are specifically requested, making these queries much more efficient.
- Does not support spatial data types.
- Does not support transactions.
- Does not support foreign keys.
- Does not support virtual columns.
- No storage limit.
- Supports row locking.
- Supports table discovery, and the server can access ARCHIVE tables even if the corresponding `.frm` file is missing.
- OPTIMIZE TABLE and REPAIR TABLE can be used to compress the table in its entirety, resulting in slightly better compression.
- With MariaDB, it is possible to upgrade from the MySQL 5.0 format without having to dump the tables.
- INSERT DELAYED is supported.
- Running many SELECTs during the insertions can deteriorate the compression, unless only multi-rows INSERTs and INSERT DELAYED are used.

No items found.

There are 3 related questions.

5.3.6 BLACKHOLE

The `BLACKHOLE` storage engine accepts data but does not store it and always returns an empty result.

A table using the `BLACKHOLE` storage engine consists of a single .frm table format file, but no associated data or index files.

This storage engine can be useful, for example, if you want to run complex filtering rules on a slave without incurring any overhead on a master. The master can run a `BLACKHOLE` storage engine, with the data replicated to the slave for processing.

Contents

Installing the Plugin

Although the plugin's shared library is distributed with MariaDB by default, the plugin is not actually installed by MariaDB by default. There are two methods that can be used to install the plugin with MariaDB.

The first method can be used to install the plugin without restarting the server. You can install the plugin dynamically by executing INSTALL SONAME or INSTALL PLUGIN. For example:

```
INSTALL SONAME 'ha_blackhole';
```

The second method can be used to tell the server to load the plugin when it starts up. The plugin can be installed this way by providing the --plugin-load or the --plugin-load-add options. This can be specified as a command-line argument to mysqld or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
plugin_load_add = ha_blackhole
```

Uninstalling the Plugin

You can uninstall the plugin dynamically by executing UNINSTALL SONAME or UNINSTALL PLUGIN. For example:

```
UNINSTALL SONAME 'ha_blackhole';
```

If you installed the plugin by providing the --plugin-load or the --plugin-load-add options in a relevant server option group in an option file, then those options should be removed to prevent the plugin from being loaded the next time the server is restarted.

Using the BLACKHOLE Storage Engine

Using with DML

INSERT, UPDATE, and DELETE statements all work with the `BLACKHOLE` storage engine. However, no data changes are actually applied.

Using with Replication

If the binary log is enabled, all SQL statements will be logged as usual, and replicated to any slave servers. However, since rows are not stored, it is important to use statement-based rather than the row or mixed format, as UPDATE and DELETE statements are neither logged nor replicated. See Binary Log Formats.

Using with Triggers

Some triggers work with the `BLACKHOLE` storage engine.

`BEFORE` triggers for INSERT statements are still activated.

Triggers for UPDATE and DELETE statements are **not** activated.

Triggers with the `FOR EACH ROW` clause do not apply, since the tables have no rows.

Using with Foreign Keys

Foreign keys are not supported. If you convert an InnoDB table to `BLACKHOLE`, then the foreign keys will disappear. If you convert the same table back to InnoDB, then you will have to recreate them.

Using with Virtual Columns

If you convert an InnoDB table which contains virtual columns to `BLACKHOLE`, then it produces an error.

Using with AUTO_INCREMENT

Because a BLACKHOLE table does not store data, it will not maintain the AUTO_INCREMENT value. If you are replicating to a table that can handle `AUTO_INCREMENT` columns, and are not explicitly setting the primary key auto-increment value in the INSERT query, or using the SET INSERT_ID statement, inserts will fail on the slave due to duplicate keys.

Limits

The maximum key size is:

- 3500 bytes (>= MariaDB 10.1.48, MariaDB 10.2.35, MariaDB 10.3.26, MariaDB 10.4.16 and MariaDB 10.5.7)
- 1000 bytes (<= MariaDB 10.1.47, MariaDB 10.2.34, MariaDB 10.3.25, MariaDB 10.4.15 and MariaDB 10.5.6).

Examples

```
CREATE TABLE table_name (
   id int unsigned primary key not null,
   v varchar(30)
) ENGINE=BLACKHOLE;

INSERT INTO table_name VALUES (1, 'bob'),(2, 'jane');

SELECT * FROM table_name;
Empty set (0.001 sec)
```

5.3.7 CONNECT

Note: You can download a PDF version of the CONNECT documentation (1.7.0003).

Connect Version	Introduced	Maturity
Connect 1.07.0002	MariaDB 10.5.9, MariaDB 10.4.18, MariaDB 10.3.28, MariaDB 10.2.36	Stable
Connect 1.07.0001	MariaDB 10.4.12, MariaDB 10.3.22, MariaDB 10.2.31, MariaDB 10.1.44	Stable
Connect 1.06.0010	MariaDB 10.4.8, MariaDB 10.3.18, MariaDB 10.2.27	Stable
Connect 1.06.0007	MariaDB 10.3.6, MariaDB 10.2.14, MariaDB 10.1.33	Stable
Connect 1.06.0005	MariaDB 10.3.3, MariaDB 10.2.10, MariaDB 10.1.29	Stable
Connect 1.06.0004	MariaDB 10.3.2, MariaDB 10.2.9, MariaDB 10.1.28	Stable
Connect 1.06.0001	MariaDB 10.3.1, MariaDB 10.2.8, MariaDB 10.1.24	Beta
Connect 1.05.0003	MariaDB 10.3.0, MariaDB 10.2.5, MariaDB 10.1.22	Stable
Connect 1.05.0001	MariaDB 10.2.4, MariaDB 10.1.21	Stable
Connect 1.04.0008	MariaDB 10.2.2, MariaDB 10.1.17	Stable
Connect 1.04.0006	MariaDB 10.2.0, MariaDB 10.1.13,	Stable
Connect 1.04.0005	MariaDB 10.1.10	Beta
Connect 1.04.0003	MariaDB 10.1.9	Beta

The CONNECT storage engine enables MariaDB to access external local or remote data (MED). This is done by defining tables based on different data types, in particular files in various formats, data extracted from other DBMS or products (such as Excel or MongoDB) via ODBC or JDBC, or data retrieved from the environment (for example DIR, WMI, and MAC tables)

This storage engine supports table partitioning, MariaDB virtual columns and permits defining *special* columns such as ROWID, FILEID, and SERVID.

No precise definition of maturity exists. Because CONNECT handles many table types, each type has a different maturity depending on whether it is old and well-tested, less well-tested or newly implemented. This will be indicated for all data types.

Introduction to the CONNECT Engine

Reasons behind the CONNECT storage engine.

Installing the CONNECT Storage Engine

Installing the CONNECT storage engine.

CONNECT Create Table Options

CREATE TABLE options for the CONNECT engine.

CONNECT Data Types

Data types supported by CONNECT.

Current Status of the CONNECT Handler

The current CONNECT handler is a stable release.

CONNECT Table Types

CONNECT Table Types Overview
CONNECT can handle many table formats.

Inward and Outward Tables
The two broad categories of CONNECT tables.

CONNECT Table Types - Data Files
CONNECT plain DOS or UNIX data files.

CONNECT Zipped File Tables
When the table file or files are compressed in one or several zip files.

CONNECT DOS and FIX Table Types
CONNECT tables based on text files

CONNECT DBF Table Type
CONNECT dBASE III or IV tables.

CONNECT BIN Table Type
CONNECT binary files in which each row is a logical record of fixed length

CONNECT VEC Table Type
CONNECT binary files organized in vectors

CONNECT CSV and FMT Table Types
Variable length CONNECT data files.

CONNECT - NoSQL Table Types
Based on files that do not match the relational format but often represent hierarchical data.

CONNECT - Files Retrieved Using Rest Queries
JSON, XML and CSV data files can be retrieved as results from REST queries.

CONNECT JSON Table Type
JSON (JavaScript Object Notation) is a widely-used lightweight data-interchange format.

CONNECT XML Table Type
CONNECT XML files

CONNECT INI Table Type
CONNECT INI Windows configuration or initialization files.

CONNECT - External Table Types
Access tables belonging to the current or another server.

CONNECT ODBC Table Type: Accessing Tables From Another DBMS
CONNECT Table Types - ODBC Table Type: Accessing Tables from other DBMS

CONNECT JDBC Table Type: Accessing Tables from Another DBMS
Using JDBC to access other tables.

CONNECT MONGO Table Type: Accessing Collections from MongoDB
Used to directly access MongoDB collections as tables.

CONNECT MYSQL Table Type: Accessing MySQL/MariaDB Tables
Accessing a MySQL or MariaDB table or view

CONNECT PROXY Table Type
Tables that access and read the data of another table or view

CONNECT XCOL Table Type
Based on another table/view, used when object tables have a column that contains a list of values

CONNECT OCCUR Table Type
Extension to the PROXY type when referring to a table/view having several c...

CONNECT PIVOT Table Type
Transform the result of another table into another table along “pivot” and "fact" columns.

CONNECT TBL Table Type: Table List
Define a table as a list of tables of any engine and type.

CONNECT - Using the TBL and MYSQL Table Types Together
Used together, the TBL and MYSQL types lift all the limitations of the FEDERATED and MERGE engines

CONNECT Table Types - Special "Virtual" Tables
VIR, WMI and MAC special table types

CONNECT Table Types - VIR
VIR virtual type for CONNECT

CONNECT Table Types - OEM: Implemented in an External LIB
CONNECT OEM table types are implemented in an external library.

CONNECT Table Types - Catalog Tables
Catalog tables return information about another table or data source

Adding DataFlex 3.1c .dat Files As An External Table Type With CONNECT
I'm using MariaDB's CONNECT engine to access / utilize a set of Visual FoxP...

CONNECT engine windows
with mariadb 10.07 installed on windows, how to setup engines, particulary ...

creating pivot table fails
I tried to create a pivot table based on an existing table "test1" and get ...

limit of number of columns
I have a table of 6MB with 50 values in the pivot-values leading to 50 colu...

Other CONNECT Articles

CONNECT - Security
CONNECT requires the FILE privilege for "outward" tables

CONNECT - OEM Table Example
Example showing how an OEM table can be implemented.

Using CONNECT

Using CONNECT - General Information
Using CONNECT - General Information.

Using CONNECT - Virtual and Special Columns
Virtual and special columns example usage

Using CONNECT - Importing File Data Into MariaDB Tables
Directly using external (file) data has many advantages

Using CONNECT - Exporting Data From MariaDB
Exporting data from MariaDB with CONNECT

Using CONNECT - Indexing
Indexing with the CONNECT handler

Using CONNECT - Condition Pushdown
Using CONNECT - Condition Pushdown.

USING CONNECT - Offline Documentation

CONNECT Plugin User Manual.

Using CONNECT - Partitioning and Sharding

Partitioning and Sharding with CONNECT

Other CONNECT Articles

CONNECT - Making the GetRest Library

Compiling the function calling the cpprestsdk package separately that will be loaded by CONNECT.

CONNECT - Adding the REST Feature as a Library Called by an OEM Table

How the REST feature can be added as a library called by an OEM table.

CONNECT - Compiling JSON UDFs in a Separate Library

There are situations when you may need to have JSON UDFs in a separate library.

CONNECT System Variables

System variables related to the CONNECT storage engine.

JSON Sample Files

expense.json sample file

There are 20 related questions.

5.3.7.1 Introduction to the CONNECT Engine

CONNECT is not just a new “YASE” (Yet another Storage Engine) that provides another way to store data with additional features. It brings a new dimension to MariaDB, already one of the best products to deal with traditional database transactional applications, further into the world of business intelligence and data analysis, including NoSQL facilities. Indeed, BI is the set of techniques and tools for the transformation of raw data into meaningful and useful information. And where is this data?

> "It's amazing in an age where relational databases reign supreme when it comes to managing data that so much information still exists outside RDBMS engines in the form of flat files and other such constructs. In most enterprises, data is passed back and forth between disparate systems in a fashion and speed that would rival the busiest expressways in the world, with much of this data existing in common, delimited files. Target systems intercept these source files and then typically proceed to load them via ETL (extract, transform, load) processes into databases that then utilize the information for business intelligence, transactional functions, or other standard operations. ETL tasks and data movement jobs can consume quite a bit of time and resources, especially if large volumes of data are present that require loading into a database. This being the case, many DBAs welcome alternative means of accessing and managing data that exists in file format."
>
> - Robin Schumacher[1]

What he describes is known as MED (Management of External Data) enabling the handling of data not stored in a DBMS database as if it were stored in tables. An ISO standard exists that describes one way to implement and use MED in SQL by defining foreign tables for which an external FDW (Foreign Data Wrapper) has been developed in C.

However, since this was written, a new source of data was developed as the “cloud”. Data are existing worldwide and, in particular, can be obtained in JSON or XML format in answer to REST queries. From Connect 1.06.0010, it is possible to create JSON, XML or CSV tables based on data retrieved from such REST queries.

MED as described above is a rather complex way to achieve this goal and MariaDB does not support the ISO SQL/MED standard. But, to cover the need, possibly in transactional but mostly in decision support applications, the CONNECT storage engine supports MED in a much simpler way.

The main features of CONNECT are:

1. No need for additional SQL language extensions.
2. Embedded wrappers for many external data types (files, data sources, virtual).
3. NoSQL query facilities for JSON, XML, HTML files and using JSON UDFs.
4. NoSQL data obtained from REST queries (requires cpprestsdk).
5. NoSQL new data type MONGO accessing MongoDB collections as relational tables.
6. Read/Write access to external files of most commonly used formats.
7. Direct access to most external data sources via ODBC, JDBC and MySQL or MongoDB API.
8. Only used columns are retrieved from external scan.

9. Push-down WHERE clauses when appropriate.
10. Support of special and virtual columns.
11. Parallel execution of multi-table tables (currently unavailable).
12. Supports partitioning by sub-files or by sub-tables (enabling table sharding).
13. Support of MRR for SELECT, UPDATE and DELETE.
14. Provides remote, block, dynamic and virtual indexing.
15. Can execute complex queries on remote servers.
16. Provides an API that allows writing additional FDW in C++.

With CONNECT, MariaDB has one of the most advanced implementations of MED and NoSQL, without the need for complex additions to the SQL syntax (foreign tables are "normal" tables using the CONNECT engine).

Giving MariaDB easy and natural access to external data enables the use of all of its powerful functions and SQL-handling abilities for developing business intelligence applications.

With version 1.07 of CONNECT, retrieving data from REST queries is available in all binary distributed version of MariaDB, and, from 1.07.002, CONNECT allows workspaces greater than 4GB.

---

1. ↑ Robin Schumacher is Vice President Products at DataStax and former Director of Product Management at MySQL. He has over 13 years of database experience in DB2, MySQL, Oracle, SQL Server and other database engines.

5.3.7.2 Installing the CONNECT Storage Engine

The `CONNECT` storage engine enables MariaDB to access external local or remote data (MED). This is done by defining tables based on different data types, in particular files in various formats, data extracted from other DBMS or products (such as Excel or MongoDB) via ODBC or JDBC, or data retrieved from the environment (for example DIR, WMI, and MAC tables)

This storage engine supports table partitioning, MariaDB virtual columns and permits defining special columns such as ROWID, FILEID, and SERVID.

The storage engine must be installed before it can be used.

Contents

Installing the Plugin's Package

The `CONNECT` storage engine's shared library is included in MariaDB packages as the `ha_connect.so` or `ha_connect.so` shared library on systems where it can be built.

Installing on Linux

The `CONNECT` storage engine is included in binary tarballs on Linux.

Installing with a Package Manager

The `CONNECT` storage engine can also be installed via a package manager on Linux. In order to do so, your system needs to be configured to install from one of the MariaDB repositories.

You can configure your package manager to install it from MariaDB Corporation's MariaDB Package Repository by using the MariaDB Package Repository setup script.

You can also configure your package manager to install it from MariaDB Foundation's MariaDB Repository by using the MariaDB Repository Configuration Tool.

Installing with yum/dnf

On RHEL, CentOS, Fedora, and other similar Linux distributions, it is highly recommended to install the relevant RPM

package from MariaDB's repository using `yum` or `dnf`. Starting with RHEL 8 and Fedora 22, `yum` has been replaced by `dnf`, which is the next major version of `yum`. However, `yum` commands still work on many systems that use `dnf`. For example:

```
sudo yum install MariaDB-connect-engine
```

Installing with apt-get

On Debian, Ubuntu, and other similar Linux distributions, it is highly recommended to install the relevant DEB package from MariaDB's repository using `apt-get`. For example:

```
sudo apt-get install mariadb-plugin-connect
```

Installing with zypper

On SLES, OpenSUSE, and other similar Linux distributions, it is highly recommended to install the relevant RPM package from MariaDB's repository using `zypper`. For example:

```
sudo zypper install MariaDB-connect-engine
```

Installing the Plugin

Once the shared library is in place, the plugin is not actually installed by MariaDB by default. There are two methods that can be used to install the plugin with MariaDB.

The first method can be used to install the plugin without restarting the server. You can install the plugin dynamically by executing `INSTALL SONAME` or `INSTALL PLUGIN`. For example:

```
INSTALL SONAME 'ha_connect';
```

The second method can be used to tell the server to load the plugin when it starts up. The plugin can be installed this way by providing the `--plugin-load` or the `--plugin-load-add` options. This can be specified as a command-line argument to `mysqld` or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
plugin_load_add = ha_connect
```

Uninstalling the Plugin

You can uninstall the plugin dynamically by executing `UNINSTALL SONAME` or `UNINSTALL PLUGIN`. For example:

```
UNINSTALL SONAME 'ha_connect';
```

If you installed the plugin by providing the `--plugin-load` or the `--plugin-load-add` options in a relevant server option group in an option file, then those options should be removed to prevent the plugin from being loaded the next time the server is restarted.

Installing Dependencies

The `CONNECT` storage engine has some external dependencies.

Installing unixODBC

The `CONNECT` storage engine requires an ODBC library. On Unix-like systems, that usually means installing unixODBC. On some systems, this is installed as the `unixODBC` package. For example:

```
sudo yum install unixODBC
```

On other systems, this is installed as the `libodbc1` package. For example:

```
sudo apt-get install libodbc1
```

If you do not have the ODBC library installed, then you may get an error about a missing library when you attempt to install the plugin. For example:

```
INSTALL SONAME 'ha_connect';
ERROR 1126 (HY000): Can't open shared library
'/home/ian/MariaDB_Downloads/10.1.17/lib/plugin/ha_connect.so'
  (errno: 2, libodbc.so.1: cannot open shared object file: No such file or directory)
```

5.3.7.3 CONNECT Create Table Options

Contents

Create Table statements for “CONNECT” tables are standard MariaDB create statements specifying `engine=CONNECT`. There are a few additional table and column options specific to CONNECT.

Table Options

Table Option	Type	Description
`AVG_ROW_LENGTH`	Integer	Can be specified to help CONNECT estimate the size of a variable record table length.
`BLOCK_SIZE`	Integer	The number of rows each block of a FIX, BIN, DBF, or VEC table contains. For an ODBC table this is the RowSet size option. For a JDBC table this is the fetch size.
`CATFUNC`	String	The catalog function used by a catalog table.
`COLIST`	String	The column list of OCCUR tables or $project of MONGO tables.
`COMPRESS`	Number	`1` or `2` if the data file is g-zip compressed. Defaults to 0. Before CONNECT 1.05.0001, this was boolean, and true if the data file is compressed.
`CONNECTION`	String	Specifies the connection of an ODBC, JDBC or MYSQL table.
`DATA_CHARSET`	String	The character set used in the external file or data source.
`DBNAME`	String	The target database for ODBC, JDBC, MYSQL, catalog, and PROXY based tables. The database concept is sometimes known as a *schema*.
`ENGINE`	String	Must be specfied as `CONNECT`.
`ENDING`	Integer	End of line length. Defaults to `1` for Unix/Linux and `2` for Windows.
`FILE_NAME`	String	The file (path) name for all table types based on files. Can be absolute or relative to the current data directory. If not specified, this is an Inward table and a default value is used.
`FILTER`	String	To filter an external table. Currently MONGO tables only.
`HEADER`	Integer	Applies to CSV, VEC, and HTML files. Its meaning depends on the table type.
`HTTP`	String	The HTTP of the client of REST queries. From Connect 1.06.0010.
`HUGE`	Boolean	To specify that a table file can be larger than 2GB. For a MYSQL table, prevents the result set from being stored in memory.
`LRECL`	Integer	The file record size (often calculated by default).
`MAPPED`	Boolean	Specifies whether *file mapping* is used to handle the table file.
`MODULE`	String	The (path) name of the DLL or shared lib implementing the access of a non-standard (OEM) table type.
`MULTIPLE`	Integer	Used to specify multiple file tables.
`OPTION_LIST`	String	Used to specify all other options not yet directly defined.
`QCHAR`	String	Specifies the character used for quoting some fields of a CSV table or the identifiers of an ODBC/JDBC tables.

QUOTED	Integer	The level of quoting used in CSV table files.
READONLY	Boolean	True if the data file must not be modified or erased.
SEP_CHAR	String	Specifies the field separator character of a CSV or XCOL table. Also, used to specify the Jpath separator for JSON tables.
SEPINDEX	Boolean	When true, indexes are saved in separate files.
SPLIT	Boolean	True for a VEC table when all columns are in separate files.
SRCDEF	String	The source definition of a table retrieved via ODBC, JDBC or the MySQL API or used by a PIVOT table.
SUBTYPE	String	The subtype of an OEM table type.
TABLE_LIST	String	The comma separated list of TBL table sub-tables.
TABLE_TYPE	String	The external table type: DOS, FIX, BIN, CSV, FMT, XML, JSON, INI, DBF, VEC, ODBC, JDBC, MYSQL, TBL, PROXY, XCOL, OCCUR, PIVOT, ZIP, VIR, DIR, WMI, MAC, and OEM. Defaults to DOS, MYSQL, or PROXY depending on what options are used.
TABNAME	String	The target table or node for ODBC, JDBC, MYSQL, PROXY, or catalog tables; or the top node name for XML tables.
URI	String	The URI of a REST request.. From Connect 1.06.0010.
XFILE_NAME	String	The file (path) base name for table index files. Can be absolute or relative to the data directory. Defaults to the file name.
ZIPPED	Boolean	True if the table file(s) is/are zipped in one or several zip files.

All integers in the above table are unsigned big integers.

Because CONNECT handles many table types; many table type specific options are not in the above list and must be entered using the `OPTION_LIST` option. The syntax to use is:

```
... option_list='opname1=opvalue1,opname2=opvalue2...'
```

Be aware that until Connect 1.5.5, no blanks should be inserted before or after the ' `=` ' and ' `,` ' characters. The option name is all that is between the start of the string or the last ' `,` ' character and the next ' `=` ' character, and the option value is all that is between this ' `=` ' character and the next ' `,` ' or end of string. For instance:

```
option_list='name=TABLE,coltype=HTML,attribute=border=1;cellpadding=5,headattr=bgcolor=yellow';
```

This defines four options, ' `name` ', ' `coltype` ', ' `attribute` ', and ' `headattr` '; with values ' `TABLE` ', ' `HTML` ', ' `border=1;cellpadding=5` ', and ' `bgcolor=yellow` ', respectively. The only restriction is that values cannot contain commas, but they can contain equal signs.

Column Options

Column Option	Type	Description
DATE_FORMAT	String	The format indicating how a date is stored in the file.
DISTRIB	Enum	“scattered”, “clustered”, “sorted” (ascending).
FIELD_FORMAT	String	The column format for some table types.
FIELD_LENGTH	Integer	Set the internal field length for DATE columns.
FLAG	Integer	An integer value whose meaning depends on the table type.
JPATH	String	The Json path of JSON table columns.
MAX_DIST	Integer	Maximum number of distinct values in this column.
SPECIAL	String	The name of the SPECIAL column that set this column value.
XPATH	String	The XML path of XML table columns.

- The `MAX_DIST` and `DISTRIB` column options are used for block indexing.
- All integers in the above table are unsigned big integers.
- JPATH and XPATH were added to make CREATE TABLE statements more readable, but they do the same thing as

FIELD_FORMAT and any of them can be used with the same result.

Index Options

Index Option	Type	Description
DYNAM	Boolean	Set the index as “dynamic”.
MAPPED	Boolean	Use index file mapping.

Note 1: Creating a CONNECT table based on file does not erase or create the file if the file name is specified in the CREATE TABLE statement (“outward” table). If the file does not exist, it will be populated by subsequent INSERT or LOAD commands or by the “AS select statement” of the CREATE TABLE command. Unlike the CSV engine, CONNECT easily permits the creation of tables based on already existing files, for instance files made by other applications. However, if the file name is not specified, a file with a name defaulting to `tablename.tabletype` will be created in the data directory (“inward” table).

Note 2: Dropping a CONNECT table is done with a standard DROP statement. For outward tables, this drops only the CONNECT table definition but does not erase the corresponding data file and index files. Use `DELETE` or `TRUNCATE` to do so. This is contrary to data and index files of inward tables are erased on DROP like for other MariaDB engines.

5.3.7.4 CONNECT Data Types

Contents

Many data types make no or little sense when applied to plain files. This why CONNECT supports only a restricted set of data types. However, ODBC, JDBC or MYSQL source tables may contain data types not supported by CONNECT. In this case, CONNECT makes an automatic conversion to a similar supported type when it is possible.

The data types currently supported by CONNECT are:

Type name	Description	Used for
`TYPE_STRING`	Zero ended string	char, varchar, text
`TYPE_INT`	4 bytes integer	int, mediumint, integer
`TYPE_SHORT`	2 bytes integer	smallint
`TYPE_TINY`	1 byte integer	tinyint
`TYPE_BIGINT`	8 bytes integer	bigint, longlong
`TYPE_DOUBLE`	8 bytes floating point	double, float, real
`TYPE_DECIM`	Numeric value	decimal, numeric, number
`TYPE_DATE`	4 bytes integer	date, datetime, time, timestamp, year

TYPE_STRING

This type corresponds to what is generally known as CHAR or VARCHAR by database users, or as strings by programmers. Columns containing characters have a maximum length but the character string is of fixed or variable length depending on the file format.

The DATA_CHARSET option must be used to specify the character set used in the data source or file. Note that, unlike

usually with MariaDB, when a multi-byte character set is used, the column size represents the number of bytes the column value can contain, not the number of characters.

TYPE_INT

The INTEGER type contains signed integer numeric 4-byte values (the *int/ of the C language) ranging from* `-2,147,483,648` *to* `2,147,483,647` *for signed type and* `0` *to* `4,294,967,295` *for unsigned type.*

TYPE_SHORT

The SHORT data type contains signed integer numeric 2-byte values (the *short integer* of the C language) ranging from `-32,768` to `32,767` for signed type and `0` to `65,535` for unsigned type.

TYPE_TINY

The TINY data type contains integer numeric 1-byte values (the *char* of the C language) ranging from `-128` to `127` for signed type and `0` to `255` for unsigned type. For some table types, TYPE_TINY is used to represent Boolean values (0 is false, anything else is true).

TYPE_BIGINT

The BIGINT data type contains signed integer 8-byte values (the *long long* of the C language) ranging from `-9,223,372,036,854,775,808` to `9,223,372,036,854,775,807` for signed type and from `0` to `18,446,744,073,709,551,615` for unsigned type.

Inside tables, the coding of all integer values depends on the table type. In tables represented by text files, the number is written in characters, while in tables represented by binary files (`BIN` or `VEC`) the number is directly stored in the binary representation corresponding to the platform.

The *length* (or *precision*) specification corresponds to the length of the table field in which the value is stored for text files only. It is used to set the output field length for all table types.

TYPE_DOUBLE

The DOUBLE data type corresponds to the C language double type, a floating-point double precision value coded with 8 bytes. Like for integers, the internal coding in tables depends on the table type, characters for text files, and platform binary representation for binary files.

The *length* specification corresponds to the length of the table field in which the value is stored for text files only. The *scale* (was *precision*) is the number of decimal digits written into text files. For binary table types (BIN and VEC) this does not apply. The *length* and *scale* specifications are used to set the output field length and number of decimals for all types of tables.

TYPE_DECIM

The DECIMAL data type corresponds to what MariaDB or ODBC data sources call NUMBER, NUMERIC, or DECIMAL: a numeric value with a maximum number of digits (the precision) some of them eventually being decimal digits (the scale). The internal coding in CONNECT is a character representation of the number. For instance:

```
colname decimal(14,6)
```

This defines a column *colname* as a number having a *precision* of 14 and a *scale* of 6. Supposing it is populated by:

```
insert into xxx values (-2658.74);
```

The internal representation of it will be the character string `-2658.740000` . The way it is stored in a file table depends on the table type. The *length* field specification corresponds to the length of the table field in which the value is stored and is calculated by CONNECT from the *precision* and the *scale* values. This length is *precision* plus 1 if *scale* is not 0 (for the decimal point) plus 1 if this column is not unsigned (for the eventual minus sign). In fix formatted tables the number is right justified in the field of width *length*, for variable formatted tables, such as CSV, the field is the representing character string.

Because this type is mainly used by CONNECT to handle numeric or decimal fields of ODBC, JDBC and MySQL table types, CONNECT does not provide decimal calculations or comparison by itself. This is why decimal columns of CONNECT

tables cannot be indexed.

DATE Data type

Internally, date/time values are stored by CONNECT as a signed 4-byte integer. The value 0 corresponds to 01 January 1970 12:00:00 am coordinated universal time (UTC). All other date/time values are represented by the number of seconds elapsed since or before midnight (00:00:00), 1 January 1970, to that date/time value. Date/time values before midnight 1 January 1970 are represented by a negative number of seconds.

CONNECT handles dates from **13 December 1901, 20:45:52** to **18 January 2038, 19:14:07**.

Although date and time information can be represented in both CHAR and INTEGER data types, the DATE data type has special associated properties. For each DATE value, CONNECT can store all or only some of the following information: century, year, month, day, hour, minute, and second.

Date Format in Text Tables

Internally, date/time values are handled as a signed 4-byte integer. But in text tables (type DOS, FIX, CSV, FMT, and DBF) dates are most of the time stored as a formatted character string (although they also can be stored as a numeric string representing their internal value). Because there are infinite ways to format a date, the format to use for decoding dates, as well as the field length in the file, must be associated to date columns (except when they are stored as the internal numeric value).

Note that this associated format is used only to describe the way the temporal value is stored internally. This format is used both for output to decode the date in a SELECT statement as well as for input to encode the date in INSERT or UPDATE statements. However, what is kept in this value depends on the data type used in the column definition (all the MariaDB temporal values can be specified). When creating a table, the format is associated to a date column using the DATE_FORMAT option in the column definition, for instance:

```
create table birthday (
  Name varchar(17),
  Bday date field_length=10 date_format='MM/DD/YYYY',
  Btime time field_length=8 date_format='hh:mm tt')
engine=CONNECT table_type=CSV;

insert into birthday values ('Charlie','2012-11-12','15:30:00');

select * from birthday;
```

The SELECT query returns:

Name	Bday	Btime
Charlie	2012-11-12	15:30:00

The values of the INSERT statement must be specified using the standard MariaDB syntax and these values are displayed as MariaDB temporal values. Sure enough, the column formats apply only to the way these values are represented inside the CSV files. Here, the inserted record will be:

```
Charlie,11/12/2012,03:30 PM
```

Note: The field_length option exists because the MariaDB syntax does not allow specifying the field length between parentheses for temporal column types. If not specified, the field length is calculated from the date format (sometimes as a max value) or made equal to the default length value if there is no date format. In the above example it could have been removed as the calculated values are the ones specified. However, if the table type would have been DOS or FIX, these values could be adjusted to fit the actual field length within the file.

A CONNECT format string consists of a series of elements that represent a particular piece of information and define its format. The elements will be recognized in the order they appear in the format string. Date and time format elements will be replaced by the actual date and time as they appear in the source string. They are defined by the following groups of characters:

Element	Description
YY	The last two digits of the year (that is, 1996 would be coded as "96").
YYYY	The full year (that is, 1996 could be entered as "96" but displayed as “1996”).
MM	The one or two-digit month number.

MMM	The three-character month abbreviation.
MMMM	The full month name.
DD	The one or two-digit month day.
DDD	The three-character weekday abbreviation.
DDDD	The full weekday name.
hh	The one or two-digit hour in 12-hour or 24-hour format.
mm	The one or two-digit minute.
ss	The one or two-digit second.
t	The one-letter AM/PM abbreviation (that is, AM is entered as "A").
tt	The two-letter AM/PM abbreviation (that is, AM is entered as "AM").

Usage Notes

- To match the source string, you can add body text to the format string, enclosing it in single quotes or double quotes if it would be ambiguous. Punctuation marks do not need to be quoted.
- The hour information is regarded as 12-hour format if a “t” or “tt” element follows the “hh” element in the format or as 24-hour format otherwise.
- The "MM", "DD", "hh", "mm", "ss" elements can be specified with one or two letters (e.g. "MM" or "M") making no difference on input, but placing a leading zero to one-digit values on output[1] for two-letter elements.
- If the format contains elements DDD or DDDD, the day of week name is skipped on input and ignored to calculate the internal date value. On output, the correct day of week name is generated and displayed.
- Temporal values are always stored as numeric in BIN and VEC tables.

Handling dates that are out of the range of supported CONNECT dates

If you want to make a table containing, for instance, historical dates not being convertible into CONNECT dates, make your column CHAR or VARCHAR and store the dates in the MariaDB format. All date functions applied to these strings will convert them to MariaDB dates and will work as if they were real dates. Of course they must be inserted and will be displayed using the MariaDB format.

NULL handling

CONNECT handles null values for data sources able to produce nulls. Currently this concerns mainly the ODBC, JDBC, MONGO, MYSQL, XML, JSON and INI table types. For INI, JSON, MONGO or XML types, null values are returned when the key is missing in the section (INI) or when the corresponding node does not exist in a row (XML, JSON, MONGO).

For other file tables, the issue is to define what a null value is. In a numeric column, 0 can sometimes be a valid value but, in some other cases, it can make no sense. The same for character columns; is a blank field a valid value or not?

A special case is DATE columns with a DATE_FORMAT specified. Any value not matching the format can be regarded as NULL.

CONNECT leaves the decision to you. When declaring a column in the CREATE TABLE statement, if it is declared NOT NULL, blank or zero values will be considered as valid values. Otherwise they will be considered as NULL values. In all cases, nulls are replaced on insert or update by pseudo null values, a zero-length character string for text types or a zero value for numeric types. Once converted to pseudo null values, they will be recognized as NULL only for columns declared as nullable.

For instance:

```
create table t1 (a int, b char(10)) engine=connect;
insert into t1 values (0,'zero'),(1,'one'),(2,'two'),(null,'???');
select * from t1 where a is null;
```

The select query replies:

a	b
NULL	zero
NULL	???

Sure enough, the value 0 entered on the first row is regarded as NULL for a nullable column. However, if we execute the

query:

```
select * from t1 where a = 0;
```

This will return no line because a NULL is not equal to 0 in an SQL where clause.

Now let us see what happens with not null columns:

```
create table t1 (a int not null, b char(10) not null) engine=connect;
insert into t1 values (0,'zero'),(1,'one'),(2,'two'),(null,'???');
```

The insert statement will produce a warning saying:

Level	Code	Message
Warning	1048	Column 'a' cannot be null

It is replaced by a pseudo null `0` on the fourth row. Let us see the result:

```
select * from t1 where a is null;
select * from t1 where a = 0;
```

The first query returns no rows, 0 are valid values and not NULL. The second query replies:

a	b
0	zero
0	???

It shows that the NULL inserted value was replaced by a valid 0 value.

Unsigned numeric types

They are supported by CONNECT since version 1.01.0010 for fixed numeric types (TINY, SHORT, INTEGER, and BITINT).

Data type conversion

CONNECT is able to convert data from one type to another in most cases. These conversions are done without warning even when this leads to truncation or loss of precision. This is true, in particular, for tables of type ODBC, JDBC, MYSQL and PROXY (via MySQL) because the source table may contain some data types not supported by CONNECT. They are converted when possible to CONNECT types.

When converted, MariaDB types are converted as:

MariaDB Types	CONNECT Type	Remark
integer, medium integer	TYPE_INT	4 byte integer
small integer	TYPE_SHORT	2 byte integer
tiny integer	TYPE_TINY	1 byte integer
char, varchar	TYPE_STRING	Same length
double, float, real	TYPE_DOUBLE	8 byte floating point
decimal, numeric	TYPE_DECIM	Length depends on precision and scale
all date related types	TYPE_DATE	Date format can be set accordingly
bigint, longlong	TYPE_BIGINT	8 byte integer
enum, set	TYPE_STRING	Numeric value not accessible
All text types	TYPE_STRING TYPE_ERROR	Depending on the value of the connect_type_conv system variable value.
Other types	TYPE_ERROR	Not supported, no conversion provided.

For ENUM, the length of the column is the length of the longest value of the enumeration. For SET the length is enough to contain all the set values concatenated with comma separator.

In the case of TEXT columns, the handling depends on the values given to the connect_type_conv and connect_conv_size system variables.

Note: BLOB is currently not converted by default until a TYPE_BIN type is added to CONNECT. However, the FORCE option (from Connect 1.06.006) can be specified for blob columns containing text and the SKIP option also applies to ODBC BLOB columns.

ODBC SQL types are converted as:

SQL Types	Connect Type	Remark
SQL_CHAR, SQL_VARCHAR	TYPE_STRING	
SQL_LONGVARCHAR	TYPE_STRING	`len = min(abs(len), connect_conv_size)` If the column is generated by discovery (columns not specified) its length is connect_conv_size.
SQL_NUMERIC, SQL_DECIMAL	TYPE_DECIM	
SQL_INTEGER	TYPE_INT	
SQL_SMALLINT	TYPE_SHORT	
SQL_TINYINT, SQL_BIT	TYPE_TINY	
SQL_FLOAT, SQL_REAL, SQL_DOUBLE	TYPE_DOUBLE	
SQL_DATETIME	TYPE_DATE	`len = 10`
SQL_INTERVAL	TYPE_STRING	`len = 8 + ((scale) ? (scale+1) : 0)`
SQL_TIMESTAMP	TYPE_DATE	`len = 19 + ((scale) ? (scale +1) : 0)`
SQL_BIGINT	TYPE_BIGINT	
SQL_GUID	TYPE_STRING	I `len=36`
SQL_BINARY, SQL_VARBINARY, SQL_LONG-VARBINARY	TYPE_STRING	`len = min(abs(len), connect_conv_size`) Only if the value of connect_type_conv is `force` . The column should use the binary charset.
Other types	TYPE_ERROR	*Not supported.*

JDBC SQL types are converted as:

JDBC Types	Connect Type	Remark
(N)CHAR, (N)VARCHAR	TYPE_STRING	
LONG(N)VARCHAR	TYPE_STRING	`len = min(abs(len), connect_conv_size)` If the column is generated by discovery (columns not specified), its length is connect_conv_size
NUMERIC, DECIMAL, VARBINARY	TYPE_DECIM	
INTEGER	TYPE_INT	
SMALLINT	TYPE_SHORT	
TINYINT, BIT	TYPE_TINY	
FLOAT, REAL, DOUBLE	TYPE_DOUBLE	
DATE	TYPE_DATE	`len = 10`
TIME	TYPE_DATE	`len = 8 + ((scale) ? (scale+1) : 0)`
TIMESTAMP	TYPE_DATE	`len = 19 + ((scale) ? (scale +1) : 0)`
BIGINT	TYPE_BIGINT	
UUID (specific to PostgreSQL)	TYPE_STRING TYPE_ERROR	`len=36` If connect_type_conv=NO
Other types	TYPE_ERROR	Not supported.

Note: The connect_type_conv SKIP option also applies to ODBC and JDBC tables.

1. ↑ Here input and output are used to specify respectively decoding the date to get its numeric value from the data file and encoding a date to write it in the table file. Input is performed within SELECT queries; output is performed in UPDATE or INSERT queries.

5.3.7.5 Current Status of the CONNECT Handler

The current CONNECT handler is a GA (stable) release. It was written starting both from an aborted project written for MySQL in 2004 and from the “DBCONNECT” program. It was tested on all the examples described in this document, and is distributed with a set of 53 test cases. Here is a not limited list of future developments:

1. Adding more table types.
2. Make more tests files (53 are already made)
3. Adding more data types, in particular unsigned ones (done for unsigned).
4. Supporting indexing on nullable and decimal columns.
5. Adding more optimize tools (block indexing, dynamic indexing, etc.) (done)
6. Supporting MRR (done)
7. Supporting partitioning (done)
8. Getting NOSQL data from the Net as answers from REST queries (done)

No programs are bug free, especially new ones. Please report all bugs or documentation errors using the means provided by MariaDB.

5.3.7.6 CONNECT Table Types

The main feature of CONNECT is to give MariaDB the ability to handle tables from many sources, native files, other DBMS’s tables, or special “virtual” tables. Moreover, for all tables physically represented by data files, CONNECT recognizes many different file formats, described below but not limited in the future to this list, because more can be easily added to it on demand (OEM tables).

Note: You can download a PDF version of the CONNECT documentation (1.7.0003).

CONNECT Table Types Overview

CONNECT can handle many table formats.

Inward and Outward Tables

The two broad categories of CONNECT tables.

CONNECT Table Types - Data Files

CONNECT plain DOS or UNIX data files.

CONNECT Zipped File Tables

When the table file or files are compressed in one or several zip files.

CONNECT DOS and FIX Table Types

CONNECT tables based on text files

CONNECT DBF Table Type

CONNECT dBASE III or IV tables.

CONNECT BIN Table Type

CONNECT binary files in which each row is a logical record of fixed length

CONNECT VEC Table Type

CONNECT binary files organized in vectors

CONNECT CSV and FMT Table Types

Variable length CONNECT data files.

CONNECT - NoSQL Table Types
Based on files that do not match the relational format but often represent hierarchical data.

CONNECT - Files Retrieved Using Rest Queries
JSON, XML and CSV data files can be retrieved as results from REST queries.

CONNECT JSON Table Type
JSON (JavaScript Object Notation) is a widely-used lightweight data-interchange format.

CONNECT XML Table Type
CONNECT XML files

CONNECT INI Table Type
CONNECT INI Windows configuration or initialization files.

CONNECT - External Table Types
Access tables belonging to the current or another server.

CONNECT ODBC Table Type: Accessing Tables From Another DBMS
CONNECT Table Types - ODBC Table Type: Accessing Tables from other DBMS

CONNECT JDBC Table Type: Accessing Tables from Another DBMS
Using JDBC to access other tables.

CONNECT MONGO Table Type: Accessing Collections from MongoDB
Used to directly access MongoDB collections as tables.

CONNECT MYSQL Table Type: Accessing MySQL/MariaDB Tables
Accessing a MySQL or MariaDB table or view

CONNECT PROXY Table Type
Tables that access and read the data of another table or view

CONNECT XCOL Table Type
Based on another table/view, used when object tables have a column that contains a list of values

CONNECT OCCUR Table Type
Extension to the PROXY type when referring to a table/view having several c...

CONNECT PIVOT Table Type
Transform the result of another table into another table along “pivot” and "fact" columns.

CONNECT TBL Table Type: Table List
Define a table as a list of tables of any engine and type.

CONNECT - Using the TBL and MYSQL Table Types Together
Used together, the TBL and MYSQL types lift all the limitations of the FEDERATED and MERGE engines

CONNECT Table Types - Special "Virtual" Tables
VIR, WMI and MAC special table types

CONNECT Table Types - VIR
VIR virtual type for CONNECT

CONNECT Table Types - OEM: Implemented in an External LIB
CONNECT OEM table types are implemented in an external library.

CONNECT Table Types - Catalog Tables
Catalog tables return information about another table or data source

There are 4 related questions.

5.3.7.6.1 CONNECT Table Types Overview

CONNECT can handle very many table formats; it is indeed one of its main features. The Type option specifies the type and format of the table. The Type options available values and their descriptions are listed in the following table:

Type	Description
BIN	Binary file with numeric values in platform representation, also with columns at fixed offset within records and fixed record length.
BSON	(Temporary) JSON table handled by the new JSON handling.
CSV*$	"Comma Separated Values" file in which each variable length record contains column values separated by a specific character (defaulting to the comma)
DBF*	File having the dBASE format.
DOS	The table is contained in one or several files. The file format can be refined by some other options of the command or more often using a specific type as many of those described below. Otherwise, it is a flat text file where columns are placed at a fixed offset within each record, the last column being of variable length.
DIR	Virtual table that returns a file list like the Unix `ls` or DOS `dir` command.
FIX	Text file arranged like DOS but with fixed length records.
FMT	File in which each record contains the column values in a non-standard format (the same for each record) This format is specified in the column definition.
INI	File having the format of the initialization or configuration files used by many applications.
JDBC*	Table accessed via a JDBC driver.
JSON*$	File having the JSON format.
MAC	Virtual table returning information about the machine and network cards (Windows only).
MONGO*	Table accessed via the MongoDB C Driver API.
MYSQL*	Table accessed using the MySQL API like the FEDERATED engine.
OCCUR*	A table based on another table existing on the current server, several columns of the object table containing values that can be grouped in only one column.
ODBC*	Table extracted from an application accessible via ODBC or unixODBC. For example from another DBMS or from an Excel spreadsheet.
OEM*	Table of any other formats not directly handled by CONNECT but whose access is implemented by an external FDW (foreign data wrapper) written in C++ (as a DLL or Shared Library).
PIVOT*	Used to "pivot" the display of an existing table or view.
PROXY*	A table based on another table existing on the current server.
TBL*	Accessing a collection of tables as one table (like the MERGE engine does for MyIsam tables)
VEC	Binary file organized in vectors, in which column values are grouped consecutively, either split in separate files or in a unique file.
VIR*	Virtual table containing only special and virtual columns.
WMI*	Windows Management Instrumentation table displaying information coming from a WMI provider. This type enables to get in tabular format all sorts of information about the machine hardware and operating system (Windows only).
XCOL*	A table based on another table existing on the current server with one of its columns containing comma separated values.
XML*$	File having the XML or HTML format.
ZIP	Table giving information about the contents of a zip file.

Catalog Tables

For all table types marked with a '*' in the table above, CONNECT is able to analyze the data source to retrieve the column definition. This can be used to define a “catalog” table that display the column description of the source, or to create a table without specifying the column definition that will be automatically constructed by CONNECT when creating the table.

When marked with a '$' the file can be the result returned by a REST query.

5.3.7.6.2 Inward and Outward Tables

Contents

There are two broad categories of file-based CONNECT tables. Inward and Outward. They are described below.

Outward Tables

Tables are "outward" when their file name is specified in the CREATE TABLE statement using the *file_name* option.

Firstly, remember that CONNECT implements MED (Management of External Data). This means that the "true" CONNECT tables – "outward tables" – are based on data that belongs to files that can be produced by other applications or data imported from another DBMS.

Therefore, their data is "precious" and should not be modified except by specific commands such as INSERT, UPDATE, or DELETE. For other commands such as CREATE, DROP, or ALTER their data is never modified or erased.

Outward tables can be created on existing files or external tables. When they are dropped, only the local description is dropped, the file or external table is not dropped or erased. Also, DROP TABLE does not erase the indexes.

ALTER TABLE produces the following warning, as a reminder:

```
Warning (Code 1105): This is an outward table, table data were not modified.
```

If the specified file does not exist, it is created when data is inserted into the table. If a SELECT is issued before the file is created, the following error is produced:

```
Warning (Code 1105): Open(rb) error 2 on <file_path>: No such file or directory
```

Altering Outward Tables

When an ALTER TABLE is issued, it just modifies the table definition accordingly without changing the data. ALTER can be used safely to, for instance, modify options such as MAPPED, HUGE or READONLY but with extreme care when modifying column definitions or order options because some column options such as FLAG should also be modified or may become wrong.

Changing the table type with ALTER often makes no sense. But many suspicious alterations can be acceptable if they are just meant to correct an existing wrong definition.

Translating a CONNECT table to another engine is fine but the opposite is forbidden when the target CONNECT table is not table based or when its data file exists (because when the target table data cannot be changed and if the source table is dropped, the table data would be lost). However, it can be done to create a new file-based tables when its file does not exist or is void.

Creating or dropping indexes is accepted because it does not modify the table data. However, it is often unsafe to do it with an ALTER TABLE statement that does other modifications.

Of course, all changes are acceptable for empty tables.

Note: Using outward tables requires the FILE privilege.

Inward Tables

A special type of file-based CONNECT tables are "inward" tables. They are file-based tables whose file name is not specified in the CREATE TABLE statement (no *file_name* option).

Their file will be located in the current database directory and their name will default to tablename.type where tablename is the table name and type is the table type folded to lower case. When they are created without using a `CREATE TABLE ... SELECT ...` statement, an empty file is made at create time and they can be populated by further inserts.

They behave like tables of other storage engines and, unlike outward CONNECT tables, they are erased when the table is dropped. Of course they should not be read-only to be usable. Even though their utility is limited, they can be used for

testing purposes or when the user does not have the FILE privilege.

Altering Inward Tables

One thing to know, because CONNECT builds indexes in a specific way, is that all index modifications are done using an "in-place" algorithm – meaning not using a temporary table. This is why, when indexing is specified in an ALTER TABLE statement containing other changes that cannot be done "in-place", the statement cannot be executed and raises an error.

Converting an inward table to an outward table, using an ALTER TABLE statement specifying a new file name and/or a new table type, is restricted the same way it is when converting a table from another engine to an outward table. However there are no restrictions to convert another engine table to a CONNECT inward table.

5.3.7.6.3 CONNECT Table Types - Data Files

Contents

Most of the tables processed by CONNECT are just plain DOS or UNIX data files, logically regarded as tables thanks to the description given when creating the table. This description comes from the `CREATE TABLE` statement. Depending on the application, these tables can already exist as data files, used as is by CONNECT, or can have been physically made by CONNECT as the result of a `CREATE TABLE ... SELECT ...` and/or INSERT statement(s).

The file *path/name* is given by the `FILE_NAME` option. If it is a relative path/name, it will be relative to the database directory, the one containing the table `.FRM` file.

Unless specified, the maturity of file table types is stable.

Multiple File Tables

A **multiple** file table is one that is physically contained in several files of the same type instead of just one. These files are processed sequentially during the process of a query and the result is the same as if all the table files were merged into one. This is great to process files coming from different sources (such as cash register log files) or made at different time periods (such as bank monthly reports) regarded as one table. Note that the operations on such files are restricted to sequential Select and Update; and that VEC multiple tables are not supported by CONNECT. The file list depends on the setting of the **multiple** option of the `CREATE TABLE` statement for that table.

Multiple tables are specified by the option MULTIPLE=*n*, which can take four values:

0	Not a multiple table (the default). This can be used in an ALTER TABLE statement.
1	The table is made from files located in the same directory. The FILE_NAME option is a pattern such as `'cash*.log'` that all the table file path/names verify.
2	The FILE_NAME gives the name of a file that contains the path/names of all the table files. This file can be made using a DIR table.
3	Like multiple=1 but also including eligible files from the directory sub-folders.

The `FILEID` special column, described here, allows query pruning by filtering the file list or doing some grouping on the files that make a multiple table.

Note: Multiple was not initially implemented for XML tables. This restriction was removed in version 1.02.

Record Format

This characteristic applies to table files handled by the operating system input/output functions. It is **fixed** for table types FIX, BIN, DBF and VEC, and it is variable for DOS, VCT, FMT and some JSON tables.

For fixed tables, most I/O operations are done by block of BLOCK_SIZE rows. This diminishes the number of I/O's and enables block indexing.

Starting with CONNECT version 1.6.6, the BLOCK_SIZE option can also be specified for variable tables. Then, a file similar

to the block indexing file is created by CONNECT that gives the size in bytes of each block of BLOCK_SIZE rows. This enables the use of block I/Os and block indexing to variable tables. It also enables CONNECT to return the exact row number for info commands

File Mapping

For file-based tables of reasonable size, processing time can be greatly enhanced under Windows(TM) and some flavors of UNIX or Linux by using the technique of “file mapping”, in which a file is processed as if it were entirely in memory. Mapping is specified when creating the table by the use of the `MAPPED=YES` option. This does not apply to tables not handled by system I/O functions (`XML` and `INI`).

Big File Tables

Because all files are handled by the standard input/output functions of the operating system, their size is limited to 2GB, the maximum size handled by standard functions. For some table types, CONNECT can deal with files that are larger than 2GB, or prone to become larger than this limit. These are the FIX, BIN and VEC types. To tell connect to use input/output functions dealing with big files, specify the option `huge=1` or `huge=YES` for that table. Note however that CONNECT cannot randomly access tables having more than 2G records.

Compressed File Tables

CONNECT can make and process some tables whose data file is compressed. The only supported compression format is the gzlib format. Zip and zlib formats are supported differently. The table types that can be compressed are DOS, FIX, BIN, CSV and FMT. This can save some disk space at the cost of a somewhat longer processing time.

Some restrictions apply to compressed tables:

- Compressed tables are not indexable.
- Update and partial delete are not supported.

Use the numeric compress option to specify a compressed table:

1. Not compressed
2. Compressed in gzlib format.
3. Made of compressed blocks of block_size records (enabling block indexing)

Relational Formatted Tables

These are based on files whose records represent one table row. Only the column representation within each record can differ. The following relational formatted tables are supported:

- DOS and FIX Table Types
- DBF Table Type
- BIN Table Type
- VEC Table Type
- CSV and FMT Table Types

NoSQL Table Types

These are based on files that do not match the relational format but often represent hierarchical data. CONNECT can handle JSON, INI-CFG, XML and some HTML files..

The way it is done is different from what PostgreSQL does. In addition to including in a table some column values of a specific data format (JSON, XML) to be handled by specific functions, CONNECT can directly use JSON, XML or INI files that can be produced by other applications and this is the table definition that describes where and how the contained information must be retrieved.

This is also different from what MariaDB does with dynamic columns, which is close to what MySQL and PostgreSQL do with the JSON column type.

The following NoSQL types are supported:

- XML Table Type
- JSON Table Type
- INI Table Type

5.3.7.6.4 CONNECT Zipped File Tables

MariaDB starting with 10.2.4
Connect can work on table files that are compressed in one or several zip files.

The specific options used when creating tables based on zip files are:

Table Option	Type	Description
ZIPPED	Boolean	Required to be set as true.
ENTRY*	String	The optional name or pattern of the zip entry or entries to be used with the table. If not specified, all entries or only the first one will be used depending on the *mulentries* option setting.
MULENTRIES*	Boolean	True if several entries are part of the table. If not specified, it defaults to false if the *entry* option is not specified. If the entry option is specified, it defaults to true if the entry name contains wildcard characters or false if it does not.
APPEND*	Boolean	Used when creating new zipped tables (see below)
LOAD*	String	Used when creating new zipped tables (see below)

Options marked with a '*' must be specified in the option list.

Examples of use:

Example 1: Single CSV File Included in a Single ZIP File

Let's suppose you have a CSV file from which you would create a table by:

```
create table emp
... optional column definition
engine=connect table_type=CSV file_name='E:/Data/employee.csv'
sep_char=';' header=1;
```

If the CSV file is included in a ZIP file, the CREATE TABLE becomes:

```
create table empzip
... optional column definition
engine=connect table_type=CSV file_name='E:/Data/employee.zip'
sep_char=';' header=1 zipped=1 option_list='Entry=emp.csv';
```

The *file_name* option is the name of the zip file. The *entry* option is the name of the entry inside the zip file. If there is only one entry file inside the zip file, this option can be omitted.

Example 2: Several CSV Files Included in a Single ZIP File

If the table is made from several files such as emp01.csv, emp02.csv, etc., the standard create table would be:

```
create table empmul (
... required column definition
) engine=connect table_type=CSV file_name='E:/Data/emp*.csv'
sep_char=';' header=1 multiple=1;
```

But if these files are all zipped inside a unique zip file, it becomes:

```
create table empzmul
... required column definition
engine=connect table_type=CSV file_name='E:/Data/emp.zip'
sep_char=';' header=1 zipped=1 option_list='Entry=emp*.csv';
```

Here the *entry* option is the pattern that the files inside the zip file must match. If all entry files are ok, the *entry* option can be omitted but the Boolean option *mulentries* must be specified as true.

Example 3: Single CSV File included in Multiple ZIP Files (Without considering subfolders)

If the table is created on several zip files, it is specified as for all other multiple tables:

```
create table zempmul (
... required column definition
) engine=connect table_type=CSV file_name='E:/Data/emp*.zip'
sep_char=';' header=1 multiple=1 zipped=yes
option_list='Entry=employee.csv';
```

Here again the *entry* option is used to restrict the entry file(s) to be used inside the zip files and can be omitted if all are ok.

The column descriptions can be retrieved by the discovery process for table types allowing it. It cannot be done for multiple tables or multiple entries.

A catalog table can be created by adding *catfunc=columns*. This can be used to show the column definitions of multiple tables. *Multiple* must be set to false and the column definitions will be the ones of the first table or entry.

This first implementation has some restrictions:

1. Zipped tables are read-only. UPDATE and DELETE are not supported. However, INSERT is supported in a specific way when making tables.
2. The inside files are decompressed into memory. Memory problems may arise with huge files.
3. Only file types that can be handled from memory are eligible for this. This includes DOS, FIX, BIN, CSV, FMT, DBF, JSON, and XML table types, as well as types based on these such as XCOL, OCCUR and PIVOT.

Optimization by indexing or block indexing is possible for table types supporting it. However, it applies to the uncompressed table. This means that the whole table is always uncompressed.

Partitioning is also supported. See how to do it in the section about partitioning.

Creating New Zipped Tables

Tables can be created to access already existing zip files. However, is it also possible to make the zip file from an existing file or table. Two ways are available to make the zip file:

Insert Method

insert can be used to make the table file for table types based on records (this excludes DBF, XML and JSON when pretty is not 0). However, the current implementation of the used package (minizip) does not support adding to an already existing zip entry. This means that when executing an insert statement the inserted records are not added but replace the existing ones. CONNECT protects existing data by not allowing such inserts, Therefore, only three ways are available to do so:

1. Using only one insert statement to make the whole table. This is possible only for small tables and is principally useful when making tests.
2. Making the table from the data of another table. This can be done by executing an “insert into table select * from another_table” or by specifying “as select * from another_table” in the create table statement.
3. Making the table from a file whose format enables to use the “load data infile” statement.

To add a new entry in an existing zip file, specify “append=YES” in the option list. When inserting several entries, use ALTER to specify the required options, for instance:

```
create table znumul (
Chiffre int(3) not null,
Lettre char(16) not null)
engine=CONNECT table_type=CSV
file_name='C:/Data/FMT/mnum.zip' header=1 lrecl=20 zipped=1
option_list='Entry=Num1';
insert into znumul select * from num1;
alter table znumul option_list='Entry=Num2,Append=YES';
insert into znumul select * from num2;
alter table znumul option_list='Entry=Num3,Append=YES';
insert into znumul select * from num3;
alter table znumul option_list='Entry=Num*,Append=YES';
select * from znumul;
```

The last ALTER is needed to display all the entries.

File Zipping Method

This method enables to make the zip file from another file when creating the table. It applies to all table types including DBF, XML and JSON. It is specified in the create table statement with the load option. For example:

```
create table XSERVZIP (
NUMERO varchar(4) not null,
LIEU varchar(15) not null,
CHEF varchar(5) not null,
FONCTION varchar(12) not null,
NOM varchar(21) not null)
engine=CONNECT table_type=XML file_name='E:/Xml/perso.zip' zipped=1
option_list='entry=services,load=E:/Xml/serv2.xml';
```

When executing this statement, the *serv2.xml* file will be zipped as /perso.zip*. The entry name can be specified or defaults to the source file name.*

If the column descriptions are specified, the table can be used later to read from the zipped table, but they are not used when creating the zip file. Thus, a fake column (there must be one) can be specified and another table created to read the zip file. This one can take advantage of the discovery process to avoid providing the columns description for table types allowing it. For instance:

```
create table mkzq (whatever int)
engine=connect table_type=DBF zipped=1
file_name='C:/Data/EAUX/dbf/CQUART.ZIP'
option_list='Load=C:/Data/EAUX/dbf/CQUART.DBF';

create table zquart
engine=connect table_type=DBF zipped=1
file_name='C:/Data/EAUX/dbf/CQUART.ZIP';
```

It is also possible to create a multi-entries table from several files:

```
CREATE TABLE znewcities (
  _id char(5) NOT NULL,
  city char(16) NOT NULL,
  lat double(18,6) NOT NULL `FIELD_FORMAT`='loc:[0]',
  lng double(18,6) NOT NULL `FIELD_FORMAT`='loc:[1]',
  pop int(6) NOT NULL,
  state char(2) NOT NULL
) ENGINE=CONNECT TABLE_TYPE=JSON FILE_NAME='E:/Json/newcities.zip' ZIPPED=1 LRECL=1000
OPTION_LIST='Load=E:/Json/city_*.json,mulentries=YES,pretty=0';
```

Here the files to load are specified with wildcard characters and the *mulentries* options must be specified. However, the *entry* option must not be specified, entry names will be made from the file names. Provide a fake column description if the files have different column layout, but specific tables will have to be created to read each of them.

ZIP Table Type

A ZIP table type is also available. It is not meant to read the inside files but to display information about the zip file contents. For instance:

```
create table xzipinfo2 (
entry varchar(256)not null,
cmpsize bigint not null flag=1,
uncsize bigint not null flag=2,
method int not null flag=3,
date datetime not null flag=4)
engine=connect table_type=ZIP file_name='E:/Data/Json/cities.zip';
```

This will display the name, compressed size, uncompressed size, and compress method of all entries inside the zip file. Column names are irrelevant; these are flag values that mean what information to retrieve.

It is possible to retrieve this information from several zip files by specifying the multiple option:

```
create table TestZip1 (
entry varchar(260)not null,
cmpsize bigint not null flag=1,
uncsize bigint not null flag=2,
method int not null flag=3,
date datetime not null flag=4,
zipname varchar(256) special='FILEID')
engine=connect table_type=ZIP multiple=1
file_name='C:/Data/Ziptest/CCAM06300_DBF_PART*.zip';
```

Here we added the special column zipname to get the name of the zip file for each entry.

5.3.7.6.5 CONNECT DOS and FIX Table Types

Contents

Overview

Tables of type DOS and FIX are based on text files (see CONNECT Table Types - Data Files). Within a record, column fields are positioned at a fixed offset from the beginning of the record. Except sometimes for the last field, column fields are also of fixed length. If the last field has varying length, the type of the table is DOS. For instance, having the file *dept.dat* formatted like:

```
0318 KINGSTON       70012 SALES       Bank/Insurance
0021 ARMONK         87777 CHQ         Corporate headquarter
0319 HARRISON       40567 SALES       Federal Administration
2452 POUGHKEEPSIE   31416 DEVELOPMENT Research & development
```

You can define a table based on it with:

```
create table department (
  number char(4) not null,
  location char(15) not null flag=5,
  director char(5) not null flag=20,
  function char(12) not null flag=26,
  name char(22) not null flag=38)
engine=CONNECT table_type=DOS file_name='dept.dat';
```

Here the flag column option represents the offset of this column inside the records. If the offset of a column is not specified, it defaults to the end of the previous column and defaults to 0 for the first one. The `lrecl` parameter that represents the maximum size of a record is calculated by default as the end of the rightmost column and can be unspecified except when some trailing information exists after the rightmost column.

> **Note:** A special case is files having an encoding such as UTF-8 (for instance specifying `charset=UTF8`) in which some characters may be represented with several bytes. Unlike the type size that MariaDB interprets as a number of characters, the `lrecl` value is the record size in bytes and the flag value represents the offset of the field in the record in bytes. If the flag and/or the `lrecl` value are not specified, they will be calculated by the number of characters in the fields multiplied by a value that is the maximum size in bytes of a character for the corresponding charset. For UTF-8 this value is 3 which is often far too much as there are very few characters requiring 3 bytes to be represented. When creating a new file, you are on the safe side by only doubling the maximum number of characters of a field to calculate the offset of the next field. Of course, for already existing files, the offset must be specified according to what it is in it.

Although the field representation is always text in the table file, you can freely choose the corresponding column type, characters, date, integer or floating point according to its contents.

Sometimes, as in the *number* column of the above *department* table, you have the choice of the type, numeric or characters. This will modify how the column is internally handled — in characters `0021` is different from `21` but not in numeric — as well as how it is displayed.

If the last field has fixed length, the table should be referred as having the type `FIX`. For instance, to create a table on the file *boys.txt*:

```
John      Boston      25/01/1986  02/06/2010
Henry     Boston      07/06/1987  01/04/2008
George    San Jose    10/08/1981  02/06/2010
Sam       Chicago     22/11/1979  10/10/2007
James     Dallas      13/05/1992  14/12/2009
Bill      Boston      11/09/1986  10/02/2008
```

You can for instance use the command:

```
create table boys (
  name char(12) not null,
  city char(12) not null,
  birth date not null date_format='DD/MM/YYYY',
  hired date not null date_format='DD/MM/YYYY' flag=36)
engine=CONNECT table_type=FIX file_name='boys.txt' lrecl=48;
```

Here some *flag* options were not specified because the fields have no intermediate space between them except for the last column. The offsets are calculated by default adding the field length to the *offset* of the preceding field. However, for formatted date columns, the offset in the file depends on the format and cannot be calculated by default. For fixed files, the *lrecl* option is the physical length of the record including the line ending character(s). It is calculated by adding to the end of the last field 2 bytes under Windows (CRLF) or 1 byte under UNIX. If the file is imported from another operating system, the `ENDING` option will have to be specified with the proper value.

For this table, the last offset and the record length must be specified anyway because the date columns have field length coming from their format that is not known by CONNECT. Do not forget to add the line ending length to the total length of the fields.

This table is displayed as:

name	city	birth	hired
John	Boston	1986-01-25	2010-06-02
Henry	Boston	1987-06-07	2008-04-01
George	San Jose	1981-08-10	2010-06-02
Sam	Chicago	1979-11-22	2007-10-10
James	Dallas	1992-05-13	2009-12-14
Bill	Boston	1986-09-11	2008-02-10

Whenever possible, the fixed format should be preferred to the varying one because it is much faster to deal with fixed tables than with variable tables. Sure enough, instead of being read or written record by record, FIX tables are processed by blocks of `BLOCK_SIZE` records, resulting in far less input/output operations to execute. The block size defaults to 100 if not specified in the Create Table statement.

Note 1: It is not mandatory to declare in the table all the fields existing in the source file. However, if some fields are ignored, the *flag* option of the following field and/or the *lrecl* option will have to be specified.

Note 2: Some files have an EOF marker (CTRL+Z 1A) that can prevent the table to be recognized as fixed because the file length is not a multiple of the fixed record size. To indicate this, use in the option list the create option EOF. For instance, if after creating the FIX table *xtab* on the file *foo.dat* that you know have fixed record size, you get, when you try to use it, a message such as:

```
File foo.dat is not fixed length, len=302587 lrecl=141
```

After checking that the LRECL default or specified specification is correct, you can indicate to ignore that extra EOF character by:

```
alter table xtab option_list='eof=1';
```

Of course, you can specify this option directly in the Create statement. All this applies to some other table types, in particular to BIN tables.

Note 3: The width of the fields is the length specified in the column declaration. For instance for a column declared as:

```
number int(3) not null,
```

The field width in the file is 3 characters. This is the value used to calculate the offset of the next field if it is not specified. If this length is not specified, it defaults to the MySQL default type length.

Specifying the Field Format

Some files have specific format for their numeric fields. For instance, the decimal point is absent and/or the field should be filled with leading zeros. To deal with such files, as well in reading as in writing, the format can be specified in the `CREATE TABLE` column definition. The syntax of the field format specification is:

```
Field_format='[Z][N][d]'
```

The optional parts of the format are:

Z The field has leading zeros

N No decimal point exist in the file

d The number of decimals, defaults to the column precision

Example

Let us see how it works in the following example. We define a table based on the file xfmt.txt having eight fields of 12 characters:

```
create table xfmt (
  col1 double(12,3) not null,
  col2 double(12,3) not null field_format='4',
  col3 double(12,2) not null field_format='N3',
  col4 double(12,3) not null field_format='Z',
  col5 double(12,3) not null field_format='Z3',
  col6 double(12,5) not null field_format='ZN5',
  col7 int(12) not null field_format='N3',
  col8 smallint(12) not null field_format='N3')
engine=CONNECT table_type=FIX file_name='xfmt.txt';

insert into xfmt values(4567.056,4567.056,4567.056,4567.056,-23456.8,
    3.14159,4567,4567);
select * from xfmt;
```

The first row is displayed as:

COL1	COL2	COL3	COL4	COL5	COL6	COL7	COL8
4567.056	4567.056	4567.06	4567.056	-23456.800	3.14159	4567	4567

The number of decimals displayed for all float columns is the column precision, the second argument of the column type option. Of course, integer columns have no decimals, although their formats specify some.

More interesting is the file layout. To see it let us define another table based on the same file but whose columns are all characters:

```
create table cfmt (
  col1 char(12) not null,
  col2 char(12) not null,
  col3 char(12) not null,
  col4 char(12) not null,
  col5 char(12) not null,
  col6 char(12) not null,
  col7 char(12) not null,
  col8 char(12) not null)
engine=CONNECT table_type=FIX file_name='xfmt.txt';
select * from cfmt;
```

The (transposed) display of the select command shows the file text layout for each field. Below a third column was added in this document to comment this result.

Column	Row 1	Comment (all numeric fields are written right justified)
COL1	`4567.056`	No format, the value was entered as is.
COL2	`4567.0560`	The format '4' forces to write 4 decimals.
COL3	`4567060`	N3 → No decimal point. The last 3 digits are decimals. However, the second decimal was rounded because of the column precision.
COL4	`00004567.056`	Z → Leading zeros, 3 decimals (the column precision)
COL5	`-0023456.800`	Z3 → (Minus sign) leading zeros, 3 decimals.
COL6	`000000314159`	ZN5 → Leading zeros, no decimal point, 5 decimals.

COL7	`4567000`	N3 → No decimal point. The last 3 digits are decimals.
COL8	`4567000`	Same. Any decimals would be ignored.

Note: For columns internally using double precision floating-point numbers, MariaDB limits the decimal precision of any calculation to the column precision. The declared column precision should be at least the number of decimals of the format to avoid a loss of decimals as it happened for `col3` of the above example.

5.3.7.6.6 CONNECT DBF Table Type

Contents

Overview

A table of type `DBF` is physically a dBASE III or IV formatted file (used by many products like dBASE, Xbase, FoxPro etc.). This format is similar to the FIX type format with in addition a prefix giving the characteristics of the file, describing in particular all the fields (columns) of the table.

Because `DBF` files have a header that contains Meta data about the file, in particular the column description, it is possible to create a table based on an existing `DBF` file without giving the column description, for instance:

```
create table cust engine=CONNECT table_type=DBF file_name='cust.dbf';
```

To see what CONNECT has done, you can use the `DESCRIBE` or `SHOW CREATE TABLE` commands, and eventually modify some options with the `ALTER TABLE` command.

The case of deleted lines is handled in a specific way for DBF tables. Deleted lines are not removed from the file but are "soft deleted" meaning they are marked as deleted. In particular, the number of lines contained in the file header does not take care of soft deleted lines. This is why if you execute these two commands applied to a DBF table named *tabdbf*:

```
select count(*) from tabdbf;
select count(*) from tabdbf where 1;
```

They can give a different result, the (fast) first one giving the number of physical lines in the file and the second one giving the number of line that are not (soft) deleted.

The commands UPDATE, INSERT, and DELETE can be used with DBF tables. The DELETE command marks the deleted lines as suppressed but keeps them in the file. The INSERT command, if it is used to populate a newly created table, constructs the file header before inserting new lines.

Note: For DBF tables, column name length is limited to 11 characters and field length to 256 bytes.

Conversion of dBASE Data Types

CONNECT handles only types that are stored as characters.

Symbol	DBF Type	CONNECT Type	Description
B	Binary (string)	TYPE_STRING	10 digits representing a .DBT block number.
C	Character	TYPE_STRING	All OEM code page characters - padded with blanks to the width of the field.
D	Date	TYPE_DATE	8 bytes - date stored as a string in the format YYYYMMDD.
N	Numeric	TYPE_INT TYPE_BIGINT TYPE_DOUBLE	Number stored as a string, right justified, and padded with blanks to the width of the field.
L	Logical	TYPE_STRING	1 byte - initialized to 0x20 otherwise T or F.
M	Memo (string)	TYPE_STRING	10 digits representing a .DBT block number.
@	Timestamp	Not supported	8 bytes - two longs, first for date, second for time. It is the number of days since 01/01/4713 BC.
I	Long	Not supported	4 bytes. Leftmost bit used to indicate sign, 0 negative.

+	Autoincrement	Not supported	Same as a Long
F	Float	TYPE_DOUBLE	Number stored as a string, right justified, and padded with blanks to the width of the field.
O	Double	Not supported	8 bytes - no conversions, stored as a double.
G	OLE	TYPE_STRING	10 digits representing a .DBT block number.

For the N numeric type, CONNECT converts it to TYPE_DOUBLE if the decimals value is not 0, to TYPE_BIGINT if the length value is greater than 10, else to TYPE_INT.

For M, B, and G types, CONNECT just returns the DBT number.

Reading soft deleted lines of a DBF table

It is possible to read these lines by changing the read mode of the table. This is specified by an option `READMODE` that can take the values:

0 Standard mode. This is the default option.

1 Read all lines including soft deleted ones.

2 Read only the soft deleted lines.

For example, to read all lines of the tabdbf table, you can do:

```
alter table tabdbf option_list='Readmode=1';
```

To come back to normal mode, specify READMODE=0.

5.3.7.6.7 CONNECT BIN Table Type

Contents

Overview

A table of type BIN is physically a binary file in which each row is a logical record of fixed length [1]. Within a record, column fields are of a fixed offset and length as with FIX tables. Specific to BIN tables is that numerical values are internally encoded using native platform representation, so no conversion is needed to handle numerical values in expressions.

It is not required that the lines of a BIN file be separated by characters such as CR and/or LF but this is possible. In such an event, the *lrecl* option must be specified accordingly.

Note: Unlike for the DOS and FIX types, the width of the fields is the length of their internal representation in the file. For instance for a column declared as:

```
number int(5) not null,
```

The field width in the file is 4 characters, the size of a binary integer. This is the value used to calculate the offset of the next field if it is not specified. Therefore, if the next field is placed 5 characters after this one, this declaration is not enough, and the flag option will have to be used on the next field.

Type Conversion in BIN Tables

Here are the correspondences between the column type and field format provided by default:

Column type	File default format
Char(*n*)	Text of *n* characters.
Date	Integer (4 bytes)
Int(*n*)	Integer (4 bytes)

Smallint(*n*)	Short integer (2 bytes)
TinyInt(*n*)	Char (1 Byte)
Bigint(*n*)	Large integer (8 bytes)
Double(*n,d*)	Double floating point (8 bytes)

However, the column type need not necessarily match the field format within the table file. In particular, this occurs for field formats that correspond to numeric types that are not handled by CONNECT[2]. Indeed, BIN table files may internally contain float numbers or binary numbers of any byte length in big-endian or little-endian representation[3]. Also, as in DOS or FIX types tables, you may want to handle some character fields as numeric or vice versa.

This is why it is possible to specify the field format when it does not correspond to the column type default using the *field_format* column option in the CREATE TABLE statement. Here are the available field formats for BIN tables:

Field_format	Internal representation
[n]{L or B or H}[n]	n bytes binary number in little endian, big endian or host endian representation.
C	Characters string (*n* bytes)
I	integer (4 bytes)
D	Double float (8 bytes)
S	Short integer (2 bytes)
T	Tiny integer (1 byte)
G	Big integer (8 bytes)
F or **R**	Real or float (Floating point number on 4 bytes)
X	Use the default format field for the column type

All field formats (except the first one) are a one-character specification [4]. 'X' is equivalent to not specifying the field format. For the 'C' character specification, *n* is the column width as specified with the column type. For one-column formats, the number of bytes of the numeric fields corresponds to what it is on most platforms. However, it could vary for some. The G, I, S and T formats are deprecated because they correspond to supported data types and may not be supported in future versions.

Example

Here is an example of a BIN table. The file record layout is supposed to be:

```
NNNNCCCCCCCCCCIIIISSFFFFSS
```

Here N represents numeric characters, C any characters, I integer bytes, S short integer bytes, and F float number bytes. The `IIII` field contains a date in numeric format.

The table could be created by:

```
create table testbal (
fig int(4) not null field_format='C',
name char(10) not null,
birth date not null field_format='L',
id char(5) not null field_format='L2',
salary double(9,2) not null default 0.00 field_format='F',
dept int(4) not null field_format='L2')
engine=CONNECT table_type=BIN block_size=5 file_name='Testbal.dat';
```

Specifying the little-endian representation for binary values is not useful on most machines, but makes the create table statement portable on a machine using big endian, as well as the table file.

The field offsets and the file record length are calculated according the column internal format and eventually modified by the field format. It is not necessary to specify them for a packed binary file without line endings. If a line ending is desired, specify the ending option or specify the `lrecl` option adding the ending width. The table can be filled by:

```
insert into testbal values
  (5500,'ARCHIBALD','1980-01-25','3789',4380.50,318),
  (123,'OLIVER','1953-08-10','23456',3400.68,2158),
  (3123,'FOO','2002-07-23','888',default,318);
```

Note that the types of the inserted values must match the column type, not the field format type.

The query:

```
select * from testbal;
```

returns:

fig	name	birth	id	salary	dept
5500	ARCHIBALD	1980-01-25	3789	4380.50	318
123	OLIVER	1953-08-10	23456	3400.68	2158
3123	FOO	2002-07-23	888	0.00	318

Numeric fields alignment

In binary files, numeric fields and record length can be aligned on 4-or-8-byte boundaries to optimize performance on certain processors. This can be modified in the OPTION_LIST with an "align" option ("packed" meaning `align=1` is the default).

---

1. ↑ Sometimes it can be a physical record if LF or CRLF have been written in the file.
2. ↑ Most of these are obsolete because CONNECT supports all column types except float
3. ↑ The default endian representation used in the table file can be specified by setting the ENDIAN option as 'L' or 'B' in the option list.
4. ↑ It can be specified with more than one character, but only the first one is significant.

5.3.7.6.8 CONNECT VEC Table Type

Contents

Warning: Avoid using this table type in production applications. This file format is specific to CONNECT and may not be supported in future versions.

Tables of type `VEC` are binary files that in some cases can provide good performance on read-intensive query workloads. CONNECT organizes their data on disk as columns of values from the same attribute, as opposed to storing it as rows of tabular records. This organization means that when a query needs to access only a few columns of a particular table, only those columns need to be read from disk. Conversely, in a row-oriented table, all values in a table are typically read from disk, wasting I/O bandwidth.

CONNECT provides two integral VEC formats, in which each column's data is adjacent.

Integral vector formats

In these true vertical formats, the VEC files are made of all the data of the first column, followed by all the data of the second column etc. All this can be in one physical file or each column data can be in a separate file. In the first case, the option max_rows=m, where m is the estimate of the maximum size (number of rows) of the table, must be specified to be able to insert some new records. This leaves an empty space after each column area in which new data can be inserted. In the second case, the "Split" option can be specified[2] at table creation and each column will be stored in a file named sequentially from the table file name followed by the rank of the column. Inserting new lines can freely augment such a table.

Differences between vector formats

These formats correspond to different needs. The integral vector format provides the best performance gain. It will be chosen when the speed of decisional queries must be optimized.

In the case of a unique file, inserting new data will be limited but there will be only one open and close to do. However, the size of the table cannot be calculated from the file size because of the eventual unused space in the file. It must be kept in a header containing the maximum number of rows and the current number of valid rows in the table. To achieve this, specify

the option Header=*n* when creating the table. If `n=1` the header will be placed at the beginning of the file, if `n=2` it will be a separate file with the type '.blk', and if `n=3` the header will be place at the end of the file. This last value is provided because batch inserting is sometimes slower when the header is at the beginning of the file. If not specified, the header option will default to 2 for this table type.

On the other hand, the "Split" format with separate files have none of these issues, and is a much safer solution when the table must frequently inserted or shared among several users.

For instance:

```
create table vtab (
a int not null,
b char(10) not null)
engine=CONNECT table_type=VEC file_name='vt.vec';
```

This table, split by default, will have the column values in files vt1.vec and vt2.vec.

For vector tables, the option *block_size=n* is used for block reading and writing; however, to have a file made of blocks of equal size, the internal value of the *max_rows=m* option is eventually increased to become a multiple of n.

Like for BIN tables, numeric values are stored using platform internal layout, the correspondence between column types and internal format being the same than the default ones given above for BIN. However, field formats are not available for VEC tables.

Header option

This applies to VEC tables that are not split. Because the file size depends on the MAX_ROWS value, CONNECT cannot know how many valid records exist in the file. Depending on the value of the HEADER option, this information is stored in a header that can be placed at the beginning of the file, at the end of the file or in a separate file called fn.blk. The valid values for the HEADER option are:

0 Defaults to 2 for standard tables and to 3 for inward tables.

1 The header is at the beginning of the file.

2 The header is in a separate file.

3 The header is at the end of the file.

The value 2 can be used when dealing with files created by another application with no header. The value 3 makes sometimes inserting in the file faster than when the header is at the beginning of the file.

Note: VEC being a file format specific to CONNECT, no big endian / little endian conversion is provided. These files are not portable between machines using a different byte order setting.

5.3.7.6.9 CONNECT CSV and FMT Table Types

Contents

CSV Type

Many source data files are formatted with variable length fields and records. The simplest format, known as `CSV` (Comma Separated Variables), has column fields separated by a separator character. By default, the separator is a comma but can be specified by the `SEP_CHAR` option as any character, for instance a semi-colon.

If the CSV file first record is the list of column names, specifying the `HEADER=1` option will skip the first record on reading. On writing, if the file is empty, the column names record is automatically written.

For instance, given the following *people.csv* file:

```
Name;birth;children
"Archibald";17/05/01;3
"Nabucho";12/08/03;2
```

You can create the corresponding table by:

```
create table people (
  name char(12) not null,
  birth date not null date_format='DD/MM/YY',
  children smallint(2) not null)
engine=CONNECT table_type=CSV file_name='people.csv'
header=1 sep_char=';' quoted=1;
```

Alternatively the engine can attempt to automatically detect the column names, data types and widths using:

```
create table people
engine=CONNECT table_type=CSV file_name='people.csv'
header=1 sep_char=';' quoted=1;
```

For CSV tables, the *flag* column option is the rank of the column into the file starting from 1 for the leftmost column. This is to enable having column displayed in a different order than in the file and/or to define the table specifying only some columns of the CSV file. For instance:

```
create table people (
  name char(12) not null,
  children smallint(2) not null flag=3,
  birth date not null flag=2 date_format='DD/MM/YY')
engine=CONNECT table_type=CSV file_name='people.csv'
header=1 sep_char=';' quoted=1;
```

In this case the command:

```
select * from people;
```

will display the table as:

name	children	birth
Archibald	3	2001-05-17
Nabucho	2	2003-08-12

Many applications produce CSV files having some fields quoted, in particular because the field text contains the separator character. For such files, specify the 'QUOTED=*n*' option to indicate the level of quoting and/or the ' `QCHAR=c` ' to specify what is this eventual quoting character, which is `"` by default. Quoting with single quotes must be specified as `QCHAR=''''` . On writing, fields will be quoted depending on the value of the quoting level, which is `-1` by default meaning no quoting:

0 The fields between quotes are read and the quotes discarded. On writing, fields will be quoted only if they contain the separator character or begin with the quoting character. If they contain the quoting character, it will be doubled.

1 Only text fields will be written between quotes, except null fields. This includes also the column names of an eventual header.

2 All fields will be written between quotes, except null fields.

3 All fields will be written between quotes, including null fields.

Files written this way are successfully read by most applications including spreadsheets.

Note 1: If only the QCHAR option is specified, the QUOTED option will default to 1.

Note 2: For CSV tables whose separator is the tab character, specify `sep_char='\t'` .

Note 3: When creating a table on an existing CSV file, you can let CONNECT analyze the file and make the column description. However, this is a not an elaborate analysis of the file and, for instance, `DATE` fields will not be recognized as such but will be regarded as string fields.

Note 4: The CSV parser only reads and buffers up to 4KB per row by default, rows longer than this will be truncated when read from the file. If the rows are expected to be longer than this use `lrecl` to increase this. For example to set an 8KB

maximum row read you would use `lrecl=8192`

Restrictions on CSV Tables

- If `secure_file_priv` is set to the path of some directory, then CSV tables can only be created with files in that directory.

FMT Type

FMT tables handle files of various formats that are an extension of the concept of CSV files. CONNECT supports these files providing all lines have the same format and that all fields present in all records are recognizable (optional fields must have recognizable delimiters). These files are made by specific application and CONNECT handles them in read only mode.

FMT tables must be created as CSV tables, specifying their type as FMT. In addition, each column description must be added to its format specification.

Column Format Specification of FMT tables

The input format for each column is specified as a FIELD_FORMAT option. A simple example is:

```
IP Char(15) not null field_format=' %n%s%n',
```

In the above example, the format for this (1st) field is `' %n%s%n'`. Note that the blank character at the beginning of this format **is** significant. No trailing blank should be specified in the column formats.

The syntax and meaning of the column input format is the one of the C **scanf** function.

However, CONNECT uses the input format in a specific way. Instead of using it to directly store the input value in the column buffer; it uses it to delimit the sub string of the input record that contains the corresponding column value. Retrieving this value is done later by the column functions as for standard CSV files.

This is why all column formats are made of five components:

1. An eventual description of what is met and ignored before the column value.
2. A marker of the beginning of the column value written as `%n`.
3. The format specification of the column value itself.
4. A marker of the end of the column value written as `%n` (or `%m` for optional fields).
5. An eventual description of what is met after the column value (not valid is `%m` was used).

For example, taking the file *funny.txt*:

```
12345,'BERTRAND',#200;5009.13
 56, 'POIROT-DELMOTTE' ,#4256 ;18009
345 ,'TRUCMUCHE' , #67; 19000.25
```

You can make a table *fmtsample* with 4 columns ID, NAME, DEPNO and SALARY, using the Create Table statement and column formats:

```
create table FMTSAMPLE (
  ID Integer(5) not null field_format=' %n%d%n',
  NAME Char(16) not null field_format=' , ''%n%[^''']%n''',
  DEPNO Integer(4) not null field_format=' , #%n%d%n',
  SALARY Double(12,2) not null field_format=' ; %n%f%n')
Engine=CONNECT table_type=FMT file_name='funny.txt';
```

Field 1 is an integer (`%d`) with eventual leading blanks.

Field 2 is separated from field 1 by optional blanks, a comma, and other optional blanks and is between single quotes. The leading quote is included in component 1 of the column format, followed by the `%n` marker. The column value is specified as `%[^']` meaning to keep any characters read until a quote is met. The ending marker (`%n`) is followed by the 5th component of the column format, the single quote that follows the column value.

Field 3, also separated by a comma, is a number preceded by a pound sign.

Field 4, separated by a semicolon eventually surrounded by blanks, is a number with an optional decimal point (`%f`).

This table will be displayed as:

ID	NAME	DEPNO	SALARY

12345	BERTRAND	200	5009.13
56	POIROT-DELMOTTE	4256	18009.00
345	TRUCMUCHE	67	19000.25

Optional Fields

To be recognized, a field normally must be at least one character long. For instance, a numeric field must have at least one digit, or a character field cannot be void. However many existing files do not follow this format.

Let us suppose for instance that the preceding example file could be:

```
12345,'BERTRAND',#200;5009.13
 56, 'POIROT-DELMOTTE' ,# ;18009
345 ,'' , #67; 19000.25
```

This will display an error message such as *"Bad format line x field y of FMTSAMPLE".* To avoid this and accept these records, the corresponding fields must be specified as "optional". In the above example, fields 2 and 3 can have null values (in lines 3 and 2 respectively). To specify them as optional, their format must be terminated by `%m` (instead of the second `%n`). A statement such as this can do the table creation:

```
create table FMTAMPLE (
  ID Integer(5) not null field_format=' %n%d%n',
  NAME Char(16) not null field_format=' , ''%n%[^''']%m',
  DEPNO Integer(4) field_format=''' , #%n%d%m',
  SALARY Double(12,2) field_format=' ; %n%f%n')
Engine=CONNECT table_type=FMT file_name='funny.txt';
```

Note that, because the statement must be terminated by `%m` with no additional characters, skipping the ending quote of field 2 was moved from the end of the second column format to the beginning of the third column format.

The table result is:

ID	NAME	DEPNO	SALARY
12345	BERTRAND	200	5,009.13
56	POIROT-DELMOTTE	NULL	18,009.00
345	NULL	67	19,000.25

Missing fields are replaced by null values if the column is nullable, blanks for character strings and 0 for numeric fields if it is not.

Note 1: Because the formats are specified between quotes, quotes belonging to the formats must be doubled or escaped to avoid a `CREATE TABLE` statement syntax error.

Note 2: Characters separating columns can be included as well in component 5 of the preceding column format or in component 1 of the succeeding column format but for blanks, which should be always included in component 1 of the succeeding column format because line trailing blanks can be sometimes lost. This is also mandatory for optional fields.

Note 3: Because the format is mainly used to find the sub-string corresponding to a column value, the field specification does not necessarily match the column type. For instance supposing a table contains two integer columns, NBONE and NBTWO, the two lines describing these columns could be:

```
NBONE integer(5) not null field_format=' %n%d%n',
NBTWO integer(5) field_format=' %n%s%n',
```

The first one specifies a required integer field (`%d`), the second line describes a field that can be an integer, but can be replaced by a "-" (or any other) character. Specifying the format specification for this column as a character field (`%s`) enables to recognize it with no error in all cases. Later on, this field will be converted to integer by the column read function, and a null 0 value will be generated for field specified in their format as non-numeric.

Bad Record Error Processing

When no match if found for a column field the process aborts with a message such as:

```
Bad format line 3 field 4 of funny.txt
```

This can mean as well that one line of the input line is ill formed or that the column format for this field has been wrongly specified. When you know that your file contains records that are ill formatted and should be eliminated from normal processing, set the “maxerr” option of the CREATE TABLE statement, for instance:

```
Option_list='maxerr=100'
```

This will indicate that no error message be raised for the 100 first wrong lines. You can set Maxerr to a number greater than the number of wrong lines in your files to ignore them and get no errors.

Additionally, the “accept” option permit to keep those ill formatted lines with the bad field, and all succeeding fields of the record, nullified. If “accept” is specified without “maxerr”, all ill formatted lines will be accepted.

Note: This error processing also applies to CSV tables.

Fields Containing a Formatted Date

A special case is one of columns containing a formatted date. In this case, two formats must be specified:

1. The field recognition format used to delimit the date in the input record.
2. The date format used to interpret the date.
3. The field length option if the date representation is different than the standard type size.

For example, let us suppose we have a web log source file containing records such as:

```
165.91.215.31 - - [17/Jul/2001:00:01:13 -0400] - "GET /usnews/home.htm HTTP/1.1" 302
```

The create table statement shall be like this:

```
create table WEBSAMP (
  IP char(15) not null field_format='%n%s%n',
  DATE datetime not null field_format=' - - [%n%s%n -0400]'
  date_format='DD/MMM/YYYY:hh:mm:ss' field_length=20,
  FILE char(128) not null field_format=' - "GET %n%s%n',
  HTTP double(4,2) not null field_format=' HTTP/%n%f%n"',
  NBONE int(5) not null field_format=' %n%d%n')
Engine=CONNECT table_type=FMT lrecl=400
file_name='e:\\data\\token\\Websamp.dat';
```

Note 1: Here, `field_length=20` was necessary because the default size for datetime columns is only 19. The `lrecl=400` was also specified because the actual file contains more information in each records making the record size calculated by default too small.

Note 2: The file name could have been specified as `'e:/data/token/Websamp.dat'`.

Note 3: FMT tables are currently read only.

5.3.7.6.10 CONNECT - NoSQL Table Types

They are based on files that do not match the relational format but often represent hierarchical data. CONNECT can handle JSON, INI-CFG, XML, and some HTML files.

The way it is done is different from what MySQL or PostgreSQL does. In addition to including in a table some column values of a specific data format (JSON, XML) to be handled by specific functions, CONNECT can directly use JSON, XML or INI files that are produced by other applications, and this is the table definition that describes where and how the contained information must be retrieved.

This is also different from what MariaDB does with dynamic columns, which is close to what MySQL and PostgreSQL do with the JSON column type.

Note: The LEVEL option used with these tables should, from Connect 1.07.0002, be specified as DEPTH. Also, what was specified with the FIELD_FORMAT column option should now also be specified using JPATH or XPATH.

5.3.7.6.11 CONNECT - Files Retrieved Using Rest Queries

Starting with CONNECT version 1.07.0001, JSON, XML and possibly CSV data files can be retrieved as results from REST queries when creating or querying such tables. This is done internally by CONNECT using the CURL program generally

available on all systems (if not just install it).

This can also be done using the Microsoft Casablanca (cpprestsdk) package. To enable it, first, install the package as explained in https://github.com/microsoft/cpprestsdk. Then make the GetRest library (dll or so) as explained in Making the GetRest Library.

Note: If both are available, cpprestsdk is used preferably because it is faster. This can be changed by specifying 'curl=1' in the option list.

Note: If you want to use this feature with an older distributed version of MariaDB not featuring REST, it is possible to add it as an OEM module as explained in Adding the REST Feature as a Library Called by an OEM Table .

Creating Tables using REST

To do so, specify the HTTP of the web client and eventually the URI of the request in the CREATE TABLE statement. For example, for a query returning JSON data:

```
CREATE TABLE webusers (
  id bigint(2) NOT NULL,
  name char(24) NOT NULL,
  username char(16) NOT NULL,
  email char(25) NOT NULL,
  address varchar(256) DEFAULT NULL,
  phone char(21) NOT NULL,
  website char(13) NOT NULL,
  company varchar(256) DEFAULT NULL
) ENGINE=CONNECT DEFAULT CHARSET=utf8
TABLE_TYPE=JSON FILE_NAME='users.json' HTTP='http://jsonplaceholder.typicode.com' URI='/users';
```

As with standard JSON tables, discovery is possible, meaning that you can leave CONNECT to define the columns by analyzing the JSON file. Here you could just do:

```
CREATE TABLE webusers
ENGINE=CONNECT DEFAULT CHARSET=utf8
TABLE_TYPE=JSON FILE_NAME='users.json'
HTTP='http://jsonplaceholder.typicode.com' URI='/users';
```

For example, executing:

```
SELECT name, address FROM webusers2 LIMIT 1;
```

returns:

name	address
Leanne Graham	Kulas Light Apt. 556 Gwenborough 92998-3874 -37.3159 81.1496

Here we see that for some complex elements such as *address*, which is a Json object containing values and objects, CONNECT by default has just listed their texts separated by blanks. But it is possible to ask it to analyze in more depth the json result by adding the DEPTH option. For instance:

```
CREATE OR REPLACE TABLE webusers
ENGINE=CONNECT DEFAULT CHARSET=utf8
TABLE_TYPE=JSON FILE_NAME='users.json'
HTTP='http://jsonplaceholder.typicode.com' URI='/users'
OPTION_LIST='Depth=2';
```

Then the table will be created as:

```
CREATE TABLE `webusers3` (
  `id` bigint(2) NOT NULL,
  `name` char(24) NOT NULL,
  `username` char(16) NOT NULL,
  `email` char(25) NOT NULL,
  `address_street` char(17) NOT NULL `JPATH`='$.address.street',
  `address_suite` char(9) NOT NULL `JPATH`='$.address.suite',
  `address_city` char(14) NOT NULL `JPATH`='$.address.city',
  `address_zipcode` char(10) NOT NULL `JPATH`='$.address.zipcode',
  `address_geo_lat` char(8) NOT NULL `JPATH`='$.address.geo.lat',
  `address_geo_lng` char(9) NOT NULL `JPATH`='$.address.geo.lng',
  `phone` char(21) NOT NULL,
  `website` char(13) NOT NULL,
  `company_name` char(18) NOT NULL `JPATH`='$.company.name',
  `company_catchPhrase` char(40) NOT NULL `JPATH`='$.company.catchPhrase',
  `company_bs` varchar(36) NOT NULL `JPATH`='$.company.bs'
) ENGINE=CONNECT DEFAULT CHARSET=utf8 `TABLE_TYPE`='JSON' `FILE_NAME`='users.json'
`OPTION_LIST`='Depth=2' `HTTP`='http://jsonplaceholder.typicode.com' `URI`='/users';
```

Allowing one to get all the values of the Json result, for example:

```
SELECT name, address_city city, company_name company FROM webusers3;
```

That results in:

name	city	company
Leanne Graham	Gwenborough	Romaguera-Crona
Ervin Howell	Wisokyburgh	Deckow-Crist
Clementine Bauch McKenziehaven	Romaguera-Jacobson	
Patricia Lebsack	South Elvis	Robel-Corkery
Chelsey Dietrich	Roscoeview	Keebler LLC
Mrs. Dennis Schulist	South Christy	Considine-Lockman
Kurtis Weissnat	Howemouth	Johns Group
Nicholas Runolfsdottir V	Aliyaview	Abernathy Group
Glenna Reichert	Bartholomebury	Yost and Sons
Clementina DuBuque	Lebsackbury	Hoeger LLC

Of course, the complete create table (obtained by SHOW CREATE TABLE) can later be edited to make your table return exactly what you want to get. See the JSON table type for details about what and how to specify these.

Note that such tables are read only. In addition, the data will be retrieved from the web each time you query the table with a SELECT statement. This is fine if the result varies each time, such as when you query a weather forecasting site. But if you want to use the retrieved file many times without reloading it, just create another table on the same file without specifying the HTTP option.

Note: For JSON tables, specifying the file name is optional and defaults to tabname.type. However, you should specify it if you want to use the file later for other tables.

See the JSON table type for changes that will occur in the new CONNECT versions (distributed in early 2021).

5.3.7.6.12 CONNECT JSON Table Type

Contents

Overview

JSON (JavaScript Object Notation) is a lightweight data-interchange format widely used on the Internet. Many applications, generally written in JavaScript or PHP use and produce JSON data, which are exchanged as files of different physical formats. JSON data is often returned from REST queries.

It is also possible to query, create or update such information in a database-like manner. MongoDB does it using a JavaScript-like language. PostgreSQL includes these facilities by using a specific data type and related functions like dynamic columns.

The CONNECT engine adds this facility to MariaDB by supporting tables based on JSON data files. This is done like for XML tables by creating tables describing what should be retrieved from the file and how it should be processed.

Starting with 1.07.0002, the internal way JSON was parsed and handled was changed. The main advantage of the new way

is to reduce the memory required to parse JSON. It was from 6 to 10 times the size of the JSON source and is now only 2 to 4 times. However, this is in Beta mode and JSON tables are still handled using the old mode. To use the new mode, tables should be created with TABLE_TYPE=BSON. Another way is the set the connect_force_bson session variable to 1 or ON. Then all JSON tables will be handled as BSON. Of course, this is temporary and when successfully tested, the new way will replace the old way and all tables be created as JSON.

Let us start from the file “biblio3.json” that is the JSON equivalent of the XML Xsample file described in the XML table chapter:

```
[
  {
    "ISBN": "9782212090819",
    "LANG": "fr",
    "SUBJECT": "applications",
    "AUTHOR": [
      {
        "FIRSTNAME": "Jean-Christophe",
        "LASTNAME": "Bernadac"
      },
      {
        "FIRSTNAME": "François",
        "LASTNAME": "Knab"
      }
    ],
    "TITLE": "Construire une application XML",
    "PUBLISHER": {
      "NAME": "Eyrolles",
      "PLACE": "Paris"
    },
    "DATEPUB": 1999
  },
  {
    "ISBN": "9782840825685",
    "LANG": "fr",
    "SUBJECT": "applications",
    "AUTHOR": [
      {
        "FIRSTNAME": "William J.",
        "LASTNAME": "Pardi"
      }
    ],
    "TITLE": "XML en Action",
    "TRANSLATED": {
       "PREFIX": "adapté de l'anglais par",
       "TRANSLATOR": {
          "FIRSTNAME": "James",
        "LASTNAME": "Guerin"
        }
    },
    "PUBLISHER": {
      "NAME": "Microsoft Press",
      "PLACE": "Paris"
    },
    "DATEPUB": 1999
  }
]
```

This file contains the different items existing in JSON.

- `Arrays` : They are enclosed in square brackets and contain a list of comma separated values.
- `Objects` : They are enclosed in curly brackets. They contain a comma separated list of pairs, each pair composed of a key name between double quotes, followed by a ‘:’ character and followed by a value.
- `Values` : Values can be an array or an object. They also can be a string between double quotes, an integer or float number, a Boolean value or a null value. The simplest way for CONNECT to locate a table in such a file is by an array containing a list of objects (this is what MongoDB calls a collection of documents). Each array value will be a table row and each pair of the row objects will represent a column, the key being the column name and the value the column value.

A first try to create a table on this file will be to take the outer array as the table:

```
create table jsample (
ISBN char(15),
LANG char(2),
SUBJECT char(32),
AUTHOR char(128),
TITLE char(32),
TRANSLATED char(80),
PUBLISHER char(20),
DATEPUB int(4))
engine=CONNECT table_type=JSON
File_name='biblio3.json';
```

If we execute the query:

```
select isbn, author, title, publisher from jsample;
```

We get the result:

isbn	author	title	publisher
9782212090819	Jean-Christophe Bernadac	Construire une application XML	Eyrolles Paris
9782840825685	William J. Pardi	XML en Action	Microsoft Press Pari

Note that by default, column values that are objects have been set to the concatenation of all the string values of the object separated by a blank. When a column value is an array, only the first item of the array is retrieved (This will change in later versions of Connect).

However, things are generally more complicated. If JSON files do not contain attributes (although object pairs are similar to attributes) they contain a new item, arrays. We have seen that they can be used like XML multiple nodes, here to specify several authors, but they are more general because they can contain objects of different types, even it may not be advisable to do so.

This is why CONNECT enables the specification of a column field_format option “JPATH” (FIELD_FORMAT until Connect 1.6) that is used to describe exactly where the items to display are and how to handles arrays.

Here is an example of a new table that can be created on the same file, allowing choosing the column names, to get some sub-objects and to specify how to handle the author array.

Until Connect 1.5:

```
create table jsampall (
ISBN char(15),
Language char(2) field_format='LANG',
Subject char(32) field_format='SUBJECT',
Author char(128) field_format='AUTHOR:[" and "]',
Title char(32) field_format='TITLE',
Translation char(32) field_format='TRANSLATOR:PREFIX',
Translator char(80) field_format='TRANSLATOR',
Publisher char(20) field_format='PUBLISHER:NAME',
Location char(16) field_format='PUBLISHER:PLACE',
Year int(4) field_format='DATEPUB')
engine=CONNECT table_type=JSON File_name='biblio3.json';
```

From Connect 1.6:

```
create table jsampall (
ISBN char(15),
Language char(2) field_format='LANG',
Subject char(32) field_format='SUBJECT',
Author char(128) field_format='AUTHOR.[" and "]',
Title char(32) field_format='TITLE',
Translation char(32) field_format='TRANSLATOR.PREFIX',
Translator char(80) field_format='TRANSLATOR',
Publisher char(20) field_format='PUBLISHER.NAME',
Location char(16) field_format='PUBLISHER.PLACE',
Year int(4) field_format='DATEPUB')
engine=CONNECT table_type=JSON File_name='biblio3.json';
```

From Connect 1.07.0002

```
create table jsampall (
ISBN char(15),
Language char(2) jpath='$.LANG',
Subject char(32) jpath='$.SUBJECT',
Author char(128) jpath='$.AUTHOR[" and "]',
Title char(32) jpath='$.TITLE',
Translation char(32) jpath='$.TRANSLATOR.PREFIX',
Translator char(80) jpath='$.TRANSLATOR',
Publisher char(20) jpath='$.PUBLISHER.NAME',
Location char(16) jpath='$.PUBLISHER.PLACE',
Year int(4) jpath='$.DATEPUB')
engine=CONNECT table_type=JSON File_name='biblio3.json';
```

Given the query:

```
select title, author, publisher, location from jsampall;
```

The result is:

title	author	publisher	location
Construire une application XML	Jean-Christophe Bernadac and François Knab	Eyrolles	Paris
XML en Action	William J. Pardi	Microsoft Press	Paris

Note: The JPATH was not specified for column ISBN because it defaults to the column name.

Here is another example showing that one can choose what to extract from the file and how to “expand” an array, meaning to generate one row for each array value:

Until Connect 1.5:

```
create table jsampex (
ISBN char(15),
Title char(32) field_format='TITLE',
AuthorFN char(128) field_format='AUTHOR:[X]:FIRSTNAME',
AuthorLN char(128) field_format='AUTHOR:[X]:LASTNAME',
Year int(4) field_format='DATEPUB')
engine=CONNECT table_type=JSON File_name='biblio3.json';
```

From Connect 1.6:

```
create table jsampex (
ISBN char(15),
Title char(32) field_format='TITLE',
AuthorFN char(128) field_format='AUTHOR.[X].FIRSTNAME',
AuthorLN char(128) field_format='AUTHOR.[X].LASTNAME',
Year int(4) field_format='DATEPUB')
engine=CONNECT table_type=JSON File_name='biblio3.json';
```

From Connect 1.06.006:

```
create table jsampex (
ISBN char(15),
Title char(32) field_format='TITLE',
AuthorFN char(128) field_format='AUTHOR[*].FIRSTNAME',
AuthorLN char(128) field_format='AUTHOR[*].LASTNAME',
Year int(4) field_format='DATEPUB')
engine=CONNECT table_type=JSON File_name='biblio3.json';
```

From Connect 1.07.0002

```
create table jsampex (
ISBN char(15),
Title char(32) jpath='TITLE',
AuthorFN char(128) jpath='AUTHOR[*].FIRSTNAME',
AuthorLN char(128) jpath='AUTHOR[*].LASTNAME',
Year int(4) jpath='DATEPUB')
engine=CONNECT table_type=JSON File_name='biblio3.json';
```

It is displayed as:

ISBN	Title	AuthorFN	AuthorLN	Year
9782212090819	Construire une application XML	Jean-Christophe	Bernadac	1999
9782212090819	Construire une application XML	François	Knab	1999
9782840825685	XML en Action	William J.	Pardi	1999

Note: The example above shows that the '$.', that means the beginning of the path, can be omitted.

The Jpath Specification

From Connect 1.6, the Jpath specification has changed to be the one of the native JSON functions and more compatible with what is generally used. It is close to the standard definition and compatible to what MongoDB and other products do. The ':' separator is replaced by '.'. Position in array is accepted MongoDB style with no square brackets. Array specification specific to CONNECT are still accepted but [*] is used for expanding and [x] for multiply. However, tables created with the previous syntax can still be used by adding SEP_CHAR=':' (can be done with alter table). Also, it can be now specified as JPATH (was FIELD_FORMAT) but FIELD_FORMAT is still accepted.

Until Connect 1.5, it is the description of the path to follow to reach the required item. Each step is the key name (case sensitive) of the pair when crossing an object, and the number of the value between square brackets when crossing an array. Each specification is separated by a ':' character.

From Connect 1.6, It is the description of the path to follow to reach the required item. Each step is the key name (case sensitive) of the pair when crossing an object, and the position number of the value when crossing an array. Key specifications are separated by a '.' character.

For instance, in the above file, the last name of the second author of a book is reached by:

$.AUTHOR[1].LASTNAME *standard style $AUTHOR.1.LASTNAME* MongoDB style AUTHOR:[1]:LASTNAME *old style when SEP_CHAR=':' or until Connect 1.5*

The '$' or "$." prefix specifies the root of the path and can be omitted with CONNECT.

The array specification can also indicate how it must be processed:

For instance, in the above file, the last name of the second author of a book is reached by:

```
AUTHOR:[1]:LASTNAME
```

The array specification can also indicate how it must be processed:

Specification	Array Type	Limit	Description
n (Connect >= 1.6) or [n][1]	All	N.A	Take the nth value of the array.
[*] (Connect >= 1.6), [X] or [x] (Connect <= 1.5)	All		Expand. Generate one row for each array value.
["string"]	String		Concatenate all values separated by the specified string.
[+]	Numeric		Make the sum of all the non-null array values.
[x] (Connect >= 1.6), [*] (Connect <= 1.5)	Numeric		Make the product of all non-null array values.
[!]	Numeric		Make the average of all the non-null array values.
[>] or [<]	All		Return the greatest or least non-null value of the array.
[#]	All	N.A	Return the number of values in the array.
[]	All		Expand if under an expanded object. Otherwise sum if numeric, else concatenation separated by ", ".
	All		Between two separators, if an array, expand it if under an expanded object or take the first value of it.

Note 1: When the LIMIT restriction is applicable, only the first *m* array items are used, *m* being the value of the LIMIT option (to be specified in option_list). The LIMIT default value is 10.

Note 2: An alternative way to indicate what is to be expanded is to use the expand option in the option list, for instance:

```
OPTION_LIST='Expand=AUTHOR'
```

`AUTHOR` is here the key of the pair that has the array as a value (case sensitive). Expand is limited to only one branch (expanded arrays must be under the same object).

Let us take as an example the file `expense.json` (found here). The table jexpall expands all under and including the week array:

From Connect 1.07.0002

```
create table jexpall (
WHO char(12),
WEEK int(2) jpath='$.WEEK[*].NUMBER',
WHAT char(32) jpath='$.WEEK[*].EXPENSE[*].WHAT',
AMOUNT double(8,2) jpath='$.WEEK[*].EXPENSE[*].AMOUNT')
engine=CONNECT table_type=JSON File_name='expense.json';
```

From Connect.1.6

```
create table jexpall (
WHO char(12),
WEEK int(2) field_format='$.WEEK[*].NUMBER',
WHAT char(32) field_format='$.WEEK[*].EXPENSE[*].WHAT',
AMOUNT double(8,2) field_format='$.WEEK[*].EXPENSE[*].AMOUNT')
engine=CONNECT table_type=JSON File_name='expense.json';
```

Until Connect 1.5:

```
create table jexpall (
WHO char(12),
WEEK int(2) field_format='WEEK:[x]:NUMBER',
WHAT char(32) field_format='WEEK:[x]:EXPENSE:[x]:WHAT',
AMOUNT double(8,2) field_format='WEEK:[x]:EXPENSE:[x]:AMOUNT')
engine=CONNECT table_type=JSON File_name='expense.json';
```

WHO	WEEK	WHAT	AMOUNT
Joe	3	Beer	18.00
Joe	3	Food	12.00
Joe	3	Food	19.00
Joe	3	Car	20.00
Joe	4	Beer	19.00
Joe	4	Beer	16.00
Joe	4	Food	17.00
Joe	4	Food	17.00
Joe	4	Beer	14.00
Joe	5	Beer	14.00
Joe	5	Food	12.00
Beth	3	Beer	16.00
Beth	4	Food	17.00
Beth	4	Beer	15.00
Beth	5	Food	12.00
Beth	5	Beer	20.00
Janet	3	Car	19.00
Janet	3	Food	18.00
Janet	3	Beer	18.00

Janet	4	Car	17.00
Janet	5	Beer	14.00
Janet	5	Car	12.00
Janet	5	Beer	19.00
Janet	5	Food	12.00

The table `jexpw` shows what was bought and the sum and average of amounts for each person and week:

From Connect 1.07.0002

```
create table jexpw (
WHO char(12) not null,
WEEK int(2) not null jpath='$.WEEK[*].NUMBER',
WHAT char(32) not null jpath='$.WEEK[].EXPENSE[", "].WHAT',
SUM double(8,2) not null jpath='$.WEEK[].EXPENSE[+].AMOUNT',
AVERAGE double(8,2) not null jpath='$.WEEK[].EXPENSE[!].AMOUNT')
engine=CONNECT table_type=JSON File_name='expense.json';
```

From Connect 1.6:

```
create table jexpw (
WHO char(12) not null,
WEEK int(2) not null field_format='$.WEEK[*].NUMBER',
WHAT char(32) not null field_format='$.WEEK[].EXPENSE[", "].WHAT',
SUM double(8,2) not null field_format='$.WEEK[].EXPENSE[+].AMOUNT',
AVERAGE double(8,2) not null field_format='$.WEEK[].EXPENSE[!].AMOUNT')
engine=CONNECT table_type=JSON File_name='expense.json';
```

Until Connect 1.5:

```
create table jexpw (
WHO char(12) not null,
WEEK int(2) not null field_format='WEEK:[x]:NUMBER',
WHAT char(32) not null field_format='WEEK::EXPENSE:[", "]:WHAT',
SUM double(8,2) not null field_format='WEEK::EXPENSE:[+]:AMOUNT',
AVERAGE double(8,2) not null field_format='WEEK::EXPENSE:[!]:AMOUNT')
engine=CONNECT table_type=JSON File_name='expense.json';
```

WHO	WEEK	WHAT	SUM	AVERAGE
Joe	3	Beer, Food, Food, Car	69.00	17.25
Joe	4	Beer, Beer, Food, Food, Beer	83.00	16.60
Joe	5	Beer, Food	26.00	13.00
Beth	3	Beer	16.00	16.00
Beth	4	Food, Beer	32.00	16.00
Beth	5	Food, Beer	32.00	16.00
Janet	3	Car, Food, Beer	55.00	18.33
Janet	4	Car	17.00	17.00
Janet	5	Beer, Car, Beer, Food	57.00	14.25

Let us see what the table `jexpz` does:

From Connect 1.6:

```
create table jexpz (
WHO char(12) not null,
WEEKS char(12) not null field_format='WEEK[", "].NUMBER',
SUMS char(64) not null field_format='WEEK["+"].EXPENSE[+].AMOUNT',
SUM double(8,2) not null field_format='WEEK[+].EXPENSE[+].AMOUNT',
AVGS char(64) not null field_format='WEEK["+"].EXPENSE[!].AMOUNT',
SUMAVG double(8,2) not null field_format='WEEK[+].EXPENSE[!].AMOUNT',
AVGSUM double(8,2) not null field_format='WEEK[!].EXPENSE[+].AMOUNT',
AVERAGE double(8,2) not null field_format='WEEK[!].EXPENSE[*].AMOUNT')
engine=CONNECT table_type=JSON File_name='expense.json';
```

From Connect 1.07.0002

```
create table jexpz (
WHO char(12) not null,
WEEKS char(12) not null jpath='WEEK[", "].NUMBER',
SUMS char(64) not null jpath='WEEK["+"].EXPENSE[+].AMOUNT',
SUM double(8,2) not null jpath='WEEK[+].EXPENSE[+].AMOUNT',
AVGS char(64) not null jpath='WEEK["+"].EXPENSE[!].AMOUNT',
SUMAVG double(8,2) not null jpath='WEEK[+].EXPENSE[!].AMOUNT',
AVGSUM double(8,2) not null jpath='WEEK[!].EXPENSE[+].AMOUNT',
AVERAGE double(8,2) not null jpath='WEEK[!].EXPENSE[*].AMOUNT')
engine=CONNECT table_type=JSON File_name='expense.json';
```

Until Connect 1.5:

```
create table jexpz (
WHO char(12) not null,
WEEKS char(12) not null field_format='WEEK:[", "]:NUMBER',
SUMS char(64) not null field_format='WEEK:["+"]:EXPENSE:[+]:AMOUNT',
SUM double(8,2) not null field_format='WEEK:[+]:EXPENSE:[+]:AMOUNT',
AVGS char(64) not null field_format='WEEK:["+"]:EXPENSE:[!]:AMOUNT',
SUMAVG double(8,2) not null field_format='WEEK:[+]:EXPENSE:[!]:AMOUNT',
AVGSUM double(8,2) not null field_format='WEEK:[!]:EXPENSE:[+]:AMOUNT',
AVERAGE double(8,2) not null field_format='WEEK:[!]:EXPENSE:[x]:AMOUNT')
engine=CONNECT table_type=JSON
File_name='E:/Data/Json/expense2.json';
```

WHO	WEEKS	SUMS	SUM	AVGS	SUMAVG	AVGSUM	AVERAGE
Joe	3, 4, 5	69.00+83.00+26.00	178.00	17.25+16.60+13.00	46.85	59.33	16.18
Beth	3, 4, 5	16.00+32.00+32.00	80.00	16.00+16.00+16.00	48.00	26.67	16.00
Janet	3, 4, 5	55.00+17.00+57.00	129.00	18.33+17.00+14.25	49.58	43.00	16.12

For all persons:

- Column 1 show the person name.
- Column 2 shows the weeks for which values are calculated.
- Column 3 lists the sums of expenses for each week.
- Column 4 calculates the sum of all expenses by person.
- Column 5 shows the week's expense averages.
- Column 6 calculates the sum of these averages.
- Column 7 calculates the average of the week's sum of expenses.
- Column 8 calculates the average expense by person.

It would be very difficult, if even possible, to obtain this result from table `jexpall` using an SQL query.

Handling of NULL Values

Json has a null explicit value that can be met in arrays or object key values. When regarding json as a relational table, a column value can be null because the corresponding json item is explicitly null, or implicitly because the corresponding item is missing in an array or object. CONNECT does not make any distinction between explicit and implicit nulls.

However, it is possible to specify how nulls are handled and represented. This is done by setting the string session variable connect_json_null. The default value of connect_json_null is "<null>"; it can be changed, for instance, by:

```
SET connect_json_null='NULL';
```

This changes its representation when a column displays the text of an object or the concatenation of the values of an array.

It is also possible to tell CONNECT to ignore nulls by:

```
SET connect_json_null=NULL;
```

When doing so, nulls do not appear in object text or array lists. However, this does not change the behavior of array calculation nor the result of array count.

Having Columns defined by Discovery

It is possible to let the MariaDB discovery process do the job of column specification. When columns are not defined in the create table statement, CONNECT endeavors to analyze the JSON file and to provide the column specifications. This is possible only for tables represented by an array of objects because CONNECT retrieves the column names from the object pair keys and their definition from the object pair values. For instance, the jsample table could be created saying:

```
create table jsample engine=connect table_type=JSON file_name='biblio3.json';
```

Let's check how it was actually specified using the show create table statement:

```
CREATE TABLE `jsample` (
  `ISBN` char(13) NOT NULL,
  `LANG` char(2) NOT NULL,
  `SUBJECT` char(12) NOT NULL,
  `AUTHOR` varchar(256) DEFAULT NULL,
  `TITLE` char(30) NOT NULL,
  `TRANSLATED` varchar(256) DEFAULT NULL,
  `PUBLISHER` varchar(256) DEFAULT NULL,
  `DATEPUB` int(4) NOT NULL
) ENGINE=CONNECT DEFAULT CHARSET=latin1 `TABLE_TYPE`='JSON' `FILE_NAME`='biblio3.json';
```

It is equivalent except for the column sizes that have been calculated from the file as the maximum length of the corresponding column when it was a normal value. For columns that are json arrays or objects, the column is specified as a varchar string of length 256, supposedly big enough to contain the sub-object's concatenated values. Nullable is set to true if the column is null or missing in some rows or if its JPATH contains arrays.

If a more complex definition is desired, you can ask CONNECT to analyse the JPATH up to a given depth using the DEPTH or LEVEL option in the option list. Its default value is 0 but can be changed setting the connect_default_depth session variable (in future versions the default will be 5). The depth value is the number of sub-objects that are taken in the JPATH2 (this is different from what is defined and returned by the native Json_Depth function).

For instance:

```
create table jsampall2 engine=connect table_type=JSON
  file_name='biblio3.json' option_list='level=1';
```

This will define the table as:

From Connect 1.07.0002

```
CREATE TABLE `jsampall2` (
  `ISBN` char(13) NOT NULL,
  `LANG` char(2) NOT NULL,
  `SUBJECT` char(12) NOT NULL,
  `AUTHOR_FIRSTNAME` char(15) NOT NULL `JPATH`='$.AUTHOR.[0].FIRSTNAME',
  `AUTHOR_LASTNAME` char(8) NOT NULL `JPATH`='$.AUTHOR.[0].LASTNAME',
  `TITLE` char(30) NOT NULL,
  `TRANSLATED_PREFIX` char(23) DEFAULT NULL `JPATH`='$.TRANSLATED.PREFIX',
  `TRANSLATED_TRANSLATOR` varchar(256) DEFAULT NULL `JPATH`='$.TRANSLATED.TRANSLATOR',
  `PUBLISHER_NAME` char(15) NOT NULL `JPATH`='$.PUBLISHER.NAME',
  `PUBLISHER_PLACE` char(5) NOT NULL `JPATH`='$.PUBLISHER.PLACE',
  `DATEPUB` int(4) NOT NULL
) ENGINE=CONNECT DEFAULT CHARSET=latin1 `TABLE_TYPE`='JSON'
  `FILE_NAME`='biblio3.json' `OPTION_LIST`='depth=1';
```

From Connect 1.6:

```
CREATE TABLE `jsampall2` (
  `ISBN` char(13) NOT NULL,
  `LANG` char(2) NOT NULL,
  `SUBJECT` char(12) NOT NULL,
  `AUTHOR_FIRSTNAME` char(15) NOT NULL `FIELD_FORMAT`='AUTHOR..FIRSTNAME',
  `AUTHOR_LASTNAME` char(8) NOT NULL `FIELD_FORMAT`='AUTHOR..LASTNAME',
  `TITLE` char(30) NOT NULL,
  `TRANSLATED_PREFIX` char(23) DEFAULT NULL `FIELD_FORMAT`='TRANSLATED.PREFIX',
  `TRANSLATED_TRANSLATOR` varchar(256) DEFAULT NULL `FIELD_FORMAT`='TRANSLATED.TRANSLATOR',
  `PUBLISHER_NAME` char(15) NOT NULL `FIELD_FORMAT`='PUBLISHER.NAME',
  `PUBLISHER_PLACE` char(5) NOT NULL `FIELD_FORMAT`='PUBLISHER.PLACE',
  `DATEPUB` int(4) NOT NULL
) ENGINE=CONNECT DEFAULT CHARSET=latin1 `TABLE_TYPE`='JSON'
  `FILE_NAME`='biblio3.json' `OPTION_LIST`='level=1';
```

Until Connect 1.5:

```
CREATE TABLE `jsampall2` (
  `ISBN` char(13) NOT NULL,
  `LANG` char(2) NOT NULL,
  `SUBJECT` char(12) NOT NULL,
  `AUTHOR_FIRSTNAME` char(15) NOT NULL `FIELD_FORMAT`='AUTHOR::FIRSTNAME',
  `AUTHOR_LASTNAME` char(8) NOT NULL `FIELD_FORMAT`='AUTHOR::LASTNAME',
  `TITLE` char(30) NOT NULL,
  `TRANSLATED_PREFIX` char(23) DEFAULT NULL `FIELD_FORMAT`='TRANSLATED:PREFIX',
  `TRANSLATED_TRANSLATOR` varchar(256) DEFAULT NULL `FIELD_FORMAT`='TRANSLATED:TRANSLATOR',
  `PUBLISHER_NAME` char(15) NOT NULL `FIELD_FORMAT`='PUBLISHER:NAME',
  `PUBLISHER_PLACE` char(5) NOT NULL `FIELD_FORMAT`='PUBLISHER:PLACE',
  `DATEPUB` int(4) NOT NULL
) ENGINE=CONNECT DEFAULT CHARSET=latin1 `TABLE_TYPE`='JSON' `
  FILE_NAME`='biblio3.json' `OPTION_LIST`='level=1';
```

For columns that are a simple value, the Json path is the column name. This is the default when the Jpath option is not specified, so it was not specified for such columns. However, you can force discovery to specify it by setting the connect_all_path variable to 1 or ON. This can be useful if you plan to change the name of such columns and relieves you of manually specifying the path (otherwise it would default to the new name and cause the column to not or wrongly be found).

Another problem is that CONNECT cannot guess what you want to do with arrays. Here the AUTHOR array is set to 0, which means that only its first value will be retrieved unless you also had specified “Expand=AUTHOR” in the option list. But of course, you can replace it with anything else.

This method can be used as a quick way to make a “template” table definition that can later be edited to make the desired definition. In particular, column names are constructed from all the object keys of their path in order to have distinct column names. This can be manually edited to have the desired names, provided their JPATH key names are not modified.

DEPTH can also be given the value -1 to create only columns that are simple values (no array or object). It normally defaults to 0 but this can be modified setting the connect_default_depth variable.

Note: Since version 1.6.4, CONNECT eliminates columns that are “void” or whose type cannot be determined. For instance given the file sresto.json:

```
{"_id":1,"name":"Corner Social","cuisine":"American","grades":[{"grade":"A","score":6}]}
{"_id":2,"name":"La Nueva Clasica Antillana","cuisine":"Spanish","grades":[]}
```

Previously, when using discovery, creating the table by:

```
create table sjr0
engine=connect table_type=JSON file_name='sresto.json'
option_list='Pretty=0,Depth=1' lrecl=128;
```

The table was previously created as:

```
CREATE TABLE `sjr0` (
  `_id` bigint(1) NOT NULL,
  `name` char(26) NOT NULL,
  `cuisine` char(8) NOT NULL,
  `grades` char(1) DEFAULT NULL,
  `grades_grade` char(1) DEFAULT NULL `JPATH`='$.grades[0].grade',
  `grades_score` bigint(1) DEFAULT NULL `JPATH`='$.grades[0].score'
) ENGINE=CONNECT DEFAULT CHARSET=latin1 `TABLE_TYPE`='JSON'
  `FILE_NAME`='sresto.json'
  `OPTION_LIST`='Pretty=0,Depth=1,Accept=1' `LRECL`=128;
```

The column “grades” was added because of the void array in line 2. Now this column is skipped and does not appear anymore (unless the option `Accept=1` is added in the option list).

JSON Catalogue Tables

Another way to see JSON table column specifications is to use a catalogue table. For instance:

```
create table bibcol engine=connect table_type=JSON file_name='biblio3.json'
  option_list='level=2' catfunc=columns;
select column_name, type_name type, column_size size, jpath from bibcol;
```

which returns:

From Connect 1.07.0002:

column_name	type	size	jpath
ISBN	CHAR	13	$.ISBN
LANG	CHAR	2	$.LANG
SUBJECT	CHAR	12	$.SUBJECT
AUTHOR_FIRSTNAME	CHAR	15	$.AUTHOR[0].FIRSTNAME
AUTHOR_LASTNAME	CHAR	8	$.AUTHOR[0].LASTNAME
TITLE	CHAR	30	$.TITLE
TRANSLATED_PREFIX	CHAR	23	$.TRANSLATED.PREFIX
TRANSLATED_TRANSLATOR_FIRSTNAME	CHAR	5	$TRANSLATED.TRANSLATOR.FIRSTNAME
TRANSLATED_TRANSLATOR_LASTNAME	CHAR	6	$.TRANSLATED.TRANSLATOR.LASTNAME
PUBLISHER_NAME	CHAR	15	$.PUBLISHER.NAME
PUBLISHER_PLACE	CHAR	5	$.PUBLISHER.PLACE
DATEPUB	INTEGER	4	$.DATEPUB

From Connect 1.6:

column_name	type	size	jpath
ISBN	CHAR	13	
LANG	CHAR	2	
SUBJECT	CHAR	12	
AUTHOR_FIRSTNAME	CHAR	15	AUTHOR..FIRSTNAME
AUTHOR_LASTNAME	CHAR	8	AUTHOR..LASTNAME
TITLE	CHAR	30	
TRANSLATED_PREFIX	CHAR	23	TRANSLATED.PREFIX
TRANSLATED_TRANSLATOR_FIRSTNAME	CHAR	5	TRANSLATED.TRANSLATOR.FIRSTNAME
TRANSLATED_TRANSLATOR_LASTNAME	CHAR	6	TRANSLATED.TRANSLATOR.LASTNAME
PUBLISHER_NAME	CHAR	15	PUBLISHER.NAME
PUBLISHER_PLACE	CHAR	5	PUBLISHER.PLACE

DATEPUB	INTEGER	4	

Until Connect 1.5:

column_name	type	size	jpath
ISBN	CHAR	13	
LANG	CHAR	2	
SUBJECT	CHAR	12	
AUTHOR_FIRSTNAME	CHAR	15	AUTHOR::FIRSTNAME
AUTHOR_LASTNAME	CHAR	8	AUTHOR::LASTNAME
TITLE	CHAR	30	
TRANSLATED_PREFIX	CHAR	23	TRANSLATED:PREFIX
TRANSLATED_TRANSLATOR_FIRSTNAME	CHAR	5	TRANSLATED:TRANSLATOR:FIRSTNAME
TRANSLATED_TRANSLATOR_LASTNAME	CHAR	6	TRANSLATED:TRANSLATOR:LASTNAME
PUBLISHER_NAME	CHAR	15	PUBLISHER:NAME
PUBLISHER_PLACE	CHAR	5	PUBLISHER:PLACE
DATEPUB	INTEGER	4	

All this is mostly useful when creating a table on a remote file that you cannot easily see.

Finding the table within a JSON file

Given the file “facebook.json”:

```
{
   "data": [
      {
         "id": "X999_Y999",
         "from": {
            "name": "Tom Brady", "id": "X12"
         },
         "message": "Looking forward to 2010!",
         "actions": [
            {
               "name": "Comment",
               "link": "http://www.facebook.com/X999/posts/Y999"
            },
            {
               "name": "Like",
               "link": "http://www.facebook.com/X999/posts/Y999"
            }
         ],
         "type": "status",
         "created_time": "2010-08-02T21:27:44+0000",
         "updated_time": "2010-08-02T21:27:44+0000"
      },
      {
         "id": "X998_Y998",
         "from": {
            "name": "Peyton Manning", "id": "X18"
         },
         "message": "Where's my contract?",
         "actions": [
            {
               "name": "Comment",
               "link": "http://www.facebook.com/X998/posts/Y998"
            },
            {
               "name": "Like",
               "link": "http://www.facebook.com/X998/posts/Y998"
            }
         ],
         "type": "status",
         "created_time": "2010-08-02T21:27:44+0000",
         "updated_time": "2010-08-02T21:27:44+0000"
      }
   ]
}
```

The table we want to analyze is represented by the array value of the “data” object. Here is how this is specified in the create table statement:

From Connect 1.07.0002:

```
create table jfacebook (
`ID` char(10) jpath='id',
`Name` char(32) jpath='from.name',
`MyID` char(16) jpath='from.id',
`Message` varchar(256) jpath='message',
`Action` char(16) jpath='actions..name',
`Link` varchar(256) jpath='actions..link',
`Type` char(16) jpath='type',
`Created` datetime date_format='YYYY-MM-DD\'T\'hh:mm:ss' jpath='created_time',
`Updated` datetime date_format='YYYY-MM-DD\'T\'hh:mm:ss' jpath='updated_time')
engine=connect table_type=JSON file_name='facebook.json' option_list='Object=data,Expand=actions'
```

From Connect 1.6:

```
create table jfacebook (
`ID` char(10) field_format='id',
`Name` char(32) field_format='from.name',
`MyID` char(16) field_format='from.id',
`Message` varchar(256) field_format='message',
`Action` char(16) field_format='actions..name',
`Link` varchar(256) field_format='actions..link',
`Type` char(16) field_format='type',
`Created` datetime date_format='YYYY-MM-DD\'T\'hh:mm:ss' field_format='created_time',
`Updated` datetime date_format='YYYY-MM-DD\'T\'hh:mm:ss' field_format='updated_time')
engine=connect table_type=JSON file_name='facebook.json' option_list='Object=data,Expand=actions'
```

Until Connect 1.5:

```
create table jfacebook (
`ID` char(10) field_format='id',
`Name` char(32) field_format='from:name',
`MyID` char(16) field_format='from:id',
`Message` varchar(256) field_format='message',
`Action` char(16) field_format='actions::name',
`Link` varchar(256) field_format='actions::link',
`Type` char(16) field_format='type',
`Created` datetime date_format='YYYY-MM-DD\'T\'hh:mm:ss' field_format='created_time',
`Updated` datetime date_format='YYYY-MM-DD\'T\'hh:mm:ss' field_format='updated_time')
engine=connect table_type=JSON file_name='facebook.json' option_list='Object=data,Expand=actions'
```

This is the object option that gives the Jpath of the table. Note also an alternate way to declare the array to be expanded by the expand option of the option_list.

Because some string values contain a date representation, the corresponding columns are declared as datetime and the date format is specified for them.

The Jpath of the object option has the same syntax as the column Jpath but of course all array steps must be specified using the [n] (until Connect 1.5) or n (from Connect 1.6) format.

Note: This applies to the whole document for tables having `PRETTY = 2` (see below). Otherwise, it applies to the document objects of each file records.

JSON File Formats

The examples we have seen so far are files that, even they can be formatted in different ways (blanks, tabs, carriage return and line feed are ignored when parsing them), respect the JSON syntax and are made of only one item (Object or Array). Like for XML files, they are entirely parsed and a memory representation is made used to process them. This implies that they are of reasonable size to avoid an out of memory condition. Tables based on such files are recognized by the option Pretty=2 that we did not specify above because this is the default.

An alternate format, which is the format of exported MongoDB files, is a file where each row is physically stored in one file record. For instance:

```
{ "_id" : "01001", "city" : "AGAWAM", "loc" : [ -72.622739, 42.070206 ], "pop" : 15338, "state"
: "MA" }
{ "_id" : "01002", "city" : "CUSHMAN", "loc" : [ -72.51564999999999, 42.377017 ], "pop" :
36963, "state" : "MA" }
{ "_id" : "01005", "city" : "BARRE", "loc" : [ -72.1083540000001, 42.409698 ], "pop" : 4546,
"state" : "MA" }
{ "_id" : "01007", "city" : "BELCHERTOWN", "loc" : [ -72.4109530000001, 42.275103 ], "pop" :
10579, "state" : "MA" }
…
{ "_id" : "99929", "city" : "WRANGELL", "loc" : [ -132.352918, 56.433524 ], "pop" : 2573,
"state" : "AK" }
{ "_id" : "99950", "city" : "KETCHIKAN", "loc" : [ -133.18479, 55.942471 ], "pop" : 422,
"state" : "AK" }
```

The original file, “cities.json”, has 29352 records. To base a table on this file we must specify the option Pretty=0 in the option list. For instance:

From Connect 1.07.0002:

```
create table cities (
`_id` char(5) key,
`city` char(32),
`lat` double(12,6) jpath='loc.0',
`long` double(12,6) jpath='loc.1',
`pop` int(8),
`state` char(2) distrib='clustered')
engine=CONNECT table_type=JSON file_name='cities.json' lrecl=128 option_list='pretty=0';
```

From Connect 1.6:

```
create table cities (
`_id` char(5) key,
`city` char(32),
`lat` double(12,6) field_format='loc.0',
`long` double(12,6) field_format='loc.1',
`pop` int(8),
`state` char(2) distrib='clustered')
engine=CONNECT table_type=JSON file_name='cities.json' lrecl=128 option_list='pretty=0';
```

Until Connect 1.5:

```
create table cities (
`_id` char(5) key,
`city` char(32),
`long` double(12,6) field_format='loc:[0]',
`lat` double(12,6) field_format='loc:[1]',
`pop` int(8),
`state` char(2) distrib='clustered')
engine=CONNECT table_type=JSON file_name='cities.json' lrecl=128 option_list='pretty=0';
```

Note the use of [n] (until Connect 1.5) or n (from Connect 1.6) array specifications for the longitude and latitude columns.

When using this format, the table is processed by CONNECT like a DOS, CSV or FMT table. Rows are retrieved and parsed by records and the table can be very large. Another advantage is that such a table can be indexed, which can be of great value for very large tables. The “distrib” option of the “state” column tells CONNECT to use block indexing when possible.

For such tables – as well as for *pretty=1* ones – the record size must be specified using the LRECL option. Be sure you don’t specify it too small as it is used to allocate the read/write buffers and the memory used for parsing the rows. If in doubt, be generous as it does not cost much in memory allocation.

Another format exists, noted by Pretty=1, which is similar to this one but has some additions to represent a JSON array. A header and a trailer records are added containing the opening and closing square bracket, and all records but the last are followed by a comma. It has the same advantages for reading and updating, but inserting and deleting are executed in the pretty=2 way.

Alternate Table Arrangement

We have seen that the most natural way to represent a table in a JSON file is to make it on an array of objects. However, other possibilities exist. A table can be an array of arrays, a one column table can be an array of values, or a one row table can be just one object or one value. Single row tables are internally handled by adding a one value array around them.

Let us see how to handle, for instance, a table that is an array of arrays. The file:

```
[
  [56, "Coucou", 500.00],
  [[2,0,1,4], "Hello World", 2.0316],
  ["1784", "John Doo", 32.4500],
  [1914, ["Nabucho","donosor"], 5.12],
  [7, "sept", [0.77,1.22,2.01]],
  [8, "huit", 13.0]
]
```

A table can be created on this file as:

From Connect 1.07.0002:

```
create table xjson (
`a` int(6) jpath='1',
`b` char(32) jpath='2',
`c` double(10,4) jpath='3')
engine=connect table_type=JSON file_name='test.json' option_list='Pretty=1,Jmode=1,Base=1' lrecl=
```

From Connect 1.6:

```
create table xjson (
`a` int(6) field_format='1',
`b` char(32) field_format='2',
`c` double(10,4) field_format='3')
engine=connect table_type=JSON file_name='test.json' option_list='Pretty=1,Jmode=1,Base=1' lrecl=
```

Until Connect 1.5:

```
create table xjson (
`a` int(6) field_format='[1]',
`b` char(32) field_format='[2]',
`c` double(10,4) field_format='[3]')
engine=connect table_type=JSON file_name='test.json'
option_list='Pretty=1,Jmode=1,Base=1' lrecl=128;
```

Columns are specified by their position in the row arrays. By default, this is zero-based but for this table the base was set to 1 by the *Base* option of the option list. Another new option in the option list is Jmode=1. It indicates what type of table this is. The Jmode values are:

1. An array of objects. This is the default.
2. An array of Array. Like this one.
3. An array of values.

When reading, this is not required as the type of the array items is specified for the columns; however, it is required when inserting new rows so CONNECT knows what to insert. For instance:

```
insert into xjson values(25, 'Breakfast', 1.414);
```

After this, it is displayed as:

a	b	c
56	Coucou	500.0000
2	Hello World	2.0316
1784	John Doo	32.4500
1914	Nabucho	5.1200
7	sept	0.7700
8	huit	13.0000
25	Breakfast	1.4140

Unspecified array values are represented by their first element.

Getting and Setting JSON Representation of a Column

We have seen that columns corresponding to a Json object or array are retrieved by default as the concatenation of all its values separated by a blank. It is also possible to retrieve and display such column contains as the full JSON string corresponding to it in the JSON file. This is specified in the JPATH by a “*” where the object or array would be specified.

Note: When having columns generated by discovery, this can be specified by adding the STRINGIFY option to ON or 1 in the option list.

For instance:

From Connect 1.07.0002:

```
create table jsample2 (
ISBN char(15),
Lng char(2) jpath='LANG',
json_Author char(255) jpath='AUTHOR.*',
Title char(32) jpath='TITLE',
Year int(4) jpath='DATEPUB')
engine=CONNECT table_type=JSON file_name='biblio3.json';
```

From Connect 1.6:

```
create table jsample2 (
ISBN char(15),
Lng char(2) field_format='LANG',
json_Author char(255) field_format='AUTHOR.*',
Title char(32) field_format='TITLE',
Year int(4) field_format='DATEPUB')
engine=CONNECT table_type=JSON file_name='biblio3.json';
```

Until Connect 1.5:

```
create table jsample2 (
ISBN char(15),
Lng char(2) field_format='LANG',
json_Author char(255) field_format='AUTHOR:*',
Title char(32) field_format='TITLE',
Year int(4) field_format='DATEPUB')
engine=CONNECT table_type=JSON file_name='biblio3.json';
```

Now the query:

```
select json_Author from jsample2;
```

will return and display :

json_Author
[{"FIRSTNAME":"Jean-Christophe","LASTNAME":"Bernadac"},{"FIRSTNAME":"François","LASTNAME":"Knab"}]
[{"FIRSTNAME":"William J.","LASTNAME":"Pardi"}]

Note: Prefixing the column name by *json_* is optional but is useful when using the column as argument to Connect UDF functions, making it to be surely recognized as valid Json without aliasing.

This also works on input, a column specified so that it can be directly set to a valid JSON string.

This feature is of great value as we will see below.

Create, Read, Update and Delete Operations on JSON Tables

The SQL commands INSERT, UPDATE and DELETE are fully supported for JSON tables except those returned by REST queries. For INSERT and UPDATE, if the target values are simple values, there are no problems.

However, there are some issues when the added or modified values are objects or arrays.

Concerning objects, the same problems exist that we have already seen with the XML type. The added or modified object will have the format described in the table definition, which can be different from the one of the JSON file. Modifications should be done using a file specifying the full path of modified objects.

New problems are raised when trying to modify the values of an array. Only updates can be done on the original table. First of all, for the values of the array to be distinct values, all update operations concerning array values must be done using a table expanding this array.

For instance, to modify the authors of the `biblio.json` based table, the `jsampex` table must be used. Doing so, updating and deleting authors is possible using standard SQL commands. For example, to change the first name of Knab from François to John:

```
update jsampex set authorfn = 'John' where authorln = 'Knab';
```

However It would be wrong to do:

```
update jsampex set authorfn = 'John' where isbn = '9782212090819';
```

Because this would change the first name of both authors as they share the same ISBN.

Where things become more difficult is when trying to delete or insert an author of a book. Indeed, a delete command will delete the whole book and an insert command will add a new complete row instead of adding a new author in the same array. Here we are penalized by the SQL language that cannot give us a way to specify this. Something like:

```
update jsampex add authorfn = 'Charles', authorln = 'Dickens'
where title = 'XML en Action';
```

However this does not exist in SQL. Does this mean that it is impossible to do it? No, but it requires us to use a table specified on the same file but adapted to this task. One way to do it is to specify a table for which the authors are no more an expanded array. Supposing we want to add an author to the “XML en Action” book. We will do it on a table containing just the author(s) of that book, which is the second book of the table.

From Connect 1.6:

```
create table jauthor (
FIRSTNAME char(64),
LASTNAME char(64))
engine=CONNECT table_type=JSON File_name='biblio3.json' option_list='Object=1.AUTHOR';
```

Until Connect 1.5

```
create table jauthor (
FIRSTNAME char(64),
LASTNAME char(64))
engine=CONNECT table_type=JSON File_name='biblio3.json' option_list='Object=[1]:AUTHOR';
```

The command:

```
select * from jauthor;
```

replies:

FIRSTNAME	LASTNAME
William J.	Pardi

It is a standard JSON table that is an array of objects in which we can freely insert or delete rows.

```
insert into jauthor values('Charles','Dickens');
```

We can check that this was done correctly by:

```
select * from jsampex;
```

This will display:

ISBN	Title	AuthorFN	AuthorLN	Year
9782212090819	Construire une application XML	Jean-Christophe	Bernadac	1999
9782212090819	Construire une application XML	John	Knab	1999
9782840825685	XML en Action	William J.	Pardi	1999
9782840825685	XML en Action	Charles	Dickens	1999

Note: If this table were a big table with many books, it would be difficult to know what the order of a specific book is in the table. This can be found by adding a special ROWID column in the table.

However, an alternate way to do it is by using direct JSON column representation as in the `JSAMPLE2` table. This can be done by:

```
update jsample2 set json_Author =
'[{"FIRSTNAME":"William J.","LASTNAME":"Pardi"},
  {"FIRSTNAME":"Charles","LASTNAME":"Dickens"}]'
where isbn = '9782840825685';
```

Here, we didn't have to find the index of the sub array to modify. However, this is not quite satisfying because we had to manually write the whole JSON value to set to the json_Author column.

Therefore we need specific functions to do so. They are introduced now.

JSON User Defined Functions

Although such functions written by other parties do exist,[2] CONNECT provides its own UDFs that are specifically adapted to the JSON table type and easily available because, being inside the CONNECT library or DLL, they require no additional module to be loaded (see CONNECT - Compiling JSON UDFs in a Separate Library to make these functions in a separate library module).

In particular, MariaDB 10.2 and 10.3 feature native JSON functions. In some cases, it is possible that these native functions can be used. However, mixing native and UDF JSON functions in the same query often does not work because the way they recognize their arguments is different and might even cause a server crash.

Here is the list of the CONNECT functions; more can be added if required.

Name	Type	Return	Description	Added
jbin_array	Function	STRING*	Make a JSON array containing its arguments.	MariaDB 10.1.9
jbin_array_add	Function	STRING*	Adds to its first array argument its second arguments.	MariaDB 10.1.9
jbin_array_add_values	Function	STRING*	Adds to its first array argument all following arguments.	
jbin_array_delete	Function	STRING*	Deletes the nth element of its first array argument.	MariaDB 10.1.9
jbin_file	Function	STRING*	Returns of a (json) file contain.	MariaDB 10.1.9
jbin_get_item	Function	STRING*	Access and returns a json item by a JPATH key.	MariaDB 10.1.9
jbin_insert_item	Function	STRING	Insert item values located to paths.	
jbin_item_merge	Function	STRING*	Merges two arrays or two objects.	MariaDB 10.1.9
jbin_object	Function	STRING*	Make a JSON object containing its arguments.	MariaDB 10.1.9
jbin_object_nonull	Function	STRING*	Make a JSON object containing its not null arguments.	MariaDB 10.1.9
jbin_object_add	Function	STRING*	Adds to its first object argument its second argument.	MariaDB 10.1.9
jbin_object_delete	Function	STRING*	Deletes the nth element of its first object argument.	MariaDB 10.1.9
jbin_object_key	Function	STRING*	Make a JSON object for key/value pairs.	
jbin_object_list	Function	STRING*	Returns the list of object keys as an array.	MariaDB 10.1.9
jbin_set_item	Function	STRING	Set item values located to paths.	
jbin_update_item	Function	STRING	Update item values located to paths.	
jfile_bjson	Function	STRING	Convert a pretty=0 file to another BJson file.	MariaDB 10.5.9, MariaDB 10.4.18, MariaDB 10.3.28, MariaDB 10.2.36
jfile_convert	Function	STRING	Convert a Json file to another pretty=0 file.	MariaDB 10.5.9, MariaDB 10.4.18, MariaDB 10.3.28, MariaDB 10.2.36
jfile_make	Function	STRING	Make a json file from its json item first argument.	MariaDB 10.1.9

json_array	Function	STRING	Make a JSON array containing its arguments.	MariaDB 10.0.17 until Connect 1.5
json_array_add	Function	STRING	Adds to its first array argument its second arguments (before MariaDB 10.1.9, all following arguments).	
json_array_add_values	Function	STRING	Adds to its first array argument all following arguments.	MariaDB 10.1.9
json_array_delete	Function	STRING	Deletes the nth element of its first array argument.	
json_array_grp	Aggregate	STRING	Makes JSON arrays from coming argument.	
json_file	Function	STRING	Returns the contains of (json) file.	MariaDB 10.1.9
json_get_item	Function	STRING	Access and returns a json item by a JPATH key.	MariaDB 10.1.9
json_insert_item	Function	STRING	Insert item values located to paths.	
json_item_merge	Function	STRING	Merges two arrays or two objects.	MariaDB 10.1.9
json_locate_all	Function	STRING	Returns the JPATH's of all occurrences of an element.	MariaDB 10.1.9
json_make_array	Function	STRING	Make a JSON array containing its arguments.	From Connect 1.6
json_make_object	Function	STRING	Make a JSON object containing its arguments.	From Connect 1.6
json_object	Function	STRING	Make a JSON object containing its arguments.	MariaDB 10.0.17 until Connect 1.5
json_object_delete	Function	STRING	Deletes the nth element of its first object argument.	MariaDB 10.1.9
json_object_grp	Aggregate	STRING	Makes JSON objects from coming arguments.	
json_object_list	Function	STRING	Returns the list of object keys as an array.	MariaDB 10.1.9
json_object_nonull	Function	STRING	Make a JSON object containing its not null arguments.	
json_serialize	Function	STRING	Serializes the return of a "Jbin" function.	MariaDB 10.1.9
json_set_item	Function	STRING	Set item values located to paths.	
json_update_item	Function	STRING	Update item values located to paths.	
jsonvalue	Function	STRING	Make a JSON value from its unique argument. Called json_value until MariaDB 10.0.22 and MariaDB 10.1.8.	MariaDB 10.0.17
jsoncontains	Function	INTEGER	Returns 0 or 1 if an element is contained in the document.	
jsoncontains_path	Function	INTEGER	Returns 0 or 1 if a JPATH is contained in the document.	
jsonget_string	Function	STRING	Access and returns a string element by a JPATH key.	MariaDB 10.1.9
jsonget_int	Function	INTEGER	Access and returns an integer element by a JPATH key.	MariaDB 10.1.9
jsonget_real	Function	REAL	Access and returns a real element by a JPATH key.	MariaDB 10.1.9
jsonlocate	Function	STRING	Returns the JPATH to access one element.	MariaDB 10.1.9

String values are mapped to JSON strings. These strings are automatically escaped to conform to the JSON syntax. The automatic escaping is bypassed when the value has an alias beginning with 'json_'. This is automatically the case when a JSON UDF argument is another JSON UDF whose name begins with "json_" (not case sensitive). This is why all functions that do not return a Json item are not prefixed by "json_".

Argument string values, for some functions, can alternatively be json file names. When this is ambiguous, alias them as *jfile_*. Full path should be used because UDF functions has no means to know what the current database is. Apparently, when the file name path is not full, it is based on the MariaDB data directory but I am not sure it is always true.

Numeric values are (big) integers, double floating point values or decimal values. Decimal values are character strings containing a numeric representation and are treated as strings. Floating point values contain a decimal point and/or an exponent. Integers are written without decimal points.

To install these functions execute the following commands :[3]

> Note: Json function names are often written on this page with leading upper case letters for clarity. It is possible to do so in SQL queries because function names are case insensitive. However, when creating or dropping them, their names must match the case they are in the library module (lower case from MariaDB 10.1.9).

On Unix systems (from Connect 1.7.02):

```
create function jsonvalue returns string soname 'ha_connect.so';
create function json_make_array returns string soname 'ha_connect.so';
create function json_array_add_values returns string soname 'ha_connect.so';
create function json_array_add returns string soname 'ha_connect.so';
create function json_array_delete returns string soname 'ha_connect.so';
create function json_make_object returns string soname 'ha_connect.so';
create function json_object_nonull returns string soname 'ha_connect.so';
create function json_object_key returns string soname 'ha_connect.so';
create function json_object_add returns string soname 'ha_connect.so';
create function json_object_delete returns string soname 'ha_connect.so';
create function json_object_list returns string soname 'ha_connect.so';
create function json_object_values returns string soname 'ha_connect.so';
create function jsonset_grp_size returns integer soname 'ha_connect.so';
create function jsonget_grp_size returns integer soname 'ha_connect.so';
create aggregate function json_array_grp returns string soname 'ha_connect.so';
create aggregate function json_object_grp returns string soname 'ha_connect.so';
create function jsonlocate returns string soname 'ha_connect.so';
create function json_locate_all returns string soname 'ha_connect.so';
create function jsoncontains returns integer soname 'ha_connect.so';
create function jsoncontains_path returns integer soname 'ha_connect.so';
create function json_item_merge returns string soname 'ha_connect.so';
create function json_get_item returns string soname 'ha_connect.so';
create function jsonget_string returns string soname 'ha_connect.so';
create function jsonget_int returns integer soname 'ha_connect.so';
create function jsonget_real returns real soname 'ha_connect.so';
create function json_set_item returns string soname 'ha_connect.so';
create function json_insert_item returns string soname 'ha_connect.so';
create function json_update_item returns string soname 'ha_connect.so';
create function json_file returns string soname 'ha_connect.so';
create function jfile_make returns string soname 'ha_connect.so';
create function jfile_convert returns string soname 'ha_connect.so';
create function jfile_bjson returns string soname 'ha_connect.so';
create function json_serialize returns string soname 'ha_connect.so';
create function jbin_array returns string soname 'ha_connect.so';
create function jbin_array_add_values returns string soname 'ha_connect.so';
create function jbin_array_add returns string soname 'ha_connect.so';
create function jbin_array_delete returns string soname 'ha_connect.so';
create function jbin_object returns string soname 'ha_connect.so';
create function jbin_object_nonull returns string soname 'ha_connect.so';
create function jbin_object_key returns string soname 'ha_connect.so';
create function jbin_object_add returns string soname 'ha_connect.so';
create function jbin_object_delete returns string soname 'ha_connect.so';
create function jbin_object_list returns string soname 'ha_connect.so';
create function jbin_item_merge returns string soname 'ha_connect.so';
create function jbin_get_item returns string soname 'ha_connect.so';
create function jbin_set_item returns string soname 'ha_connect.so';
create function jbin_insert_item returns string soname 'ha_connect.so';
create function jbin_update_item returns string soname 'ha_connect.so';
create function jbin_file returns string soname 'ha_connect.so';
```

On Unix systems (from Connect 1.6):

```
create function jsonvalue returns string soname 'ha_connect.so';
create function json_make_array returns string soname 'ha_connect.so';
create function json_array_add_values returns string soname 'ha_connect.so';
create function json_array_add returns string soname 'ha_connect.so';
create function json_array_delete returns string soname 'ha_connect.so';
create function json_make_object returns string soname 'ha_connect.so';
create function json_object_nonull returns string soname 'ha_connect.so';
create function json_object_key returns string soname 'ha_connect.so';
create function json_object_add returns string soname 'ha_connect.so';
create function json_object_delete returns string soname 'ha_connect.so';
create function json_object_list returns string soname 'ha_connect.so';
create function jsonset_grp_size returns integer soname 'ha_connect.so';
create function jsonget_grp_size returns integer soname 'ha_connect.so';
create aggregate function json_array_grp returns string soname 'ha_connect.so';
create aggregate function json_object_grp returns string soname 'ha_connect.so';
create function jsonlocate returns string soname 'ha_connect.so';
create function json_locate_all returns string soname 'ha_connect.so';
create function jsoncontains returns integer soname 'ha_connect.so';
create function jsoncontains_path returns integer soname 'ha_connect.so';
create function json_item_merge returns string soname 'ha_connect.so';
create function json_get_item returns string soname 'ha_connect.so';
create function jsonget_string returns string soname 'ha_connect.so';
create function jsonget_int returns integer soname 'ha_connect.so';
create function jsonget_real returns real soname 'ha_connect.so';
create function json_set_item returns string soname 'ha_connect.so';
create function json_insert_item returns string soname 'ha_connect.so';
create function json_update_item returns string soname 'ha_connect.so';
create function json_file returns string soname 'ha_connect.so';
create function jfile_make returns string soname 'ha_connect.so';
create function json_serialize returns string soname 'ha_connect.so';
create function jbin_array returns string soname 'ha_connect.so';
create function jbin_array_add_values returns string soname 'ha_connect.so';
create function jbin_array_add returns string soname 'ha_connect.so';
create function jbin_array_delete returns string soname 'ha_connect.so';
create function jbin_object returns string soname 'ha_connect.so';
create function jbin_object_nonull returns string soname 'ha_connect.so';
create function jbin_object_key returns string soname 'ha_connect.so';
create function jbin_object_add returns string soname 'ha_connect.so';
create function jbin_object_delete returns string soname 'ha_connect.so';
create function jbin_object_list returns string soname 'ha_connect.so';
create function jbin_item_merge returns string soname 'ha_connect.so';
create function jbin_get_item returns string soname 'ha_connect.so';
create function jbin_set_item returns string soname 'ha_connect.so';
create function jbin_insert_item returns string soname 'ha_connect.so';
create function jbin_update_item returns string soname 'ha_connect.so';
create function jbin_file returns string soname 'ha_connect.so';
```

On Unix systems (from MariaDB 10.1.9 until Connect 1.5):

```
create function jsonvalue returns string soname 'ha_connect.so';
create function json_array returns string soname 'ha_connect.so';
create function json_array_add_values returns string soname 'ha_connect.so';
create function json_array_add returns string soname 'ha_connect.so';
create function json_array_delete returns string soname 'ha_connect.so';
create function json_object returns string soname 'ha_connect.so';
create function json_object_nonull returns string soname 'ha_connect.so';
create function json_object_key returns string soname 'ha_connect.so';
create function json_object_add returns string soname 'ha_connect.so';
create function json_object_delete returns string soname 'ha_connect.so';
create function json_object_list returns string soname 'ha_connect.so';
create function jsonset_grp_size returns integer soname 'ha_connect.so';
create function jsonget_grp_size returns integer soname 'ha_connect.so';
create aggregate function json_array_grp returns string soname 'ha_connect.so';
create aggregate function json_object_grp returns string soname 'ha_connect.so';
create function jsonlocate returns string soname 'ha_connect.so';
create function json_locate_all returns string soname 'ha_connect.so';
create function jsoncontains returns integer soname 'ha_connect.so';
create function jsoncontains_path returns integer soname 'ha_connect.so';
create function json_item_merge returns string soname 'ha_connect.so';
create function json_get_item returns string soname 'ha_connect.so';
create function jsonget_string returns string soname 'ha_connect.so';
create function jsonget_int returns integer soname 'ha_connect.so';
create function jsonget_real returns real soname 'ha_connect.so';
create function json_set_item returns string soname 'ha_connect.so';
create function json_insert_item returns string soname 'ha_connect.so';
create function json_update_item returns string soname 'ha_connect.so';
create function json_file returns string soname 'ha_connect.so';
create function jfile_make returns string soname 'ha_connect.so';
create function json_serialize returns string soname 'ha_connect.so';
create function jbin_array returns string soname 'ha_connect.so';
create function jbin_array_add_values returns string soname 'ha_connect.so';
create function jbin_array_add returns string soname 'ha_connect.so';
create function jbin_array_delete returns string soname 'ha_connect.so';
create function jbin_object returns string soname 'ha_connect.so';
create function jbin_object_nonull returns string soname 'ha_connect.so';
create function jbin_object_key returns string soname 'ha_connect.so';
create function jbin_object_add returns string soname 'ha_connect.so';
create function jbin_object_delete returns string soname 'ha_connect.so';
create function jbin_object_list returns string soname 'ha_connect.so';
create function jbin_item_merge returns string soname 'ha_connect.so';
create function jbin_get_item returns string soname 'ha_connect.so';
create function jbin_set_item returns string soname 'ha_connect.so';
create function jbin_insert_item returns string soname 'ha_connect.so';
create function jbin_update_item returns string soname 'ha_connect.so';
create function jbin_file returns string soname 'ha_connect.so';
```

On WIndows (from Connect 1.7.02):

```
create function jsonvalue returns string soname 'ha_connect';
create function json_make_array returns string soname 'ha_connect';
create function json_array_add_values returns string soname 'ha_connect';
create function json_array_add returns string soname 'ha_connect';
create function json_array_delete returns string soname 'ha_connect';
create function json_make_object returns string soname 'ha_connect';
create function json_object_nonull returns string soname 'ha_connect';
create function json_object_key returns string soname 'ha_connect';
create function json_object_add returns string soname 'ha_connect';
create function json_object_delete returns string soname 'ha_connect';
create function json_object_list returns string soname 'ha_connect';
create function json_object_values returns string soname 'ha_connect';
create function jsonset_grp_size returns integer soname 'ha_connect';
create function jsonget_grp_size returns integer soname 'ha_connect';
create aggregate function json_array_grp returns string soname 'ha_connect';
create aggregate function json_object_grp returns string soname 'ha_connect';
create function jsonlocate returns string soname 'ha_connect';
create function json_locate_all returns string soname 'ha_connect';
create function jsoncontains returns integer soname 'ha_connect';
create function jsoncontains_path returns integer soname 'ha_connect';
create function json_item_merge returns string soname 'ha_connect';
create function json_get_item returns string soname 'ha_connect';
create function jsonget_string returns string soname 'ha_connect';
create function jsonget_int returns integer soname 'ha_connect';
create function jsonget_real returns real soname 'ha_connect';
create function json_set_item returns string soname 'ha_connect';
create function json_insert_item returns string soname 'ha_connect';
create function json_update_item returns string soname 'ha_connect';
create function json_file returns string soname 'ha_connect';
create function jfile_make returns string soname 'ha_connect';
create function jfile_convert returns string soname 'ha_connect';
create function jfile_bjson returns string soname 'ha_connect';
create function json_serialize returns string soname 'ha_connect';
create function jbin_array returns string soname 'ha_connect';
create function jbin_array_add_values returns string soname 'ha_connect';
create function jbin_array_add returns string soname 'ha_connect';
create function jbin_array_delete returns string soname 'ha_connect';
create function jbin_object returns string soname 'ha_connect';
create function jbin_object_nonull returns string soname 'ha_connect';
create function jbin_object_key returns string soname 'ha_connect';
create function jbin_object_add returns string soname 'ha_connect';
create function jbin_object_delete returns string soname 'ha_connect';
create function jbin_object_list returns string soname 'ha_connect';
create function jbin_item_merge returns string soname 'ha_connect';
create function jbin_get_item returns string soname 'ha_connect';
create function jbin_set_item returns string soname 'ha_connect';
create function jbin_insert_item returns string soname 'ha_connect';
create function jbin_update_item returns string soname 'ha_connect';
create function jbin_file returns string soname 'ha_connect';
```

On WIndows (from Connect 1.6):

```
create function jsonvalue returns string soname 'ha_connect';
create function json_make_array returns string soname 'ha_connect';
create function json_array_add_values returns string soname 'ha_connect';
create function json_array_add returns string soname 'ha_connect';
create function json_array_delete returns string soname 'ha_connect';
create function json_make_object returns string soname 'ha_connect';
create function json_object_nonull returns string soname 'ha_connect';
create function json_object_key returns string soname 'ha_connect';
create function json_object_add returns string soname 'ha_connect';
create function json_object_delete returns string soname 'ha_connect';
create function json_object_list returns string soname 'ha_connect';
create function jsonset_grp_size returns integer soname 'ha_connect';
create function jsonget_grp_size returns integer soname 'ha_connect';
create aggregate function json_array_grp returns string soname 'ha_connect';
create aggregate function json_object_grp returns string soname 'ha_connect';
create function jsonlocate returns string soname 'ha_connect';
create function json_locate_all returns string soname 'ha_connect';
create function jsoncontains returns integer soname 'ha_connect';
create function jsoncontains_path returns integer soname 'ha_connect';
create function json_item_merge returns string soname 'ha_connect';
create function json_get_item returns string soname 'ha_connect';
create function jsonget_string returns string soname 'ha_connect';
create function jsonget_int returns integer soname 'ha_connect';
create function jsonget_real returns real soname 'ha_connect';
create function json_set_item returns string soname 'ha_connect';
create function json_insert_item returns string soname 'ha_connect';
create function json_update_item returns string soname 'ha_connect';
create function json_file returns string soname 'ha_connect';
create function jfile_make returns string soname 'ha_connect';
create function json_serialize returns string soname 'ha_connect';
create function jbin_array returns string soname 'ha_connect';
create function jbin_array_add_values returns string soname 'ha_connect';
create function jbin_array_add returns string soname 'ha_connect';
create function jbin_array_delete returns string soname 'ha_connect';
create function jbin_object returns string soname 'ha_connect';
create function jbin_object_nonull returns string soname 'ha_connect';
create function jbin_object_key returns string soname 'ha_connect';
create function jbin_object_add returns string soname 'ha_connect';
create function jbin_object_delete returns string soname 'ha_connect';
create function jbin_object_list returns string soname 'ha_connect';
create function jbin_item_merge returns string soname 'ha_connect';
create function jbin_get_item returns string soname 'ha_connect';
create function jbin_set_item returns string soname 'ha_connect';
create function jbin_insert_item returns string soname 'ha_connect';
create function jbin_update_item returns string soname 'ha_connect';
create function jbin_file returns string soname 'ha_connect';
```

On WIndows (until Connect 1.5):

```
create function jsonvalue returns string soname 'ha_connect';
create function json_array returns string soname 'ha_connect';
create function json_array_add_values returns string soname 'ha_connect';
create function json_array_add returns string soname 'ha_connect';
create function json_array_delete returns string soname 'ha_connect';
create function json_object returns string soname 'ha_connect';
create function json_object_nonull returns string soname 'ha_connect';
create function json_object_key returns string soname 'ha_connect';
create function json_object_add returns string soname 'ha_connect';
create function json_object_delete returns string soname 'ha_connect';
create function json_object_list returns string soname 'ha_connect';
create function jsonset_grp_size returns integer soname 'ha_connect';
create function jsonget_grp_size returns integer soname 'ha_connect';
create aggregate function json_array_grp returns string soname 'ha_connect';
create aggregate function json_object_grp returns string soname 'ha_connect';
create function jsonlocate returns string soname 'ha_connect';
create function json_locate_all returns string soname 'ha_connect';
create function jsoncontains returns integer soname 'ha_connect';
create function jsoncontains_path returns integer soname 'ha_connect';
create function json_item_merge returns string soname 'ha_connect';
create function json_get_item returns string soname 'ha_connect';
create function jsonget_string returns string soname 'ha_connect';
create function jsonget_int returns integer soname 'ha_connect';
create function jsonget_real returns real soname 'ha_connect';
create function json_set_item returns string soname 'ha_connect';
create function json_insert_item returns string soname 'ha_connect';
create function json_update_item returns string soname 'ha_connect';
create function json_file returns string soname 'ha_connect';
create function jfile_make returns string soname 'ha_connect';
create function json_serialize returns string soname 'ha_connect';
create function jbin_array returns string soname 'ha_connect';
create function jbin_array_add_values returns string soname 'ha_connect';
create function jbin_array_add returns string soname 'ha_connect';
create function jbin_array_delete returns string soname 'ha_connect';
create function jbin_object returns string soname 'ha_connect';
create function jbin_object_nonull returns string soname 'ha_connect';
create function jbin_object_key returns string soname 'ha_connect';
create function jbin_object_add returns string soname 'ha_connect';
create function jbin_object_delete returns string soname 'ha_connect';
create function jbin_object_list returns string soname 'ha_connect';
create function jbin_item_merge returns string soname 'ha_connect';
create function jbin_get_item returns string soname 'ha_connect';
create function jbin_set_item returns string soname 'ha_connect';
create function jbin_insert_item returns string soname 'ha_connect';
create function jbin_update_item returns string soname 'ha_connect';
create function jbin_file returns string soname 'ha_connect';
```

Jfile_Bjson

MariaDB starting with 10.2.36

JFile_Bjson was introduced in MariaDB 10.5.9, MariaDB 10.4.18, MariaDB 10.3.28 and MariaDB 10.2.36.

```
Jfile_Bjson(in_file_name, out_file_name, lrecl)
```

Converts the first argument pretty=0 json file to Bjson file. B(inary)json is a pre-parsed json format. It is described below in the Performance chapter (available in next Connect versions).

Jfile_Convert

MariaDB starting with 10.2.36

JFile_Convert was introduced in MariaDB 10.5.9, MariaDB 10.4.18, MariaDB 10.3.28 and MariaDB 10.2.36.

```
Jfile_Convert(in_file_name, out_file_name, lrecl)
```

Converts the first argument json file to another *pretty=0* json file. The third integer argument is the record length to use. This is often required to process huge json files that would be very slow if they were in *pretty=2* format.

This is done without completely parsing the file, is very fast and requires no big memory.

Jfile_Make

MariaDB starting with 10.1.9

Jfile_Make was added in CONNECT 1.4 (from MariaDB 10.1.9).

```
Jfile_Make(arg1, arg2, [arg3], …)
```

The first argument must be a json item (if it is just a string, Jfile_Make will try its best to see if it is a json item or an input file name). The following arguments are a string file name and an integer pretty value (defaulting to 2) in any order. This function creates a json file containing the first argument item.

The returned string value is the created file name. If not specified as an argument, the file name can in some cases be retrieved from the first argument; in such cases the file itself is modified.

This function can be used to create or format a json file. For instance, supposing we want to format the file tb.json, this can be done with the query:

```
select Jfile_Make('tb.json' jfile_, 2);
```

The tb.json file will be changed to:

```
[
  {
    "_id": 5,
    "type": "food",
    "ratings": [
      5,
      8,
      9
    ]
  },
  {
    "_id": 6,
    "type": "car",
    "ratings": [
      5,
      9
    ]
  }
]
```

Json_Array_Add

```
Json_Array_Add(arg1, arg2, [arg3][, arg4][, ...])
```

Note: In CONNECT version 1.3 (before MariaDB 10.1.9), this function behaved like the new `Json_Array_Add_Values` function. The following describes this function for CONNECT version 1.4 (from MariaDB 10.1.9) only. The first argument must be a JSON array. The second argument is added as member of this array. For example:

```
select Json_Array_Add(Json_Array(56,3.1416,'machin',NULL),
'One more') Array;
```

Array
[56,3.141600,"machin",null,"One more"]

Note: The first array is not escaped, its (alias) name beginning with ‘json_’.

Now we can see how adding an author to the JSAMPLE2 table can alternatively be done:

```
update jsample2 set
  json_author = json_array_add(json_author, json_object('Charles' FIRSTNAME, 'Dickens' LASTNAME))
  where isbn = '9782840825685';
```

Note: Calling a column returning JSON a name prefixed by json_ (like json_author here) is good practice and removes the need to give it an alias to prevent escaping when used as an argument.

Additional arguments: If a third integer argument is given, it specifies the position (zero based) of the added value:

```
select Json_Array_Add('[5,3,8,7,9]' json_, 4, 2) Array;
```

Array
[5,3,4,8,7,9]

If a string argument is added, it specifies the Json path to the array to be modified. For instance:

```
select Json_Array_Add('{"a":1,"b":2,"c":[3,4]}' json_, 5, 1, 'c');
```

Json_Array_Add('{"a":1,"b":2,"c":[3, 4]}' json_, 5, 1, 'c')
{"a":1,"b":2,"c":[3,5,4]}

Json_Array_Add_Values

Json_Array_Add_Values added in CONNECT 1.4 replaces the function Json_Array_Add of CONNECT version 1.3 (before MariaDB 10.1.9).

```
Json_Array_Add_Values(arg, arglist)
```

The first argument must be a JSON array string. Then all other arguments are added as members of this array. For example:

```
select Json_Array_Add_Values
  (Json_Array(56, 3.1416, 'machin', NULL), 'One more', 'Two more') Array;
```

Array
[56,3.141600,"machin",null,"One more","Two more"]

Json_Array_Delete

```
Json_Array_Delete(arg1, arg2 [,arg3] [...])
```

The first argument should be a JSON array. The second argument is an integer indicating the rank (0 based conforming to general json usage) of the element to delete. For example:

```
select Json_Array_Delete(Json_Array(56,3.1416,'foo',NULL),1) Array;
```

Array
[56,"foo",null]

Now we can see how to delete the second author from the JSAMPLE2 table:

```
update jsample2 set json_author = json_array_delete(json_author, 1)
  where isbn = '9782840825685';
```

A Json path can be specified as a third string argument

Json_Array_Grp

```
Json_Array_Grp(arg)
```

This is an aggregate function that makes an array filled from values coming from the rows retrieved by a query. Let us suppose we have the pet table:

name	race	number
John	dog	2
Bill	cat	1
Mary	dog	1
Mary	cat	1
Lisbeth	rabbit	2
Kevin	cat	2
Kevin	bird	6
Donald	dog	1
Donald	fish	3

The query:

```
select name, json_array_grp(race) from pet group by name;
```

will return:

name	json_array_grp(race)
Bill	["cat"]
Donald	["dog","fish"]
John	["dog"]
Kevin	["cat","bird"]
Lisbeth	["rabbit"]
Mary	["dog","cat"]

One problem with the JSON aggregate functions is that they construct their result in memory and cannot know the needed amount of storage, not knowing the number of rows of the used table.

Therefore, the number of values for each group is limited. This limit is the value of JsonGrpSize whose default value is 10 but can be set using the JsonSet_Grp_Size function. Nevertheless, working on a larger table is possible, but only after setting JsonGrpSize to the ceiling of the number of rows per group for the table. Try not to set it to a very large value to avoid memory exhaustion.

JsonContains

```
JsonContains(json_doc, item [, int])<
```

This function can be used to check whether an item is contained in a document. Its arguments are the same than the ones of the JsonLocate function; only the return value changes. The integer returned value is 1 is the item is contained in the document or 0 otherwise.

JsonContains_Path

```
JsonContains_Path(json_doc, path)
```

This function can be used to check whether a Json path is contained in the document. The integer returned value is 1 is the path is contained in the document or 0 otherwise.

Json_File

```
Json_File(arg1, [arg2, [arg3]], …)
```

The first argument must be a file name. This function returns the text of the file that is supposed to be a json file. If only one argument is specified, the file text is returned without being parsed. Up to two additional arguments can be specified:

A string argument is the path to the sub-item to be returned. An integer argument specifies the pretty format value of the file.

This function is chiefly used to get the json item argument of other json functions from a json file. For instance, supposing the file tb.json is:

```
{ "_id" : 5, "type" : "food", "ratings" : [ 5, 8, 9 ] }
{ "_id" : 6, "type" : "car", "ratings" : [ 5, 9 ] }
```

Extracting a value from it can be done with a query such as:

```
select JsonGet_String(Json_File('tb.json', 0), '[1]:type') "Type";
```

or, from MariaDB 10.2.8:

```
select JsonGet_String(Json_File('tb.json', 0), '$[1].type') "Type";
```

This query returns:

Type
car

However, we'll see that, most of the time, it is better to use Jbin_File or to directly specify the file name in queries. In particular this function should not be used for queries that must modify the json item because, even if the modified json is returned, the file itself would be unchanged.

Json_Get_Item

MariaDB starting with 10.1.9

Json_Get_Item was added in CONNECT 1.4 (from MariaDB 10.1.9).

```
Json_Get_Item(arg1, arg2, …)
```

This function returns a subset of the json document passed as first argument. The second argument is the json path of the item to be returned and should be one returning a json item (terminated by a '*'). If not, the function will try to make it right but this is not foolproof. For instance:

```
select Json_Get_Item(Json_Object('foo' as "first", Json_Array('a', 33)
  as "json_second"), 'second') as "item";
```

The correct path should have been 'second:*' (or from MariaDB 10.2.8, 'second.*'), but in this simple case the function was able to make it right. The returned item:

item
["a",33]

Note: The array is aliased "json_second" to indicate it is a json item and avoid escaping it. However, the "json_" prefix is skipped when making the object and must not be added to the path.

JsonGet_Grp_Size

```
JsonGet_Grp_Size(val)
```

This function returns the JsonGrpSize value.

JsonGet_String / JsonGet_Int / JsonGet_Real

MariaDB starting with 10.1.9

JsonGet_String, JsonGet_Int and JsonGet_Real were added in CONNECT 1.4 (from MariaDB 10.1.9).

```
JsonGet_String(arg1, arg2, [arg3] …)
JsonGet_Int(arg1, arg2, [arg3] …)
JsonGet_Real(arg1, arg2, [arg3] …)
```

The first argument should be a JSON item. If it is a string with no alias, it will be converted as a json item. The second argument is the path of the item to be located in the first argument and returned, eventually converted according to the used function. For example:

```
select
JsonGet_String('{"qty":7,"price":29.50,"garanty":null}','price') "String",
JsonGet_Int('{"qty":7,"price":29.50,"garanty":null}','price') "Int",
JsonGet_Real('{"qty":7,"price":29.50,"garanty":null}','price') "Real";
```

This query returns:

String	Int	Real
29.50	29	29.500000000000000

The function *JsonGet_Real* can be given a third argument to specify the number of decimal digits of the returned value. For instance:

```
select
JsonGet_Real('{"qty":7,"price":29.50,"garanty":null}','price',4) "Real";
```

This query returns:

String
29.50

The given path can specify all operators for arrays except the “expand” [X] operator (or from MariaDB 10.2.8, the“expand” [*] operator). For instance:

```
select
JsonGet_Int(Json_Array(45,28,36,45,89), '[4]') "Rank",
JsonGet_Int(Json_Array(45,28,36,45,89), '[#]') "Number",
JsonGet_String(Json_Array(45,28,36,45,89), '[","]') "Concat",
JsonGet_Int(Json_Array(45,28,36,45,89), '[+]') "Sum",
JsonGet_Real(Json_Array(45,28,36,45,89), '[!]', 2) "Avg";
```

The result:

Rank	Number	Concat	Sum	Avg
89	5	45,28,36,45,89	243	48.60

Json_Item_Merge

```
Json_Item_Merge(arg1, arg2, …)
```

This function merges two arrays or two objects. For arrays, this is done by adding to the first array all the values of the second array. For instance:

```
select Json_Item_Merge(Json_Array('a','b','c'), Json_Array('d','e','f')) as "Result";
```

The function returns:

Result
["a","b","c","d","e","f"]

For objects, the pairs of the second object are added to the first object if the key does not yet exist in it; otherwise the pair of the first object is set with the value of the matching pair of the second object. For instance:

```
select Json_Item_Merge(Json_Object(1 "a", 2 "b", 3 "c"), Json_Object(4 "d",5 "b",6 "f"))
  as "Result";
```

The function returns:

Result

```
{"a":1,"b":5,"c":3,"d":4,"f":6}
```

JsonLocate

```
JsonLocate(arg1, arg2, [arg3], …):
```

The first argument must be a JSON tree. The second argument is the item to be located. The item to be located can be a constant or a json item. Constant values must be equal in type and value to be found. This is "shallow equality" – strings, integers and doubles won't match.

This function returns the json path to the located item or null if it is not found. For example:

```
select JsonLocate('{"AUTHORS":[{"FN":"Jules", "LN":"Verne"},
  {"FN":"Jack", "LN":"London"}]}' json_, 'Jack') Path;
```

This query returns:

Path
AUTHORS:[1]:FN

or, from MariaDB 10.2.8:

Path
$.AUTHORS[1].FN

The path syntax is the same used in JSON CONNECT tables.

By default, the path of the first occurrence of the item is returned. The third parameter can be used to specify the occurrence whose path is to be returned. For instance:

```
select
JsonLocate('[45,28,[36,45],89]',45) first,
JsonLocate('[45,28,[36,45],89]',45,2) second,
JsonLocate('[45,28,[36,45],89]',45.0) `wrong type`,
JsonLocate('[45,28,[36,45],89]','[36,45]' json_) json;
```

first	second	wrong type	json
[0]	[2]:[1]	<null>	[2]

or, from MariaDB 10.2.8:

first	second	wrong type	json
$[0]	$[2][1]	<null>	$[2]

For string items, the comparison is case sensitive by default. However, it is possible to specify a string to be compared case insensitively by giving it an alias beginning by “ci”:

```
select JsonLocate('{"AUTHORS":[{"FN":"Jules", "LN":"Verne"},
  {"FN":"Jack", "LN":"London"}]}' json_, 'VERNE' ci) Path;
```

Path
AUTHORS:[0]:LN

or, from MariaDB 10.2.8:

Path
$.AUTHORS[0].LN

Json_Locate_All

```
Json_Locate_All(arg1, arg2, [arg3], …):
```

The first argument must be a JSON item. The second argument is the item to be located. This function returns the paths to all locations of the item as an array of strings. For example:

```
select Json_Locate_All('[[45,28],[[36,45],89]]',45);
```

This query returns:

All paths
["[0]:[0]","[1]:[0]:[1]"]

or, from MariaDB 10.2.8:

All paths
["$[0][0]","$[1][0][1]"]

The returned array can be applied other functions. For instance, to get the number of occurrences of an item in a json tree, you can do:

```
select JsonGet_Int(Json_Locate_All('[[45,28],[[36,45],89]]',45), '[#]') "Nb of occurs";
```

or, from MariaDB 10.2.8:

```
select JsonGet_Int(Json_Locate_All('[[45,28],[[36,45],89]]',45), '$[#]') "Nb of occurs";
```

The displayed result:

Nb of occurs
2

If specified, the third integer argument set the depth to search in the document. This means the maximum items in the paths (until MariaDB 10.2.7, the number of ':' separator characters in them plus one). This value defaults to 10 but can be increased for complex documents or reduced to set the maximum wanted depth of the returned paths.

Json_Make_Array

```
Json_Make_Array(val1, …, valn)
```

This function was named "Json_Array" in previous versions of CONNECT. It was renamed because MariaDB 10.2 features native JSON functions including a Json_Array function. The native function does almost the same as the UDF one, but does not accept CONNECT-specific arguments such as the result from JBIN functions.

Json_Make_Array returns a string denoting a JSON array with all its arguments as members. For example:

```
select Json_Make_Array(56, 3.1416, 'My name is "Foo"', NULL);
```

Json_Make_Array(56, 3.1416, 'My name is "Foo"',N ULL)
[56,3.141600,"My name is \"Foo\"",null]

Note: The argument list can be void. If so, a void array is returned.

This function was named "Json_Array" in previous versions of CONNECT. It was renamed because MariaDB 10.2 features native JSON functions including a "Json_Array" function. The native function does almost the same as the UDF one but does not accept CONNECT specific arguments such as the result from JBIN functions.

Json_Make_Object

```
Json_Make_Object(arg1, …, argn)
```

This function was named "Json_Object" in previous versions of CONNECT. It was renamed because MariaDB 10.2 features native JSON functions including a Json_Object function. The native function does what the UDF Json_Object_Key does.

Json_Make_Object returns a string denoting a JSON object. For instance:

```
select Json_Make_Object(56, 3.1416, 'machin', NULL);
```

The object is filled with pairs corresponding to the given arguments. The key of each pair is made from the argument (default or specified) alias.

Json_Make_Object(56, 3.1416, 'machin', NULL)
{"56":56,"3.1416":3.141600,"machin":"machin","NULL":null}

When needed, it is possible to specify the keys by giving an alias to the arguments:

```
select Json_Make_Object(56 qty, 3.1416 price, 'machin' truc, NULL garanty);
```

Json_Make_Object(56 qty,3.1416 price,'machin' truc, NULL garanty)
{"qty":56,"price":3.141600,"truc":"machin","garanty":null}

If the alias is prefixed by 'json_' (to prevent escaping) the key name is stripped from that prefix.

This function is chiefly useful when entering values retrieved from a table, the key being by default the column name:

```
select Json_Make_Object(matricule, nom, titre, salaire) from connect.employe where nom = 'PANTIER
```

Json_Make_Object(matricule, nom, titre, salaire)
{"matricule":40567,"nom":"PANTIER","titre":"DIRECTEUR","salaire":14000.000000}

This function was named "Json_Object" in previous versions of CONNECT. It was renamed because MariaDB 10.2 features native JSON functions including a "Json_Object" function. The native function does what the UDF Json_Object_Key does.

Json_Object_Add

```
Json_Object_Add(arg1, arg2, [arg3] …)
```

The first argument must be a JSON object. The second argument is added as a pair to this object. For example:

```
select Json_Object_Add
  ('{"item":"T-shirt","qty":27,"price":24.99}' json_old,'blue' color) newobj;
```

newobj
{"item":"T-shirt","qty":27,"price":24.990000,"color":"blue"}

Note: If the specified key already exists in the object, its value is replaced by the new one.

The third string argument is a Json path to the target object.

Json_Object_Delete

```
Json_Object_Delete(arg1, arg2, [arg3] …):
```

The first argument must be a JSON object. The second argument is the key of the pair to delete. For example:

```
select Json_Object_Delete('{"item":"T-shirt","qty":27,"price":24.99}' json_old, 'qty') newobj;
```

newobj
{"item":"T-shirt","price":24.99}

The third string argument is a Json path to the object to be the target of deletion.

Json_Object_Grp

```
Json_Object_Grp(arg1,arg2)
```

This function works like Json_Array_Grp. It makes a JSON object filled with value pairs whose keys are passed from its first argument and values are passed from its second argument.

This can be seen with the query:

```
select name, json_object_grp(number,race) from pet group by name;
```

This query returns:

name	json_object_grp(number,race)
Bill	{"cat":1}
Donald	{"dog":1,"fish":3}
John	{"dog":2}
Kevin	{"cat":2,"bird":6}
Lisbeth	{"rabbit":2}
Mary	{"dog":1,"cat":1}

Json_Object_Key

```
Json_Object_Key([key1, val1 [, …, keyn, valn]])
```

Return a string denoting a JSON object. For instance:

```
select Json_Object_Key('qty', 56, 'price', 3.1416, 'truc', 'machin', 'garanty', NULL);
```

The object is filled with pairs made from each key/value arguments.

Json_Object_Key('qty', 56, 'price', 3.1416, 'truc', 'machin', 'garanty', NULL)
{"qty":56,"price":3.141600,"truc":"machin","garanty":null}

Json_Object_List

```
Json_Object_List(arg1, …):
```

The first argument must be a JSON object. This function returns an array containing the list of all keys existing in the object. For example:

```
select Json_Object_List(Json_Object(56 qty,3.1416 price,'machin' truc, NULL garanty))
  "Key List";
```

Key List
["qty","price","truc","garanty"]

Json_Object_Nonull

```
Json_Object_Nonull(arg1, …, argn)
```

This function works like Json_Make_Object but “null” arguments are ignored and not inserted in the object. Arguments are regarded as “null” if they are JSON null values, void arrays or objects, or arrays or objects containing only null members.

It is mainly used to avoid constructing useless null items when converting tables (see later).

Json_Object_Values

```
Json_Object_Values(json_object)
```

The first argument must be a JSON object. This function returns an array containing the list of all values existing in the object. For example:

```
select Json_Object_Values('{"One":1,"Two":2,"Three":3}') "Value List";
```

Value List
[1,2,3]

JsonSet_Grp_Size

```
JsonSet_Grp_Size(val)
```

This function is used to set the JsonGrpSize value. This value is used by the following aggregate functions as a ceiling value of the number of items in each group. It returns the JsonGrpSize value that can be its default value when passed 0 as argument.

Json_Set_Item / Json_Insert_Item / Json_Update_Item

```
Json_{Set | Insert | Update}_Item(json_doc, [item, path [, val, path …]])
```

These functions insert or update data in a JSON document and return the result. The value/path pairs are evaluated left to right. The document produced by evaluating one pair becomes the new value against which the next pair is evaluated.

- Json_Set_Item replaces existing values and adds non-existing values.
- Json_Insert_Item inserts values without replacing existing values.
- Json_Update_Item replaces only existing values.

Example:

```
set @j = Json_Array(1, 2, 3, Json_Object_Key('quatre', 4));
select Json_Set_Item(@j, 'foo', '[1]', 5, '[3]:cinq') as "Set",
Json_Insert_Item(@j, 'foo', '[1]', 5, '[3]:cinq') as "Insert",
Json_Update_Item(@j, 'foo', '[1]', 5, '[3]:cinq') as "Update";
```

or, from MariaDB 10.2.8:

```
set @j = Json_Array(1, 2, 3, Json_Object_Key('quatre', 4));
select Json_Set_Item(@j, 'foo', '$[1]', 5, '$[3].cinq') as "Set",
Json_Insert_Item(@j, 'foo', '$[1]', 5, '$[3].cinq') as "Insert",
Json_Update_Item(@j, 'foo', '$[1]', 5, '$[3].cinq') as "Update";
```

This query returns:

Set	Insert	Update
[1,"foo",3,{"quatre":4,"cinq":5}]	[1,2,3,{"quatre":4,"cinq":5}]	[1,"foo",3,{"quatre":4}]

JsonValue

```
JsonValue (val)
```

Returns a JSON value as a string, for instance:

```
select JsonValue(3.1416);
```

JsonValue(3.1416)
3.141600

Before MariaDB 10.1.9, this function was called Json_Value, but was renamed to avoid clashing with the JSON_VALUE

function.

The “JBIN” return type

Almost all functions returning a json string - whose name begins with *Json_* - have a counterpart with a name beginning with *Jbin_*. This is both for performance (speed and memory) as well as for better control of what the functions should do.

This is due to the way CONNECT UDFs work internally. The Json functions, when receiving json strings as parameters, parse them and construct a binary tree in memory. They work on this tree and before returning; serialize this tree to return a new json string.

If the json document is large, this can take up a large amount of time and storage space. It is all right when one simple json function is called – it must be done anyway – but is a waste of time and memory when json functions are used as parameters to other json functions.

To avoid multiple serializing and parsing, the Jbin functions should be used as parameters to other functions. Indeed, they do not serialize the memory document tree, but return a structure allowing the receiving function to have direct access to the memory tree. This saves the serialize-parse steps otherwise needed to pass the argument and removes the need to reallocate the memory of the binary tree, which by the way is 6 to 7 times the size of the json string. For instance:

```
select Json_Object(Jbin_Array_Add(Jbin_Array('a','b','c'), 'd') as "Jbin_foo") as "Result";
```

This query returns:

Result
{"foo":["a","b","c","d"]}

Here the binary json tree allocated by *Jbin_Array* is completed by *Jbin_Array_Add* and *Json_Object* and serialized only once to make the final result string. It would be serialized and parsed two more times if using “Json” functions.

Note that Jbin results are recognized as such because they are aliased beginning with “Jbin_”. This is why in the *Json_Object* function the alias is specified as “Jbin_foo”.

What happens if it is not recognized as such? These functions are declared as returning a string and to take care of this, the returned structure begins with a zero-terminated string. For instance:

```
select Jbin_Array('a','b','c');
```

This query replies:

Jbin_Array('a','b','c')
Binary Json array

Note: When testing, the tree returned by a “Jbin” function can be seen using the *Json_Serialize* function whose unique parameter must be a “Jbin” result. For instance:

```
select Json_Serialize(Jbin_Array('a','b','c'));
```

This query returns:

Json_Serialize(Jbin_Array('a','b','c'))
["a","b","c"]

Note: For this simple example, this is equivalent to using the *Json_Array* function.

Using a file as json UDF first argument

We have seen that many json UDFs can have an additional argument not yet described. This is in the case where the json item argument was referring to a file. Then the additional integer argument is the pretty value of the json file. It matters only when the first argument is just a file name (to make the UDF understand this argument is a file name, it should be aliased with a name beginning with jfile_) or if the function modifies the file, in which case it will be rewritten with this pretty format.

The json item is created by extracting the required part from the file. This can be the whole file but more often only some of it. There are two ways to specify the sub-item of the file to be used:

1. Specifying it in the *Json_File* or *Jbin_File* arguments.
2. Specifying it in the receiving function (not possible for all functions).

It doesn't make any difference when the *Jbin_File* is used but it does with *Json_File*. For instance:

```
select Jfile_Make('{"a":1, "b":[44, 55]}' json_, 'test.json');
select Json_Array_Add(Json_File('test.json', 'b'), 66);
```

The second query returns:

Json_Array_Add(Json_File('test.json', 'b'), 66)
[44,55,66]

It just returns the – modified -- subset returned by the Json_File function, while the query:

```
select Json_Array_Add(Json_File('test.json'), 66, 'b');
```

returns what was received from *Json_File* with the modification made on the subset.

Json_Array_Add(Json_File('test.json'), 66, 'b')
{"a":1,"b":[44,55,66]}

Note that in both case the test.json file is not modified. This is because the *Json_File* function returns a string representing all or part of the file text but no information about the file name. This is all right to check what would be the effect of the modification to the file.

However, to have the file modified, use the *Jbin_File* function or directly give the file name. *Jbin_File* returns a structure containing the file name, a pointer to the file parsed tree and eventually a pointer to the subset when a path is given as a second argument:

```
select Json_Array_Add(Jbin_File('test.json', 'b'), 66);
```

This query returns:

Json_Array_Add(Jbin_File('test.json', 'b'), 66)
test.json

This time the file is modified. This can be checked with:

```
select Json_File('test.json', 3);
```

Json_File('test.json', 3)
{"a":1,"b":[44,55,66]}

The reason why the first argument is returned by such a query is because of tables such as:

```
create table tb (
n int key,
jfile_cols char(10) not null);
insert into tb values(1,'test.json');
```

In this table, the *jfile_cols* column just contains a file name. If we update it by:

```
update tb set jfile_cols = select Json_Array_Add(Jbin_File('test.json', 'b'), 66)
where n = 1;
```

This is the test.json file that must be modified, not the jfile_cols column. This can be checked by:

```
select JsonGet_String(jfile_cols, '[1]:*') from tb;
```

JsonGet_String(jfile_cols, '[1]:*')
{"a":1,"b":[44,55,66]}

Note: It was an important facility to name the second column of the table beginning by "jfile_" so the json functions knew it was a file name without obliging to specify an alias in the queries.

Using “Jbin” to control what the query execution does

This is applying in particular when acting on json files. We have seen that a file was not modified when using the *Json_File* function as an argument to a modifying function because the modifying function just received a copy of the json file. This is not true when using the *Jbin_File* function that does not serialize the binary document and make it directly accessible. Also, as we have seen earlier, json functions that modify their first file parameter modify the file and return the file name. This is done by directly serializing the internal binary document as a file.

However, the “Jbin” counterpart of these functions does not serialize the binary document and thus does not modify the json file. For example let us compare these two queries:

/* First query */

```
select Json_Object(Jbin_Object_Add(Jbin_File('bt2.json'), 4 as "d") as "Jbin_bt1")
  as "Result";
```

/* Second query */

```
select Json_Object(Json_Object_Add(Jbin_File('bt2.json'), 4 as "d") as "Jfile_bt1")
  as "Result";
```

Both queries return:

Result
{"bt1":{"a":1,"b":2,"c":3,"d":4}}

In the first query *Jbin_Object_Add* does not serialize the document (no “Jbin” functions do) and *Json_Object* just returns a serialized modified tree. Consequently, the file bt2.json is not modified. This query is all right to copy a modified version of the json file without modifying it.

However, in the second query *Json_Object_Add* does modify the json file and returns the file name. The *Json_Object* function receives this file name, reads and parses the file, makes an object from it and returns the serialized result. This modification can be done willingly but can be an unwanted side effect of the query.

Therefore, using “Jbin” argument functions, in addition to being faster and using less memory, are also safer when dealing with json files that should not be modified.

Using JSON as Dynamic Columns

The JSON nosql language has all the features to be used as an alternative to dynamic columns. For instance, take the following example of dynamic columns:

```
create table assets (
   item_name varchar(32) primary key, /* A common attribute for all items */
   dynamic_cols  blob  /* Dynamic columns will be stored here */
 );

INSERT INTO assets VALUES
   ('MariaDB T-shirt', COLUMN_CREATE('color', 'blue', 'size', 'XL'));

INSERT INTO assets VALUES
   ('Thinkpad Laptop', COLUMN_CREATE('color', 'black', 'price', 500));

SELECT item_name, COLUMN_GET(dynamic_cols, 'color' as char) AS color FROM assets;
+-----------------+-------+
| item_name       | color |
+-----------------+-------+
| MariaDB T-shirt | blue  |
| Thinkpad Laptop | black |
+-----------------+-------+
```

/* Remove a column: */

```
UPDATE assets SET dynamic_cols=COLUMN_DELETE(dynamic_cols, "price")
  WHERE COLUMN_GET(dynamic_cols, 'color' as char)='black';
```

/* Add a column: */

```
UPDATE assets SET dynamic_cols=COLUMN_ADD(dynamic_cols, 'warranty', '3 years')
   WHERE item_name='Thinkpad Laptop';
```

/* You can also list all columns, or get them together with their values in JSON format: */

```
SELECT item_name, column_list(dynamic_cols) FROM assets;
+----------------+--------------------------+
| item_name      | column_list(dynamic_cols) |
+----------------+--------------------------+
| MariaDB T-shirt | `size`,`color`           |
| Thinkpad Laptop | `color`,`warranty`       |
+----------------+--------------------------+

SELECT item_name, COLUMN_JSON(dynamic_cols) FROM assets;
+----------------+----------------------------------------+
| item_name      | COLUMN_JSON(dynamic_cols)              |
+----------------+----------------------------------------+
| MariaDB T-shirt | {"size":"XL","color":"blue"}          |
| Thinkpad Laptop | {"color":"black","warranty":"3 years"} |
+----------------+----------------------------------------+
```

The same result can be obtained with json columns using the json UDF's:

/* JSON equivalent */

```
create table jassets (
   item_name varchar(32) primary key, /* A common attribute for all items */
   json_cols varchar(512)  /* Jason columns will be stored here */
 );

INSERT INTO jassets VALUES
   ('MariaDB T-shirt', Json_Object('blue' color, 'XL' size));

INSERT INTO jassets VALUES
   ('Thinkpad Laptop', Json_Object('black' color, 500 price));

SELECT item_name, JsonGet_String(json_cols, 'color') AS color FROM jassets;
+----------------+-------+
| item_name      | color |
+----------------+-------+
| MariaDB T-shirt | blue  |
| Thinkpad Laptop | black |
+----------------+-------+
```

/* Remove a column: */

```
UPDATE jassets SET json_cols=Json_Object_Delete(json_cols, 'price')
 WHERE JsonGet_String(json_cols, 'color')='black';
```

/* Add a column */

```
UPDATE jassets SET json_cols=Json_Object_Add(json_cols, '3 years' warranty)
 WHERE item_name='Thinkpad Laptop';
```

/* You can also list all columns, or get them together with their values in JSON format: */

```
SELECT item_name, Json_Object_List(json_cols) FROM jassets;
+----------------+-----------------------------+
| item_name      | Json_Object_List(json_cols) |
+----------------+-----------------------------+
| MariaDB T-shirt | ["color","size"]           |
| Thinkpad Laptop | ["color","warranty"]       |
+----------------+-----------------------------+

SELECT item_name, json_cols FROM jassets;
+----------------+---------------------------------------+
| item_name      | json_cols                             |
+----------------+---------------------------------------+
| MariaDB T-shirt | {"color":"blue","size":"XL"}          |
| Thinkpad Laptop | {"color":"black","warranty":"3 years"} |
+----------------+---------------------------------------+
```

However, using JSON brings features not existing in dynamic columns:

- Use of a language used by many implementation and developers.
- Full support of arrays, currently missing from dynamic columns.
- Access of subpart of json by JPATH that can include calculations on arrays.
- Possible references to json files.

With more experience, additional UDFs can be easily written to support new needs.

New Set of BSON Functions

All these functions have been rewritten using the new JSON handling way and are temporarily available changing the J starting name to B. Then Json_Make_Array new style is called using Bson_Make_Array. Some, such as Bson_Item_Delete, are new and some fix bugs found in their Json counterpart.

Converting Tables to JSON

The JSON UDF's and the direct Jpath "*" facility are powerful tools to convert table and files to the JSON format. For instance, the file `biblio3.json` we used previously can be obtained by converting the `xsample.xml file` . This can be done like this:

From Connect 1.07.0002

```
create table xj1 (row varchar(500) jpath='*') engine=connect table_type=JSON
file_name='biblio3.json' option_list='jmode=2';
```

Before Connect 1.07.0002

```
create table xj1 (row varchar(500) field_format='*')
 engine=connect table_type=JSON file_name='biblio3.json' option_list='jmode=2';
```

And then :

```
insert into xj1
  select json_object_nonull(ISBN, language LANG, SUBJECT,
    json_array_grp(json_object(authorfn FIRSTNAME, authorln LASTNAME)) json_AUTHOR, TITLE,
    json_object(translated PREFIX, json_object(tranfn FIRSTNAME, tranln LASTNAME) json_TRANSLATOR
    json_TRANSLATED, json_object(publisher NAME, location PLACE) json_PUBLISHER, date DATEPUB)
from xsampall2 group by isbn;
```

The xj1 table rows will directly receive the Json object made by the select statement used in the insert statement and the table file will be made as shown (xj1 is pretty=2 by default) Its mode is Jmode=2 because the values inserted are strings even if they denote json objects.

Another way to do this is to create a table describing the file format we want before the `biblio3.json` file existed:

From Connect 1.07.0002

```
create table jsampall3 (
ISBN char(15),
LANGUAGE char(2) jpath='LANG',
SUBJECT char(32),
AUTHORFN char(128) jpath='AUTHOR:[X]:FIRSTNAME',
AUTHORLN char(128) jpath='AUTHOR:[X]:LASTNAME',
TITLE char(32),
TRANSLATED char(32) jpath='TRANSLATOR:PREFIX',
TRANSLATORFN char(128) jpath='TRANSLATOR:FIRSTNAME',
TRANSLATORLN char(128) jpath='TRANSLATOR:LASTNAME',
PUBLISHER char(20) jpath='PUBLISHER:NAME',
LOCATION char(20) jpath='PUBLISHER:PLACE',
DATE int(4) jpath='DATEPUB')
engine=CONNECT table_type=JSON file_name='biblio3.json';
```

Before Connect 1.07.0002

```
create table jsampall3 (
ISBN char(15),
LANGUAGE char(2) field_format='LANG',
SUBJECT char(32),
AUTHORFN char(128) field_format='AUTHOR:[X]:FIRSTNAME',
AUTHORLN char(128) field_format='AUTHOR:[X]:LASTNAME',
TITLE char(32),
TRANSLATED char(32) field_format='TRANSLATOR:PREFIX',
TRANSLATORFN char(128) field_format='TRANSLATOR:FIRSTNAME',
TRANSLATORLN char(128) field_format='TRANSLATOR:LASTNAME',
PUBLISHER char(20) field_format='PUBLISHER:NAME',
LOCATION char(20) field_format='PUBLISHER:PLACE',
DATE int(4) field_format='DATEPUB')
engine=CONNECT table_type=JSON file_name='biblio3.json';
```

and to populate it by:

```
insert into jsampall3 select * from xsampall;
```

This is a simpler method. However, the issue is that this method cannot handle the multiple column values. This is why we inserted from `xsampall` not from `xsampall2`. How can we add the missing multiple authors in this table? Here again we must create a utility table able to handle JSON strings. From Connect 1.07.0002

```
create table xj2 (ISBN char(15), author varchar(150) jpath='AUTHOR:*') engine=connect
table_type=JSON file_name='biblio3.json' option_list='jmode=1';
```

Before Connect 1.07.0002

```
create table xj2 (ISBN char(15), author varchar(150) field_format='AUTHOR:*')
  engine=connect table_type=JSON file_name='biblio3.json' option_list='jmode=1';
```

```
update xj2 set author =
(select json_array_grp(json_object(authorfn FIRSTNAME, authorln LASTNAME))
  from xsampall2 where isbn = xj2.isbn);
```

Voilà !

Converting json files

We have seen that json files can be formatted differently depending on the pretty option. In particular, big data files should be formatted with pretty equal to 0 when used by a CONNECT json table. The best and simplest way to convert a file from one format to another is to use the *Jfile_Make* function. Indeed this function makes a file of specified format using the syntax:

```
Jfile_Make(json_document, [file_name], [pretty]);
```

The file name is optional when the json document comes from a Jbin_File function because the returned structure makes it available. For instance, to convert back the json file tb.json to pretty= 0, this can be simply done by:

```
select Jfile_Make(Jbin_File('tb.json'), 0);
```

Performance Consideration

MySQL and PostgreSQL have a JSON data type that is not just text but an internal encoding of JSON data. This is to save parsing time when executing JSON functions. Of course, the parse must be done anyway when creating the data and serializing must be done to output the result.

CONNECT directly works on character strings impersonating JSON values with the need of parsing them all the time but with the advantage of working easily on external data. Generally, this is not too penalizing because JSON data are often of some or reasonable size. The only case where it can be a serious problem is when working on a big JSON file.

Then, the file should be formatted or converted to *pretty=0*.

From Connect 1.7.002, this easily done using the Jfile_Convert function, for instance:

```
select jfile_convert('bibdoc.json','bibdoc0.json',350);
```

Such a json file should not be used directly by JSON UDFs because they parse the whole file, even when only a subset is used. Instead, it should be used by a JSON table created on it. Indeed, JSON tables do not parse the whole document but just the item corresponding to the row they are working on. In addition, indexing can be used by the table as explained previously on this page.

Generally speaking, the maximum flexibility offered by CONNECT is by using JSON tables and JSON UDFs together. Some things are better handled by tables, other by UDFs. The tools are there but it is up to you to discover the best way to resolve your problems.

Bjson files

Starting with Connect 1.7.002, *pretty=0* json files can be converted to a binary format that is a pre-parsed representation of json. This can be done with the Jfile_Bjson UDF function, for instance:

```
select jfile_bjson('bigfile.json','binfile.json',3500);
```

Here the third argument, the record length, must 6 to 10 times larger than the lrecl of the initial json file because the parsed representation is bigger than the original json text representation.

Tables using such Bjson files must specify ‘Pretty=-1’ in the option list.

It is probably similar to the BSON used by MongoDB and PostgreSQL and permits to process queries up to 10 times faster than working on text json files. Indexing is also available for tables using this format making even more performance improvement. For instance, some queries on a json table of half a million rows, that were previously done in more than 10 seconds, took only 0.1 second when converted and indexed.

Here again, this has been remade to use the new way Json is handled. The files made using the bfile_bjson function are only from two to four times the size of the source files. This new representation is not compatible with the old one. Therefore, these files must be used with BSON tables only.

Specifying a JSON table Encoding

An important feature of JSON is that strings should in UNICODE. As a matter of fact, all examples we have found on the Internet seemed to be just ASCII. This is because UNICODE is generally encoded in JSON files using UTF8 or UTF16 or UTF32.

To specify the required encoding, just use the data_charset CONNECT option or the native DEFAULT CHARSET option.

Retrieving JSON data from MongoDB

Classified as a NoSQL database program, MongoDB uses JSON-like documents (BSON) grouped in collections. The simplest way, and only method available before Connect 1.6, to access MongoDB data was to export a collection to a JSON file. This produces a file having the pretty=0 format. Viewed as SQL, a collection is a table and documents are table rows.

Since CONNECT version 1.6, it is now possible to directly access MongoDB collections via their MongoDB C Driver. This is the purpose of the MONGO table type described later. However, JSON tables can also do it in a somewhat different way (providing MONGO support is installed as described for MONGO tables).

It is achieved by specifying the MongoDB connection URI while creating the table. For instance:

From Connect 1.7.002

```
create or replace table jinvent (
_id char(24) not null,
item char(12) not null,
instock varchar(300) not null jpath='instock.*')
engine=connect table_type=JSON tabname='inventory' lrecl=512
connection='mongodb://localhost:27017';
```

Before Connect 1.7.002

```
create or replace table jinvent (
_id char(24) not null,
item char(12) not null,
instock varchar(300) not null field_format='instock.*')
engine=connect table_type=JSON tabname='inventory' lrecl=512
connection='mongodb://localhost:27017';
```

In this statement, the *file_name* option was replaced by the *connection* option. It is the URI enabling to retrieve data from a local or remote MongoDB server. The *tabname* option is the name of the MongoDB collection that will be used and the *dbname* option could have been used to indicate the database containing the collection (it defaults to the current database).

The way it works is that the documents retrieved from MongoDB are serialized and CONNECT uses them as if they were read from a file. This implies serializing by MongoDB and parsing by CONNECT and is not the best performance wise. CONNECT tries its best to reduce the data transfer when a query contains a reduced column list and/or a where clause. This way makes all the possibilities of the JSON table type available, such as calculated arrays.

However, to work on large JSON collations, using the MONGO table type is generally the normal way.

Note: JSON tables using the MongoDB access accept the specific MONGO options colist, filter and pipeline. They are described in the MONGO table chapter.

Summary of Options and Variables Used with Json Tables

Options and variables that can be used when creating Json tables are listed here:

Table Option	Type	Description
ENGINE	String	Must be specified as CONNECT.
TABLE_TYPE	String	Must be JSON or BSON.
FILE_NAME	String	The optional file (path) name of the Json file. Can be absolute or relative to the current data directory. If not specified, it defaults to the table name and json file type.
DATA_CHARSET	String	Set it to ‘utf8’ for most Unicode Json documents.
LRECL	Number	The file record size for pretty < 2 json files.
HTTP	String	The HTTP of the server of REST queries.
URI	String	THE URI of REST queries
CONNECTION*	String	Specifies a connection to `MONGODB` .
ZIPPED	Boolean	True if the json file(s) is/are zipped in one or several zip files.
MULTIPLE	Number	Used to specify a multiple file table.
SEP_CHAR	String	Set it to ‘:’ for old tables using the old json path syntax.
CATFUNC	String	The catalog function (column) used when creating a catalog table.
OPTION_LIST	String	Used to specify all other options listed below.

(*) For Json tables connected to MongoDB, Mongo specific options can also be used.

Other options must be specified in the option list:

Table Option	Type	Description

DEPTH LEVEL	Number	Specifies the depth in the document CONNECT looks when defining columns by discovery or in catalog tables
PRETTY	Number	Specifies the format of the Json file (-1 for Bjson files)
EXPAND	String	The name of the column to expand.
OBJECT	String	The json path of the sub-document used for the table.
BASE	Number	The numbering base for arrays: 0 (the default) or 1.
LIMIT	Number	The maximum number of array values to use when concatenating, calculating or expanding arrays. Defaults to 50 (>= Connect 1.7.0003), 10 (<= Connect 1.7.0002).
FULLARRAY	Boolean	Used when creating with Discovery. Make a column for each value of arrays (up to LIMIT).
JMODE	Number	The Json mode (array of objects, array of arrays, or array of values) Only used when inserting new rows.
ACCEPT	Boolean	Keep null columns (for discovery).
AVGLEN	Number	An estimate average length of rows. This is used only when indexing and can be set if indexing fails by miscalculating the table max size.
STRINGIFY	String	Ask discovery to make a column to return the Json representation of this object.

Column options:

Column Option	Type	Description
JPATH FIELD_FORMAT	String	Defaults to the column name.
DATE_FORMAT	String	Specifies the date format into the Json file when defining a DATE, DATETIME or TIME column.

Variables used with Json tables are:

- connect_default_depth
- connect_json_null
- connect_json_all_path
- connect_force_bson

Notes

1. ↑ The value n can be 0 based or 1 based depending on the base table option. The default is 0 to match what is the current usage in the Json world but it can be set to 1 for tables created in old versions.
2. ↑ See for instance: json-functions, https://github.com/mysqludf/lib_mysqludf_json#readme and https://blogs.oracle.com/svetasmirnova/entry/json_udf_functions_version_04
3. ↑ This will not work when CONNECT is compiled embedded

5.3.7.6.13 CONNECT XML Table Type

Contents

Overview

CONNECT supports tables represented by XML files. For these tables, the standard input/output functions of the operating

system are not used but the parsing and processing of the file is delegated to a specialized library. Currently two such systems are supported: libxml2, a part of the GNOME framework, but which does not require GNOME and, on Windows, MS-DOM (DOMDOC), the Microsoft standard support of XML documents.

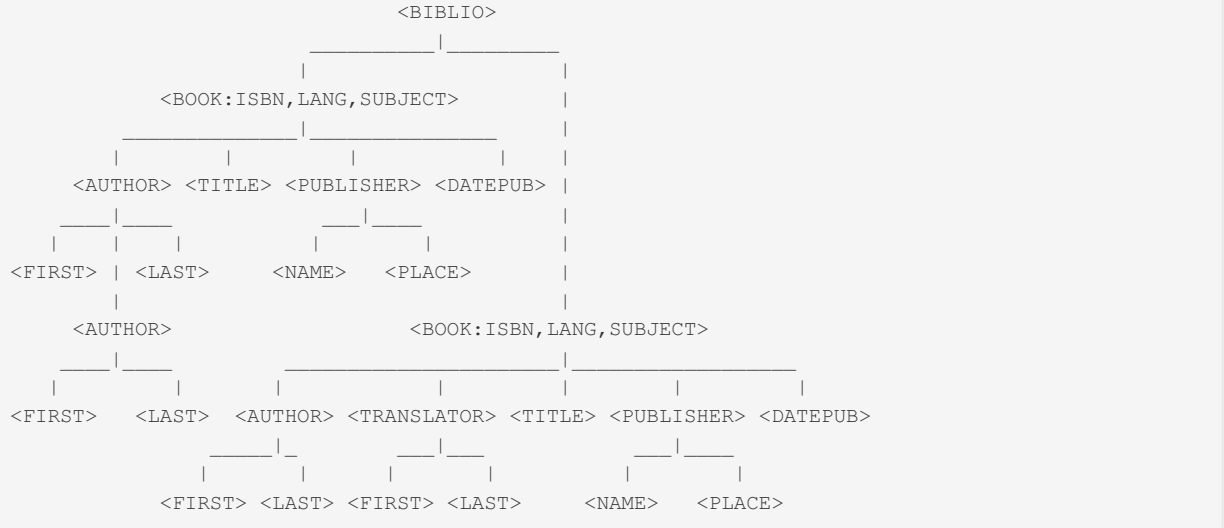
DOMDOC is the default for the Windows version of CONNECT and libxml2 is always used on other systems. On Windows the choice can be specified using the XMLSUP CREATE TABLE list option, for instance specifying `option_list='xmlsup=libxml2'`.

Creating XML tables

First of all, it must be understood that XML is a very general language used to encode data having any structure. In particular, the tag hierarchy in an XML file describes a tree structure of the data. For instance, consider the file:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<BIBLIO SUBJECT="XML">
   <BOOK ISBN="9782212090819" LANG="fr" SUBJECT="applications">
      <AUTHOR>
         <FIRSTNAME>Jean-Christophe</FIRSTNAME>
         <LASTNAME>Bernadac</LASTNAME>
      </AUTHOR>
      <AUTHOR>
         <FIRSTNAME>François</FIRSTNAME>
         <LASTNAME>Knab</LASTNAME>
      </AUTHOR>
      <TITLE>Construire une application XML</TITLE>
      <PUBLISHER>
         <NAME>Eyrolles</NAME>
         <PLACE>Paris</PLACE>
      </PUBLISHER>
      <DATEPUB>1999</DATEPUB>
   </BOOK>
   <BOOK ISBN="9782840825685" LANG="fr" SUBJECT="applications">
      <AUTHOR>
         <FIRSTNAME>William J.</FIRSTNAME>
         <LASTNAME>Pardi</LASTNAME>
      </AUTHOR>
      <TRANSLATOR PREFIX="adapté de l'anglais par">
         <FIRSTNAME>James</FIRSTNAME>
         <LASTNAME>Guerin</LASTNAME>
      </TRANSLATOR>
      <TITLE>XML en Action</TITLE>
      <PUBLISHER>
         <NAME>Microsoft Press</NAME>
         <PLACE>Paris</PLACE>
      </PUBLISHER>
      <DATEPUB>1999</DATEPUB>
   </BOOK>
</BIBLIO>
```

It represents data having the structure:



This structure seems at first view far from being tabular. However, modern database management systems, including

MariaDB, implement something close to the relational model and work on tables that are structurally not hierarchical but tabular with rows and columns.

Nevertheless, CONNECT can do it. Of course, it cannot guess what you want to extract from the XML structure, but gives you the possibility to specify it when you create the table[1].

Let us take a first example. Suppose you want to make a table from the above document, displaying the node contents.

For this, you can define a table *xsamptag* as:

```
create table xsamptag (
  AUTHOR char(50),
  TITLE char(32),
  TRANSLATOR char(40),
  PUBLISHER char(40),
  DATEPUB int(4))
engine=CONNECT table_type=XML file_name='Xsample.xml';
```

It will be displayed as:

AUTHOR	TITLE	TRANSLATOR	PUBLISHER	DATEPUB
Jean-Christophe Bernadac	Construire une application XML	<null>	Eyrolles Paris	1999
William J. Pardi	XML en Action	James Guerin	Microsoft Press Paris	1999

Let us try to understand what happened. By default the column names correspond to tag names. Because this file is rather simple, CONNECT was able to default the top tag of the table as the root node `<BIBLIO>` of the file, and the row tags as the `<BOOK>` children of the table tag. In a more complex file, this should have been specified, as we will see later. Note that we didn't have to worry about the sub-tags such as `<FIRSTNAME>` or `<LASTNAME>` because CONNECT automatically retrieves the entire text contained in a tag and its sub-tags[2].

> Only the first author of the first book appears. This is because only the first occurrence of a column tag has been retrieved so the result has a proper tabular structure. We will see later what we can do about that.

How can we retrieve the values specified by attributes? By using a *Coltype* table option to specify the default column type. The value '@' means that column names match attribute names. Therefore, we can retrieve them by creating a table such as:

```
create table xsampattr (
  ISBN char(15),
  LANG char(2),
  SUBJECT char(32))
engine=CONNECT table_type=XML file_name='Xsample.xml'
option_list='Coltype=@';
```

This table returns the following:

ISBN	LANG	SUBJECT
9782212090819	fr	applications
9782840825685	fr	applications

Now to define a table that will give us all the previous information, we must specify the column type for each column. Because in the next statement the column type defaults to Node, the *field_format* column parameter was used to indicate which columns are attributes:

From Connect 1.7.0002

```
create table xsamp (
ISBN char(15) xpath='@',
LANG char(2) xpath='@',
SUBJECT char(32) xpath='@',
AUTHOR char(50),
TITLE char(32),
TRANSLATOR char(40),
PUBLISHER char(40),
DATEPUB int(4))
engine=CONNECT table_type=XML file_name='Xsample.xml'
tabname='BIBLIO' option_list='rownode=BOOK';
```

Before Connect 1.7.0002

```
create table xsamp (
  ISBN char(15) field_format='@',
  LANG char(2) field_format='@',
  SUBJECT char(32) field_format='@',
  AUTHOR char(50),
  TITLE char(32),
  TRANSLATOR char(40),
  PUBLISHER char(40),
  DATEPUB int(4))
engine=CONNECT table_type=XML file_name='Xsample.xml'
tabname='BIBLIO' option_list='rownode=BOOK';
```

Once done, we can enter the query:

```
select subject, lang, title, author from xsamp;
```

This will return the following result:

SUBJECT	LANG	TITLE	AUTHOR
applications	fr	Construire une application XML	Jean-Christophe Bernadac
applications	fr	XML en Action	William J. Pardi

Note that we have been lucky. Because unlike SQL, XML is case sensitive and the column names have matched the node names only because the column names were given in upper case. Note also that the order of the columns in the table could have been different from the order in which the nodes appear in the XML file.

Using Xpaths with XML tables

Xpath is used by XML to locate and retrieve nodes. The table's main node Xpath is specified by the `tabname` option. If just the node name is given, CONNECT constructs an Xpath such as `'BIBLIO'` in the example above that should retrieve the `BIBLIO` node wherever it is within the XML file.

The row nodes are by default the children of the table node. However, for instance to eliminate some children nodes that are not real row nodes, the row node name can be specified using the `rownode` sub-option of the `option_list` option.

The field_format options we used above can be specified to locate more precisely where and what information to retrieve using an Xpath-like syntax. For instance:

From Connect 1.7.0002

```
create table xsampall (
isbn char(15) xpath='@ISBN',
language char(2) xpath='@LANG',
subject char(32) xpath='@SUBJECT',
authorfn char(20) xpath='AUTHOR/FIRSTNAME',
authorln char(20) xpath='AUTHOR/LASTNAME',
title char(32) xpath='TITLE',
translated char(32) xpath='TRANSLATOR/@PREFIX',
tranfn char(20) xpath='TRANSLATOR/FIRSTNAME',
tranln char(20) xpath='TRANSLATOR/LASTNAME',
publisher char(20) xpath='PUBLISHER/NAME',
location char(20) xpath='PUBLISHER/PLACE',
year int(4) xpath='DATEPUB')
engine=CONNECT table_type=XML file_name='Xsample.xml'
tabname='BIBLIO' option_list='rownode=BOOK';
```

Before Connect 1.7.0002

```
create table xsampall (
  isbn char(15) field_format='@ISBN',
  language char(2) field_format='@LANG',
  subject char(32) field_format='@SUBJECT',
  authorfn char(20) field_format='AUTHOR/FIRSTNAME',
  authorln char(20) field_format='AUTHOR/LASTNAME',
  title char(32) field_format='TITLE',
  translated char(32) field_format='TRANSLATOR/@PREFIX',
  tranfn char(20) field_format='TRANSLATOR/FIRSTNAME',
  tranln char(20) field_format='TRANSLATOR/LASTNAME',
  publisher char(20) field_format='PUBLISHER/NAME',
  location char(20) field_format='PUBLISHER/PLACE',
  year int(4) field_format='DATEPUB')
engine=CONNECT table_type=XML file_name='Xsample.xml'
tabname='BIBLIO' option_list='rownode=BOOK';
```

This very flexible column parameter serves several purposes:

- To specify the tag name, or the attribute name if different from the column name.
- To specify the type (tag or attribute) by a prefix of '@' for attributes.
- To specify the path for sub-tags using the '/' character.

This path is always relative to the current context (the column top node) and cannot be specified as an absolute path from the document root, therefore a leading '/' cannot be used. The path cannot be variable in node names or depth, therefore using ' `//` ' is not allowed.

The query:

```
select isbn, title, translated, tranfn, tranln, location from
    xsampall where translated is not null;
```

replies:

ISBN	TITLE	TRANSLATED	TRANFN	TRANLN	LOCATION
9782840825685	XML en Action	adapté de l'anglais par	James	Guerin	Paris

Libxml2 default name space issue

An issue with libxml2 is that some files can declare a default name space in their root node. Because Xpath only searches in that name space, the nodes will not be found if they are not prefixed. If this happens, specify the tabname option as an Xpath ignoring the current name space:

```
TABNAME="//*[local-name()='BIBLIO']"
```

This must also be done for the default of specified Xpath of the not attribute columns. For instance:

```
title char(32) field_format="*[local-name()='TITLE']",
```

Note: This raises an error (and is useless anyway) with DOMDOC.

Direct access on XML tables

Direct access is available on XML tables. This means that XML tables can be sorted and used in joins, even in the one-side of the join.

However, building a permanent index is not yet implemented. It is unclear whether this can be useful. Indeed, the DOM implementation that is used to access these tables firstly parses the whole file and constructs a node tree in memory. This may often be the longest part of the process, so the use of an index would not be of great value. Note also that this limits the XML files to a reasonable size. Anyway, when speed is important, this table type is not the best to use. Therefore, in these cases, it is probably better to convert the file to another type by inserting the XML table into another table of a more appropriate type for performance.

Accessing tags with namespaces

With the Windows DOMDOC support, this can be done using the prefix in the tabname column option and/or xpath column option. For instance, given the file gns.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<gpx xmlns:gns="http:dummy">
<gns:trkseg>
<trkpt lon="-121.9822235107421875" lat="37.3884925842285156">
<gns:ele>6.610851287841797</gns:ele>
<time>2014-04-01T14:54:05.000Z</time>
</trkpt>
<trkpt lon="-121.9821929931640625" lat="37.3885803222656250">
<ele>6.787827968597412</ele>
<time>2014-04-01T14:54:08.000Z</time>
</trkpt>
<trkpt lon="-121.9821624755859375" lat="37.3886299133300781">
<ele>6.771987438201904</ele>
<time>2014-04-01T14:54:10.000Z</time>
</trkpt>
</gns:trkseg>
</gpx>
```

and the defined CONNECT table:

```
CREATE TABLE xgns (
`lon` double(21,16) NOT NULL `xpath`='@',
`lat` double(20,16) NOT NULL `xpath`='@',
`ele` double(21,16) NOT NULL `xpath`='gns:ele',
`time` datetime date_format="YYYY-MM-DD 'T' hh:mm:ss '.000Z'"
)
  ENGINE=CONNECT DEFAULT CHARSET=latin1 `table_type`=XML
  `file_name`='gns.xml' tabname='gns:trkseg' option_list='xmlsup=domdoc';
```

```
select * from xgns;
```

Displays:

lon	lat	ele	time
-121,982223510742	37,3884925842285	6,6108512878418	01/04/2014 14:54:05
-121,982192993164	37,3885803222656	0	01/04/2014 14:54:08
-121,982162475586	37,3886299133301	0	01/04/2014 14:54:10

Only the prefixed ‘ele’ tag is recognized.

However, this does not work with the libxml2 support. The solution is then to use a function ignoring the name space:

```
CREATE TABLE xgns2 (
`lon` double(21,16) NOT NULL `xpath`='@',
`lat` double(20,16) NOT NULL `xpath`='@',
`ele` double(21,16) NOT NULL `xpath`="*[local-name()='ele']",
`time` datetime date_format="YYYY-MM-DD 'T' hh:mm:ss '.000Z'"
)
  ENGINE=CONNECT DEFAULT CHARSET=latin1 `table_type`=XML
  `file_name`='gns.xml' tabname="*[local-name()='trkseg']" option_list='xmlsup=libxml2';
```

Then :

```
select * from xgns2;
```

Displays:

lon	lat	ele	time
-121,982223510742	37,3884925842285	6,6108512878418	01/04/2014 14:54:05
-121,982192993164	37,3885803222656	6.7878279685974	01/04/2014 14:54:08
-121,982162475586	37,3886299133301	6.7719874382019	01/04/2014 14:54:10

This time, all ‘ele` tags are recognized. This solution does not work with DOMDOC.

Having Columns defined by Discovery

It is possible to let the MariaDB discovery process do the job of column specification. When columns are not defined in the CREATE TABLE statement, CONNECT endeavours to analyze the XML file and to provide the column specifications. This is possible only for true XML tables, but not for HTML tables.

For instance, the *xsamp* table could have been created specifying:

```
create table xsamp
engine=CONNECT table_type=XML file_name='Xsample.xml'
tabname='BIBLIO' option_list='rownode=BOOK';
```

Let's check how it was actually specified using the SHOW CREATE TABLE statement:

```
CREATE TABLE `xsamp` (
  `ISBN` char(13) NOT NULL `FIELD_FORMAT`='@',
  `LANG` char(2) NOT NULL `FIELD_FORMAT`='@',
  `SUBJECT` char(12) NOT NULL `FIELD_FORMAT`='@',
  `AUTHOR` char(24) NOT NULL,
  `TRANSLATOR` char(12) DEFAULT NULL,
  `TITLE` char(30) NOT NULL,
  `PUBLISHER` char(21) NOT NULL,
  `DATEPUB` char(4) NOT NULL
) ENGINE=CONNECT DEFAULT CHARSET=latin1 `TABLE_TYPE`='XML'
  `FILE_NAME`='E:/Data/Xml/Xsample.xml' `TABNAME`='BIBLIO' `OPTION_LIST`='rownode=BOOK';
```

It is equivalent except for the column sizes that have been calculated from the file as the maximum length of the corresponding column when it was a normal value. Also, all columns are specified as type CHAR because XML does not provide information about the node content data type. Nullable is set to true if the column is missing in some rows.

If a more complex definition is desired, you can ask CONNECT to analyse the XPATH up to a given level using the level option in the option list. The level value is the number of nodes that are taken in the XPATH. For instance:

```
create table xsampall
engine=CONNECT table_type=XML file_name='Xsample.xml'
tabname='BIBLIO' option_list='rownode=BOOK,Level=1';
```

This will define the table as:

From Connect 1.7.0002

```
CREATE TABLE `xsampall` (
  `ISBN` char(13) NOT NULL `XPATH`='@',
  `LANG` char(2) NOT NULL `XPATH`='@',
  `SUBJECT` char(12) NOT NULL `XPATH`='@',
  `AUTHOR_FIRSTNAME` char(15) NOT NULL `XPATH`='AUTHOR/FIRSTNAME',
  `AUTHOR_LASTNAME` char(8) NOT NULL `XPATH`='AUTHOR/LASTNAME',
  `TRANSLATOR_PREFIX` char(24) DEFAULT NULL `XPATH`='TRANSLATOR/@PREFIX',
  `TRANSLATOR_FIRSTNAME` char(7) DEFAULT NULL `XPATH`='TRANSLATOR/FIRSTNAME',
  `TRANSLATOR_LASTNAME` char(6) DEFAULT NULL `XPATH`='TRANSLATOR/LASTNAME',
  `TITLE` char(30) NOT NULL,
  `PUBLISHER_NAME` char(15) NOT NULL `XPATH`='PUBLISHER/NAME',
  `PUBLISHER_PLACE` char(5) NOT NULL `XPATH`='PUBLISHER/PLACE',
  `DATEPUB` char(4) NOT NULL
) ENGINE=CONNECT DEFAULT CHARSET=latin1 `TABLE_TYPE`='XML' `FILE_NAME`='Xsample.xml'
`TABNAME`='BIBLIO' `OPTION_LIST`='rownode=BOOK,Depth=1';
<</sql>>


Before Connect 1.7.0002
<<sql>>
CREATE TABLE `xsampall` (
  `ISBN` char(13) NOT NULL `FIELD_FORMAT`='@',
  `LANG` char(2) NOT NULL `FIELD_FORMAT`='@',
  `SUBJECT` char(12) NOT NULL `FIELD_FORMAT`='@',
  `AUTHOR_FIRSTNAME` char(15) NOT NULL `FIELD_FORMAT`='AUTHOR/FIRSTNAME',
  `AUTHOR_LASTNAME` char(8) NOT NULL `FIELD_FORMAT`='AUTHOR/LASTNAME',
  `TRANSLATOR_PREFIX` char(24) DEFAULT NULL `FIELD_FORMAT`='TRANSLATOR/@PREFIX',
  `TRANSLATOR_FIRSTNAME` char(7) DEFAULT NULL `FIELD_FORMAT`='TRANSLATOR/FIRSTNAME',
  `TRANSLATOR_LASTNAME` char(6) DEFAULT NULL `FIELD_FORMAT`='TRANSLATOR/LASTNAME',
  `TITLE` char(30) NOT NULL,
  `PUBLISHER_NAME` char(15) NOT NULL `FIELD_FORMAT`='PUBLISHER/NAME',
```

```
  `PUBLISHER_PLACE` char(5) NOT NULL `FIELD_FORMAT`='PUBLISHER/PLACE',
  `DATEPUB` char(4) NOT NULL
) ENGINE=CONNECT DEFAULT CHARSET=latin1 `TABLE_TYPE`='XML' `FILE_NAME`='Xsample.xml'
  `TABNAME`='BIBLIO' `OPTION_LIST`='rownode=BOOK,Level=1';
<</sql>>

This method can be used as a quick way to make a "template" table definition that can later be
edited to make the desired definition. In particular, column names are constructed from all the
nodes of their path in order to have distinct column names. This can be manually edited to have
the desired names, provided their XPATH is not modified.

To have a preview of how columns will be defined, you can use a catalog table like this:

<<sql>>
create table xsacol
engine=CONNECT table_type=XML file_name='Xsample.xml'
tabname='BIBLIO' option_list='rownode=BOOK,Level=1' catfunc=col;
<</sql>>

And when asking:

<<sql>>
select column_name Name, type_name Type, column_size Size, nullable, xpath from xsacol;
<</sql>>

You get the description of what the table columns will be:

<<style class="darkheader-nospace-borders">>
|= Name |= Type |= Size |= nullable |= xpath |
| ISBN | CHAR | 13 | 0 | @ |
| LANG | CHAR | 2 | 0 | @ |
| SUBJECT | CHAR | 12 | 0 | @ |
| AUTHOR_FIRSTNAME | CHAR | 15 | 0 | AUTHOR/FIRSTNAME |
| AUTHOR_LASTNAME | CHAR | 8 | 0 | AUTHOR/LASTNAME |
| TRANSLATOR_PREFIX | CHAR | 24 | 1 | TRANSLATOR/@PREFIX |
| TRANSLATOR_FIRSTNAME | CHAR | 7 | 1 | TRANSLATOR/FIRSTNAME |
| TRANSLATOR_LASTNAME | CHAR | 6 | 1 | TRANSLATOR/LASTNAME |
| TITLE | CHAR | 30 | 0 |  |
| PUBLISHER_NAME | CHAR | 15 | 0 | PUBLISHER/NAME |
| PUBLISHER_PLACE | CHAR | 5 | 0 | PUBLISHER/PLACE |
| DATEPUB | CHAR | 4 | 0 | |
<</style>>

== Write operations on XML tables
You can freely use the Update, Delete and Insert commands with XML tables.
However, you must understand that the format of the updated or inserted data
follows the specifications of the table you created, not the ones of the
original source file. For instance, let us suppose we insert a new book using
the //xsamp// table (not the //xsampall// table) with the command:

<<code lang=mysql inline=false>>
insert into xsamp
  (isbn, lang, subject, author, title, publisher,datepub)
  values ('9782212090529','fr','général','Alain Michard',
         'XML, Langage et Applications','Eyrolles Paris',1998);
```

Then if we ask:

```
select subject, author, title, translator, publisher from xsamp;
```

Everything seems correct when we get the result:

SUBJECT	AUTHOR	TITLE	TRANSLATOR	PUBLISHER
applications	Jean-Christophe Bernadac	Construire une application XML		Eyrolles Paris
applications	William J. Pardi	XML en Action	James Guerin	Microsoft Press Paris
général	Alain Michard	XML, Langage et Applications		Eyrolles Paris

However if we enter the apparently equivalent query on the *xsampall* table, based on the same file:

```
select subject,
concat(authorfn, ' ', authorln) author , title,
concat(tranfn, ' ', tranln) translator,
concat(publisher, ' ', location) publisher from xsampall;
```

this returns an apparently wrong answer:

SUBJECT	AUTHOR	TITLE	TRANSLATOR	PUBLISHER
applications	Jean-Christophe Bernadac	Construire une application XML		Eyrolles Paris
applications	William J. Pardi	XML en Action	James Guerin	Microsoft Press Paris
général		XML, Langage et Applications		

What happened here? Simply, because we used the *xsamp* table to do the Insert, what has been inserted within the XML file had the structure described for *xsamp*:

```
   <BOOK ISBN="9782212090529" LANG="fr" SUBJECT="général">
      <AUTHOR>Alain Michard</AUTHOR>
      <TITLE>XML, Langage et Applications</TITLE>
      <TRANSLATOR></TRANSLATOR>
      <PUBLISHER>Eyrolles Paris</PUBLISHER>
      <DATEPUB>1998</DATEPUB>
   </BOOK>
```

CONNECT cannot "invent" sub-tags that are not part of the *xsamp* table. Because these sub-tags do not exist, the *xsampall* table cannot retrieve the information that should be attached to them. If we want to be able to query the XML file by all the defined tables, the correct way to insert a new book to the file is to use the *xsampall* table, the only one that addresses all the components of the original document:

```
delete from xsamp where isbn = '9782212090529';

insert into xsampall (isbn, language, subject, authorfn, authorln,
      title, publisher, location, year)
   values('9782212090529','fr','général','Alain','Michard',
      'XML, Langage et Applications','Eyrolles','Paris',1998);
```

Now the added book, in the XML file, will have the required structure:

```
   <BOOK ISBN="9782212090529" LANG="fr" SUBJECT="général">
      <AUTHOR>
         <FIRSTNAME>Alain</FIRSTNAME>
         <LASTNAME>Michard</LASTNAME>
      </AUTHOR>
      <TITLE>XML, Langage et Applications</TITLE>
      <PUBLISHER>
         <NAME>Eyrolles</NAME>
         <PLACE>Paris</PLACE>
      </PUBLISHER>
      <DATEPUB>1998</DATEPUB>
   </BOOK>
```

Note: We used a column list in the Insert statements when creating the table to avoid generating a `<TRANSLATOR>` node with sub-nodes, all containing null values (this works on Windows only).

Multiple nodes in the XML document

Let us come back to the above example XML file. We have seen that the author node can be "multiple" meaning that there can be more than one author of a book. What can we do to get the complete information fitting the relational model? CONNECT provides you with two possibilities, but is restricted to only one such multiple node per table.

The first and most challenging one is to return as many rows than there are authors, the other columns being repeated as if we had make a join between the author column and the rest of the table. To achieve this, simply specify the “multiple” node name and the “expand” option when creating the table. For instance, we can create the *xsamp2* table like this:

```
create table xsamp2 (
  ISBN char(15) field_format='@',
  LANG char(2) field_format='@',
  SUBJECT char(32) field_format='@',
  AUTHOR char(40),
  TITLE char(32),
  TRANSLATOR char(32),
  PUBLISHER char(32),
  DATEPUB int(4))
engine=CONNECT table_type=XML file_name='Xsample.xml'
tabname='BIBLIO'
option_list='rownode=BOOK,Expand=1,Mulnode=AUTHOR,Limit=2';
```

In this statement, the Limit option specifies the maximum number of values that will be expanded. If not specified, it defaults to `10`. Any values above the limit will be ignored and a warning message issued[3]. Now you can enter a query such as:

```
select isbn, subject, author, title from xsamp2;
```

This will retrieve and display the following result:

ISBN	SUBJECT	AUTHOR	TITLE
9782212090819	applications	Jean-Christophe Bernadac	Construire une application XML
9782212090819	applications	François Knab	Construire une application XML
9782840825685	applications	William J. Pardi	XML en Action
9782212090529	général	Alain Michard	XML, Langage et Applications

In this case, this is as if the table had four rows. However if we enter the query:

```
select isbn, subject, title, publisher from xsamp2;
```

this time the result will be:

ISBN	SUBJECT	TITLE	PUBLISHER
9782212090819	applications	Construire une application XML	Eyrolles Paris
9782840825685	applications	XML en Action	Microsoft Press Paris
9782212090529	général	XML, Langage et Applications	Eyrolles Paris

Because the author column does not appear in the query, the corresponding row was not expanded. This is somewhat strange because this would have been different if we had been working on a table of a different type. However, it is closer to the relational model for which there should not be two identical rows (tuples) in a table. Nevertheless, you should be aware of this somewhat erratic behavior. For instance:

```
select count(*) from xsamp2;                /* Replies 3 */
select count(author) from xsamp2;           /* Replies 4 */
select count(isbn) from xsamp2;             /* Replies 3 */
select isbn, subject, title, publisher from xsamp2 where author <> '';
```

This last query replies:

ISBN	SUBJECT	TITLE	PUBLISHER
9782212090819	applications	Construire une application XML	Eyrolles Paris
9782212090819	applications	Construire une application XML	Eyrolles Paris
9782840825685	applications	XML en Action	Microsoft Press Paris
9782212090529	général	XML, Langage et Applications	Eyrolles Paris

Even though the author column does not appear in the result, the corresponding row was expanded because the multiple column was used in the where clause.

Intermediate multiple node

The "multiple" node can be an intermediate node. If we want to do the same expanding with the *xsampall* table, there will be nothing more to do. The *xsampall2* table can be created with:

From Connect 1.7.0002

```
create table xsampall2 (
isbn char(15) xpath='@ISBN',
language char(2) xpath='@LANG',
subject char(32) xpath='@SUBJECT',
authorfn char(20) xpath='AUTHOR/FIRSTNAME',
authorln char(20) xpath='AUTHOR/LASTNAME',
title char(32) xpath='TITLE',
translated char(32) xpath='TRANSLATOR/@PREFIX',
tranfn char(20) xpath='TRANSLATOR/FIRSTNAME',
tranln char(20) xpath='TRANSLATOR/LASTNAME',
publisher char(20) xpath='PUBLISHER/NAME',
location char(20) xpath='PUBLISHER/PLACE',
year int(4) xpath='DATEPUB')
engine=CONNECT table_type=XML file_name='Xsample.xml'
tabname='BIBLIO' option_list='rownode=BOOK,Expand=1,Mulnode=AUTHOR,Limit=2';
```

Before Connect 1.7.0002

```
create table xsampall2 (
  isbn char(15) field_format='@ISBN',
  language char(2) field_format='@LANG',
  subject char(32) field_format='@SUBJECT',
  authorfn char(20) field_format='AUTHOR/FIRSTNAME',
  authorln char(20) field_format='AUTHOR/LASTNAME',
  title char(32) field_format='TITLE',
  translated char(32) field_format='TRANSLATOR/@PREFIX',
  tranfn char(20) field_format='TRANSLATOR/FIRSTNAME',
  tranln char(20) field_format='TRANSLATOR/LASTNAME',
  publisher char(20) field_format='PUBLISHER/NAME',
  location char(20) field_format='PUBLISHER/PLACE',
  year int(4) field_format='DATEPUB')
engine=CONNECT table_type=XML file_name='Xsample.xml'
tabname='BIBLIO'
option_list='rownode=BOOK,Expand=1,Mulnode=AUTHOR,Limit=2';
```

The only difference is that the "multiple" node is an intermediate node in the path. The resulting table can be seen with a query such as:

```
select subject, language lang, title, authorfn first, authorln
    last, year from xsampall2;
```

This query displays:

SUBJECT	LANG	TITLE	FIRST	LAST	YEAR
applications	fr	Construire une application XML	Jean-Christophe	Bernadac	1999
applications	fr	Construire une application XML	François	Knab	1999
applications	fr	XML en Action	William J.	Pardi	1999
général	fr	XML, Langage et Applications	Alain	Michard	1998

These composite tables, half array half tree, reserve some surprises for us when updating, deleting from or inserting into them. Insert just cannot generate this structure; if two rows are inserted with just a different author, two book nodes will be generated in the XML file. Delete always deletes one book node and all its children nodes even if specified against only one author. Update is more complicated:

```
update xsampall2 set authorfn = 'Simon' where authorln = 'Knab';
update xsampall2 set year = 2002 where authorln = 'Bernadac';
update xsampall2 set authorln = 'Mercier' where year = 2002;
```

After these three updates, the first two responding "Affected rows: 1" and the last one responding "Affected rows: 2", the last query answers:

subject	lang	title	first	last	year

applications	fr	Construire une application XML	Jean-Christophe	Mercier	2002
applications	fr	Construire une application XML	François	Knab	2002
applications	fr	XML en Action	William J.	Pardi	1999
général	fr	XML, Langage et Applications	Alain	Michard	1998

What must be understood here is that the Update modifies node values in the XML file, not cell values in the relational table. The first update worked normally. The second update changed the year value of the book and this shows for the two expanded rows because there is only one DATEPUB node for that book. Because the third update applies to a row having a certain date value, both author names were updated.

Making a List of Multiple Values

Another way to see multiple values is to ask CONNECT to make a comma separated list of the multiple node values. This time, it can only be done if the "multiple" node is not intermediate. For example, we can modify the *xsamp2* table definition by:

```
alter table xsamp2 option_list='rownode=BOOK,Mulnode=AUTHOR,Limit=3';
```

This time 'Expand' is not specified, and Limit gives the maximum number of items in the list. Now if we enter the query:

```
select isbn, subject, author "AUTHOR(S)", title from xsamp2;
```

We will get the following result:

ISBN	SUBJECT	AUTHOR(S)	TITLE
9782212090819	applications	Jean-Christophe Bernadac, François Knab	Construire une application XML
9782840825685	applications	William J. Pardi	XML en Action
9782212090529	général	Alain Michard	XML, Langage et Applications

Note that updating the "multiple" column is not possible because CONNECT does not know which of the nodes to update.

This could not have been done with the *xsampall2* table because the author node is intermediate in the path, and making two lists, one of first names and another one of last names would not make sense anyway.

What if a table contains several multiple nodes

This can be handled by creating several tables on the same file, each containing only one multiple node and constructing the desired result using joins.

Support of HTML Tables

Most tables included in HTML documents cannot be processed by CONNECT because the HTML language is often not compatible with the syntax of XML. In particular, XML requires all open tags to be matched by a closing tag while it is sometimes optional in HTML. This is often the case concerning column tags.

However, you can meet tables that respect the XML syntax but have some of the features of HTML tables. For instance:

```
<?xml version="1.0"?>
<Beers>
  <table>
    <th><td>Name</td><td>Origin</td><td>Description</td></th>
    <tr>
      <td><brandName>Huntsman</brandName></td>
      <td><origin>Bath, UK</origin></td>
      <td><details>Wonderful hop, light alcohol</details></td>
    </tr>
    <tr>
      <td><brandName>Tuborg</brandName></td>
      <td><origin>Danmark</origin></td>
      <td><details>In small bottles</details></td>
    </tr>
  </table>
</Beers>
```

Here the different column tags are included in `<td></td>` tags as for HTML tables. You cannot just add this tag in the Xpath of the columns, because the search is done on the first occurrence of each tag, and this would cause this search to fail for all columns except the first one. This case is handled by specifying the *Colnode* table option that gives the name of these column tags, for example:

From Connect 1.7.0002

```
create table beers (
`Name` char(16) xpath='brandName',
`Origin` char(16) xpath='origin',
`Description` char(32) xpath='details')
engine=CONNECT table_type=XML file_name='beers.xml'
tabname='table' option_list='rownode=tr,colnode=td';
```

Before Connect 1.7.0002

```
create table beers (
  `Name` char(16) field_format='brandName',
  `Origin` char(16) field_format='origin',
  `Description` char(32) field_format='details')
engine=CONNECT table_type=XML file_name='beers.xml'
tabname='table' option_list='rownode=tr,colnode=td';
```

The table will be displayed as:

Name	Origin	Description
Huntsman	Bath, UK	Wonderful hop, light alcohol
Tuborg	Danmark	In small bottles

However, you can deal with tables even closer to the HTML model. For example the *coffee.htm* file:

```
<TABLE summary="This table charts the number of cups of coffe
                consumed by each senator, the type of coffee (decaf
                or regular), and whether taken with sugar.">
  <CAPTION>Cups of coffee consumed by each senator</CAPTION>
  <TR>
    <TH>Name</TH>
    <TH>Cups</TH>
    <TH>Type of Coffee</TH>
    <TH>Sugar?</TH>
  </TR>
  <TR>
    <TD>T. Sexton</TD>
    <TD>10</TD>
    <TD>Espresso</TD>
    <TD>No</TD>
  </TR>
  <TR>
    <TD>J. Dinnen</TD>
    <TD>5</TD>
    <TD>Decaf</TD>
    <TD>Yes</TD>
  </TR>
</TABLE>
```

Here column values are directly represented by the TD tag text. You cannot declare them as tags nor as attributes. In addition, they are not located using their name but by their position within the row. Here is how to declare such a table to CONNECT:

```
create table coffee (
  `Name` char(16),
  `Cups` int(8),
  `Type` char(16),
  `Sugar` char(4))
engine=connect table_type=XML file_name='coffee.htm'
tabname='TABLE' header=1 option_list='Coltype=HTML';
```

You specify the fact that columns are located by position by setting the *Coltype* option to 'HTML'. Each column position (0 based) will be the value of the *flag* column parameter that is set by default in sequence. Now we are able to display the

table:

Name	Cups	Type	Sugar
T. Sexton	10	Espresso	No
J. Dinnen	5	Decaf	Yes

Note 1: We specified ' `header=n` ' in the create statement to indicate that the first n rows of the table are not data rows and should be skipped.

Note 2: In this last example, we did not specify the node names using the Rownode and Colnode options because when *Coltype* is set to 'HTML' they default to ' `Rownode=TR` ' and ' `Colnode=TD` '.

Note 3: The *Coltype* option is a word only the first character of which is significant. Recognized values are:

T(ag) or N(ode)	Column names match a tag name (the default).
A(ttribute) or @	Column names match an attribute name.
H(tml) or C(ol) or P(os)	Column are retrieved by their position.

New file setting

Some create options are used only when creating a table on a new file, i. e. when inserting into a file that does not exist yet. When specified, the 'Header' option will create a header row with the name of the table columns. This is chiefly useful for HTML tables to be displayed on a web browser.

Some new list-options are used in this context:

Encoding	The encoding of the new document, defaulting to UTF-8.
Attribute	A list of 'attname=attvalue' separated by ';' to add to the table node.
HeadAttr	An attribute list to be added to the header row node.

Let us see for instance, the following create statement:

```
create table handlers (
  handler char(64),
  version char(20),
  author char(64),
  description char(255),
  maturity char(12))
engine=CONNECT table_type=XML file_name='handlers.htm'
tabname='TABLE' header=yes
option_list='coltype=HTML,encoding=ISO-8859-1,
attribute=border=1;cellpadding=5,headattr=bgcolor=yellow';
```

Supposing the table file does not exist yet, the first insert into that table, for instance by the following statement:

```
insert into handlers select plugin_name, plugin_version,
  plugin_author, plugin_description, plugin_maturity from
  information_schema.plugins where plugin_type = 'DAEMON';
```

will generate the following file:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Created by CONNECT Version 3.05.0005 August 17, 2012 -->
<TABLE border="1" cellpadding="5">
  <TR bgcolor="yellow">
    <TH>handler</TH>
    <TH>version</TH>
    <TH>author</TH>
    <TH>description</TH>
    <TH>maturity</TH>
  </TR>
  <TR>
    <TD>Maria</TD>
    <TD>1.5</TD>
    <TD>Monty Program Ab</TD>
    <TD>Compatibility aliases for the Aria engine</TD>
    <TD>Gamma</TD>
  </TR>
</TABLE>
```

This file can be used to display the table on a web browser (encoding should be `ISO-8859-x`)

handler	version	author	description	maturity
Maria	1.5	Monty Program Ab	Compatibility aliases for the Aria engine	Gamma

Note: The XML document encoding is generally specified in the XML header node and can be different from the DATA_CHARSET, which is always UTF-8 for XML tables. Therefore the table DATA_CHARSET character set should be unspecified, or specified as UTF8. The Encoding specification is useful only for new XML files and ignored for existing files having their encoding already specified in the header node.

Notes

1. ↑ CONNECT does not claim to be able to deal with any XML document. Besides, those that can usefully be processed for data analysis are likely to have a structure that can easily be transformed into a table.
2. ↑ With libxml2, sub tags text can be separated by 0 or several blanks depending on the structure and indentation of the data file.
3. ↑ This may cause some rows to be lost because an eventual where clause on the “multiple” column is applied only on the limited number of retrieved rows.

5.3.7.6.14 CONNECT INI Table Type

Contents

Overview

The INI type is one of the configuration or initialization files often found on Windows machines. For instance, let us suppose you have the following contact file *contact.ini*:

```
[BER]
name=Bertrand
forename=Olivier
address=21 rue Ferdinand Buisson
city=Issy-les-Mlx
zipcode=92130
tel=09.54.36.29.60
cell=06.70.06.04.16

[WEL]
name=Schmitt
forename=Bernard
hired=19/02/1985
address=64 tiergarten strasse
city=Berlin
zipcode=95013
tel=03.43.377.360

[UK1]
name=Smith
forename=Henry
hired=08/11/2003
address=143 Blum Rd.
city=London
zipcode=NW1 2BP
```

CONNECT lets you view it as a table in two different ways.

Column layout

The first way is to regard it as a table having one line per section, the columns being the keys you want to display. In this case, the CREATE statement could be:

```
create table contact (
  contact char(16) flag=1,
  name char(20),
  forename char(32),
  hired date date_format='DD/MM/YYYY',
  address char(64),
  city char(20),
  zipcode char(8),
  tel char(16))
engine=CONNECT table_type=INI file_name='contact.ini';
```

The column that will contain the section name can have any name but must specify `flag=1` . All other columns must have the names of the keys we want to display (case insensitive). The type can be character or numeric depending on the key value type, and the length is the maximum expected length for the key value. Once done, the statement:

```
select contact, name, hired, city, tel from contact;
```

This statement will display the file in tabular format.

contact	name	hired	city	tel
BER	Bertrand	1970-01-01	Issy-les-Mlx	09.54.36.29.60
WEL	Schmitt	1985-02-19	Berlin	03.43.377.360
UK1	Smith	2003-11-08	London	NULL

Only the keys defined in the create statements are visible; keys that do not exist in a section are displayed as null or pseudo null (blank for character, 1/1/70 for dates, and 0 for numeric) for columns declared NOT NULL.

All relational operations can be applied to this table. The table (and the file) can be updated, inserted and conditionally deleted. The only constraint is that when inserting values, the section name must be the first in the list of values.

Note 1: When inserting, if a section already exists, no new section will be created but the new values will be added or replace those of the existing section. Thus, the following two commands are equivalent:

```
update contact set forename = 'Harry' where contact = 'UK1';
insert into contact (contact,forename) values('UK1','Harry');
```

Note 2: Because sections represent one line, a DELETE statement on a section key will delete the whole section.

Row layout

To be a good candidate for tabular representation, an INI file should have often the same keys in all sections. In practice, many files commonly found on computers, such as the *win.ini* file of the Windows directory or the *my.ini* file cannot be viewed that way because each section has different keys. In this case, a second way is to regard the file as a table having one row per section key and whose columns can be the section name, the key name, and the key value.

For instance, let us define the table:

```
create table xcont (
  section char(16) flag=1,
  keyname char(16) flag=2,
  value char(32))
engine=CONNECT table_type=INI file_name='contact.ini'
option_list='Layout=Row';
```

In this statement, the "Layout" option sets the display format, Column by default or anything else not beginning by 'C' for row layout display. The names of the three columns can be freely chosen. The Flag option gives the meaning of the column. Specify `flag=1` for the section name and `flag=2` for the key name. Otherwise, the column will contain the key value.

Once done, the command:

```
select * from xcont;
```

Will display the following result:

section	keyname	value
BER	name	Bertrand
BER	forename	Olivier
BER	address	21 rue Ferdinand Buisson
BER	city	Issy-les-Mlx
BER	zipcode	92130
BER	tel	09.54.36.29.60
BER	cell	06.70.06.04.16
WEL	name	Schmitt
WEL	forename	Bernard
WEL	hired	19/02/1985
WEL	address	64 tiergarten strasse
WEL	city	Berlin
WEL	zipcode	95013
WEL	tel	03.43.377.360
UK1	name	Smith
UK1	forename	Henry
UK1	hired	08/11/2003
UK1	address	143 Blum Rd.
UK1	city	London
UK1	zipcode	NW1 2BP

Note: When processing an INI table, all section names are retrieved in a buffer of 8K bytes (2048 bytes before 10.0.17). For a big file having many sections, this size can be increased using for example:

```
option_list='seclen=16K';
```

5.3.7.6.15 CONNECT - External Table Types

Because so many ODBC and JDBC drivers exist and only the main ones have been heavily tested, these table types cannot be ranked as stable. Use them with care in production applications.

These types can be used to access tables belonging to the current or another database server. Six types are currently provided:

ODBC: To be used to access tables from a database management system providing an ODBC connector. ODBC is a standard of Microsoft and is currently available on Windows. On Linux, it can also be used provided a specific application emulating ODBC is installed. Currently only unixODBC is supported.

JDBC: To be used to access tables from a database management system providing a JDBC connector. JDBC is an Oracle standard implemented in Java and principally meant to be used by Java applications. Using it directly from C or C++ application seems to be almost impossible due to an Oracle bug still not fixed. However, this can be achieved using a Java wrapper class used as an interface between C++ and JDBC. On another hand, JDBC is available on all platforms and operating systems.

Mongo: To access MongoDB collections as tables via their MongoDB C Driver. Because this requires both MongoDB and the C Driver to be installed and operational, this table type is not currently available in binary distributions but only when compiling MariaDB from source.

MySQL: This type is the preferred way to access tables belonging to another MySQL or MariaDB server. It uses the MySQL API to access the external table. Even though this can be obtained using the FEDERATED(X) plugin, this specific type is used internally by CONNECT because it also makes it possible to access tables belonging to the current server.

PROXY: Internally used by some table types to access other tables from one table.

External Table Specification

The four main external table types – odbc, jdbc, mongo and mysql – are specified giving the following information:

1. The data source. This is specified in the connection option.
2. The remote table or view to access. This can be specified within the connection string or using specific CONNECT options.
3. The column definitions. This can be also left to CONNECT to find them using the discovery MariaDB feature.
4. The optional Quoted option. Can be set to 1 to quote the identifiers in the query sent to the remote server. This is required if columns or table names can contain blanks.

The way this works is by establishing a connection to the external data source and by sending it an SQL statement (or its equivalent using API functions for MONGO) enabling it to execute the original query. To enhance performance, it is necessary to have the remote data source do the maximum processing. This is needed in particular to reduce the amount of data returned by the data source.

This is why, for SELECT queries, CONNECT uses the cond_push MariaDB feature to retrieve the maximum of the where clause of the original query that can be added to the query sent to the data source. This is automatic and does not require anything to be done by the user.

However, more can be done. In addition to accessing a remote table, CONNECT offers the possibility to specify what the remote server must do. This is done by specifying it as a view in the srcdef option. For example:

```
CREATE TABLE custnum ENGINE=CONNECT TABLE_TYPE=XXX
CONNECTION='connecton string'
SRCDEF='select pays as country, count(*) as customers from custnum group by pays';
```

Doing so, the group by clause will be done by the remote server considerably reducing the amount of data sent back on the connection.

This may even be increased by adding to the srcdef part of the “compatible” part of the query where clauses like this are done for table-based tables. Note that for MariaDB, this table has two columns, country and customers. Supposing the original query is:

```
SELECT * FROM custnum WHERE (country = 'UK' OR country = 'USA') AND customers > 5;
```

How can we make the where clause be added to the sent srcdef? There are many problems:

1. Where to include the additional information.
2. What about the use of alias.

3. How to know what will be a where clause or a having clause.

The first problem is solved by preparing the srcdef view to receive clauses. The above example srcdef becomes:

```
SRCDEF='select pays as country, count(*) as customers from custnum where %s group by pays having
%s';
```

The *%s* in the srcdef are place holders for eventual compatible parts of the original query where clause. If the select query does not specify a where clause, or a gives an unacceptable where clause, place holders will be filled by dummy clauses (1=1).

The other problems must be solved by adding to the create table a list of columns that must be translated because they are aliases or/and aliases on aggregate functions that must become a having clause. For example, in this case:

```
CREATE TABLE custnum ENGINE=CONNECT TABLE_TYPE=XXX
CONNECTION='connecton string'
SRCDEF='select pays as country, count(*) as customers from custnum where %s group by pays having
%s'
OPTION_LIST='Alias=customers=*count(*);country=pays';
```

This is specified by the alias option, to be used in the option list. It is made of a semi-colon separated list of items containing:

1. The local column name (alias in the remote server)
2. An equal sign.
3. An eventual '*' indicating this is column correspond to an aggregate function.
4. The remote column name.

With this information, CONNECT will be able to make the query sent to the remote data source:

```
select pays as country, count(*) as customers from custnum where (pays = 'UK' OR pays = 'USA')
group by country having count(*) > 5
```

Note: Some data sources, including MySQL and MariaDB, accept aliases in the having clause. In that case, the alias option could have been specified as:

```
OPTION_LIST='Alias=customers=*;country=pays';
```

Another option exists, phpos, enabling to specify what place holders are present and in what order. To be specified as "W", "WH", "H", or "HW". It is rarely used because by default CONNECT can set it from the srcdef content. The only cases it is needed is when the srcdef contains only a having place holder or when the having place holder occurs before the where place holder, which can occur on queries containing joins. CONNECT cannot handle more than one place holder of each type.

SRCDEF is not available for MONGO tables, but other ways of achieving this exist and are described in the MONGO table type chapter.

5.3.7.6.16 CONNECT ODBC Table Type: Accessing Tables From Another DBMS

Contents

ODBC (Open Database Connectivity) is a standard API for accessing database management systems (DBMS). CONNECT uses this API to access data contained in other DBMS without having to implement a specific application for each one. An exception is the access to MySQL that should be done using the MYSQL table type.

> Note: On Linux, unixODBC must be installed.

These tables are given the type ODBC. For example, if a "Customers" table is contained in an Access ™ database you can define it with a command such as:

```
create table Customer (
  CustomerID varchar(5),
  CompanyName varchar(40),
  ContactName varchar(30),
  ContactTitle varchar(30),
  Address varchar(60),
  City varchar(15),
  Region varchar(15),
  PostalCode varchar(10),
  Country varchar(15),
  Phone varchar(24),
  Fax varchar(24))
engine=connect table_type=ODBC block_size=10
tabname='Customers'
Connection='DSN=MS Access Database;DBQ=C:/Program
Files/Microsoft Office/Office/1033/FPNWIND.MDB;';
```

Tabname option defaults to the table name. It is required if the source table name is different from the name of the CONNECT table. Note also that for some data sources this name is case sensitive.

Often, because CONNECT can retrieve the table description using ODBC catalog functions, the column definitions can be unspecified. For instance this table can be simply created as:

```
create table Customer engine=connect table_type=ODBC
  block_size=10 tabname='Customers'
  Connection='DSN=MS Access Database;DBQ=C:/Program Files/Microsoft Office/Office/1033/FPNWIND.MD
```

The `BLOCK_SIZE` specification will be used later to set the RowsetSize when retrieving rows from the ODBC table. A reasonably large RowsetSize can greatly accelerate the fetching process.

If you specify the column description, the column names of your table must exist in the data source table. However, you are not obliged to define all the data source columns and you can change the order of the columns. Some type conversion can also be done if appropriate. For instance, to access the FireBird sample table EMPLOYEE, you could define your table as:

```
create table empodbc (
  EMP_NO smallint(5) not null,
  FULL_NAME varchar(37) not null),
  PHONE_EXT varchar(4) not null,
  HIRE_DATE date,
  DEPT_NO smallint(3) not null,
  JOB_COUNTRY varchar(15),
  SALARY double(12,2) not null)
engine=CONNECT table_type=ODBC tabname='EMPLOYEE'
connection='DSN=firebird';
```

This definition ignores the FIRST_NAME, LAST_NAME, JOB_CODE, and JOB_GRADE columns. It places the FULL_NAME last column of the original table in second position. The type of the HIRE_DATE column was changed from *timestamp* to *date* and the type of the DEPT_NO column was changed from *char* to *integer*.

Currently, some restrictions apply to ODBC tables:

1. Cursor type is forward only (sequential reading).
2. No indexing of ODBC tables (do not specify any columns as key). However, because CONNECT can often add a where clause to the query sent to the data source, indexing will be used by the data source if it supports it. (Remote indexing is available with version 1.04, released with MariaDB 10.1.6)
3. CONNECT ODBC supports SELECT and INSERT. UPDATE and DELETE are also supported in a somewhat restricted way (see below). For other operations, use an ODBC table with the EXECSRC option (see below) to directly send proper commands to the data source.

Random Access of ODBC Tables

In CONNECT version 1.03 (until MariaDB 10.1.5) ODBC tables are not indexable. Version 1.04 (from MariaDB 10.1.6) adds remote indexing facility to the ODBC table type.

However, some queries require random access to an ODBC table; for instance when it is joined to another table or used in an order by queries applied to a long column or large tables.

There are several ways to enable random (position) access to a CONNECT ODBC table. They are dependant on the following table options:

Option	Type	Used For
Block_Size	Integer	Specifying the rowset size.
Memory*	Integer	Storing the result set in memory.
Scrollable*	Boolean	Using a scrollable cursor.

* - To be specified in the option_list.

When dealing with small tables, the simpler way to enable random access is to specify a rowset size equal or larger than the table size (or the result set size if a push down where clause is used). This means that the whole result is in memory on the first fetch and CONNECT will use it for further positional accesses.

Another way to have the result set in memory is to use the memory option. This option can be set to the following values:

0. No memory used (the default). Best when the table is read sequentially as in SELECT statements with only eventual WHERE clauses.
1. Memory size required is calculated during the first sequential table read. The allocated memory is filled during the second sequential read. Then the table rows are retrieved from the memory. This should be used when the table will be accessed several times randomly, such as in sub-selects or being the target table of a join.
2. A first query is executed to get the result set size and the needed memory is allocated. It is filled on the first sequential reading. Then random access of the table is possible. This can be used in the case of ORDER BY clauses, when MariaDB uses position reading.

Note that the best way to handle ORDER BY is to set the max_length_for_sort_data variable to a larger value (its default value is 1024 that is pretty small). Indeed, it requires less memory to be used, particularly when a WHERE clause limits the retrieved data set. This is because in the case of an order by query, MariaDB firstly retrieves the sequentially the result set and the position of each records. Often the sort can be done from the result set if it is not too big. But if too big, or if it implies some “long” columns, only the positions are sorted and MariaDB retrieves the final result from the table read in random order. If setting the max_length_for_sort_data variable is not feasible or does not work, to be able to retrieve table data from memory after the first sequential read, the memory option must be set to 2.

For tables too large to be stored in memory another possibility is to make your table to use a scrollable cursor. In this case each randomly accessed row can be retrieved from the data source specifying its cursor position, which is reasonably fast. However, scrollable cursors are not supported by all data sources.

With CONNECT version 1.04 (from MariaDB 10.1.6), another way to provide random access is to specify some columns to be indexed. This should be done only when the corresponding column of the source table is also indexed. This should be used for tables too large to be stored in memory and is similar to the remote indexing used by the MYSQL table type and by the FEDERATED engine.

There remains the possibility to extract data from the external table and to construct another table of any file format from the data source. For instance to construct a fixed formatted DOS table containing the CUSTOMER table data, create the table as

```
create table Custfix engine=connect File_name='customer.txt'
  table_type=fix block_size=20 as select * from customer;
```

Now you can use *custfix* for fast database operations on the copied *customer* table data.

Retrieving data from a spreadsheet

ODBC can also be used to create tables based on tabular data belonging to an Excel spreadsheet:

```
create table XLCONT
engine=CONNECT table_type=ODBC tabname='CONTACT'
Connection='DSN=Excel Files;DBQ=D:/Ber/Doc/Contact_BP.xls;';
```

This supposes that a tabular zone of the sheet including column headers is defined as a table named CONTACT or using a “named reference”. Refer to the Excel documentation for how to specify tables inside sheets. Once done, you can ask:

```
select * from xlcont;
```

This will extract the data from Excel and display:

Nom	Fonction	Societe
Boisseau Frederic		9 Telecom
Martelliere Nicolas		Vidal SA (Groupe UBM)
Remy Agathe		Price Minister
Du Halgouet Tanguy		Danone
Vandamme Anna		GDF
Thomas Willy		Europ Assistance France
Thomas Dominique		Acoss (DG des URSSAF)
Thomas Berengere	Responsable SI Decisionnel	DEXIA Credit Local
Husy Frederic	Responsable Decisionnel	Neuf Cegetel
Lemonnier Nathalie	Directeur Marketing Client	Louis Vuitton
Louis Loic	Reporting International Decisionnel	Accor
Menseau Eric		Orange France

Here again, the columns description was left to CONNECT when creating the table.

Multiple ODBC tables

The concept of multiple tables can be extended to ODBC tables when they are physically represented by files, for instance to Excel or Access tables. The condition is that the connect string for the table must contain a field DBQ=*filename*, in which wildcard characters can be included as for multiple=1 tables in their filename. For instance, a table contained in several Excel files CA200401.xls, CA200402.xls, ...CA200412.xls can be created by a command such as:

```
create table ca04mul (Date char(19), Operation varchar(64),
  Debit double(15,2), Credit double(15,2))
engine=CONNECT table_type=ODBC multiple=1
qchar= '"' tabname='bank account'
connection='DSN=Excel Files;DBQ=D:/Ber/CA/CA2004*.xls;';
```

Providing that in each file the applying information is internally set for Excel as a table named "bank account". This

extension to ODBC does not support *multiple*=2. The *qchar* option was specified to make the identifiers quoted in the select statement sent to ODBC, in particular the when the table or column names contain blanks, to avoid SQL syntax errors.

Caution: Avoid accessing tables belonging to the currently running MariaDB server via the MySQL ODBC connector. This may not work and may cause the server to be restarted.

Performance consideration

To avoid extracting entire tables from an ODBC source, which can be a lengthy process, CONNECT extracts the "compatible" part of query WHERE clauses and adds it to the ODBC query. Compatible means that it must be understood by the data source. In particular, clauses involving scalar functions are not kept because the data source may have different functions than MariaDB or use a different syntax. Of course, clauses involving sub-select are also skipped. This will transfer eventual indexing to the data source.

Take care with clauses involving string items because you may not know whether they are treated by the data source as case sensitive or case insensitive. If in doubt, make your queries as if the data source was processing strings as case sensitive to avoid incomplete results.

Using ODBC Tables inside correlated sub-queries

Unlike not correlated subqueries that are executed only once, correlated subqueries are executed many times. It is what ODBC calls a "requery". Several methods can be used by CONNECT to deal with this depending on the setting of the MEMORY or SCROLLABLE Boolean options:

Option	Description
Default	Implementing "requery" by discarding the current result set and re submitting the query (as MFC does)
Memory=1 or 2	Storing the result set in memory as MYSQL tables do.
Scrollable=Yes	Using a scrollable cursor.

Note: the MEMORY and SCROLLABLE options must be specified in the OPTION _ LIST.

Because the table is accessed several times, this can make queries last very long except for small tables and is almost unacceptable for big tables. However, if it cannot be avoided, using the memory method is the best choice and can be more than four times faster than the default method. If it is supported by the driver, using a scrollable cursor is slightly slower than using memory but can be an alternative to avoid memory problems when the sub-query returns a huge result set.

If the result set is of reasonable size, it is also possible to specify the block_size option equal or slightly larger than the result set. The whole result set being read on the first fetch, can be accessed many times without having to do anything else.

Another good workaround is to replace within the correlated sub-query the ODBC table by a local copy of it because MariaDB is often able to optimize the query and to provide a very fast execution.

Accessing specified views

Instead of specifying a source table name via the TABNAME option, it is possible to retrieve data from a “view” whose definition is given in a new option SRCDEF. For instance:

```
CREATE TABLE custnum (
  country varchar(15) NOT NULL,
  customers int(6) NOT NULL)
ENGINE=CONNECT TABLE_TYPE=ODBC BLOCK_SIZE=10
CONNECTION='DSN=MS Access Database;DBQ=C:/Program Files/Microsoft Office/Office/1033/FPNWIND.MDB;
SRCDEF='select country, count(*) as customers from customers group by country';
```

Or simply, because CONNECT can retrieve the returned column definition:

```
CREATE TABLE custnum ENGINE=CONNECT TABLE_TYPE=ODBC BLOCK_SIZE=10
CONNECTION='DSN=MS Access Database;DBQ=C:/Program Files/Microsoft Office/Office/1033/FPNWIND.MDB;
SRCDEF='select country, count(*) as customers from customers group by country';
```

Then, when executing for instance:

```
select * from custnum where customers > 3;
```

The processing of the group by is done by the data source, which returns only the generated result set on which only the where clause is performed locally. The result:

country	customers
Brazil	9
France	11
Germany	11
Mexico	5
Spain	5
UK	7
USA	13
Venezuela	4

This makes possible to let the data source do complicated operations, such as joining several tables or executing procedures returning a result set. This minimizes the data transfer through ODBC.

Data Modifying Operations

The only data modifying operations are the INSERT , UPDATE and DELETE commands. They can be executed successfully only if the data source database or tables are not read/only.

INSERT Command

When inserting values to an ODBC table, local values are used and sent to the ODBC table. This does not make any difference when the values are constant but in a query such as:

```
insert into t1 select * from t2;
```

Where t1 is an ODBC table, t2 is a locally defined table that must exist on the local server. Besides, it is a good way to create a distant ODBC table from local data.

CONNECT does not directly support INSERT commands such as:

```
insert into t1 values(2,'Deux') on duplicate key update msg = 'Two';
```

Sure enough, the “on duplicate key update” part of it is ignored, and will result in error if the key value is duplicated.

UPDATE and DELETE Commands

Unlike the INSERT command, UPDATE and DELETE are supported in a simplified way. Only simple table commands are supported; CONNECT does not support multi-table commands, commands sent from a procedure, or issued via a trigger. These commands are just rephrased to correspond to the data source syntax and sent to the data source for execution. Let us suppose we created the table:

```
create table tolite (
  id int(9) not null,
  nom varchar(12) not null,
  nais date default null,
  rem varchar(32) default null)
ENGINE=CONNECT TABLE_TYPE=ODBC tabname='lite'
CONNECTION='DSN=SQLite3 Datasource;Database=test.sqlite3'
CHARSET=utf8 DATA_CHARSET=utf8;
```

We can populate it by:

```
insert into tolite values(1,'Toto',now(),'First'),
(2,'Foo','2012-07-14','Second'),(4,'Machin','1968-05-30','Third');
```

The function `now()` will be executed by MariaDB and it returned value sent to the ODBC table.

Let us see what happens when updating the table. If we use the query:

```
update tolite set nom = 'Gillespie' where id = 10;
```

CONNECT will rephrase the command as:

```
update lite set nom = 'Gillespie' where id = 10;
```

What it did is just to replace the local table name with the remote table name and change all the back ticks to blanks or to the data source identifier quoting characters if QUOTED is specified. Then this command will be sent to the data source to be executed by it.

This is simpler and can be faster than doing a positional update using a cursor and commands such as “select ... for update of ...” that are not supported by all data sources. However, there are some restrictions that must be understood due to the way it is handled by MariaDB.

1. MariaDB does not know about all the above. The command will be parsed as if it were to be executed locally. Therefore, it must respect the MariaDB syntax.
2. Being executed by the data source, the (rephrased) command must also respect the data source syntax.
3. All data referenced in the SET and WHERE clause belongs to the data source.

This is possible because both MariaDB and the data source are using the SQL language. But you must use only the basic features that are part of the core SQL language. For instance, keywords like IGNORE or LOW_PRIORITY will cause syntax error with many data source.

Scalar function names also can be different, which severely restrict the use of them. For instance:

```
update tolite set nais = now() where id = 2;
```

This will not work with SQLite3, the data source returning an “unknown scalar function” error message. Note that in this particular case, you can rephrase it to:

```
update tolite set nais = date('now') where id = 2;
```

This understood by both parsers, and even if this function would return NULL executed by MariaDB, it does return the current date when executed by SQLite3. But this begins to become too trickery so to overcome all these restrictions, and permit to have all types of commands executed by the data source, CONNECT provides a specific ODBC table subtype described now.

Sending commands to a Data Source

This can be done using a special subtype of ODBC table. Let us see this in an example:

```
create table crlite (
  command varchar(128) not null,
  number int(5) not null flag=1,
  message varchar(255) flag=2)
engine=connect table_type=odbc
connection='Driver=SQLite3 ODBC Driver;Database=test.sqlite3;NoWCHAR=yes'
option_list='Execsrc=1';
```

The key points in this create statement are the EXECSRC option and the column definition.

The EXECSRC option tells that this table will be used to send a command to the data source. Most of the sent commands do not return result set. Therefore, the table columns are used to specify the command to be executed and to get the result of the execution. The name of these columns can be chosen arbitrarily, their function coming from the FLAG value:

Flag=0: The command to execute.

Flag=1: The affected rows, or -1 in case of error, or the result number of column if the command returns a result set.

Flag=2: The returned (eventually error) message.

How to use this table and specify the command to send? By executing a command such as:

```
select * from crlite where command = 'a command';
```

This will send the command specified in the WHERE clause to the data source and return the result of its execution. The syntax of the WHERE clause must be exactly as shown above. For instance:

```
select * from crlite where command =
'CREATE TABLE lite (
ID integer primary key autoincrement,
name char(12) not null,
birth date,
rem varchar(32))';
```

This command returns:

command	number	message
CREATE TABLE lite (ID integer primary key autoincrement, name...	0	Affected rows

Now we can create a standard ODBC table on the newly created table:

```
CREATE TABLE tlite
ENGINE=CONNECT TABLE_TYPE=ODBC tabname='lite'
CONNECTION='Driver=SQLite3 ODBC Driver;Database=test.sqlite3;NoWCHAR=yes'
CHARSET=utf8 DATA_CHARSET=utf8;
```

We can populate it directly using the supported INSERT statement:

```
insert into tlite(name,birth) values('Toto','2005-06-12');
insert into tlite(name,birth,rem) values('Foo',NULL,'No ID');
insert into tlite(name,birth) values('Truc','1998-10-27');
insert into tlite(name,birth,rem) values('John','1968-05-30','Last');
```

And see the result:

```
select * from tlite;
```

ID	name	birth	rem
1	Toto	2005-06-12	NULL
2	Foo	NULL	No ID
3	Truc	1998-10-27	NULL
4	John	1968-05-30	Last

Any command, for instance UPDATE, can be executed from the *crlite* table:

```
select * from crlite where command =
'update lite set birth = ''2012-07-14'' where ID = 2';
```

This command returns:

command	number	message
update lite set birth = '2012-07-15' where ID = 2	1	Affected rows

Let us verify it:

```
select * from tlite where ID = 2;
```

ID	name	birth	rem
2	Foo	2012-07-15	No ID

The syntax to send a command is rather strange and may seem unnatural. It is possible to use an easier syntax by defining a stored procedure such as:

```
create procedure send_cmd(cmd varchar(255))
MODIFIES SQL DATA
select * from crlite where command = cmd;
```

Now you can send commands like this:

```
call send_cmd('drop tlite');
```

This is possible only when sending one single command.

Sending several commands together

Grouping commands uses an easier syntax and is faster because only one connection is made for the all of them. To send several commands in one call, use the following syntax:

```
select * from crlite where command in (
  'update lite set birth = ''2012-07-14'' where ID = 2',
  'update lite set birth = ''2009-08-10'' where ID = 3');
```

When several commands are sent, the execution stops at the end of them or after a command that is in error. To continue after *n* errors, set the option maxerr=*n* (0 by default) in the option list.

Note 1: It is possible to specify the SRCDEF option when creating an EXECSRC table. It will be the command sent by default when a WHERE clause is not specified.

Note 2: Most data sources do not allow sending several commands separated by semi-colons.

Note 3: Quotes inside commands must be escaped. This can be avoided by using a different quoting character than the one used in the command

Note 4: The sent command must obey the data source syntax.

Note 5: Sent commands apply in the specified database. However, they can address any table within this database, or belonging to another database using the name syntax *schema.tabname*.

Connecting to a Data Source

There are two ways to establish a connection to a data source:

1. Using SQLDriverConnect and a Connection String
2. Using SQLConnect and a Data Source Name (DSN)

The first way uses a Connection String whose components describe what is needed to establish the connection. It is the most complete way to do it and by default CONNECT uses it.

The second way is a simplified way in which ODBC is just given the name of a DSN that must have been defined to ODBC or UnixOdbc and that contains the necessary information to establish the connection. Only the user name and password can be specified out of the DSN specification.

Defining the Connection String

Using the first way, the connection string must be specified. This is sometimes the most difficult task when creating ODBC tables because, depending on the operating system and the data source, this string can widely differ.

The format of the ODBC Connection String is:

```
connection-string::= empty-string[;] | attribute[;] | attribute; connection-string
empty-string ::=
attribute ::= attribute-keyword=attribute-value | DRIVER=[{]attribute-value[}]
attribute-keyword ::= DSN | UID | PWD | driver-defined-attribute-keyword
attribute-value ::= character-string
driver-defined-attribute-keyword = identifier
```

Where character-string has zero or more characters; identifier has one or more characters; attribute- keyword is not case-sensitive; attribute-value may be case-sensitive; and the value of the DSN keyword does not consist solely of blanks. Due to the connection string grammar, keywords and attribute values that contain the characters `[]{}(),;?*=!@` should be avoided. The value of the DSN keyword cannot consist only of blanks, and should not contain leading blanks. Because of the grammar of the system information, keywords and data source names cannot contain the backslash (\) character. Applications do not have to add braces around the attribute value after the DRIVER keyword unless the attribute contains a

semicolon (;), in which case the braces are required. If the attribute value that the driver receives includes the braces, the driver should not remove them, but they should be part of the returned connection string.

ODBC Defined Connection Attributes

The ODBC defined attributes are:

- DSN - the name of the data source to connect to. You must create this before attempting to refer to it. You create new DSNs through the ODBC Administrator (Windows), ODBCAdmin (unixODBC's GUI manager) or in the odbc.ini file.
- DRIVER - the name of the driver to connect to. You can use this in DSN-less connections.
- FILEDSN - the name of a file containing the connection attributes.
- UID/PWD - any username and password the database requires for authentication.
- SAVEFILE - request the DSN attributes are saved in this file.

Other attributes are DSN dependent attributes. The connection string can give the name of the driver in the DRIVER field or the data source in the DSN field (attention! meet the spelling and case) and has other fields that depend on the data source. When specifying a file, the DBQ field must give the **full** path and name of the file containing the table. Refer to the specific ODBC connector documentation for the exact syntax of the connection string.

Using a Predefined DSN

This is done by specifying in the option list the Boolean option “UseDSN” as yes or 1. In addition, string options “user” and “password” can be optionally specified in the option list.

When doing so, the connection string just contains the name of the predefined Data Source. For instance:

```
CREATE TABLE tlite ENGINE=CONNECT TABLE_TYPE=ODBC tabname='lite'
CONNECTION='SQLite3 Datasource'
OPTION_LIST='UseDSN=Yes,User=me,Password=mypass';
```

Note: the connection data source name (limited to 32 characters) should not be preceded by “DSN=”.

ODBC Tables on Linux/Unix

In order to use ODBC tables, you will need to have unixODBC installed. Additionally, you will need the ODBC driver for your foreign server's protocol. For example, for MS SQL Server or Sybase, you will need to have FreeTDS installed.

Make sure the user running mysqld (usually the mysql user) has permission to the ODBC data source configuration and the ODBC drivers. If you get an error on Linux/Unix when using TABLE_TYPE=ODBC:

```
Error Code: 1105 [unixODBC][Driver Manager]Can't open lib
'/usr/cachesys/bin/libcacheodbc.so' : file not found
```

You must make sure that the user running mysqld (usually "mysql") has enough permission to load the ODBC driver library. It can happen that the driver file does not have enough read privileges (use chmod to fix this), or loading is prevented by SELinux configuration (see below).

Try this command in a shell to check if the driver had enough permission:

```
sudo -u mysql ldd /usr/cachesys/bin/libcacheodbc.so
```

SELinux

SELinux can cause various problems. If you think SELinux is causing problems, check the system log (e.g. /var/log/messages) or the audit log (e.g. /var/log/audit/audit.log).

mysqld can't load some executable code, so it can't use the ODBC driver.

Example error:

```
Error Code: 1105 [unixODBC][Driver Manager]Can't open lib
'/usr/cachesys/bin/libcacheodbc.so' : file not found
```

Audit log:

```
type=AVC msg=audit(1384890085.406:76): avc: denied { execute }
for pid=1433 comm="mysqld"
path="/usr/cachesys/bin/libcacheodbc.so" dev=dm-0 ino=3279212
scontext=unconfined_u:system_r:mysqld_t:s0
tcontext=unconfined_u:object_r:usr_t:s0 tclass=file
```

mysqld can't open TCP sockets on some ports, so it can't connect to the foreign server.

Example error:

```
ERROR 1296 (HY000): Got error 174 '[unixODBC][FreeTDS][SQL Server]Unable to connect to data
source' from CONNECT
```

Audit log:

```
type=AVC msg=audit(1423094175.109:433): avc:  denied  { name_connect } for  pid=3193
comm="mysqld" dest=1433 scontext=system_u:system_r:mysqld_t:s0
tcontext=system_u:object_r:mssql_port_t:s0 tclass=tcp_socket
```

ODBC Catalog Information

Depending on the version of the used ODBC driver, some additional information on the tables are existing, such as table QUALIFIER or OWNER for old versions, now named CATALOG or SCHEMA since version 3.

CATALOG is apparently rarely used by most data sources, but SCHEMA (formerly OWNER) is and corresponds to the DATABASE information of MySQL.

The issue is that if no schema name is specified, some data sources return information for all schemas while some others only return the information of the “default” schema. In addition, the used “schema” or “database” is sometimes implied by the connection string and sometimes is not. Sometimes, it also can be included in a data source definition.

CONNECT offers two ways to specify this information:

1. When specified, the DBNAME create table option is regarded by ODBC tables as the SCHEMA name.
2. Table names can be specified as “*cat.sch.tab*” allowing to set the catalog and schema info.

When both are used, the qualified table name has precedence over DBNAME . For instance:

Tabname	DBname	Description
test.t1		The t1 table of the test schema.
test.t1	mydb	The t1 table of the test schema (test has precedence)
t1	mydb	The t1 table of the mydb schema
%.%.%		All tables in all catalogs and all schemas
t1		The t1 table in the default or all schema depending on the DSN
%.t1		The t1 table in all schemas for all DSN
test.%		All tables in the test schema

When creating a standard ODBC table, you should make sure only one source table is specified. Specifying more than one source table must be done only for CONNECT catalog tables (with CATFUNC=tables or columns).

In particular, when column definition is left to the Discovery feature, if tables with the same name are present in several schemas and the schema name is not specified, several columns with the same name will be generated. This will make the creation fail with a not very explicit error message.

Note: With some ODBC drivers, the DBNAME option or qualified table name is useless because the schema implied by the connection string or the definition of the data source has priority over the specified DBNAME .

Table name case

Another issue when dealing with ODBC tables is the way table and column names are handled regarding of the case.

For instance, Oracle follows to the SQL standard here. It converts non-quoted identifiers to upper case. This is correct and expected. PostgreSQL is not standard. It converts identifiers to lower case. MySQL/MariaDB is not standard. They preserve identifiers on Linux, and convert to lower case on Windows.

Think about that if you fail to see a table or a column on an ODBC data source.

Non-ASCII Character Sets with Oracle

When connecting through ODBC, the MariaDB Server operates as a client to the foreign database management system. As such, it requires that you configure MariaDB as you would configure native clients for the given database server.

In the case of connecting to Oracle, when using non-ASCI character sets, you need to properly set the NLS_LANG environment variable before starting the MariaDB Server.

For instance, to test this on Oracle, create a table that contains a series of special characters:

```
CREATE TABLE t1 (letter VARCHAR(4000));

INSERT INTO t1 VALUES
   (UTL_RAW.CAST_TO_VARCHAR2(HEXTORAW('C4'))),
   (UTL_RAW.CAST_TO_VARCHAR2(HEXTORAW('C5'))),
   (UTL_RAW.CAST_TO_VARCHAR2(HEXTORAW('C6')));

SELECT letter, RAWTOHEX(letter) FROM t1;

letter | RAWTOHEX(letter)
-------|-----------------
Ä      | C4
Å      | C5
Æ      | C6
```

Then create a connecting table on MariaDB and attempt the same query:

```
CREATE TABLE t1 (
   letter VARCHAR(4000))
ENGINE=CONNECT
DEFAULT CHARSET=utf8mb4
CONNECTION='DSN=YOUR_DSN'
TABLE_TYPE = 'ODBC'
DATA_CHARSET = latin1
TABNAME = 'YOUR_SCHEMA.T1';

SELECT letter, HEX(letter) FROM t1;

+--------+-------------+
| letter | HEX(letter) |
+--------+-------------+
| A      |       41 |
| ?      |       3F |
| ?      |       3F |
+--------+-------------+
```

While the character set is defined in a way that satisfies MariaDB, it has not been defined for Oracle, (that is, setting the NLS_LANG environment variable). As a result, Oracle is not providing the characters you want to MariaDB and Connect. The specific method of setting the NLS_LANG variable can vary depending on your operating system or distribution. If you're experiencing this issue, check your OS documentation for more details on how to properly set environment variables.

Using systemd

With Linux distributions that use systemd, you need to set the environment variable in the service file, (systemd doesn't read from the /etc/environment file).

This is done by setting the Environment variable in the [Service] unit. For instance,

```
# systemctl edit mariadb.service

[Service]
Environment=NLS_LANG=GERMAN_GERMANY.WE8ISO8859P1
```

Then restart MariaDB,

```
# systemctl restart mariadb.service
```

You can now retrieve the appropriate characters from Oracle tables:

```
SELECT letter, HEX(letter) FROM t1;

+--------+-------------+
| letter | HEX(letter) |
+--------+-------------+
| Ä      | C384        |
| Å      | C385        |
| Æ      | C386        |
+--------+-------------+
```

Using Windows

Microsoft Windows doesn't ignore environment variables the way systemd does on Linux, but it does require that you set the NLS_LANG environment variable on your system. In order to do so, you need to open an elevated command-prompt, (that is, Cmd.exe with administrative privileges).

From here, you can use the Setx command to set the variable. For instance,

```
Setx NLS_LANG GERMAN_GERMANY.WE8ISO8859P1 /m
```

Note: For more detail about this, see MDEV-17501.

`OPTION_LIST` Values Supported by the ODBC Tables

The following options can be given as comma-separated string to the `OPTION_LIST` value in the `CREATE TABLE` statement.

Name	Default	Description
MaxRes	0	Maximum number of rows returned by catalog functions
ConnectTimeout	-1	Connection timeout in seconds, unlimited by default
QueryTimeout	-1	Query timeout in seconds, unlimited by default
UseDSN	false	Use pre-configured DSN

5.3.7.6.17 CONNECT JDBC Table Type: Accessing Tables from Another DBMS

Contents

The JDBC table type should be distributed with all recent versions of MariaDB. However, if the automatic compilation of it is possible after the java JDK was installed, the complete distribution of it is not fully implemented in older versions. The distributed JdbcInterface.jar file contains the JdbcInterface wrapper only. New versions distribute a JavaWrappers.jar that contains all currently existing wrappers.

This will require that:

1. The Java SDK is installed on your system.

2. The java wrapper class files are available on your system.
3. And of course, some JDBC drivers exist to be used with the matching DBMS.

Point 2 was made automatic in the newest versions of MariaDB.

Compiling From Source Distribution

Even when the Java JDK has been installed, CMake sometimes cannot find the location where it stands. For instance on Linux the Oracle Java JDK package might be installed in a path not known by the CMake lookup functions causing error message such as:

```
CMake Error at /usr/share/cmake/Modules/FindPackageHandleStandardArgs.cmake:148 (message):
  Could NOT find Java (missing: Java_JAR_EXECUTABLE Java_JAVAC_EXECUTABLE
  Java_JAVAH_EXECUTABLE Java_JAVADOC_EXECUTABLE)
```

When this happen, provide a Java prefix as a hint on where the package was loaded. For instance on Ubuntu I was obliged to enter:

```
export JAVA_HOME=/usr/lib/jvm/java-8-oracle
```

After that, the compilation of the CONNECT JDBC type was completed successfully.

Compiling the Java source files

They are the source of the java wrapper classes used to access JDBC drivers. In the source distribution, they are located in the CONNECT source directory.

The default wrapper, JdbcInterface, is the only one distributed with binary distribution. It uses the standard way to get a connection to the drivers via the DriverManager.getConnection method. Other wrappers, only available with source distribution, enable connection to a Data Source, eventually implementing pooling. However, they must be compiled and installed manually.

The available wrappers are:

Wrapper	Description
JdbcInterface	Used to make the connection with available drivers the standard way.
ApacheInterface	Based on the Apache common-dbcp2 package this interface enables making connections to DBCP data sources with any JDBC drivers.
MariadbInterface	Makes connection to a MariaDB data source.
MysqlInterface	Makes connection to a Mysql data source. Must be used with a MySQL driver that implements data sources.
OracleInterface	Makes connection to an Oracle data source.
PostgresqlInterface	Makes connection to a Postgresql data source.

The wrapper used by default is specified by the connect_java_wrapper session variable and is initially set to `wrappers/JdbcInterface`. The wrapper to use for a table can also be specified in the option list as a wrapper option of the “create table” statements.

Note: Conforming java naming usage, class names are preceded by the java package name with a slash separator. However, this is not mandatory for CONNECT which adds the package name if it is missing.

The JdbcInterface wrapper is always usable when Java is present on your machine. Binary distributions have this wrapper already compiled as a JdbcInterface.jar file installed in the plugin directory whose path is automatically included in the class path of the JVM. Recent versions also add a JavaWrappers.jar that contains all these wrappers. Therefore there is no need to worry about its path.

Compiling the ApacheInterface wrapper requires that the Apache common-DBCP2 package be installed. Other wrappers are to be used only with the matching JDBC drivers that must be available when compiling them.

Installing the jar file in the plugin directory is the best place because it is part of the class path. Depending on what is installed on your system, the source files can be reduced accordingly. To compile only the JdbcInterface.java file the CMAKE_JAVA_INCLUDE_PATH is not required. Here the paths are the ones existing on my Windows 7 machine and should be localized.

Setting the Required Information

Before any operation with a JDBC driver can be made, CONNECT must initialize the environment that will make working with Java possible. This will consist of:

1. Loading dynamically the JVM library module.
2. Creating the Java Virtual Machine.
3. Establishing contact with the java wrapper class.
4. Connecting to the used JDBC driver.

Indeed, the JVM library module is not statically linked to the CONNECT plugin. This is to make it possible to use a CONNECT plugin that has been compiled with the JDBC table type on a machine where the Java SDK is not installed. Otherwise, users not interested in the JDBC table type would be obliged to install the Java SDK on their machine to be able to load the CONNECT storage engine.

JVM Library Location

If the JVM library (jvm.dll on Windows, libjvm.so on Linux) was not placed in the standard library load path, CONNECT cannot find it and must be told where to search for it. This happens in particular on Linux when the Oracle Javapackage was installed in a private location.

If the JAVA_HOME variable was exported as explained above, CONNECT can sometimes find it using this information. Otherwise, its search path can be added to the LD_LIBRARY_PATH environment variable. But all this is complicated because making environment variables permanent on Linux is painful (many different methods must be used depending on the Linux version and the used shell).

This is why CONNECT introduced a new global variable connect_jvm_path to store this information. It can be set when starting the server as a command line option or even afterwards before the first use of the JDBC table type. For example:

```
set global connect_jvm_path="/usr/lib/jvm/java-8-oracle/jre/lib/i386/client"
```

or

```
set global connect_jvm_path="/usr/lib/jvm/java-8-oracle/jre/lib/i386/server"
```

The client library is smaller and faster for connection. The server library is more optimized and can be used in case of heavy load usage.

Note that this may not be required on Windows because the path to the JVM library can sometimes be found in the registry.

Once this library is loaded, CONNECT can create the required Java Virtual Machine.

Java Class Path

This is the list of paths Java searches when loading classes. With CONNECT, the classes to load will be the java wrapper classes used to communicate with the drivers , and the used JDBC driver classes that are grouped inside jar files. If the ApacheInterface wrapper must be used, the class path must also include all three jars used by the Apache package.

> Caution: This class path is passed as a parameter to the Java Virtual Machine (JVM) when creating it and cannot be modified as it is a read only property. In addition, because MariaDB is a multi-threading application, this JVM cannot be destroyed and will be used throughout the entire life of the MariaDB server. Therefore, be sure it is correctly set before you use the JDBC table type for the first time. Otherwise, there will be practically no alternative than to shut down the server and restart it.

The path to the wrapper classes must point to the directory containing the wrappers sub-directory. If a JdbcInterface.jar file was made, its path is the directory where it is located followed by the jar file name. It is unclear where because this will depend on the installation process. If you start from a source distribution, it can be in the storage/connect directory where the CONNECT source files are or where you moved them or compiled the JdbcInterface.jar file.

For binary distributions, there is nothing to do because the jar file has been installed in the mysql share directory whose path is always automatically included in the class path available to the JVM.

Remaining are the paths of all the installed JDBC drivers that you intend to use. Remember that their path must include the jar file itself. Some applications use an environment variable CLASSPATH to contain them. Paths are separated by ':' on Linux and by ';' on Windows.

If the CLASSPATH variable actually exists and if it is available inside MariaDB, so far so good. You can check this using an UDF function provided by CONNECT that returns environment variable values:

```
create function envar returns string soname 'ha_connect.so';
select envar('CLASSPATH');
```

Most of the time, this will return null or some required files are missing. This is why CONNECT introduced a global variable to store this information. The paths specified in this variable will be added and have precedence to the ones, if any, of the CLASSPATH environment variable. As for the jvm path, this variable connect_class_path should be specified when starting the server but can also be set before using the JDBC table type for the first time.

The current directory (sql/data) is also placed by CONNECT at the beginning of the class path.

As an example, here is how I start MariaDB when doing tests on Linux:

```
olivier@olivier-Aspire-8920:~$ sudo /usr/local/mysql/bin/mysqld -u root --console --default-
storage-engine=myisam --skip-innodb --connect_jvm_path="/usr/lib/jvm/java-8-
oracle/jre/lib/i386/server" --
connect_class_path="/home/olivier/mariadb/10.1/storage/connect:/media/olivier/SOURCE/mysql-
connector-java-6.0.2/mysql-connector-java-6.0.2-bin.jar"
```

CONNECT JDBC Tables

These tables are given the type JDBC. For instance, supposing you want to access the boys table located on and external local or remote database management system providing a JDBC connector:

```
create table boys (
name char(12),
city char(12),
birth date,
hired date);
```

To access this table via JDBC you can create a table such as:

```
create table jboys engine=connect table_type=JDBC tabname=boys
connection='jdbc:mysql://localhost/dbname?user=root';
```

The CONNECTION option is the URL used to establish the connection with the remote server. Its syntax depends on the external DBMS and in this example is the one used to connect as root to a MySQL or MariaDB local database using the MySQL JDBC connector.

As for ODBC, the columns definition can be omitted and will be retrieved by the discovery process. The restrictions concerning column definitions are the same as for ODBC.

Note: The dbname indicated in the URL corresponds for many DBMS to the catalog information. For MySQL and MariaDB it is the schema (often called database) of the connection.

Using a Federated Server

Alternatively, a JDBC table can specify its connection options via a Federated server. For instance, supposing you have a table accessing an external Postgresql table defined as:

```
create table juuid engine=connect table_type=JDBC tabname=testuuid
connection='jdbc:postgresql:test?user=postgres&password=pwd';
```

You can create a Federated server:

```
create server 'post1' foreign data wrapper 'postgresql' options (
HOST 'localhost',
DATABASE 'test',
USER 'postgres',
PASSWORD 'pwd',
PORT 0,
SOCKET '',
OWNER 'postgres');
```

Now the JDBC table can be created by:

```
create table juuid engine=connect table_type=JDBC connection='post1' tabname=testuuid;
```

or by:

```
create table juuid engine=connect table_type=JDBC connection='post1/testuuid';
```

In any case, the location of the remote table can be changed in the Federated server without having to alter all the tables using this server.

JDBC needs a URL to establish a connection. CONNECT was able to construct that URL from the information contained in such Federated server definition when the URL syntax is similar to the one of MySQL, MariaDB or Postgresql. However, other DBMSs such as Oracle use a different URL syntax. In this case, simply replace the HOST information by the required URL in the Federated server definition. For instance:

```
create server 'oracle' foreign data wrapper 'oracle' options (
HOST 'jdbc:oracle:thin:@localhost:1521:xe',
DATABASE 'SYSTEM',
USER 'system',
PASSWORD 'manager',
PORT 0,
SOCKET '',
OWNER 'SYSTEM');
```

Now you can create an Oracle table with something like this:

```
create table empor engine=connect table_type=JDBC connection='oracle/HR.EMPLOYEES';
```

Note: Oracle, as Postgresql, does not seem to understand the DATABASE setting as the table schema that must be specified in the Create Table statement.

Connecting to a JDBC driver

When the connection to the driver is established by the JdbcInterface wrapper class, it uses the options that are provided when creating the CONNECT JDBC tables. Inside the default Java wrapper, the driver's main class is loaded by the DriverManager.getConnection function that takes three arguments:

URL	That is the URL that you specified in the CONNECTION option.
User	As specified in the OPTION_LIST or NULL if not specified.
Password	As specified in the OPTION_LIST or NULL if not specified.

The URL varies depending on the connected DBMS. Refer to the documentation of the specific JDBC driver for a description of the syntax to use. User and password can also be specified in the option list.

Beware that the database name in the URL can be interpreted differently depending on the DBMS. For MySQL this is the schema in which the tables are found. However, for Postgresql, this is the catalog and the schema must be specified using the CONNECT dbname option.

For instance a table accessing a Postgresql table via JDBC can be created with a create statement such as:

```
create table jt1 engine=connect table_type=JDBC
connection='jdbc:postgresql://localhost/mtr' dbname=public tabname=t1
option_list='User=mtr,Password=mtr';
```

Note: In previous versions of JDBC, to obtain a connection, java first had to initialize the JDBC driver by calling the method Class.forName. In this case, see the documentation of your DBMS driver to obtain the name of the class that implements the interface java.sql.Driver. This name can be specified as an option DRIVER to be put in the option list. However, most modern JDBC drivers since version 4 are self-loading and do not require this option to be specified.

The wrapper class also creates some required items and, in particular, a statement class. Some characteristics of this statement will depend on the options specified when creating the table:

Scrollable	To be specified in the option list. Determines the cursor type: no= forward_only or yes=scroll_insensitive.
Block_size	Will be used to set the statement fetch size.

Fetch Size

The fetch size determines the number of rows that are internally retrieved by the driver on each interaction with the DBMS. Its default value depends on the JDBC driver. It is equal to 10 for some drivers but not for the MySQL or MariaDB

connectors.

The MySQL/MariaDB connectors retrieve all the rows returned by one query and keep them in a memory cache. This is generally fine in most cases, but not when retrieving a large result set that can make the query fail with a memory exhausted exception.

To avoid this, when accessing a big table and expecting large result sets, you should specify the BLOCK_SIZE option to 1 (the only acceptable value). However a problem remains:

Suppose you execute a query such as:

```
select id, name, phone from jbig limit 10;
```

Not knowing the limit clause, CONNECT sends to the remote DBMS the query:

```
SELECT id, name, phone FROM big;
```

In this query big can be a huge table having million rows. Having correctly specified the block size as 1 when creating the table, the wrapper just reads the 10 first rows and stops. However, when closing the statement, these MySQL/MariaDB drivers must still retrieve all the rows returned by the query. This is why, the wrapper class when closing the statement also cancels the query to stop that extra reading.

The bad news is that if it works all right for some previous versions of the MySQL driver, it does not work for new versions as well as for the MariaDB driver that apparently ignores the cancel command. The good news is that you can use an old MySQL driver to access MariaDB databases. It is also possible that this bug will be fixed in future versions of the drivers.

Connection to a Data Source

This is the java preferred way to establish a connection because a data source can keep a pool of connections that can be re-used when necessary. This makes establishing connections much faster once it was done for the first time.

CONNECT provide additional wrappers whose files are located in the CONNECT source directory. The wrapper to use can be specified in the global variable connect_java_wrapper, which defaults to “JdbcInterface”.

It can also be specified for a table in the option list by setting the option wrapper to its name. For instance:

```
create table jboys
engine=CONNECT table_type=JDBC tabname='boys'
connection='jdbc:mariadb://localhost/connect?user=root&useSSL=false'
option_list='Wrapper=MariadbInterface,Scrollable=1';
```

They can be used instead of the standard JdbcInterface and are using created data sources.

The Apache one uses data sources implemented by the Apache-commons-dbcp2 package and can be used with all drivers including those not implementing data sources. However, the Apache package must be installed and its three required jar files accessible via the class path.

1. commons-dbcp2-2.1.1.jar
2. commons-pool2-2.4.2.jar
3. commons-logging-1.2.jar

Note: the versions numbers can be different on your installation.

The other ones use data sources provided by the matching JDBC driver. There are currently four wrappers to be used with mysql-6.0.2, mariadb, oracle and postgresql.

Unlike the class path, the used wrapper can be changed even after the JVM machine was created.

Random Access to JDBC Tables

The same methods described for ODBC tables can be used with JDBC tables.

Note that in the case of the MySQL or MariaDB connectors, because they internally read the whole result set in memory, using the MEMORY option would be a waste of memory. It is much better to specify the use of a scrollable cursor when needed.

Other Operations with JDBC Tables

Except for the way the connection string is specified and the table type set to JDBC, all operations with ODBC tables are done for JDBC tables the same way. Refer to the ODBC chapter to know about:

- Accessing specified views (SRCDEF)
- Data modifying operations.
- Sending commands to a data source.
- JDBC catalog information.

Note: Some JDBC drivers fail when the global time_zone variable is ambiguous, which sometimes happens when it is set to SYSTEM. If so, reset it to a not ambiguous value, for instance:

```
set global time_zone = '+2:00';
```

JDBC Specific Restrictions

Connecting via data sources created externally (for instance using Tomcat) is not supported yet.

Other restrictions are the same as for the ODBC table type.

Handling the UUID Data Type

PostgreSQL has a native UUID data type, internally stored as BIN(16). This is neither an SQL nor a MariaDB data type. The best we can do is to handle it by its character representation.

UUID will be translated to CHAR(36) when column definitions are set using discovery. Locally a PostgreSQL UUID column will be handled like a CHAR or VARCHAR column. Example:

Using the PostgreSQL table testuuid in the text database:

```
 Table « public.testuuid »
 Column | Type | Default
--------+------+--------------------
 id     | uuid | uuid_generate_v4()
 msg    | text |
```

Its column definitions can be queried by:

```
create or replace table juuidcol engine=connect table_type=JDBC tabname=testuuid
catfunc=columns
connection='jdbc:postgresql:test?user=postgres&password=pwd';
```

```
select table_name "Table", column_name "Column", data_type "Type",
  type_name "Name", column_size "Size"
 from juuidcol;
```

This query returns:

Table	Column	Type	Name	Size
testuuid	id	1111	uuid	2147483647
testuuid	msg	12	text	2147483647

Note: PostgreSQL, when a column size is undefined, returns 2147483647, which is not acceptable for MariaDB. CONNECT change it to the value of the connect_conv_size session variable. Also, for TEXT columns the data type returned is 12 (SQL_VARCHAR) instead of -1 the SQL_TEXT value.

Accessing this table via JDBC by:

```
CREATE TABLE juuid ENGINE=connect TABLE_TYPE=JDBC TABNAME=testuuid
CONNECTION='jdbc:postgresql:test?user=postgres&password=pwd';
```

it will be created by discovery as:

```
CREATE TABLE `juuid` (
  `id` char(36) DEFAULT NULL,
  `msg` varchar(8192) DEFAULT NULL
) ENGINE=CONNECT DEFAULT CHARSET=latin1 CONNECTION='jdbc:postgresql:test?
user=postgres&password=pwd' `TABLE_TYPE`='JDBC' `TABNAME`='testuuid';
```

Note: 8192 being here the _connect_conv_size_ value.

Let's populate it:

```
insert into juuid(msg) values('First');
insert into juuid(msg) values('Second');
select * from juuid;
```

Result:

id	msg
4b173ee1-1488-4355-a7ed-62ba59c2b3e7	First
6859f850-94a7-4903-8d3c-fc3c874fc274	Second

Here the id column values come from the DEFAULT of the PostgreSQL column that was specified as uuid_generate_v4().

It can be set from MariaDB. For instance:

```
insert into juuid
  values('2f835fb8-73b0-42f3-a1d3-8a532b38feca','inserted');
insert into juuid values(NULL,'null');
insert into juuid values('','random');
select * from juuid;
```

Result:

id	msg
4b173ee1-1488-4355-a7ed-62ba59c2b3e7	First
6859f850-94a7-4903-8d3c-fc3c874fc274	Second
2f835fb8-73b0-42f3-a1d3-8a532b38feca	inserted
<null>	null
8fc0a30e-dc66-4b95-ba57-497a161f4180	random

The first insert specifies a valid UUID character representation. The second one set it to NULL. The third one (a void string) generates a Java random UUID. UPDATE commands obey the same specification.

These commands both work:

```
select * from juuid where id = '2f835fb8-73b0-42f3-a1d3-8a532b38feca';
delete from juuid where id = '2f835fb8-73b0-42f3-a1d3-8a532b38feca';
```

However, this one fails:

```
select * from juuid where id like '%42f3%';
```

Returning:

1296: Got error 174 'ExecuteQuery: org.postgresql.util.PSQLException: ERROR: operator does not exist: uuid ~ unknown hint: no operator corresponds to the data name and to the argument types.

because CONNECT cond_push feature added the WHERE clause to the query sent to PostgreSQL:

```
SELECT id, msg FROM testuuid WHERE id LIKE '%42f3%'
```

and the LIKE operator does not apply to UUID in PostgreSQL.

To handle this, a new session variable was added to CONNECT: connect_cond_push. It permits to specify if cond_push is enabled or not for CONNECT and defaults to 1 (enabled). In this case, you can execute:

```
set connect_cond_push=0;
```

Doing so, the where clause will be executed by MariaDB only and the query will not fail anymore.

Executing the JDBC tests

Four tests exist but they are disabled because requiring some work to localized them according to the operating system and

available java package and JDBC drivers and DBMS.

Two of them, jdbc.test and jdbc_new.test, are accessing MariaDB via JDBC drivers that are contained in a fat jar file that is part of the test. They should be executable without anything to do on Windows; simply adding the option –enable-disabled when running the tests.

However, on Linux these tests can fail to locate the JVM library. Before executing them, you should export the JAVA_HOME environment variable set to the prefix of the java installation or export the LD_LIBRARY_PATH containing the path to the JVM lib.

Fixing Problem With mariadb-dump

In some case or some platform, when CONNECT is set up for use with JDBC table types, this causes mariadb-dump with the option --all-databases to fail.

This was reported by Robert Dyas who found the cause - see the discussion at MDEV-11238.

5.3.7.6.18 CONNECT MONGO Table Type: Accessing Collections from MongoDB

Contents

Classified as a NoSQL database program, MongoDB uses JSON-like documents (BSON) grouped in collections. The MONGO type is used to directly access MongoDB collections as tables.

Accessing MongDB from CONNECT

Accessing MongoDB from CONNECT can be done in different ways:

1. As a MONGO table via the MongoDB C Driver.
2. As a MONGO table via the MongoDB Java Driver.
3. As a JDBC table using some commercially available MongoDB JDBC drivers.
4. As a JSON table via the MongoDB C or Java Driver.

Using the MongoDB C Driver

This is currently not available from binary distributions but only for versions compiled from source. The preferred version of the MongoDB C Driver is 1.7, because they provide package recognition. What must be done is:

1. Install libbson and the MongoDB C Driver 1.7.
2. Configure, compile and install MariaDB.

With earlier versions of the Mongo C Driver, the additional include directories and libraries will have to be specified manually when compiling.

When possible, this is the preferred means of access because it does not require all the Java path settings etc. and is faster than using the Java driver.

Using the Mongo Java Driver

This is possible with all distributions including JDBC support, or compiling from source. With a binary distribution that does

not enable the MONGO table type, it is possible to access MongoDB using an OEM module. See CONNECT OEM Table Example for details. The only additional things to do are:

1. Install the MongoDB Java Driver by downloading its jar file. Several versions are available. If possible use the latest version 3 one.
2. Add the path to it in the CLASSPATH environment variable or in the connect_class_path variable. This is like what is done to declare JDBC drivers.

Connection is established by new Java wrappers Mongo3Interface and Mongo2Interface. They are available in a JDBC distribution in the Mongo2.jar and Mongo3.jar files (previously JavaWrappers.jar). If version 2 of the Java Driver is used, specify “Version=2” in the option list when creating tables.

Using JDBC

See the documentation of the existing commercial JDBC Mongo drivers.

Using JSON

See the specific chapter of the JSON Table Type.

The following describes the MONGO table type.

CONNECT MONGO Tables

Creating and running MONGO tables requires a connection to a running local or remote MongoDB server.

A MONGO table is defined to access a MongoDB collection. The table rows will be the collection documents. For instance, to create a table based on the MongoDB sample collection restaurants, you can do something such as the following:

```
create table resto (
_id varchar(24) not null,
name varchar(64) not null,
cuisine char(200) not null,
borough char(16) not null,
restaurant_id varchar(12) not null)
engine=connect table_type=MONGO tabname='restaurants'
data_charset=utf8 connection='mongodb://localhost:27017';
```

Note: The used driver is by default the C driver if only the MongoDB C Driver is installed and the Java driver if only the MongoDB Java Driver is installed. If both are available, it can be specified by the DRIVER option to be specified in the option list and defaults to C.

Here we did not define all the items of the collection documents but only those that are JSON values. The database is test by default. The connection value is the URI used to establish a connection to a local or remote MongoDB server. The value shown in this example corresponds to a local server started with its default port. It is the default connection value for MONGO tables so we could have omit specifying it.

Using discovery is available. This table could have been created by:

```
create table resto
engine=connect table_type=MONGO tabname='restaurants'
data_charset=utf8 option_list='level=-1';
```

Here “depth=-1” is used to create only columns that are simple values (no array or object). Without this, with the default value “depth=0” the table had been created as:

```
CREATE TABLE `resto` (
  `_id` char(24) NOT NULL,
  `address` varchar(136) NOT NULL,
  `borough` char(13) NOT NULL,
  `cuisine` char(64) NOT NULL,
  `grades` varchar(638) NOT NULL,
  `name` char(98) NOT NULL,
  `restaurant_id` char(8) NOT NULL
) ENGINE=CONNECT DEFAULT CHARSET=latin1 `TABLE_TYPE`='MONGO' `TABNAME`='restaurants'
`DATA_CHARSET`='utf8';
```

Fixing Problems With mariadb-dump

In some case or some platforms, when CONNECT is set up for use with JDBC table types, this causes mariadb-dump with the --all-databases option to fail.

This was reported by Robert Dyas who found the cause of it and how to fix it (see MDEV-11238).

This occurs when the Java JRE “Usage Tracker” is enabled. In that case, Java creates a directory #mysql50#.oracle_jre_usage in the mysql data directory that shows up as a database but cannot be accessed via MySQL Workbench nor apparently backed up by mariadb-dump --all-databases.

Per the Oracle documentation (https://docs.oracle.com/javacomponents/usage-tracker/overview/) the “Usage Tracker” is disabled by default. It is enabled only when creating the properties file <JRE directory>/lib/management/usagetracker.properties. This turns out to be WRONG on some platforms as the file does exist by default on a new installation, and the existence of this file enables the usage tracker.

The solution on CentOS 7 with the Oracle JVM is to rename or delete the usagetracker.properties file (to disable it) and then delete the bogus folder it created in the mysql database directory, then restart.

For example, the following works:

```
sudo mv /usr/java/default/jre/lib/management/management.properties
/usr/java/default/jre/lib/management/management.properties.TRACKER-OFF
sudo reboot
sudo rm -rf  /var/lib/mysql/.oracle_jre_usage
sudo reboot
```

In this collection, the address column is a JSON object and the column grades is a JSON array. Unlike the JSON table, just specifying the column name with no Jpath result in displaying the JSON representation of them. For instance:

```
select name, address from resto limit 3;
```

name	address
Morris Park Bake Shop	{"building":"1007","coord":[-73.8561,40.8484], "street":"Morris ParkAve", "zipcode":"10462"}
Wendy'S	{"building":"469","coord":[-73.9617,40.6629], "street":"Flatbush Avenue", "zipcode":"11225"}
Reynolds Restaurant	{"building":"351","coord":[-73.9851,40.7677], "street":"West 57Street", "zipcode":"10019"}

MongoDB Dot Notation

To address the items inside object or arrays, specify the Jpath in MongoDB syntax (if using Discovery, specify the Depth option accordingly):

From Connect 1.7.0002

```
create table newresto (
_id varchar(24) not null,
name varchar(64) not null,
cuisine char(200) not null,
borough char(16) not null,
street varchar(65) jpath='address.street',
building char(16) jpath='address.building',
zipcode char(5) jpath='address.zipcode',
grade char(1) jpath='grades.0.grade',
score int(4) not null jpath='grades.0.score',
`date` date jpath='grades.0.date',
restaurant_id varchar(255) not null)
engine=connect table_type=MONGO tabname='restaurants'
data_charset=utf8 connection='mongodb://localhost:27017';
```

Before Connect 1.7.0002

```
create table newresto (
_id varchar(24) not null,
name varchar(64) not null,
cuisine char(200) not null,
borough char(16) not null,
street varchar(65) field_format='address.street',
building char(16) field_format='address.building',
zipcode char(5) field_format='address.zipcode',
grade char(1) field_format='grades.0.grade',
score int(4) not null field_format='grades.0.score',
`date` date field_format='grades.0.date',
restaurant_id varchar(255) not null)
engine=connect table_type=MONGO tabname='restaurants'
data_charset=utf8 connection='mongodb://localhost:27017';
```

If this is not done, the Oracle JVM will start the usage tracker, which will create the hidden folder .oracle_jre_usage in the mysql home directory, which will cause a mariadb-dump of the server to fail.

```
select name, street, score, date from newresto limit 5;
```

name	street	score	date
Morris Park Bake Shop	Morris Park Ave	2	03/03/2014
Wendy'S	Flatbush Avenue	8	30/12/2014
Dj Reynolds Pub And Restaurant	West 57 Street	2	06/09/2014
Riviera Caterer	Stillwell Avenue	5	10/06/2014
Tov Kosher Kitchen	63 Road	20	24/11/2014

MONGO Specific Options

The MongoDB syntax for Jpath does not allow the CONNECT specific items on arrays. The same effect can still be obtained by a different way. For this, additional options are used when creating MONGO tables.

Option	Type	Description
Colist	String	Options to pass to the MongoDB cursor.
Filter	String	Query used by the MongoDB cursor.
Pipeline*	Boolean	If True, Colist is a pipeline.
Fullarray*	Boolean	Used when creating with Discovery.
Driver*	String	C or Java.
Version*	Integer	The Java Driver version (defaults to 3)

- : To be specified in the option list.

Note: For the content of these options, refer to the MongoDB documentation.

Colist Option

Used to pass different options when making the MongoDB cursor used to retrieve the collation documents. One of them is the projection, allowing to limit the items retrieved in documents. It is hardly useful because this limitation is made automatically by CONNECT. However, it can be used when using discovery to eliminate the _id (or another) column when you are not willing to keep it:

```
create table restest
engine=connect table_type=MONGO tabname='restaurants'
data_charset=utf8 option_list='depth=-1'
colist='{"projection":{"_id":0},"limit":5}';
```

In this example, we added another cursor option, the limit option that works like the limit SQL clause.

This additional option works only with the C driver. When using the Java driver, colist should be:

```
colist='{"_id":0}';
```

And limit would be specified with select statements.

Note: When used with a JSON table, to specify the projection list (or 'all' to get all columns) makes JPATH to be Connect Json paths, not MongoDB ones, allowing JPATH options not available to MongoDB.

Filter Option

This option is used to specify a "filter" that works as a where clause on the table. Supposing we want to create a table restricted to the restaurant making English cuisine that are not located in the Manhattan borough, we can do it by:

```
create table english
engine=connect table_type=MONGO tabname='restaurants'
data_charset=utf8
colist='{"projection":{"cuisine":0}}'
filter='{"cuisine":"English","borough":{"$ne":"Manhattan"}}'
option_list='Depth=-1';
```

And if we ask:

```
select * from english;
```

This query will return:

_id	borough	name	restaurant_id
58ada47de5a51ddfcd5ee1f3	Brooklyn	The Park Slope Chipshop	40816202
58ada47de5a51ddfcd5ee999	Brooklyn	Chip Shop	41076583
58ada47ee5a51ddfcd5f13d5	Brooklyn	The Monro	41660253
58ada47ee5a51ddfcd5f176e	Brooklyn	Dear Bushwick	41690534
58ada47ee5a51ddfcd5f1e91	Queens	Snowdonia Pub	50000290

Pipeline Option

When this option is specified as true (by YES or 1) the Colist option contains a MongoDB pipeline applying to the table collation. This is a powerful mean for doing things such as expanding arrays like we do with JSON tables. For instance:

```
create table resto2 (
name varchar(64) not null,
borough char(16) not null,
date datetime not null,
grade char(1) not null,
score int(4) not null)
engine=connect table_type=MONGO tabname='restaurants' data_charset=utf8
colist='{"pipeline":[{"$match":{"cuisine":"French"}},{"$unwind":"$grades"},{"$project":
{"_id":0,"name":1,"borough":1,"date":"$grades.date","grade":"$grades.grade","score":"$grades.sc
ore"}}]}'
option_list='Pipeline=1';
```

In this pipeline "$match" is an early filter, "$unwind" means that the grades array will be expanded (one Document for each array values) and "$project" eliminates the _id and cuisine columns and gives the Jpath for the date, grade and score columns.

```
select name, grade, score, date from resto2
where borough = 'Bronx';
```

This query replies:

name	grade	score	date
Bistro Sk	A	10	21/11/2014 01:00:00
Bistro Sk	A	12	19/02/2014 01:00:00

Bistro Sk	B	18	12/06/2013 02:00:00

This make possible to get things like we do with JSON tables:

```
select name, avg(score) average from resto2
group by name having average >= 25;
```

Can be used to get the average score inside the grades array.

name	average
Bouley Botanical	25,0000
Cheri	46,0000
Graine De Paris	30,0000
Le Pescadeux	29,7500

Fullarray Option

This option, like the Depth option, is only interpreted when creating a table with Discovery (meaning not specifying the columns). It tells CONNECT to generate a column for all existing values in the array. For instance, let us see the MongoDB collection tar by:

From Connect 1.7.0002

```
create table seetar (
Collection varchar(300) not null jpath='*')
engine=CONNECT table_type=MONGO tabname=tar;
```

Before Connect 1.7.0002

```
create table seetar (
Collection varchar(300) not null field_format='*')
engine=CONNECT table_type=MONGO tabname=tar;
```

The format '*' indicates we want to see the Json documents. This small collection is:

Collection
{"_id":{"$oid":"58f63a5099b37d9c930f9f3b"},"item":"journal","prices":[87.0,45.0,63.0,12.0,78.0]}
{"_id":{"$oid":"58f63a5099b37d9c930f9f3c"},"item":"notebook","prices":[123.0,456.0,789.0]}

The Fullarray option can be used here to generate enough columns to see all the prices of the document prices array.

```
create table tar
engine=connect table_type=MONGO
colist='{"projection":{"_id":0}}'
option_list='depth=1,Fullarray=YES';
```

The table has been created as:

From Connect 1.7.0002

```
CREATE TABLE `tar` (
  `item` char(8) NOT NULL,
  `prices_0` double(12,6) NOT NULL `JPATH`='prices.0',
  `prices_1` double(12,6) NOT NULL `JPATH`='prices.1',
  `prices_2` double(12,6) NOT NULL `JPATH`='prices.2',
  `prices_3` double(12,6) DEFAULT NULL `JPATH`='prices.3',
  `prices_4` double(12,6) DEFAULT NULL `JPATH`='prices.4'
) ENGINE=CONNECT DEFAULT CHARSET=latin1 `TABLE_TYPE`='MONGO' `COLIST`='{"projection":
{"_id":0}}' `OPTION_LIST`='depth=1,Fullarray=YES';
```

Before Connect 1.7.0002

```
CREATE TABLE `tar` (
  `item` char(8) NOT NULL,
  `prices_0` double(12,6) NOT NULL `FIELD_FORMAT`='prices.0',
  `prices_1` double(12,6) NOT NULL `FIELD_FORMAT`='prices.1',
  `prices_2` double(12,6) NOT NULL `FIELD_FORMAT`='prices.2',
  `prices_3` double(12,6) DEFAULT NULL `FIELD_FORMAT`='prices.3',
  `prices_4` double(12,6) DEFAULT NULL `FIELD_FORMAT`='prices.4'
) ENGINE=CONNECT DEFAULT CHARSET=latin1 `TABLE_TYPE`='MONGO' `COLIST`='{"projection":
{"_id":0}}' `OPTION_LIST`='level=1,Fullarray=YES';
```

And is displayed as:

item	prices_0	prices_1	prices_2	prices_3	prices_4
journal	87.00	45.00	63.00	12.00	78.00
notebook	123.00	456.00	789.00	NULL	NULL

Create, Read, Update and Delete Operations

All modifying operations are supported. However, inserting into arrays must be done in a specific way. Like with the Fullarray option, we must have enough columns to specify the array values. For instance, we can create a new table by:

From Connect 1.7.0002

```
create table testin (
n int not null,
m char(12) not null,
surname char(16) not null jpath='person.name.first',
name char(16) not null jpath='person.name.last',
age int(3) not null jpath='person.age',
price_1 double(8,2) jpath='d.0',
price_2 double(8,2) jpath='d.1',
price_3 double(8,2) jpath='d.2')
engine=connect table_type=MONGO tabname='tin'
connection='mongodb://localhost:27017';
```

Before Connect 1.7.0002

```
create table testin (
n int not null,
m char(12) not null,
surname char(16) not null field_format='person.name.first',
name char(16) not null field_format='person.name.last',
age int(3) not null field_format='person.age',
price_1 double(8,2) field_format='d.0',
price_2 double(8,2) field_format='d.1',
price_3 double(8,2) field_format='d.2')
engine=connect table_type=MONGO tabname='tin'
connection='mongodb://localhost:27017';
```

Now it is possible to populate it by:

```
insert into testin values
(1789, 'Welcome', 'Olivier','Bertrand',56, 3.14, 2.36, 8.45),
(1515, 'Hello', 'John','Smith',32, 65.17, 98.12, NULL),
(2014, 'Coucou', 'Foo','Bar',20, -1.0, 74, 81356);
```

The result will be:

n	m	surname	name	age	price_1	price_2	price_3
1789	Welcome	Olivier	Bertrand	56	3,14	2,36	8,45
1515	Hello	John	Smith	32	65,17	98,12	NULL
2014	Coucou	Foo	Bar	20	-1	74	81356

Note: If the collection does not exist yet when creating the table and inserting in it, MongoDB creates it automatically.

It can be updated by queries such as:

```
update tintin set price_3 = 83.36 where n = 2014;
```

To look how the array is generated, let us create another table:

From Connect 1.7.0002

```
create table tintin (
n int not null,
name char(16) not null jpath='person.name.first',
prices varchar(255) jpath='d')
engine=connect table_type=MONGO tabname='tin';
```

Before Connect 1.7.002

```
create table tintin (
n int not null,
name char(16) not null field_format='person.name.first',
prices varchar(255) field_format='d')
engine=connect table_type=MONGO tabname='in';
```

This table is displayed as:

From Connect 1.7.0002

n	name	prices
1789	Olivier	[3.1400000000000001243,2.3599999999999998757,8.4499999999999992895]
1515	John	[65.170000000000001705,98.120000000000004547,null]
2014	Foo	[null,74.0,83.359999999999999432]

Before Connect 1.7.002

n	name	prices
1789	Olivier	[3.14, 2.36, 8.45]
1515	John	[65.17, 98.12]
2014	Foo	[<null>, 74.0, 83.36]

Note: This last table can be used to make array calculations like with JSON tables using the JSON UDF functions. For instance:

```
select name, jsonget_real(prices,'[+]') sum_prices, jsonget_real(prices,'[!]') avg_prices from ti
```

This query returns:

name	sum_prices	avg_prices
Olivier	13.95	4.65
John	163.29	81.64
Foo	157,36	78.68

Note: When calculating on arrays, null values are ignored.

Status of MONGO Table Type

This table type is still under development. It has significant advantages over the JSON type to access MongoDB collections. Firstly, the access being direct, tables are always up to date whether the collection has been modified by another application. Performance wise, it can be faster than JSON, because most processing is done by MongoDB on BSON, its internal representation of JSON data, which is designed to optimize all operations. Note that using the MongoDB C Driver can be faster than using the MongoDB Java Driver.

Current Restrictions

- Option “CATFUNC=tables” is not implemented yet.

- Options SRCDEF and EXECSRC do not apply to MONGO tables.

5.3.7.6.19 CONNECT MYSQL Table Type: Accessing MySQL/MariaDB Tables

Contents

This table type uses libmysql API to access a MySQL or MariaDB table or view. This table must be created on the current server or on another local or remote server. This is similar to what the FederatedX storage engine provides with some differences.

Currently the Federated-like syntax can be used to create such a table, for instance:

```
create table essai (
  num integer(4) not null,
  line char(15) not null)
engine=CONNECT table_type=MYSQL
connection='mysql://root@localhost/test/people';
```

The connection string can have the same syntax as that used by FEDERATED

```
scheme://username:password@hostname:port/database/tablename
scheme://username@hostname/database/tablename
scheme://username:password@hostname/database/tablename
scheme://username:password@hostname/database/tablename
```

However, it can also be mixed with connect standard options. For instance:

```
create table essai (
  num integer(4) not null,
  line char(15) not null)
engine=CONNECT table_type=MYSQL dbname=test tabname=people
connection='mysql://root@localhost';
```

It can also be specified as a reference to a federated server:

```
connection="connection_one"
connection="connection_one/table_foo"
```

The pure (deprecated) CONNECT syntax is also accepted:

```
create table essai (
  num integer(4) not null,
  line char(15) not null)
engine=CONNECT table_type=MYSQL dbname=test tabname=people
option_list='user=root,host=localhost';
```

The specific connection items are:

Option	Default value	Description
Table	The table name	The name of the table to access.
Database	The current DB name	The database where the table is located.
Host	localhost*	The host of the server, a name or an IP address.

User	The current user	The connection user name.
Password	No password	An optional user password.
Port	The currently used port	The port of the server.
Quoted	0	1 if remote Tabname must be quoted.

- - When the host is specified as “localhost”, the connection is established on Linux using Linux sockets. On Windows, the connection is established by default using shared memory if it is enabled. If not, the TCP protocol is used. An alternative is to specify the host as “.” to use a named pipe connection (if it is enabled). This makes possible to use these table types with server skipping networking.

Caution: Take care not to refer to the MYSQL table itself to avoid an infinite loop!

MYSQL table can refer to the current server as well as to another server. Views can be referred by name or directly giving a source definition, for instance:

```
create table grp engine=connect table_type=mysql
CONNECTION='mysql://root@localhost/test/people'
SRCDEF='select title, count(*) as cnt from employees group by title';
```

When specified, the columns of the mysql table must exist in the accessed table with the same name, but can be only a subset of them and specified in a different order. Their type must be a type supported by CONNECT and, if it is not identical to the type of the accessed table matching column, a conversion can be done according to the rules given in Data type conversion.

Note: For columns prone to be targeted by a where clause, keep the column type compatible with the source table column type (numeric or character) to have a correct rephrasing of the where clause.

If you do not want to restrict or change the column definition, do not provide it and leave CONNECT get the column definition from the remote server. For instance:

```
create table essai engine=CONNECT table_type=MYSQL
connection='mysql://root@localhost/test/people';
```

This will create the *essai* table with the same columns than the people table. If the target table contains CONNECT incompatible type columns, see Data type conversion to know how these columns can be converted or skipped.

Charset Specification

When accessing the remote table, CONNECT sets the connection charset set to the default local table charset as the FEDERATED engine does.

Do not specify a column character set if it is different from the table default character set even when it is the case on the remote table. This is because the remote column is translated to the local table character set when reading it. This is the default but it can be modified by the setting the character_set_results variable of the target server. If it must keep its setting, for instance to UTF8 when containing Unicode characters, specify the local default charset to its character set.

This means that it is not possible to correctly retrieve a remote table if it contains columns having different character sets. A solution is to retrieve it by several local tables, each accessing only columns with the same character set.

Indexing of MYSQL tables

Indexes are rarely useful with MYSQL tables. This is because CONNECT tries to access only the requested rows. For instance if you ask:

```
select * from essai where num = 23;
```

CONNECT will construct and send to the server the query:

```
SELECT num, line FROM people WHERE num = 23
```

If the *people* table is indexed on *num*, indexing will be used on the remote server. This, in all cases, will limit the amount of data to retrieve on the network.

However, an index can be specified for columns that are prone to be used to join another table to the MYSQL table. For

instance:

```
select d.id, d.name, f.dept, f.salary
from loc_tab d straight_join cnc_tab f on d.id = f.id
where f.salary > 10000;
```

If the *id* column of the remote table addressed by the *cnc_tab* MYSQL table is indexed (which is likely if it is a key) you should also index the *id* column of the MYSQL *cnc_tab* table. If so, using “remote” indexing as does FEDERATED, only the useful rows of the remote table will be retrieved during the join process. However, because these rows are retrieved by separate SELECT statements, this will be useful only when retrieving a few rows of a big table.

In particular, you should not specify an index for columns not used for joining and above all DO NOT index a joined column if it is not indexed in the remote table. This would cause multiple scans of the remote table to retrieve the joined rows one by one.

Data Modifying Operations

The CONNECT MYSQL type supports SELECT and INSERT and a somewhat limited form of UPDATE and DELETE. These are described below.

The MYSQL type uses similar methods than the ODBC type to implement the INSERT, UPDATE and DELETE commands. Refer to the ODBC chapter for the restrictions concerning them.

For the UPDATE and DELETE commands, there are fewer restrictions because the remote server being a MySQL server, the syntax of the command will be always acceptable by the remote server.

For instance, you can freely use keywords like IGNORE or LOW_PRIORITY as well as scalar functions in the SET and WHERE clauses.

However, there is still an issue on multi-table statements. Let us suppose you have a *t1* table on the remote server and want to execute a query such as:

```
update essai as x set line = (select msg from t1 where id = x.num)
where num = 2;
```

When parsed locally, you will have errors if no *t1* table exists or if it does not have the referenced columns. When *t1* does not exist, you can overcome this issue by creating a local dummy *t1* table:

```
create table t1 (id int, msg char(1)) engine=BLACKHOLE;
```

This will make the local parser happy and permit to execute the command on the remote server. Note however that having a local MySQL table defined on the remote *t1* table does not solve the problem unless it is also names *t1* locally.

This is why, to permit to have all types of commands executed by the data source without any restriction, CONNECT provides a specific MySQL table subtype described now.

Sending commands to a MariaDB Server

This can be done like for ODBC or JDBC tables by defining a specific table that will be used to send commands and get the result of their execution..

```
create table send (
  command varchar(128) not null,
  warnings int(4) not null flag=3,
  number int(5) not null flag=1,
  message varchar(255) flag=2)
engine=connect table_type=mysql
connection='mysql://user@host/database'
option_list='Execsrc=1,Maxerr=2';
```

The key points in this create statement are the EXECSRC option and the column definition.

The EXECSRC option tells that this table will be used to send commands to the MariaDB server. Most of the sent commands do not return result set. Therefore, the table columns are used to specify the command to be executed and to get the result of the execution. The name of these columns can be chosen arbitrarily, their function coming from the FLAG value:

Flag=0: The command to execute (the default)

Flag=1: The number of affected rows, or the result number of columns if the command would return a result set.

Flag=2: The returned (eventually error) message.

Flag=3: The number of warnings.

How to use this table and specify the command to send? By executing a command such as:

```
select * from send where command = 'a command';
```

This will send the command specified in the WHERE clause to the data source and return the result of its execution. The syntax of the WHERE clause must be exactly as shown above. For instance:

```
select * from send where command =
'CREATE TABLE people (
num integer(4) primary key autoincrement,
line char(15) not null';
```

This command returns:

command	warnings	number	message
CREATE TABLE people (num integer(4) primary key aut...	0	0	Affected rows

Sending several commands in one call

It can be faster to execute because there will be only one connection for all of them. To send several commands in one call, use the following syntax:

```
select * from send where command in (
"update people set line = 'Two' where id = 2",
"update people set line = 'Three' where id = 3");
```

When several commands are sent, the execution stops at the end of them or after a command that is in error. To continue after n errors, set the option maxerr=*n* (0 by default) in the option list.

Note 1: It is possible to specify the SRCDEF option when creating an EXECSRC table. It will be the command sent by default when a WHERE clause is not specified.

Note 2: Backslashes inside commands must be escaped. Simple quotes must be escaped if the command is specified between simple quotes, and double quotes if it is specified between double quotes.

Note 3: Sent commands apply in the specified database. However, they can address any table within this database.

Note 4: Currently, all commands are executed in mode AUTOCOMMIT.

Retrieving Warnings and Notes

If a sent command causes warnings to be issued, it is useless to resend a “show warnings” command because the MariaDB server is opened and closed when sending commands. Therefore, getting warnings requires a specific (and tricky) way.

To indicate that warning text must be added to the returned result, you must send a multi-command query containing “pseudo” commands that are not sent to the server but directly interpreted by the EXECSRC table. These “pseudo” commands are:

Warning To get warnings

Note To get notes

Error To get errors returned as warnings (?)

Note that they must be spelled (case insensitive) exactly as above, no final “s”. For instance:

```
select * from send where command in ('Warning','Note',
'drop table if exists try',
'create table try (id int key auto_increment, msg varchar(32) not
null) engine=aria',
"insert into try(msg) values('One'),(NULL),('Three') ",
"insert into try values(2,'Deux') on duplicate key update msg =
'Two'",
"insert into try(message) values('Four'),('Five'),('Six')",
'insert into try(id) values(NULL)',
"update try set msg = 'Four' where id = 4",
'select * from try');
```

This can return something like this:

command	warnings	number	message
`drop table if exists try`	1	0	Affected rows
Note	0	1051	Unknown table 'try'
`create table try (id int key auto_increment, msg...`	0	0	Affected rows
`insert into try(msg) values('One'),(NULL), ('Three')`	1	3	Affected rows
Warning	0	1048	Column 'msg' cannot be null
`insert into try values(2,'Deux') on duplicate key...`	0	2	Affected rows
`insert into try(msge) values('Four'),('Five'), ('Six')`	0	1054	Unknown column 'msge' in 'field list'
`insert into try(id) values(NULL)`	1	1	Affected rows
Warning	0	1364	Field 'msg' doesn't have a default value
`update try set msg = 'Four' where id = 4`	0	1	Affected rows
`select * from try`	0	2	Result set columns

The execution continued after the command in error because of the MAXERR option. Normally this would have stopped the execution.

Of course, the last “select” command is useless here because it cannot return the table contain. Another MYSQL table without the EXECSRC option and with proper column definition should be used instead.

Connection Engine Limitations

Data types

There is a maximum key.index length of 255 bytes. You may be able to declare the table without an index and rely on the engine condition pushdown and remote schema.

The following types can't be used:

- BIT
- BINARY
- TINYBLOB, BLOB, MEDIUMBLOB, LONGBLOB
- TINYTEXT, MEDIUMTEXT, LONGTEXT
- ENUM
- SET
- Geometry types

Note: TEXT is allowed. However, the handling depends on the values given to the connect_type_conv and connect_conv_size system variables, and by default no conversion of TEXT columns is permitted.

SQL Limitations

The following SQL queries are not supported

- REPLACE INTO
- INSERT ... ON DUPLICATE KEY UPDATE

CONNECT MYSQL versus FEDERATED

The CONNECT MYSQL table type should not be regarded as a replacement for the FEDERATED(X) engine. The main use of the MYSQL type is to access other engine tables as if they were CONNECT tables. This was necessary when accessing tables from some CONNECT table types such as TBL, XCOL, OCCUR, or PIVOT that are designed to access CONNECT tables only. When their target table is not a CONNECT table, these types are silently using internally an intermediate MYSQL table.

However, there are cases where you can use MYSQL CONNECT tables yourself, for instance:

1. When the table will be used by a TBL table. This enables you to specify the connection parameters for each sub-table and is more efficient than using a local FEDERATED sub-table.
2. When the desired returned data is directly specified by the SRCDEF option. This is great to let the remote server do most of the job, such as grouping and/or joining tables. This cannot be done with the FEDERATED engine.
3. To take advantage of the *push_cond* facility that adds a where clause to the command sent to the remote table. This restricts the size of the result set and can be crucial for big tables.
4. For tables with the EXECSRC option on.
5. When doing tests. For instance to check a connection string.

If you need multi-table updating, deleting, or bulk inserting on a remote table, you can alternatively use the FEDERATED engine or a “send” table specifying the EXECSRC option on.

5.3.7.6.20 CONNECT PROXY Table Type

Contents

A `PROXY` table is a table that accesses and reads the data of another table or view. For instance, to create a table based on the boys `FIX` table:

```
create table xboy engine=connect
  table_type=PROXY tabname=boys;
```

Simply, `PROXY` being the default type when `TABNAME` is specified:

```
create table xboy engine=connect tabname=boys;
```

Because the boys table can be directly used, what can be the use of a proxy table? Well, its main use is to be internally used by other table types such as TBL, XCOL, OCCUR, or PIVOT. Sure enough, PROXY tables are CONNECT tables, meaning that they can be based on tables of any engines and accessed by table types that need to access CONNECT tables.

Proxy on non-CONNECT Tables

When the sub-table is a view or not a CONNECT table, CONNECT internally creates a temporary CONNECT table of MYSQL type to access it. This connection uses the same default parameters as for a `MYSQL` table. It is also possible to specify them to the `PROXY` table using in the `PROXY` declaration the same `OPTION_LIST` options as for a `MYSQL` table. Of course, it is simpler and more natural to use directly the MYSQL type in this case.

Normally, the default parameters should enable the `PROXY` table to reconnect the server. However, an issue is when the current user was logged using a password. The security protocol prevents CONNECT to retrieve this password and requires it to be given in the `PROXY` table create statement. For instance adding to it:

```
... option_list='Password=mypass';
```

However, it is often not advisable to write in clear a password that can be seen by all user able to see the table declaration by show create table, in particular, if the table is used when the current user is root. To avoid this, a specific user should be created on the local host that will be used by proxy tables to retrieve local tables. This user can have minimum grant options, for instance SELECT on desired directories, and needs no password. Supposing ‘proxy’ is such a user, the option list to add will be:

```
... option_list='user=proxy';
```

Using a PROXY Table as a View

A `PROXY` table can also be used by itself to modify the way a table is viewed. For instance, a proxy table does not use the indexes of the object table. It is also possible to define its columns with different names or type, to use only some of them or to changes their order. For instance:

```
create table city (
  city varchar(11),
  boy char(12) flag=1,
  birth date)
engine=CONNECT tabname=boys;
select * from city;
```

This will display:

city	boy	birth
Boston	John	1986-01-25
Boston	Henry	1987-06-07
San Jose	George	1981-08-10
Chicago	Sam	1979-11-22
Dallas	James	1992-05-13
Boston	Bill	1986-09-11

Here we did not have to specify column format or offset because data are retrieved from the boys table, not directly from the boys.txt file. The flag option of the *boy* column indicates that it correspond to the first column of the boys table, the *name* column.

Avoiding PROXY table loop

CONNECT is able to test whether a `PROXY` , or `PROXY` -based, table refers directly or indirectly to itself. If a direct reference can tested at table creation, an indirect reference can only be tested when executing a query on the table. However, this is possible only for local tables. When using remote tables or views, a problem can occur if the remote table or the view refers back to one of the local tables of the chain. The same caution should be used than when using `FEDERATEDX` tables.

Note: All `PROXY` or `PROXY` -based tables are read-only in this version.

Modifying Operations

All INSERT / UPDATE / DELETE operations can be used with proxy tables. However, the same restrictions applying to the source table also apply to the proxy table.

> Note: All PROXY and PROXY-based table types are not indexable.

5.3.7.6.21 CONNECT XCOL Table Type

Contents

`XCOL` tables are based on another table or view, like `PROXY` tables. This type can be used when the object table has a column that contains a list of values.

Suppose we have a *'children'* table that can be displayed as:

name	childlist
Sophie	Vivian, Antony

Lisbeth	Lucy,Charles,Diana
Corinne	
Claude	Marc
Janet	Arthur, Sandra, Peter, John

We can have a different view on these data, where each child will be associated with his/her mother by creating an `XCOL` table by:

```
CREATE TABLE xchild (
  mother char(12) NOT NULL,
  child char(12) DEFAULT NULL flag=2
) ENGINE=CONNECT table_type=XCOL tabname='chlist'
option_list='colname=child';
```

The `COLNAME` option specifies the name of the column receiving the list items. This will return from:

```
select * from xchild;
```

The requested view:

mother	child
Sophia	Vivian
Sophia	Antony
Lisbeth	Lucy
Lisbeth	Charles
Lisbeth	Diana
Corinne	NULL
Claude	Marc
Janet	Arthur
Janet	Sandra
Janet	Peter
Janet	John

Several things should be noted here:

- When the original *children* field is void, what happens depends on the NULL specification of the "multiple" column. If it is nullable, like here, a void string will generate a NULL value. However, if the column is not nullable, no row will be generated at all.
- Blanks after the separator are ignored.
- No copy of the original data was done. Both tables use the same source data.
- Specifying the column definitions in the `CREATE TABLE` statement is optional.

The "multiple" column *child* can be used as any other column. For instance:

```
select * from xchild where substr(child,1,1) = 'A';
```

This will return:

Mother	Child
Sophia	Antony
Janet	Arthur

If a query does not involve the "multiple" column, no row multiplication will be done. For instance:

```
select mother from xchild;
```

This will just return all the mothers:

mother
Sophia
Lisbeth
Corinne
Claude
Janet

The same occurs with other types of select statements, for instance:

```
select count(*) from xchild;       -- returns 5
select count(child) from xchild;   -- returns 10
select count(mother) from xchild; -- returns 5
```

Grouping also gives different result:

```
select mother, count(*) from xchild group by mother;
```

Replies:

mother	count(*)
Claude	1
Corinne	1
Janet	1
Lisbeth	1
Sophia	1

While the query:

```
select mother, count(child) from xchild group by mother;
```

Gives the more interesting result:

mother	count(child)
Claude	1
Corinne	0
Janet	4
Lisbeth	3
Sophia	2

Some more options are available for this table type:

Option	Description
Sep_char	The separator character used in the "multiple" column, defaults to the comma.
Mult	Indicates the max number of multiple items. It is used to internally calculate the max size of the table and defaults to 10. (To be specified in `OPTION_LIST`).

Using Special Columns with XCOL

Special columns can be used in XCOL tables. The mostly useful one is ROWNUM that gives the rank of the value in the list of values. For instance:

```
CREATE TABLE xchild2 (
rank int NOT NULL SPECIAL=ROWID,
mother char(12) NOT NULL,
child char(12) NOT NULL flag=2
) ENGINE=CONNECT table_type=XCOL tabname='chlist' option_list='colname=child';
```

This table will be displayed as:

rank	mother	child
1	Sophia	Vivian
2	Sophia	Antony
1	Lisbeth	Lucy
2	Lisbeth	Charles
3	Lisbeth	Diana
1	Claude	Marc
1	Janet	Arthur
2	Janet	Sandra
3	Janet	Peter
4	Janet	John

To list only the first child of each mother you can do:

```
SELECT mother, child FROM xchild2 where rank = 1 ;
```

returning:

mother	child
Sophia	Vivian
Lisbeth	Lucy
Claude	Marc
Janet	Arthur

However, note the following pitfall: trying to get the names of all mothers having more than 2 children cannot be done by:

```
SELECT mother FROM xchild2 where rank > 2;
```

This is because with no row multiplication being done, the rank value is always 1. The correct way to obtain this result is longer but cannot use the ROWNUM column:

```
SELECT mother FROM xchild2 group by mother having count(child) > 2;
```

XCOL tables based on specified views

Instead of specifying a source table name via the TABNAME option, it is possible to retrieve data from a “view” whose definition is given in a new option SRCDEF . For instance:

```
create table xsvars engine=connect table_type=XCOL
srcdef='show variables like "optimizer_switch"'
option_list='Colname=Value';
```

Then, for instance:

```
select value from xsvars limit 10;
```

This will display something like:

value
index_merge=on
index_merge_union=on
index_merge_sort_union=on
index_merge_intersection=on

index_merge_sort_intersection=off
engine_condition_pushdown=off
index_condition_pushdown=on
derived_merge=on
derived_with_keys=on
firstmatch=on

Note: All XCOL tables are read only.

5.3.7.6.22 CONNECT OCCUR Table Type

Similarly to the `XCOL` table type, `OCCUR` is an extension to the `PROXY` type when referring to a table or view having several columns containing the same kind of data. It enables having a different view of the table where the data from these columns are put in a single column, eventually causing several rows to be generated from one row of the object table. For example, supposing we have a *pets* table:

name	dog	cat	rabbit	bird	fish
John	2	0	0	0	0
Bill	0	1	0	0	0
Mary	1	1	0	0	0
Lisbeth	0	0	2	0	0
Kevin	0	2	0	6	0
Donald	1	0	0	0	3

We can create an occur table by:

```
create table xpet (
  name varchar(12) not null,
  race char(6) not null,
  number int not null)
engine=connect table_type=occur tabname=pets
option_list='OccurCol=number,RankCol=race'
Colist='dog,cat,rabbit,bird,fish';
```

When displaying it by

```
select * from xpet;
```

We will get the result:

name	race	number
John	dog	2
Bill	cat	1
Mary	dog	1
Mary	cat	1
Lisbeth	rabbit	2
Kevin	cat	2
Kevin	bird	6
Donald	dog	1
Donald	fish	3

First of all, the values of the column listed in the Colist option have been put in a unique column whose name is given by the OccurCol option. When several columns have non null (or pseudo-null) values, several rows are generated, with the other normal columns values repeated.

In addition, an optional special column was added whose name is given by the RankCol option. This column contains the

name of the source column from which the value of the OccurCol column comes from. It permits here to know the race of the pets whose number is given in *number*.

This table type permit to make queries that would be more complicated to make on the original tables. For instance to know who as more than 1 pet of a kind, you can simply ask:

```
select * from xpet where number > 1;
```

You will get the result:

name	race	number
John	dog	2
Lisbeth	rabbit	2
Kevin	cat	2
Kevin	bird	6
Donald	fish	3

Note 1: Like for XCOL tables, no row multiplication for queries not implying the Occur column.

Note 2: Because the OccurCol was declared "not null" no rows were generated for null or pseudo-null values of the column list. If the OccurCol is declared as nullable, rows are also generated for columns containing null or pseudo-null values.

Occur tables can be also defined from views or source definition. Also, CONNECT is able to generate the column definitions if not specified. For example:

```
create table ocsrc engine=connect table_type=occur
colist='january,february,march,april,may,june,july,august,september,
october,november,december' option_list='rankcol=month,occurcol=day'
srcdef='select ''Foo'' name, 8 january, 7 february, 2 march, 1 april,
  8 may, 14 june, 25 july, 10 august, 13 september, 22 october, 28
  november, 14 december';
```

This table is displayed as:

name	month	day
Foo	january	8
Foo	february	7
Foo	march	2
Foo	april	1
Foo	may	8
Foo	june	14
Foo	july	25
Foo	august	10
Foo	september	13
Foo	october	22
Foo	november	28
Foo	december	14

5.3.7.6.23 CONNECT PIVOT Table Type

Contents

This table type can be used to transform the result of another table or view (called the source table) into a pivoted table along “pivot” and “facts” columns. A pivot table is a great reporting tool that sorts and sums (by default) independent of the original data layout in the source table.

For example, let us suppose you have the following “Expenses” table:

Who	Week	What	Amount
Joe	3	Beer	18.00
Beth	4	Food	17.00
Janet	5	Beer	14.00
Joe	3	Food	12.00
Joe	4	Beer	19.00
Janet	5	Car	12.00
Joe	3	Food	19.00
Beth	4	Beer	15.00
Janet	5	Beer	19.00
Joe	3	Car	20.00
Joe	4	Beer	16.00
Beth	5	Food	12.00
Beth	3	Beer	16.00
Joe	4	Food	17.00
Joe	5	Beer	14.00
Janet	3	Car	19.00
Joe	4	Food	17.00
Beth	5	Beer	20.00
Janet	3	Food	18.00
Joe	4	Beer	14.00
Joe	5	Food	12.00
Janet	3	Beer	18.00
Janet	4	Car	17.00
Janet	5	Food	12.00

Pivoting the table contents using the 'Who' and 'Week' fields for the left columns, and the 'What' field for the top heading and summing the 'Amount' fields for each cell in the new table, gives the following desired result:

Who	Week	Beer	Car	Food
Beth	3	16.00	0.00	0.00
Beth	4	15.00	0.00	17.00
Beth	5	20.00	0.00	12.00
Janet	3	18.00	19.00	18.00
Janet	4	0.00	17.00	0.00

Janet	5	33.00	12.00	12.00
Joe	3	18.00	20.00	31.00
Joe	4	49.00	0.00	34.00
Joe	5	14.00	0.00	12.00

Note that SQL enables you to get the same result presented differently by using the “group by” clause, namely:

```
select who, week, what, sum(amount) from expenses
       group by who, week, what;
```

However there is no way to get the pivoted layout shown above just using SQL. Even using embedded SQL programming for some DBMS is not quite simple and automatic.

The Pivot table type of CONNECT makes doing this much simpler.

Using the PIVOT Tables Type

To get the result shown in the example above, just define it as a new table with the statement:

```
create table pivex
engine=connect table_type=pivot tabname=expenses;
```

You can now use it as any other table, for instance to display the result shown above, just say:

```
select * from pivex;
```

The CONNECT implementation of the PIVOT table type does much of the work required to transform the source table:

1. Finding the “Facts” column, by default the last column of the source table. Finding “Facts” or “Pivot” columns work only for table based pivot tables. They do not for view or srcdef based pivot tables, for which they must be explicitly specified.
2. Finding the “Pivot” column, by default the last remaining column.
3. Choosing the aggregate function to use, “SUM” by default.
4. Constructing and executing the “Group By” on the “Facts” column, getting its result in memory.
5. Getting all the distinct values in the “Pivot” column and defining a “Data” column for each.
6. Spreading the result of the intermediate memory table into the final table.

> The source table “Pivot” column must not be nullable (there are no such things as a “null” column) The creation will be refused even is this nullable column actually does not contain null values.

If a different result is desired, Create Table options are available to change the defaults used by Pivot. For instance if we want to display the average expense for each person and product, spread in columns for each week, use the following statement:

```
create table pivex2
engine=connect table_type=pivot tabname=expenses
option_list='PivotCol=Week,Function=AVG';
```

Now saying:

```
select * from pivex2;
```

Will display the resulting table:

Who	What	3	4	5
Beth	Beer	16.00	15.00	20.00
Beth	Food	0.00	17.00	12.00
Janet	Beer	18.00	0.00	16.50
Janet	Car	19.00	17.00	12.00
Janet	Food	18.00	0.00	12.00

Joe	Beer	18.00	16.33	14.00
Joe	Car	20.00	0.00	0.00
Joe	Food	15.50	17.00	12.00

Restricting the Columns in a Pivot Table

Let us suppose that we want a Pivot table from expenses summing the expenses for all people and products whatever week it was bought. We can do this just by removing from the pivex table the week column from the column list.

```
alter table pivex drop column week;
```

The result we get from the new table is:

Who	Beer	Car	Food
Beth	51.00	0.00	29.00
Janet	51.00	48.00	30.00
Joe	81.00	20.00	77.00

Note: Restricting columns is also needed when the source table contains extra columns that should not be part of the pivot table. This is true in particular for key columns that prevent a proper grouping.

PIVOT Create Table Syntax

The Create Table statement for PIVOT tables uses the following syntax:

```
create table pivot_table_name
[(column_definition)]
engine=CONNECT table_type=PIVOT
{tabname='source_table_name' | srcdef='source_table_def'}
[option_list='pivot_table_option_list'];
```

The column definition has two sets of columns:

1. A set of columns belonging to the source table, not including the “facts” and “pivot” columns.
2. “Data” columns receiving the values of the aggregated “facts” columns named from the values of the “pivot” column. They are indicated by the “flag” option.

The **options** and **sub-options** available for Pivot tables are:

Option	Type	Description
Tabname	*[DB.]Name*	The name of the table to “pivot”. If not set SrcDef must be specified.
SrcDef	*SQL_statement*	The statement used to generate the intermediate mysql table.
DBname	*name*	The name of the database containing the source table. Defaults to the current database.
Function*	*name*	The name of the aggregate function used for the data columns, SUM by default.
PivotCol*	*name*	Specifies the name of the Pivot column whose values are used to fill the “data” columns having the flag option.
FncCol*	*[func(]name[)]*	Specifies the name of the data “Facts” column. If the form func(name) is used, the aggregate function name is set to func.
Groupby*	*Boolean*	Set it to True (1 or Yes) if the table already has a GROUP BY format.
Accept*	*Boolean*	To accept non matching Pivot column values.

- : These options must be specified in the OPTION_LIST.

Additional Access Options

There are four cases where pivot must call the server containing the source table or on which the SrcDef statement must be executed:

1. The source table is not a CONNECT table. 2. The SrcDef option is specified. 3. The source table is on another server. 4.

The columns are not specified.

By default, pivot tries to call the currently used server using host=localhost, user=root not using password, and port=3306. However, this may not be what is needed, in particular if the local root user has a password in which case you can get an “access denied” error message when creating or using the pivot table.

Specify the host, user, password and/or port options in the option_list to override the default connection options used to access the source table, get column specifications, execute the generated group by or SrcDef query.

Defining a Pivot Table

There are principally two ways to define a PIVOT table:

1. From an existing table or view. 2. Directly giving the SQL statement returning the result to pivot.

Defining a Pivot Table from a Source Table

The **tabname** standard table option is used to give the name of the source table or view.

For tables, the internal Group By will be internally generated, except when the GROUPBY option is specified as true. Do it only when the table or view has a valid GROUP BY format.

Directly Defining the Source of a Pivot Table in SQL

Alternatively, the internal source can be directly defined using the **SrcDef** option that must have the proper group by format.

As we have seen above, a proper Pivot Table is made from an internal intermediate table resulting from the execution of a `GROUP BY` statement. In many cases, it is simpler or desirable to directly specify this when creating the pivot table. This may be because the source is the result of a complex process including filtering and/or joining tables.

To do this, use the **SrcDef** option, often replacing all other options. For instance, suppose that in the first example we are only interested in weeks 4 and 5. We could of course display it by:

```
select * from pivex where week in (4,5);
```

However, what if this table is a huge table? In this case, the correct way to do it is to define the pivot table as this:

```
create table pivex4
engine=connect table_type=pivot
option_list='PivotCol=what,FncCol=amount'
SrcDef='select who, week, what, sum(amount) from expenses
where week in (4,5) group by who, week, what';
```

If your source table has millions of records and you plan to pivot only a small subset of it, doing so will make a lot of a difference performance wise. In addition, you have entire liberty to use expressions, scalar functions, aliases, join, where and having clauses in your SQL statement. The only constraint is that you are responsible for the result of this statement to have the correct format for the pivot processing.

Using SrcDef also permits to use expressions and/or scalar functions. For instance:

```
create table xpivot (
Who char(10) not null,
What char(12) not null,
First double(8,2) flag=1,
Middle double(8,2) flag=1,
Last double(8,2) flag=1)
engine=connect table_type=PIVOT
option_list='PivotCol=wk,FncCol=amnt'
Srcdef='select who, what, case when week=3 then ''First'' when
week=5 then ''Last'' else ''Middle'' end as wk, sum(amount) *
6.56 as amnt from expenses group by who, what, wk';
```

Now the statement:

```
select * from xpivot;
```

Will display the result:

Who	What	First	Middle	Last

Beth	Beer	104.96	98.40	131.20
Beth	Food	0.00	111.52	78.72
Janet	Beer	118.08	0.00	216.48
Janet	Car	124.64	111.52	78.72
Janet	Food	118.08	0.00	78.72
Joe	Beer	118.08	321.44	91.84
Joe	Car	131.20	0.00	0.00
Joe	Food	203.36	223.04	78.72

Note 1: to avoid multiple lines having the same fixed column values, it is mandatory in **SrcDef** to place the pivot column at the end of the group by list.

Note 2: in the create statement **SrcDef**, it is mandatory to give aliases **to** the columns containing expressions so they are recognized by the other options.

Note 3: in the **SrcDef** select statement, quotes must be escaped because the entire statement is passed to MariaDB between quotes. Alternatively, specify it between double quotes.

Note 4: We could have left CONNECT do the column definitions. However, because they are defined from the sorted names, the Middle column had been placed at the end of them.

Specifying the Columns Corresponding to the Pivot Column

These columns must be named from the values existing in the “pivot” column. For instance, supposing we have the following *pet* table:

name	race	number
John	dog	2
Bill	cat	1
Mary	dog	1
Mary	cat	1
Lisbeth	rabbit	2
Kevin	cat	2
Kevin	bird	6
Donald	dog	1
Donald	fish	3

Pivoting it using *race* as the pivot column is done with:

```
create table pivet
engine=connect table_type=pivot tabname=pet
option_list='PivotCol=race,groupby=1';
```

This gives the result:

name	dog	cat	rabbit	bird	fish
John	2	0	0	0	0
Bill	0	1	0	0	0
Mary	1	1	0	0	0
Lisbeth	0	0	2	0	0
Kevin	0	2	0	6	0
Donald	1	0	0	0	3

By the way, does this ring a bell? It shows that in a way PIVOT tables are doing the opposite of what OCCUR tables do.

We can alternatively define specifically the table columns but what happens if the Pivot column contains values that is not matching a “data” column? There are three cases depending on the specified options and flags.

First case: If no specific options are specified, this is an error an when trying to display the table. The query will abort with an error message stating that a non-matching value was met. Note that because the column list is established when creating the table, this is prone to occur if some rows containing new values for the pivot column are inserted in the source table. If this happens, you should re-create the table or manually add the new columns to the pivot table.

Second case: The accept option was specified. For instance:

```
create table xpivet2 (
name varchar(12) not null,
dog int not null default 0 flag=1,
cat int not null default 0 flag=1)
engine=connect table_type=pivot tabname=pet
option_list='PivotCol=race,groupby=1,Accept=1';
```

No error will be raised and the non-matching values will be ignored. This table will be displayed as:

name	dog	cat
John	2	0
Bill	0	1
Mary	1	1
Lisbeth	0	0
Kevin	0	2
Donald	1	0

Third case: A “dump” column was specified with the flag value equal to 2. All non-matching values will be added in this column. For instance:

```
create table xpivet (
name varchar(12) not null,
dog int not null default 0 flag=1,
cat int not null default 0 flag=1,
other int not null default 0 flag=2)
engine=connect table_type=pivot tabname=pet
option_list='PivotCol=race,groupby=1';
```

This table will be displayed as:

name	dog	cat	other
John	2	0	0
Bill	0	1	0
Mary	1	1	0
Lisbeth	0	0	2
Kevin	0	2	6
Donald	1	0	3

> It is a good idea to provide such a “dump” column if the source table is prone to be inserted new rows that can have a value for the pivot column that did not exist when the pivot table was created.

Pivoting Big Source Tables

This may sometimes be risky. If the pivot column contains too many distinct values, the resulting table may have too many columns. In all cases the process involved, finding distinct values when creating the table or doing the group by when using it, can be very long and sometimes can fail because of exhausted memory.

Restrictions by a where clause should be applied to the source table when creating the pivot table rather than to the pivot table itself. This can be done by creating an intermediate table or using as source a view or a srcdef option.

All PIVOT tables are read only.

5.3.7.6.24 CONNECT TBL Table Type: Table List

Contents

This type allows defining a table as a list of tables of any engine and type. This is more flexible than multiple tables that must be all of the same file type. This type does, but is more powerful than, what is done with the MERGE engine.

The list of the columns of the TBL table may not necessarily include all the columns of the tables of the list. If the name of some columns is different in the sub-tables, the column to use can be specified by its position given by the `FLAG` option of the column. If the `ACCEPT` option is set to true (Y or 1) columns that do not exist in some of the sub-tables are accepted and their value will be null or pseudo-null (this depends on the nullability of the column) for the tables not having this column. The column types can also be different and an automatic conversion will be done if necessary.

Note: If not specified, the column definitions are retrieved from the first table of the table list.

The default database of the sub-tables is the current database or if not, can be specified in the DBNAME option. For the tables that are not in the default database, this can be specified in the table list. For instance, to create a table based on the French table *employe* in the current database and on the English table *employee* of the *db2* database, the syntax of the create statement can be:

```
CREATE TABLE allemp (
  SERIALNO char(5) NOT NULL flag=1,
  NAME varchar(12) NOT NULL flag=2,
  SEX smallint(1),
  TITLE varchar(15) NOT NULL flag=3,
  MANAGER char(5) DEFAULT NULL flag=4,
  DEPARTMENT char(4) NOT NULL flag=5,
  SECRETARY char(5) DEFAULT NULL flag=6,
  SALARY double(8,2) NOT NULL flag=7)
ENGINE=CONNECT table_type=TBL
table_list='employe,db2.employee' option_list='Accept=1';
```

The search for columns in sub tables is done by name and, if they exist with a different name, by their position given by a not null `FLAG` option. Column *sex* exists only in the English table (`FLAG` is `0`). Its values will null value for the French table.

For instance, the query:

```
select name, sex, title, salary from allemp where department = 318;
```

Can reply:

NAME	SEX	TITLE	SALARY
BARBOUD	NULL	VENDEUR	9700.00
MARCHANT	NULL	VENDEUR	8800.00
MINIARD	NULL	ADMINISTRATIF	7500.00
POUPIN	NULL	INGENIEUR	7450.00
ANTERPE	NULL	INGENIEUR	6850.00
LOULOUTE	NULL	SECRETAIRE	4900.00
TARTINE	NULL	OPERATRICE	2800.00
WERTHER	NULL	DIRECTEUR	14500.00
VOITURIN	NULL	VENDEUR	10130.00
BANCROFT	2	SALESMAN	9600.00

MERCHANT	1	SALESMAN	8700.00
SHRINKY	2	ADMINISTRATOR	7500.00
WALTER	1	ENGINEER	7400.00
TONGHO	1	ENGINEER	6800.00
HONEY	2	SECRETARY	4900.00
PLUMHEAD	2	TYPIST	2800.00
WERTHER	1	DIRECTOR	14500.00
WHEELFOR	1	SALESMAN	10030.00

The first 9 rows, coming from the French table, have a null for the *sex* value. They would have 0 if the sex column had been created NOT NULL.

Sub-tables of not CONNECT engines

Sub-tables are accessed as `PROXY` tables. For not CONNECT sub-tables that are accessed via the MySQL API, it is possible like with `PROXY` to change the MYSQL default options. Of course, this will apply to all not CONNECT tables of the list.

Using the TABID special column

The TABID special column can be used to see from which table the rows come from and to restrict the access to only some of the sub-tables.

Let us see the following example where t1 and t2 are MyISAM tables similar to the ones given in the `MERGE` description:

```
create table xt1 (
  a int(11) not null,
  message char(20))
engine=CONNECT table_type=MYSQL tabname='t1'
option_list='database=test,user=root';

create table xt2 (
  a int(11) not null,
  message char(20))
engine=CONNECT table_type=MYSQL tabname='t2'
option_list='database=test,user=root';

create table toto (
  tabname char(8) not null special='TABID',
  a int(11) not null,
  message char(20))
engine=CONNECT table_type=TBL table_list='xt1,xt2';

select * from total;
```

The result returned by the SELECT statement is:

tabname	a	message
xt1	1	Testing
xt1	2	table
xt1	3	t1
xt2	1	Testing
xt2	2	table
xt2	3	t2

Now if you send the query:

```
select * from total where tabname = 'xt2';
```

CONNECT will analyze the where clause and only read the *xt1* table. This can save time if you want to retrieve only a few

sub-tables from a TBL table containing many sub-tables.

Parallel Execution

> Parallel Execution is currently unavailable until some bugs are fixed.

When the sub-tables are located on different servers, it is possible to execute the remote queries simultaneously instead of sequentially. To enable this, set the thread option to yes.

Additional options available for this table type:

Option	Description
Maxerr	The max number of missing tables in the table list before an error is raised. Defaults to 0.
Accept	If true, missing columns are accepted and return null values. Defaults to false.
Thread	If true, enables parallel execution of remote sub-tables.

These options can be specified in the `OPTION_LIST`.

5.3.7.6.25 CONNECT - Using the TBL and MYSQL Table Types Together

Contents

Used together, these types lift all the limitations of the FEDERATED and MERGE engines.

MERGE: Its limitation is obvious, the merged tables must be identical MyISAM tables, and MyISAM is not even the default engine for MariaDB. However, TBL accesses a collection of CONNECT tables, but because these tables can be user specified or internally created MYSQL tables, there is no limitation to the type of the tables that can be merged.

TBL is also much more flexible. The merged tables must not be "identical", they just should have the columns defined in the TBL table. If the type of one column in a merged table is not the one of the corresponding column of the TBL table, the column value will be converted. As we have seen, if one column of the TBL table of the TBL column does not exist in one of the merged table, the corresponding value will be set to null. If columns in a sub-table have a different name, they can be accessed by position using the FLAG column option of CONNECT.

However, one limitation of the TBL type regarding MERGE is that TBL tables are currently read-only; INSERT is not supported by TBL. Also, keep using MERGE to access a list of identical MyISAM tables because it will be faster, not passing by the MySQL API.

FEDERATED(X): The main limitation of FEDERATED is to access only MySQL/MariaDB tables. The MYSQL table type of CONNECT has the same limitation but CONNECT provides the ODBC table type and JDBC table type that can access tables of any RDBS providing an ODBC or JDBC driver (including MySQL even it is not really useful!)

Another major limitation of FEDERATED is to access only one table. By combining TBL and MYSQL tables, CONNECT enables to access a collection of local or remote tables as one table. Of course the sub-tables can be on different servers. With one SELECT statement, a company manager will be able to interrogate results coming from all of his subsidiary computers. This is great for distribution, banking, and many other industries.

Remotely executing complex queries

Many companies or administrations must deal with distributed information. CONNECT enables to deal with it efficiently without having to copy it to a centralized database. Let us suppose we have on some remote network machines *m1, m2, … mn* some information contained in two tables *t1* and *t2*.

Suppose we want to execute on all servers a query such as:

```
select c1, sum(c2) from t1 a, t2 b where a.id = b.id group by c1;
```

This raises many problems. Returning the column values of the *t1* and *t2* tables from all servers can be a lot of network traffic. The group by on the possibly huge resulting tables can be a long process. In addition, the join on the *t1* and *t2* tables may be relevant only if the joined tuples belong to the same machine, obliging to add a condition on an additional tabid or servid special column.

All this can be avoided and optimized by forcing the query to be locally executed on each server and retrieving only the small results of the group by queries. Here is how to do it. For each remote machine, create a table that will retrieve the locally executed query. For instance for m1:

```
create table rt1 engine=connect option_list='host=m1'
srcdef='select c1, sum(c2) as sc2 from t1 a, t2 b where a.id = b.id group by c1';
```

Note the alias for the functional column. An alias would be required for the c1 column if its name was different on some machines. The t1 and t2 table names can also be eventually different on the remote machines. The true names must be used in the `SRCDEF` parameter. This will create a set of tables with two columns named c1 and sc2 [1].

Then create the table that will retrieve the result of all these tables:

```
create table rtall engine=connect table_type=tbl
table_list='rt1,rt2,…,rtn' option_list='thread=yes';
```

Now you can retrieve the desired result by:

```
select c1, sum(sc2) from rtall;
```

Almost all the work will be done on the remote machines, simultaneously thanks to the thread option, making this query super-fast even on big tables placed on many remote machines.

Thread is currently experimental. Use it only for test and report any malfunction on JIRA.

Providing a list of servers

An interesting case is when the query to run on remote machines is the same for all of them. It is then possible to avoid declaring all sub-tables. In this case, the table list option will be used to specify the list of servers the `SRCDEF` query must be sent. This will be a list of URL's and/or Federated server names.

For instance, supposing that federated servers srv1, srv2, … srv*n* were created for all remote servers, it will be possible to create a tbl table allowing getting the result of a query executed on all of them by:

```
create table qall [column definition]
engine=connect table_type=TBL srcdef='a query'
table_list='srv1,srv2,…,srvn' [option_list='thread=yes'];
```

For instance:

```
create table verall engine=connect table_type=TBL srcdef='select @@version' table_list=',server_c
select * from verall;
```

This reply:

@@version
10.0.3-MariaDB-debug
10.0.2-MariaDB

Here the server list specifies a void server corresponding to the local running MariaDB and a federated server named *server_one*.

1. ↑ To generate the columns from the `SRCDEF` query, CONNECT must execute it. This will make sure it is ok. However, if the remote server is not connected yet, or the remote table not existing yet, you can alternatively specify the columns in the create table statement.

5.3.7.6.26 CONNECT Table Types - Special "Virtual" Tables

Contents

The special table types supported by CONNECT are the Virtual table type (VIR - introduced in MariaDB 10.0.15), Directory Listing table type (DIR), the Windows Management Instrumentation Table Type (WMI), and the “Mac Address” type (MAC).

These tables are “virtual tables”, meaning they have no physical data but rather produce result data using specific algorithms. Note that this is close to what Views are, so they could be regarded as special views.

DIR Type

A table of type DIR returns a list of file name and description as a result set. To create a DIR table, use a Create Table statement such as:

```
create table source (
  DRIVE char(2) NOT NULL,
  PATH varchar(256) NOT NULL,
  FNAME varchar(256) NOT NULL,
  FTYPE char(4) NOT NULL,
  SIZE double(12,0) NOT NULL flag=5,
  MODIFIED datetime NOT NULL)
engine=CONNECT table_type=DIR file_name='..\\*.cc';
```

When used in a query, the table returns the same file information listing than the system "DIR `*.cc` " statement would return if executed in the same current directory (here supposedly ..\)

For instance, the query:

```
select fname, size, modified from source
  where fname like '%handler%';
```

Displays:

fname	size	modified
handler	152177	2011-06-13 18:08:29
sql_handler	25321	2011-06-13 18:08:31

Note: the important item in this table is the flag option value (set sequentially from 0 by default) because it determines which particular information item is returned in the column:

Flag value	Information
0	The disk drive (Windows)
1	The file path
2	The file name
3	The file type
4	The file attribute
5	The file size
6	The last write access date
7	The last read access date
8	The file creation date

The Subdir option

When specified in the create table statement, the subdir option indicates to list, in addition to the files contained in the specified directory, all the files verifying the filename pattern that are contained in sub-directories of the specified directory. For instance, using:

```
create table data (
  PATH varchar(256) NOT NULL flag=1,
  FNAME varchar(256) NOT NULL,
  FTYPE char(4) NOT NULL,
  SIZE double(12,0) NOT NULL flag=5)
engine=CONNECT table_type=DIR file_name='*.frm'
option_list='subdir=1';

select path, count(*), sum(size) from data group by path;
```

You will get the following result set showing how many tables are created in the MariaDB databases and what is the total length of the FRM files:

path	count(*)	sum(size)
\CommonSource\mariadb-5.2.7\sql\data\connect\	30	264469
\CommonSource\mariadb-5.2.7\sql\data\mysql\	23	207168
\CommonSource\mariadb-5.2.7\sql\data\test\	22	196882

The Nodir option (Windows)

The Boolean Nodir option can be set to false (0 or no) to add directories that match the file name pattern from the listed files (it is true by default). This is an addition to CONNECT version 1.6. Previously, directory names matching pattern were listed on Windows. Directories were and are never listed on Linux.

Note: The way file names are retrieved makes positional access to them impossible. Therefore, DIR tables cannot be indexed or sorted when it is done using positions.

Be aware, in particular when using the subdir option, that queries on DIR tables are slow and can last almost forever if made on a directory that contains a great number of files in it and its sub-directories.

dir tables can be used to populate a list of files used to create a multiple=2 table. However, this is not as useful as it was when the multiple 3 did not exist.

Windows Management Instrumentation Table Type “WMI”

Note: This table type is available on Windows only.

WMI provides an operating system interface through which instrumented components provide information. Some Microsoft tools to retrieve information through WMI are the WMIC console command and the WMI CMI Studio application.

The CONNECT WMI table type enables administrators and operators not capable of scripting or programming on top of WMI to enjoy the benefit of WMI without even learning about it. It permits to present this information as tables that can be queried, transformed, copied in documents or other tables.

To create a WMI table displaying information coming from a WMI provider, you must provide the namespace and the class name that characterize the information you want to retrieve. The best way to find them is to use the WMI CIM Studio that have tools to browse namespaces and classes and that can display the names of the properties of that class.

The column names of the tables must be the names (case insensitive) of the properties you want to retrieve. For instance:

```
create table alias (
  friendlyname char(32) not null,
  target char(50) not null)
engine=CONNECT table_type='WMI'
option_list='Namespace=root\\cli,Class=Msft_CliAlias';
```

WMI tables returns one row for each instance of the related information. The above example is handy to get the class equivalent of the alias of the WMIC command and also to have a list of many classes commonly used.

Because most of the useful classes belong to the 'root\cimv2' namespace, this is the default value for WMI tables when the namespace is not specified. Some classes have many properties whose name and type may not be known when creating the table. To find them, you can use the WMI CMI Studio application but his will be rarely required because CONNECT is

able to retrieve them.

Actually, the class specification also has default values for some namespaces. For the ‘root\cli’ namespace the class name defaults to ‘Msft_CliAlias’ and for the ‘root_cimv2’ namespace the class default value is ‘Win32_ComputerSystemProduct’. Because many class names begin with ‘Win32_’ it is not necessary to say it and specifying the class as ‘Product’ will effectively use class ‘Win32_Product’.

For example if you define a table as:

```
create table CSPROD engine=CONNECT table_type='WMI';
```

It will return the information on the current machine, using the class ComputerSystemProduct of the CIMV2 namespace. For instance:

```
select * from csprod;
```

Will return a result such as:

Column	Row 1
Caption	Computer system product
Description	Computer system product
IdentifyingNumber	LXAP50X32982327A922300
Name	Aspire 8920
SKUNumber	
UUID	00FC523D-B8F7-DC12-A70E-00B0D1A46136
Vendor	Acer
Version	Aspire 8920

Note: This is a transposed display that can be obtained with some GUI.

Getting column information

An issue, when creating a WMI table, is to make its column definition. Indeed, even when you know the namespace and the class for the wanted information, it is not easy to find what are the names and types of its properties. However, because CONNECT can retrieve this information from the WMI provider, you can simply omit defining columns and CONNECT will do the job.

Alternatively, you can get this information using a catalog table (see below).

Performance Consideration

Some WMI providers can be very slow to answer. This is not an issue for those that return few object instances, such as the ones returning computer, motherboard, or Bios information. They generally return only one row (instance). However, some can return many rows, in particular the "CIM_DataFile" class. This is why care must be taken about them.

Firstly, it is possible to limit the allocated result size by using the ‘Estimate’ create table option. To avoid result truncation, CONNECT allocates a result of 100 rows that is enough for almost all tables.The 'Estimate' option permits to reduce this size for all classes that return only a few rows, and in some rare case to increase it to avoid truncation.

However, it is not possible to limit the time taken by some WMI providers to answer, in particular the CIM_DATAFILE class. Indeed the Microsoft documentation says about it:

> "Avoid enumerating or querying for all instances of CIM_DataFile on a computer because the volume of data is likely to either affect performance or cause the computer to stop responding."

Sure enough, even a simple query such as:

```
select count(*) from cim where drive = 'D:' and path like '\\MariaDB\\%';
```

is prone to last almost forever (probably due to the LIKE clause). This is why, when not asking for some specific items, you should consider using the DIR table type instead.

Syntax of WMI queries

Queries to WMI providers are done using the WQL language, not the SQL language. CONNECT does the job of making the WQL query. However, because of the restriction of the WQL syntax, the WHERE clause will be generated only when respecting the following restrictions:

1. No function.
2. No comparison between two columns.
3. No expression (currently a CONNECT restriction)
4. No BETWEEN and IN predicates.

Filtering with WHERE clauses not respecting these conditions will still be done by MariaDB only, except in the case of CIM_Datafile class for the reason given above.

However, there is one point that is not covered yet, the syntax used to specify dates in queries. WQL does not recognize dates as number items but translates them to its internal format dates specified as text. Many formats are recognized as described in the Microsoft documentation but only one is useful because common to WQL and MariaDB SQL. Here is an example of a query on a table named "cim" created by:

```
create table cim (
  Name varchar(255) not null,
  LastModified datetime not null)
engine=CONNECT table_type='WMI'
option_list='class=CIM_DataFile,estimate=5000';
```

The date must be specified with the format in which CIM DATETIME values are stored (WMI uses the date and time formats defined by the Distributed Management Task Force).

```
select * from cim where drive = 'D:' and path = '\\PlugDB\\Bin\\'
     and lastmodified > '20120415000000.000000+120';
```

This syntax must be strictly respected. The text has the format:

```
yyyymmddHHMMSS.mmmmmmsUUU
```

It is: year, month, day, hour, minute, second, millisecond, and signed minute deviation from UTC. This format is locale-independent so you can write a query that runs on any machine.

Note 1: The WMI table type is available only in Windows versions of CONNECT.

Note 2: WMI tables are read only.

Note 3: WMI tables are not indexable.

Note 4: WMI consider all strings as case insensitive.

MAC Address Table Type “MAC”

> **Note:** This table type is available on Windows only.

This type is used to display various general information about the computer and, in particular, about its network cards. To create such a table, the syntax to use is:

```
create table tabname (column definition)
engine=CONNECT table_type=MAC;
```

Column names can be freely chosen because their signification, i.e. the values they will display, comes from the specified Flag option. The valid values for Flag are:

Flag	Valeur	Type
1	Host name	varchar(132)
2	Domain	varchar(132)
3	DNS address	varchar(24)
4	Node type	int(1)
5	Scope ID	varchar(256)
6	Routing	int(1)

7	Proxy	int(1)
8	DNS	int(1)
10	Name	varchar(260)
11	Description	varchar(132)
12	MAC address	char(24)
13	Type	int(3)
14	DHCP	int(1)
15	IP address	char(16)
16	SUBNET mask	char(16)
17	GATEWAY	char(16)
18	DHCP server	char(16)
19	Have WINS	int(1)
20	Primary WINS	char(16)
21	Secondary WINS	char(16)
22	Lease obtained	datetime
23	Lease expires	datetime

Note: The information of columns having a Flag value less than 10 are unique for the computer, the other ones are specific to the network cards of the computer.

For instance, you can define the table *macaddr* as:

```
create table macaddr (
  Host varchar(132) flag=1,
  Card varchar(132) flag=11,
  Address char(24) flag=12,
  IP char(16) flag=15,
  Gateway char(16) flag=17,
  Lease datetime flag=23)
engine=CONNECT table_type=MAC;
```

If you execute the query:

```
select host, address, ip, gateway, lease from macaddr;
```

It will return, for example:

Host	Address	IP	Gateway	Lease
OLIVIER	00-A0-D1-A4-61-36	0.0.0.0	0.0.0.0	1970-01-01 00:00:00
OLIVIER	00-1D-E0-9B-90-0B	192.168.0.10	192.168.0.254	2011-09-18 10:28:5

5.3.7.6.27 CONNECT Table Types - VIR

Contents

VIR Type

A VIR table is a virtual table having only Special or Virtual columns. Its only property is its “size”, or cardinality, meaning the number of virtual rows it contains. It is created using the syntax:

```
CREATE TABLE name [coldef] ENGINE=CONNECT TABLE_TYPE=VIR
[BLOCK_SIZE=n];
```

The optional `BLOCK_SIZE` option gives the size of the table, defaulting to 1 if not specified. When its columns are not specified, it is almost equivalent to a SEQUENCE table “seq_1_to_Size”.

Displaying constants or expressions

Many DBMS use a no-column one-line table to do this, often call “dual”. MySQL and MariaDB use syntax where no table is specified. With CONNECT, you can achieve the same purpose with a virtual table, with the noticeable advantage of being able to display several lines. For example:

```
create table virt engine=connect table_type=VIR block_size=10;
select concat('The square root of ', n, ' is') what,
round(sqrt(n),16) value from virt;
```

This will return:

what	value
The square root of 1 is	1.0000000000000000
The square root of 2 is	1.4142135623730951
The square root of 3 is	1.7320508075688772
The square root of 4 is	2.0000000000000000
The square root of 5 is	2.2360679774997898
The square root of 6 is	2.4494897427831779
The square root of 7 is	2.6457513110645907
The square root of 8 is	2.8284271247461903
The square root of 9 is	3.0000000000000000
The square root of 10 is	3.1622776601683795

What happened here? First of all, unlike Oracle “dual” tableS that have no columns, a MariaDB table must have at least one column. By default, CONNECT creates VIR tables with one special column. This can be seen with the SHOW CREATE TABLE statement:

```
CREATE TABLE `virt` (
`n` int(11) NOT NULL `SPECIAL`=ROWID,
PRIMARY KEY (`n`)
) ENGINE=CONNECT DEFAULT CHARSET=latin1 `TABLE_TYPE`='VIR'
`BLOCK_SIZE`=10
```

This special column is called “n” and its value is the row number starting from 1. It is purely a virtual table and no data file exists corresponding to it and to its index. It is possible to specify the columns of a VIR table but they must be CONNECT special columns or virtual columns. For instance:

```
create table virt2 (
n int key not null special=rowid,
sig1 bigint as ((n*(n+1))/2) virtual,
sig2 bigint as(((2*n+1)*(n+1)*n)/6) virtual)
engine=connect table_type=VIR block_size=10000000;
select * from virt2 limit 995, 5;
```

This table shows the sum and the sum of the square of the n first integers:

n	sig1	sig2
996	496506	329845486
997	497503	330839495
998	498501	331835499
999	499500	332833500

1000	500500	333833500

Note that the size of the table can be made very big as there no physical data. However, the result should be limited in the queries. For instance:

```
select * from virt2 where n = 1664510;
```

Such a query could last very long if the rowid column were not indexed. Note that by default, CONNECT declares the “n” column as a primary key. Actually, VIR tables can be indexed but only on the ROWID (or ROWNUM) columns of the table. This is a virtual index for which no data is stored.

Generating a Table filled with constant values

An interesting use of virtual tables, which often cannot be achieved with a table of any other type, is to generate a table containing constant values. This is easily done with a virtual table. Let us define the table FILLER as:

```
create table filler engine=connect table_type=VIR block_size=5000000;
```

Here we choose a size larger than the biggest table we want to generate. Later if we need a table pre- filled with default and/or null values, we can do for example:

```
create table tp (
id int(6) key not null,
name char(16) not null,
salary float(8,2));
insert into tp select n, 'unknown', NULL from filler where n <= 10000;
```

This will generate a table having 10000 rows that can be updated later when needed. Note that a SEQUENCE table could have been used here instead of FILLING .

VIR tables vs. SEQUENCE tables

With just its default column, a VIR table is almost equivalent to a SEQUENCE table. The syntax used is the main difference, for instance:

```
select * from seq_100_to_150_step_10;
```

can be obtained with a VIR table (of size >= 15) by:

```
select n*10 from vir where n between 10 and 15;
```

Therefore, the main difference is to be able to define the columns of VIR tables. Unfortunately, there are currently many limitations to virtual columns that hopefully should be removed in the future.

5.3.7.6.28 CONNECT Table Types - OEM: Implemented in an External LIB

Contents

Although CONNECT provides a rich set of table types, specific applications may need to access data organized in a way that is not handled by its existing foreign data wrappers (FDW). To handle these cases, CONNECT features an interface that enables developers to implement in C++ the required table wrapper and use it as if it were part of the standard CONNECT table type list. CONNECT can use these additional handlers providing the corresponding external module (dll or shared lib) be available.

To create such a table on an existing handler, use a Create Table statement as shown below.

```
create table xtab (column definitions)
engine=CONNECT table_type=OEM module='libname'
subtype='MYTYPE' [standard table options]
Option_list='Myopt=foo';
```

The option module gives the name of the DLL or shared library implementing the OEM wrapper for the table type. This library must be located in the plugin directory like all other plugins or UDF's.

This library must export a function *GetMYTYPE*. The option subtype enables CONNECT to have the name of the exported function and to use the new table type. Other options are interpreted by the OEM type and can also be specified within the *option_list* option.

Column definitions can be unspecified only if the external wrapper is able to return this information. For this it must export a function ColMYTYPE returning these definitions in a format acceptable by the CONNECT discovery function.

Which and how options must be specified and the way columns must be defined may vary depending on the OEM type used and should be documented by the OEM type implementer(s).

An OEM Table Example

The OEM table REST described in Adding the REST Feature as a Library Called by an OEM Table permits using REST-like tables with MariaDB binary distributions containing but not enabling the REST table type

Of course, the mongo (dll or so) exporting the GetREST and colREST functions must be available in the plugin directory for all this to work.

Some Currently Available OEM Table Modules and Subtypes

Module	Subtype	Description
libhello	HELLO	A sample OEM wrapper displaying a one line table saying "Hello world"
mongo	MONGO	Enables using tables based on MongoDB collections.
Tabfic	FIC	Handles files having the Windev HyperFile format.
Tabofx	OFC	Handles Open Financial Connectivity files.
Tabofx	QIF	Handles Quicken Interchange Format files.
Cirpack	CRPK	Handles CDR's from Cirpack UTP's.
Tabplg	PLG	Access tables from the PlugDB DBMS.

How to implement an OEM handler is out of the scope of this document.

5.3.7.6.29 CONNECT Table Types - Catalog Tables

A catalog table is one that returns information about another table, or data source. It is similar to what MariaDB commands such as `DESCRIBE` or `SHOW` do. Applied to local tables, this just duplicates what these commands do, with the noticeable difference that they are tables and can be used inside queries as joined tables or inside sub-selects.

But their main interest is to enable querying the structure of external tables that cannot be directly queried with description commands. Let's see an example:

Suppose we want to access the tables from a Microsoft Access database as an ODBC type table. The first information we must obtain is the list of tables existing in this data source. To get it, we will create a catalog table that will return it extracted from the result set of the SQLTables ODBC function:

```
create table tabinfo (
  table_name varchar(128) not null,
  table_type varchar(16) not null)
engine=connect table_type=ODBC catfunc=tables
Connection='DSN=MS Access Database;DBQ=C:/Program
Files/Microsoft Office/Office/1033/FPNWIND.MDB;';
```

The SQLTables function returns a result set having the following columns:

Field	Data Type	Null	Info Type	Flag Value
Table_Cat	char(128)	NO	FLD_CAT	17
Table_Name	char(128)	NO	FLD_SCHEM	18
Table_Name	char(128)	NO	FLD_NAME	1

Table_Type	char(16)	NO	FLD_TYPE	2
Remark	char(128)	NO	FLD_REM	5

Note: The Info Type and Flag Value are CONNECT interpretations of this result.

Here we could have omitted the column definitions of the catalog table or, as in the above example, chose the columns returning the name and type of the tables. If specified, the columns must have the exact name of the corresponding SQLTables result set, or be given a different name with the matching flag value specification.

(The Table_Type can be TABLE, SYSTEM TABLE, VIEW, etc.)

For instance, to get the tables we want to use we can ask:

```
select table_name from tabinfo where table_type = 'TABLE';
```

This will return:

table_name
Categories
Customers
Employees
Products
Shippers
Suppliers

Now we want to create the table to access the CUSTOMERS table. Because CONNECT can retrieve the column description of ODBC tables, it not necessary to specify them in the create table statement:

```
create table Customers engine=connect table_type=ODBC
Connection='DSN=MS Access Database;DBQ=C:/Program
Files/Microsoft Office/Office/1033/FPNWIND.MDB;';
```

However, if we prefer to specify them (to eventually modify them) we must know what the column definitions of that table are. We can get this information with a catalog table. This is how to do it:

```
create table custinfo engine=connect table_type=ODBC
tabname=customers catfunc=columns
Connection='DSN=MS Access Database;DBQ=C:/Program
Files/Microsoft Office/Office/1033/FPNWIND.MDB;';
```

Alternatively it is possible to specify what columns of the catalog table we want:

```
create table custinfo (
  column_name char(128) not null,
  type_name char(20) not null,
  length int(10) not null flag=7,
  prec smallint(6) not null flag=9)
  nullable smallint(6) not null)
engine=connect table_type=ODBC tabname=customers
catfunc=columns
Connection='DSN=MS Access Database;DBQ=C:/Program
Files/Microsoft Office/Office/1033/FPNWIND.MDB;';
```

To get the column info:

```
select * from custinfo;
```

which results in this table:

column_name	type_name	length	prec	nullable
CustomerID	VARCHAR	5	0	1
CompanyName	VARCHAR	40	0	1
ContactName	VARCHAR	30	0	1

ContactTitle	VARCHAR	30	0	1
Address	VARCHAR	60	0	1
City	VARCHAR	15	0	1
Region	VARCHAR	15	0	1
PostalCode	VARCHAR	10	0	1
Country	VARCHAR	15	0	1
Phone	VARCHAR	24	0	1
Fax	VARCHAR	24	0	1

Now you can create the CUSTOMERS table as:

```
create table Customers (
  CustomerID varchar(5),
  CompanyName varchar(40),
  ContactName varchar(30),
  ContactTitle varchar(30),
  Address varchar(60),
  City varchar(15),
  Region varchar(15),
  PostalCode varchar(10),
  Country varchar(15),
  Phone varchar(24),
  Fax varchar(24))
engine=connect table_type=ODBC block_size=10
Connection='DSN=MS Access Database;DBQ=C:/Program
Files/Microsoft Office/Office/1033/FPNWIND.MDB;';
```

Let us explain what we did here: First of all, the creation of the catalog table. This table returns the result set of an ODBC SQLColumns function sent to the ODBC data source. Columns functions always return a data set having some of the following columns, depending on the table type:

Field	Data Type	Null	Info Type	Flag Value	Returned by
Table_Cat*	char(128)	NO	FLD_CAT	17	ODBC, JDBC
Table_Schema*	char(128)	NO	FLD_SCEM	18	ODBC, JDBC
Table_Name	char(128)	NO	FLD_TABNAME	19	ODBC, JDBC
Column_Name	char(128)	NO	FLD_NAME	1	ALL
Data_Type	smallint(6)	NO	FLD_TYPE	2	ALL
Type_Name	char(30)	NO	FLD_TYPENAME	3	ALL
Column_Size*	int(10)	NO	FLD_PREC	4	ALL
Buffer_Length*	int(10)	NO	FLD_LENGTH	5	ALL
Decimal_Digits*	smallint(6)	NO	FLD_SCALE	6	ALL
Radix	smallint(6)	NO	FLD_RADIX	7	ODBC, JDBC, MYSQL
Nullable	smallint(6)	NO	FLD_NULL	8	ODBC, JDBC, MYSQL
Remarks	char(255)	NO	FLD_REM	9	ODBC, JDBC, MYSQL
Collation	char(32)	NO	FLD_CHARSET	10	MYSQL
Key	char(4)	NO	FLD_KEY	11	MYSQL
Default_value	N.A.		FLD_DEFAULT	12	
Privilege	N.A.		FLD_PRIV	13	
Date_fmt	char(32)	NO	FLD_DATEFMT	15	MYSQL
Xpath/Jpath	Varchar(256)	NO	FLD_FORMAT	16	XML/JSON

'*': These names have changed since earlier versions of CONNECT.

Note: ALL includes the ODBC, JDBC, MYSQL, DBF, CSV, PROXY, TBL, XML, JSON, XCOL, and WMI table types. More could be added later.

We chose among these columns the ones that were useful for our create statement, using the flag value when we gave them a different name (case insensitive).

The options used in this definition are the same as the one used later for the actual CUSTOMERS data tables except that:

1. The `TABNAME` option is mandatory here to specify what the queried table name is.
2. The `CATFUNC` option was added both to indicate that this is a catalog table, and to specify that we want column information.

Note: If the `TABNAME` option had not been specified, this table would have returned the columns of all the tables defined in the connected data source.

Currently the available `CATFUNC` are:

Function	Specified as:	Applies to table types:
FNC_TAB	**tab**les	ODBC, JDBC, MYSQL
FNC_COL	**col**umns	ODBC, JDBC, MYSQL, DBF, CSV, PROXY, XCOL, TBL, WMI
FNC_DSN	**datasource**s **dsn** **sqldatasource**s	ODBC
FNC_DRIVER	**driver**s **sqldriver**s	ODBC, JDBC

Note: Only the bold part of the function name specification is required.

The `DATASOURCE` and `DRIVERS` functions respectively return the list of available data sources and ODBC drivers available on the system.

The SQLDataSources function returns a result set having the following columns:

Field	Data Type	Null	Info Type	Flag value
Name	varchar(256)	NO	FLD_NAME	1
Description	varchar(256)	NO	FLD_REM	9

To get the data source, you can do for instance:

```
create table datasources (
engine=CONNECT table_type=ODBC catfunc=DSN;
```

The SQLDrivers function returns a result set having the following columns:

Field	Type	Null	Info Type	Flag value
Description	varchar(128)	YES	FLD_NAME	1
Attributes	varchar(256)	YES	FLD_REM	9

You can get the driver list with:

```
create table drivers
engine=CONNECT table_type=ODBC catfunc=drivers;
```

Another example, WMI table

To create a catalog table returning the attribute names of a WMI class, use the same table options as the ones used with the normal WMI table plus the additional option ‘catfunc=columns’. If specified, the columns of such a catalog table can be chosen among the following:

Name	Type	Flag	Description
Column_Name	CHAR	1	The name of the property
Data_Type	INT	2	The SQL data type
Type_Name	CHAR	3	The SQL type name
Column_Size	INT	4	The field length in characters

Buffer_Length	INT	5	Depends on the coding
Scale	INT	6	Depends on the type

If you wish to use a different name for a column, set the Flag column option.

For example, before creating the "csprod" table, you could have created the info table:

```
create table CSPRODCOL (
  Column_name char(64) not null,
  Data_Type int(3) not null,
  Type_name char(16) not null,
  Length int(6) not null,
  Prec int(2) not null flag=6)
engine=CONNECT table_type='WMI' catfunc=col;
```

Now the query:

```
select * from csprodcol;
```

will display the result:

Column_name	Data_Type	Type_name	Length	Prec
Caption	1	CHAR	255	1
Description	1	CHAR	255	1
IdentifyingNumber	1	CHAR	255	1
Name	1	CHAR	255	1
SKUNumber	1	CHAR	255	1
UUID	1	CHAR	255	1
Vendor	1	CHAR	255	1
Version	1	CHAR	255	1

This can help to define the columns of the matching normal table.

Note 1: The column length, for the Info table as well as for the normal table, can be chosen arbitrarily, it just must be enough to contain the returned information.

Note 2: The Scale column returns 1 for text columns (meaning case insensitive); 2 for float and double columns; and 0 for other numeric columns.

Catalog Table result size limit

Because catalog tables are processed like the information retrieved by “Discovery” when table columns are not specified in a Create Table statement, their result set is entirely retrieved and memory allocated.

By default, this allocation is done for a maximum return line number of:

Catfunc	Max lines
Drivers	256
Data Sources	512
Columns	20,000
Tables	10,000

When the number of lines retrieved for a table is more than this maximum, a warning is issued by CONNECT. This is mainly prone to occur with columns (and also tables) with some data sources having many tables when the table name is not specified.

If this happens, it is possible to increase the default limit using the MAXRES option, for instance:

```
create table allcols engine=connect table_type=odbc
connection='DSN=ORACLE_TEST;UID=system;PWD=manager'
option_list='Maxres=110000' catfunc=columns;
```

Indeed, because the entire table result is memorized before the query is executed; the returned value would be limited even on a query such as:

```
select count(*) from allcols;
```

5.3.7.7 CONNECT - Security

The use of the CONNECT engine requires the `FILE` privilege for "outward" tables. This should not be an important restriction. The use of CONNECT "outward" tables on a remote server seems of limited interest without knowing the files existing on it and must be protected anyway. On the other hand, using it on the local client machine is not an issue because it is always possible to create locally a user with the `FILE` privilege.

5.3.7.8 CONNECT - OEM Table Example

This is an example showing how an OEM table can be implemented.

The header File `my_global.h` :

```
/*****************************************************************/
/*  Definitions needed by the included files.                    */
/*****************************************************************/
#if !defined(MY_GLOBAL_H)
#define MY_GLOBAL_H
typedef unsigned int uint;
typedef unsigned int uint32;
typedef unsigned short ushort;
typedef unsigned long ulong;
typedef unsigned long DWORD;
typedef char *LPSTR;
typedef const char *LPCSTR;
typedef int BOOL;
#if defined(__WIN__)
typedef void *HANDLE;
#else
typedef int HANDLE;
#endif
typedef char *PSZ;
typedef const char *PCSZ;
typedef unsigned char BYTE;
typedef unsigned char uchar;
typedef long long longlong;
typedef unsigned long long ulonglong;
typedef char my_bool;
struct charset_info_st {};
typedef const charset_info_st CHARSET_INFO;
#define FALSE 0
#define TRUE  1
#define Item char
#define MY_MAX(a,b) ((a>b)?(a):(b))
#define MY_MIN(a,b) ((a<b)?(a):(b))
#endif // MY_GLOBAL_H
```

Note: This is a fake `my_global.h` that just contains what is useful for the `jmgoem.cpp` source file.

The source File `jmgoem.cpp` :

```
/************* jmgoem C++ Program Source Code File (.CPP) **************/
/* PROGRAM NAME: jmgoem    Version 1.0                                 */
/*  (C) Copyright to the author Olivier BERTRAND          2017         */
/*  This program is the Java MONGO OEM module definition.             */
/***********************************************************************/

/***********************************************************************/
/*  Definitions needed by the included files.                          */
/***********************************************************************/
#include "my_global.h"

/***********************************************************************/
/*  Include application header files:                                  */
```

```
/*  global.h    is header containing all global declarations.          */
/*  plgdbsem.h  is header containing the DB application declarations.  */
/*  (x)table.h  is header containing the TDBASE declarations.          */
/*  tabext.h    is header containing the TDBEXT declarations.          */
/*  mongo.h     is header containing the MONGO declarations.           */
/*********************************************************************/
#include "global.h"
#include "plgdbsem.h"
#if defined(HAVE_JMGO)
#include "csort.h"
#include "javaconn.h"
#endif   // HAVE_JMGO
#include "xtable.h"
#include "tabext.h"
#include "mongo.h"

/*********************************************************************/
/*  These functions are exported from the MONGO library.             */
/*********************************************************************/
extern "C" {
  PTABDEF __stdcall GetMONGO(PGLOBAL, void*);
  PQRYRES __stdcall ColMONGO(PGLOBAL, PTOS, void*, char*, char*, bool);
} // extern "C"

/*********************************************************************/
/*  DB static variables.                                            */
/*********************************************************************/
int TDB::Tnum;
int DTVAL::Shift;
#if defined(HAVE_JMGO)
int    CSORT::Limit = 0;
double CSORT::Lg2 = log(2.0);
size_t CSORT::Cpn[1000] = {0};          /* Precalculated cmpnum values */
#if defined(HAVE_JAVACONN)
char *JvmPath = NULL;
char *ClassPath = NULL;
char *GetPluginDir(void)
{return "C:/mongo-java-driver/mongo-java-driver-3.4.2.jar;"
        "C:/MariaDB-10.1/MariaDB/storage/connect/";}
char *GetJavaWrapper(void) {return (char*)"wrappers/Mongo3Interface";}
#else   // !HAVE_JAVACONN
HANDLE JAVAConn::LibJvm;              // Handle to the jvm DLL
CRTJVM JAVAConn::CreateJavaVM;
GETJVM JAVAConn::GetCreatedJavaVMs;
#if defined(_DEBUG)
GETDEF JAVAConn::GetDefaultJavaVMInitArgs;
#endif  //  _DEBUG
#endif // !HAVE_JAVACONN
#endif   // HAVE_JMGO

/*********************************************************************/
/*  This function returns a Mongo definition class.                  */
/*********************************************************************/
PTABDEF __stdcall GetMONGO(PGLOBAL g, void *memp)
{
  return new(g, memp) MGODEF;
} // end of GetMONGO

#ifdef NOEXP
/*********************************************************************/
/* Functions to be defined if not exported by the CONNECT version.   */
/*********************************************************************/
bool IsNum(PSZ s)
{
  for (char *p = s; *p; p++)
    if (*p == ']')
      break;
    else if (!isdigit(*p) || *p == '-')
      return false;

  return true;
} // end of IsNum
#endif

/*********************************************************************/
```

```
/*  Return the columns definition to MariaDB.                          */
/**********************************************************************/
PQRYRES __stdcall ColMONGO(PGLOBAL g, PTOS tp, char *tab,
                                               char *db, bool info)
{
#ifdef NOMGOCOL
  // Cannot use discovery
  strcpy(g->Message, "No discovery, MGOColumns is not accessible");
  return NULL;
#else
  return MGOColumns(g, db, NULL, tp, info);
#endif
} // end of ColMONGO
```

The file `mongo.def` : (required only on Windows)

```
LIBRARY     MONGO
EXPORTS
   GetMONGO     @1
   ColMONGO     @2
```

Compiling this OEM

To compile this OEM module, first make the two or three required files by copy/pasting from the above listings.

Even if this module is to be used with a binary distribution, you need some source files in order to successfully compile it. At least the CONNECT header files that are included in `jmgoem.cpp` and the ones they can include. This can be obtained by downloading the MariaDB source file tar.gz and extracting from it the CONNECT sources files in a directory that will be added to the additional source directories if it is not the directory containing the above files.

The module must be linked to the `ha_connect.lib` of the binary version it will used with. Recent distributions add this lib in the plugin directory.

The resulting module, for instance `mongo.so` or `mongo.dll` , must be placed in the plugin directory of the MariaDB server. Then, you will be able to use MONGO like tables simply replacing in the CREATE TABLE statement the option `TABLE_TYPE=MONGO` with `TABLE_TYPE=OEM SUBTYPE=MONGO MODULE='mongo.(so|dll)'` . Actually, the module name, here supposedly 'mongo', can be anything you like.

This will work with the last (not yet) distributed versions of MariaDB 10.0 and 10.1 because, even it is not enabled, the MONGO type is included in them. This is also the case for MariaDB 10.2.9 but then, on Windows, you will have to define NOEXP and NOMGOCOL because these functions are not exported by this version.

To implement for older versions that do not contain the MONGO type, you can add the corresponding source files, namely `javaconn.cpp` , `jmgfam.cpp` , `jmgoconn.cpp` , `mongo.cpp` and `tabjmg.cpp` that you should find in the CONNECT extracted source files if you downloaded a recent version. As they include `my_global.h` , this is the reason why the included file was named this way. In addition, your compiling should define `HAVE_JMGO` and `HAVE_JAVACONN` . Of course, this is possible only if `ha_connect.lib` is available.

5.3.7.9 Using CONNECT

Using CONNECT - General Information
Using CONNECT - General Information.

Using CONNECT - Virtual and Special Columns
Virtual and special columns example usage

Using CONNECT - Importing File Data Into MariaDB Tables
Directly using external (file) data has many advantages

Using CONNECT - Exporting Data From MariaDB
Exporting data from MariaDB with CONNECT

Using CONNECT - Indexing
Indexing with the CONNECT handler

Using CONNECT - Condition Pushdown
Using CONNECT - Condition Pushdown.

5.3.7.9.1 Using CONNECT - General Information

Contents

The main characteristic of CONNECT is to enable accessing data scattered on a machine as if it was a centralized database. This, and the fact that locking is not used by connect (data files are open and closed for each query) makes CONNECT very useful for importing or exporting data into or from a MariaDB database and also for all types of Business Intelligence applications. However, it is not suited for transactional applications.

For instance, the index type used by CONNECT is closer to bitmap indexing than to B-trees. It is very fast for retrieving result but not when updating is done. In fact, even if only one indexed value is modified in a big table, the index is entirely remade (yet this being four to five times faster than for a b-tree index). But normally in Business Intelligence applications, files are not modified so often.

If you are using CONNECT to analyze files that can be modified by an external process, the indexes are of course not modified by it and become outdated. Use the OPTIMIZE TABLE command to update them before using the tables based on them.

This means also that CONNECT is not designed to be used by centralized servers, which are mostly used for transactions and often must run a long time without human intervening.

Performance

Performances vary a great deal depending on the table type. For instance, ODBC tables are only retrieved as fast as the other DBMS can do. If you have a lot of queries to execute, the best way to optimize your work can be sometime to translate the data from one type to another. Fortunately this is very simple with CONNECT. Fixed formats like FIX, BIN or VEC tables can be created from slower ones by commands such as:

```
Create table fastable table_specs select * from slowtable;
```

`FIX` and `BIN` are often the better choice because the I/O functions are done on blocks of `BLOCK_SIZE` rows. `VEC` tables can be very efficient for tables having many columns only a few being used in each query. Furthermore, for tables of reasonable size, the `MAPPED` option can very often speed up many queries.

Create Table statement

Be aware of the two broad kinds of CONNECT tables:

Inward	They are table whose file name is not specified at create. An empty file will be given a default name (*tabname.tabtype*) and will be populated like for other engines. They do not require the FILE privilege and can be used for testing purpose.
Outward	They are all other `CONNECT` tables and access external data sources or files. They are the true useful tables but require the `FILE` privilege.

Drop Table statement

For outward tables, the DROP TABLE statement just removes the table definition but does not erase the table data. However, dropping an inward tables also erase the table data as well.

Alter Table statement

Be careful using the ALTER TABLE statement. Currently the data compatibility is not tested and the modified definition can become incompatible with the data. In particular, Alter modifies the table definition only but does not modify the table data. Consequently, the table type should not be modified this way, except to correct an incorrect definition. Also adding, dropping or modifying columns may be wrong because the default offset values (when not explicitly given by the FLAG option) may be wrong when recompiled with missing columns.

Safe use of ALTER is for indexing, as we have seen earlier, and to change options such as MAPPED or HUGE those do not impact the data format but just the way the data file is accessed. Modifying the BLOCK_SIZE option is all right with FIX, BIN, DBF, split VEC tables; however it is unsafe for VEC tables that are not split (only one data file) because at their creation the estimate size has been made a multiple of the block size. This can cause errors if this estimate is not a multiple of the new value of the block size.

In all cases, it is safer to drop and re-create the table (outward tables) or to make another one from the table that must be modified.

Update and Delete for File Tables

CONNECT can execute these commands using two different algorithms:

- It can do it in place, directly modifying rows (update) or moving rows (delete) within the table file. This is a fast way to do it in particular when indexing is used.
- It can do it using a temporary file to make the changes. This is required when updating variable record length tables and is more secure in all cases.

The choice between these algorithms depends on the session variable connect_use_tempfile.

5.3.7.9.2 Using CONNECT - Virtual and Special Columns

CONNECT supports MariaDB virtual and persistent columns. It is also possible to declare a column as being a CONNECT special column. Let us see on an example how this can be done. The boys table we have seen previously can be recreated as:

```
create table boys (
  linenum int(6) not null default 0 special=rowid,
  name char(12) not null,
  city char(12) not null,
  birth date not null date_format='DD/MM/YYYY',
  hired date not null date_format='DD/MM/YYYY' flag=36,
  agehired int(3) as (floor(datediff(hired,birth)/365.25))
  virtual,
  fn char(100) not null default '' special=FILEID)
engine=CONNECT table_type=FIX file_name='boys.txt' mapped=YES lrecl=47;
```

We have defined two CONNECT special columns. You can give them any name; it is the field SPECIAL option that specifies the special column functional name.

Note: the default values specified for the special columns do not mean anything. They are specified just to prevent getting warning messages when inserting new rows.

For the definition of the *agehired* virtual column, no CONNECT options can be specified as it has no offset or length, not being stored in the file.

The command:

```
select * from boys where city = 'boston';
```

will return:

linenum	name	city	birth	hired	agehired	fn
1	John	Boston	1986-01-25	2010-06-02	24	d:\mariadb\sql\data\boys.txt
2	Henry	Boston	1987-06-07	2008-04-01	20	d:\mariadb\sql\data\boys.txt
6	Bill	Boston	1986-09-11	2008-02-10	21	d:\mariadb\sql\data\boys.txt

Existing special columns are listed in the following table:

Special Name	Type	Description of the column value

ROWID	Integer	The row ordinal number in the table. This is not quite equivalent to a virtual column with an auto increment of 1 because rows are renumbered when deleting rows.
ROWNUM	Integer	The row ordinal number in the file. This is different from ROWID for multiple tables, TBL/XCOL/OCCUR/PIVOT tables, XML tables with a multiple column, and for DBF tables where ROWNUM includes soft deleted rows.
FILEID FDISK FPATH FNAME FTYPE	String	FILEID returns the full name of the file this row belongs to. Useful in particular for multiple tables represented by several files. The other special columns can be used to retrieve only one part of the full name.
TABID	String	The name of the table this row belongs to. Useful for TBL tables.
PARTID	String	The name of the partition this row belongs to. Specific to partitioned tables.
SERVID	String	The name of the federated server or server host used by a MYSQL table. “ODBC” for an ODBC table, "JDBC" for a JDBC table and “Current” for all other tables.

Note: CONNECT does not currently support auto incremented columns. However, a `ROWID` special column will do the job of a column auto incremented by 1.

5.3.7.9.3 Using CONNECT - Importing File Data Into MariaDB Tables

Directly using external (file) data has many advantages, such as to work on “fresh” data produced for instance by cash registers, telephone switches, or scientific apparatus. However, you may want in some case to import external data into your MariaDB database. This is extremely simple and flexible using the CONNECT handler. For instance, let us suppose you want to import the data of the xsample.xml XML file previously given in example into a MyISAM table called *biblio* belonging to the connect database. All you have to do is to create it by:

```
create table biblio engine=myisam select * from xsampall2;
```

This last statement creates the MyISAM table and inserts the original XML data, translated to tabular format by the *xsampall2* CONNECT table, into the MariaDB *biblio* table. Note that further transformation on the data could have been achieved by using a more elaborate Select statement in the Create statement, for instance using filters, alias or applying functions to the data. However, because the Create Table process copies table data, later modifications of the *xsample.xml* file will not change the *biblio* table and changes to the *biblio* table will not modify the *xsample.xml* file.

All these can be combined or transformed by further SQL operations. This makes working with CONNECT much more flexible than just using the LOAD statement.

5.3.7.9.4 Using CONNECT - Exporting Data From MariaDB

Exporting data from MariaDB is obviously possible with CONNECT in particular for all formats not supported by the `SELECT INTO OUTFILE` statement. Let us consider the query:

```
select plugin_name handler, plugin_version version, plugin_author
author, plugin_description description, plugin_maturity maturity
from information_schema.plugins where plugin_type = 'STORAGE ENGINE';
```

Supposing you want to get the result of this query into a file handlers.htm in XML/HTML format, allowing displaying it on an Internet browser, this is how you can do it:

Just create the CONNECT table that will be used to make the file:

```
create table handout
engine=CONNECT table_type=XML file_name='handout.htm' header=yes
option_list='name=TABLE,coltype=HTML,attribute=border=1;cellpadding=5
     ,headattr=bgcolor=yellow'
select plugin_name handler, plugin_version version, plugin_author
author, plugin_description description, plugin_maturity maturity
from information_schema.plugins where plugin_type = 'STORAGE ENGINE';
```

Here the column definition is not given and will come from the Select statement following the Create. The CONNECT options are the same we have seen previously. This will do both actions, creating the matching *handlers* CONNECT table and 'filling'

it with the query result.

Note 1: This could not be done in only one statement if the table type had required using explicit CONNECT column options. In this case, firstly create the table, and then populate it with an Insert statement.

Note 2: The source “plugins” table column “description” is a long text column, data type not supported for CONNECT tables. It has been silently internally replaced by varchar(256).

5.3.7.9.5 Using CONNECT - Indexing

Contents

Indexing is one of the main ways to optimize queries. Key columns, in particular when they are used to join tables, should be indexed. But what should be done for columns that have only few distinct values? If they are randomly placed in the table they should not be indexed because reading many rows in random order can be slower than reading the entire table sequentially. However, if the values are sorted or clustered, indexing can be acceptable because CONNECT indexes store the values in the order they appear into the table and this will make retrieving them almost as fast as reading them sequentially.

CONNECT provides four indexing types:

1. Standard Indexing
2. Block Indexing
3. Remote Indexing
4. Dynamic Indexing

Standard Indexing

CONNECT standard indexes are created and used as the ones of other storage engines although they have a specific internal format. The CONNECT handler supports the use of standard indexes for most of the file based table types.

You can define them in the CREATE TABLE statement, or either using the CREATE INDEX statement or the ALTER TABLE statement. In all cases, the index files are automatically made. They can be dropped either using the DROP INDEX statement or the ALTER TABLE statement, and this erases the index files.

Indexes are automatically reconstructed when the table is created, modified by INSERT, UPDATE or DELETE commands, or when the SEPINDEX option is changed. If you have a lot of changes to do on a table at one moment, you can use table locking to prevent indexes to be reconstructed after each statement. The indexes will be reconstructed when unlocking the table. For instance:

```
lock table t1 write;
insert into t1 values(...);
insert into t1 values(...);
...
unlock tables;
```

If a table was modified by an external application that does not handle indexing, the indexes must be reconstructed to prevent returning false or incomplete results. To do this, use the OPTIMIZE TABLE command.

For outward tables, index files are not erased when dropping the table. This is the same as for the data file and preserves the possibility of several users using the same data file via different tables.

Unlike other storage engines, CONNECT constructs the indexes as files that are named by default from the data file name, not from the table name, and located in the data file directory. Depending on the SEPINDEX table option, indexes are saved in a unique file or in separate files (if SEPINDEX is true). For instance, if indexes are in separate files, the primary index of the table *dept.dat* of type DOS is a file named *dept_PRIMARY.dnx*. This makes possible to define several tables on the same data file, with eventual different options such as mapped or not mapped, and to share the index files as well.

If the index file should have a different name, for instance because several tables are created on the same data file with different indexes, specify the base index file name with the XFILE_NAME option.

Note1: Indexed columns must be declared NOT NULL; CONNECT doesn't support indexes containing null values.

Note 2: MRR is used by standard indexing if it is enabled.

Note 3: Prefix indexing is not supported. If specified, the CONNECT engine ignores the prefix and builds a whole index.

Handling index errors

The way CONNECT handles indexing is very specific. All table modifications are done regardless of indexing. Only after a table has been modified, or when an `OPTIMIZE TABLE` command is sent are the indexes made. If an error occurs, the corresponding index is not made. However, CONNECT being a non-transactional engine, it is unable to roll back the changes made to the table. The main causes of indexing errors are:

- Trying to index a nullable column. In this case, you can alter the table to declare the column as not nullable or, if the column is nullable indeed, make it not indexed.
- Entering duplicate values in a column indexed by a unique index. In this case, if the index was wrongly declared as unique, alter is declaration to reflect this. If the column should really contain unique values, you must manually remove or update the duplicate values.

In both cases, after correcting the error, remake the indexes with the OPTIMIZE TABLE command.

Index file mapping

To accelerate the indexing process, CONNECT makes an index structure in memory from the index file. This can be done by reading the index file or using it as if it was in memory by “file mapping”. On enabled versions, file mapping is used according to the boolean connect_indx_map system variable. Set it to 0 (file read) or 1 (file mapping).

Block Indexing

To accelerate input/output, CONNECT uses when possible a read/write mode by blocks of n rows, n being the value given in the BLOCK _ SIZE option of the Create Table, or a default value depending on the table type. This is automatic for fixed files (FIX, BIN, DBF or VEC), but must be specified for variable files (DOS , CSV or FMT).

For blocked tables, further optimization can be achieved if the data values for some columns are “clustered” meaning that they are not evenly scattered in the table but grouped in some consecutive rows. Block indexing permits to skip blocks in which no rows fulfill a conditional predicate without having even to read the block. This is true in particular for sorted columns.

You indicate this when creating the table by using the DISTRIB =d column option. The enum value d can be *scattered*, *clustered*, or *sorted*. In general only one column can be sorted. Block indexing is used only for clustered and sorted columns.

Difference between standard indexing and block indexing

- Block indexing is internally handled by CONNECT while reading sequentially a table data. This means in particular that when standard indexing is used on a table, block indexing is not used.
- In a query, only one standard index can be used. However, block indexing can combine the restrictions coming from a where clause implying several clustered/sorted columns.
- The block index files are faster to make and much smaller than standard index files.

Notes for this Release:

- On all operations that create or modify a table, CONNECT automatically calculates or recalculates and saves the mini/maxi or bitmap values for each block, enabling it to skip block containing no acceptable values. In the case where the optimize file does not correspond anymore to the table, because it has been accidentally destroyed, or because some column definitions have been altered, you can use the OPTIMIZE TABLE command to reconstruct the optimization file.
- Sorted column special processing is currently restricted to ascending sort. Column sorted in descending order must be flagged as clustered. Improper sorting is not checked in Update or Insert operations but is flagged when optimizing the table.
- Block indexing can be done in two ways. Keeping the min/max values existing for each block, or keeping a bitmap allowing knowing what column distinct values are met in each blocks. This second ways often gives a better optimization, except for sorted columns for which both are equivalent. The bitmap approach can be done only on columns having not too many distinct values. This is estimated by the MAX _ DIST option value associated to the column when creating the table. ~~Bitmap block indexing will be used if this number is not greater than the MAXBMP setting for the database~~.
- CONNECT cannot perform block indexing on case insensitive character columns. To force block indexing on a

character column, specify its charset as not case insensitive, for instance as binary. However this will also apply to all other clauses, this column being now case sensitive.

Remote Indexing

Remote indexing is specific to the MYSQL table type. It is equivalent to what the FEDERATED storage does. A MYSQL table does not support indexes per se. Because access to the table is handled remotely, it is the remote table that supports the indexes. What the MYSQL table does is just to add a WHERE clause to the SELECT command sent to the remote server allowing the remote server to use indexing when applicable. Note however that because CONNECT adds when possible all or part of the where clause of the original query, this happens often even if the remote indexed column is not declared locally indexed. The only, but very important, case a column should be locally declared indexed is when it is used to join tables. Otherwise, the required where clause would not be added to the sent SELECT query.

See Indexing of MYSQL tables for more.

Dynamic Indexing

An indexed created as “dynamic” is a standard index which, in some cases, can be reconstructed for a specific query. This happens in particular for some queries where two tables are joined by an indexed key column. If the “from” table is big and the “to” big table reduced in size because of a where clause, it can be worthwhile to reconstruct the index on this reduced table.

Because of the time added by reconstructing the index, this will be valuable only if the time gained by reducing the index size if more than this reconstruction time. This is why this should not be done if the “from” table is small because there will not be enough row joining to compensate for the additional time. Otherwise, the gain of using a dynamic index is:

- Indexing time is a little faster if the index is smaller.
- The join process will return only the rows fulfilling the where clause.
- Because the table is read sequentially when reconstructing the index there no need for MRR.
- Constructing the index can be faster if the table is reduced by block indexing.
- While constructing the index, CONNECT also stores in memory the values of other used columns.

This last point is particularly important. It means that after the index is reconstructed, the join is done on a temporary memory table.

Unfortunately, storage engines being called independently by MariaDB for each table, CONNECT has no global information to decide when it is good to use dynamic indexing. This is why you should use it only on cases where you see that some important join queries take a very long time and only on columns used for joining the table. How to declare an index to be dynamic is by using the Boolean DYNAM index option. For instance, the query:

```
select d.diag, count(*) cnt from diag d, patients p where d.pnb =
p.pnb and ageyears < 17 and county = 30 and drg <> 11 and d.diag
between 4296 and 9434 group by d.diag order by cnt desc;
```

Such a query joining the diag table to the patients table may last a very long time if the tables are big. To declare the primary key on the pnb column of the patients table to be dynamic:

```
alter table patients drop primary key;
alter table patients add primary key (pnb) comment 'DYNAMIC' dynam=1;
```

Note 1: The comment is not mandatory here but useful to see that the index is dynamic if you use the SHOW INDEX command.

Note 2: There is currently no way to just change the DYNAM option without dropping and adding the index. This is unfortunate because it takes time.

Virtual Indexing

It applies only to the virtual tables of type VIR and must be made on a column specifying `SPECIAL=ROWID` or `SPECIAL=ROWNUM`.

5.3.7.9.6 Using CONNECT - Condition Pushdown

The ODBC, JDBC, MYSQL, TBL and WMI table types use engine condition pushdown in order to restrict the number of

rows returned by the RDBS source or the WMI component.

The CONDITION_PUSHDOWN argument used in old versions of CONNECT is no longer needed because CONNECT uses condition pushdown unconditionally.

5.3.7.9.7 USING CONNECT - Offline Documentation

Note: You can download a PDF version of the CONNECT documentation (1.7.0003).

5.3.7.9.8 Using CONNECT - Partitioning and Sharding

Contents

CONNECT supports the MySQL/MariaDB partition specification. It is done similar to the way MyISAM or InnoDB do by using the PARTITION engine that must be enabled for this to work. This type of partitioning is sometimes referred as “horizontal partitioning”.

Partitioning enables you to distribute portions of individual tables across a file system according to rules which you can set largely as needed. In effect, different portions of a table are stored as separate tables in different locations. The user-selected rule by which the division of data is accomplished is known as a partitioning function, which in MariaDB can be the modulus, simple matching against a set of ranges or value lists, an internal hashing function, or a linear hashing function.

CONNECT takes this notion a step further, by providing two types of partitioning:

1. File partitioning. Each partition is stored in a separate file like in multiple tables.
2. Table partitioning. Each partition is stored in a separate table like in TBL tables.

Partition engine issues

Using partitions sometimes requires creating the tables in an unnatural way to avoid some error due to several partition engine bugs:

1. Engine specific column and index options are not recognized and cause a syntax error when the table is created. The workaround is to create the table in two steps, a CREATE TABLE statement followed by an ALTER TABLE statement.
2. The connection string, when specified for the table, is lost by the partition engine. The workaround is to specify the connection string in the *option_list*.
3. MySQL upstream bug #71095. In case of list columns partitioning it sometimes causes a false “impossible where” clause to be raised. This makes a wrong void result returned when it should not be void. There is no workaround but this bug should be hopefully fixed.

The following examples are using the above workaround syntax to address these issues.

File Partitioning

File partitioning applies to file-based CONNECT table types. As with multiple tables, physical data is stored in several files instead of just one. The differences to multiple tables are:

1. Data is distributed amongst the different files following the partition rule.
2. Unlike multiple tables, partitioned tables are not read only.
3. Unlike multiple tables, partitioned tables can be indexable.
4. The file names are generated from the partition names.
5. Query pruning is automatically made by the partition engine.

The table file names are generated differently depending on whether the table is an inward or outward table. For inward tables, for which the file name is not specified, the partition file names are:

```
Data file name: table_name#P#partition_name.table_file_type
Index file name: table_name#P#partition_name.index_file_type
```

For instance for the table:

```
CREATE TABLE t1 (
id INT KEY NOT NULL,
msg VARCHAR(32))
ENGINE=CONNECT TABLE_TYPE=FIX
partition by range(id) (
partition first values less than(10),
partition middle values less than(50),
partition last values less than(MAXVALUE));
```

CONNECT will generate in the current data directory the files:

```
| t1#P#first.fix
| t1#P#first.fnx
| t1#P#middle.fix
| t1#P#middle.fnx
| t1#P#last.fix
| t1#P#last.fnx
```

This is similar to what the partition engine does for other engines - CONNECT partitioned inward tables behave like other engines partition tables do. Just the data format is different.

Note: If sub-partitioning is used, inward table files and index files are named:

```
| table_name#P#partition_name#SP#subpartition_name.type
| table_name#P#partition_name#SP#subpartition_name.index_type
```

Outward Tables

The real problems occur with outward tables, in particular when they are created from already existing files. The first issue is to make the partition table use the correct existing file names. The second one, only for already existing not void tables, is to be sure the partitioning function match the distribution of the data already existing in the files.

The first issue is addressed by the way data file names are constructed. For instance let us suppose we want to make a table from the fixed formatted files:

```
E:\Data\part1.txt
E:\Data\part2.txt
E:\Data\part3.txt
```

This can be done by creating a table such as:

```
create table t2 (
id int not null,
msg varchar(32),
index XID(id))
engine=connect table_type=FIX file_name='E:/Data/part%s.txt'
partition by range(id) (
partition `1` values less than(10),
partition `2` values less than(50),
partition `3` values less than(MAXVALUE));
```

The rule is that for each partition the matching file name is internally generated by replacing in the given FILE _ NAME option value the “%s” part by the partition name.

If the table was initially void, further inserts will populate it according to the partition function. However, if the files did exist

and contained data, this is your responsibility to determine what partition function actually matches the data distribution in them. This means in particular that partitioning by key or by hash cannot be used (except in exceptional cases) because you have almost no control over what the used algorithm does.

In the example above, there is no problem if the table is initially void, but if it is not, serious problems can be met if the initial distribution does not match the table distribution. Supposing a row in which “id” as the value 12 was initially contained in the part1.txt file, it will be seen when selecting the whole table but if you ask:

```
select * from t2 where id = 12;
```

The result will have 0 rows. This is because according to the partition function query pruning will only look inside the second partition and will miss the row that is in the wrong partition.

One way to check for wrong distribution if for instance to compare the results from queries such as:

```
SELECT partition_name, table_rows FROM
information_schema.partitions WHERE table_name = 't2';
```

And

```
SELECT CASE WHEN id < 10 THEN 1 WHEN id < 50 THEN 2 ELSE 3 END
AS pn, COUNT(*) FROM part3 GROUP BY pn;
```

If they match, the distribution can be correct although this does not prove it. However, if they do not match, the distribution is surely wrong.

Partitioning on a Special Column

There are some cases where the files of a multiple table do not contain columns that can be used for range or list partitioning. For instance, let’s suppose we have a multiple table based on the following files:

```
tmp/boston.txt
tmp/chicago.txt
tmp/atlanta.txt
```

Each of them containing the same kind of data:

```
ID: int
First_name: varchar(16)
Last_name: varchar(30)
Birth: date
Hired: date
Job: char(10)
Salary: double(8,2)
```

A multiple table can be created on them, for instance by:

```
create table mulemp (
id int NOT NULL,
first_name varchar(16) NOT NULL,
last_name varchar(30) NOT NULL,
birth date NOT NULL date_format='DD/MM/YYYY',
hired date NOT NULL date_format='DD/MM/YYYY',
job char(10) NOT NULL,
salary double(8,2) NOT NULL
) engine=CONNECT table_type=FIX file_name='tmp/*.txt' multiple=1;
```

The issue is that if we want to create a partitioned table on these files, there are no columns to use for defining a partition function. Each city file can have the same kind of column values and there is no way to distinguish them.

However, there is a solution. It is to add to the table a special column that will be used by the partition function. For instance, the new table creation can be done by:

```
create table partemp (
id int NOT NULL,
first_name varchar(16) NOT NULL,
last_name varchar(30) NOT NULL,
birth date NOT NULL date_format='DD/MM/YYYY',
hired date NOT NULL date_format='DD/MM/YYYY',
job char(16) NOT NULL,
salary double(10,2) NOT NULL,
city char(12) default 'boston' special=PARTID,
index XID(id)
) engine=CONNECT table_type=FIX file_name='E:/Data/Test/%s.txt';
alter table partemp
partition by list columns(city) (
partition `atlanta` values in('atlanta'),
partition `boston` values in('boston'),
partition `chicago` values in('chicago'));
```

Note 1: we had to do it in two steps because of the column CONNECT options.

Note 2: the special column PARTID returns the name of the partition in which the row is located.

Note 3: here we could have used the FNAME special column instead because the file name is specified as being the partition name.

This may seem rather stupid because it means for instance that a row will be in partition boston if it belongs to the partition boston! However, it works because the partition engine doesn't know about special columns and behaves as if the city column was a real column.

What happens if we populate it by?

```
insert into partemp(id,first_name,last_name,birth,hired,job,salary) values
(1205,'Harry','Cover','1982-10-07','2010-09-21','MANAGEMENT',125000.00);
insert into partemp values
(1524,'Jim','Beams','1985-06-18','2012-07-25','SALES',52000.00,'chicago'),
(1431,'Johnny','Walker','1988-03-12','2012-08-09','RESEARCH',46521.87,'boston'),
(1864,'Jack','Daniels','1991-12-01','2013-02-16','DEVELOPMENT',63540.50,'atlanta');
```

The value given for the city column (explicitly or by default) will be used by the partition engine to decide in which partition to insert the rows. It will be ignored by CONNECT (a special column cannot be given a value) but later will return the matching value. For instance:

```
select city, first_name, job from partemp where id in (1524,1431);
```

This query returns:

city	first_name	job
boston	Johnny	RESEARCH
chicago	Jim	SALES

Everything works as if the city column was a real column contained in the table data files.

Partitioning of zipped tables

Two cases are currently supported:

If a table is based on several zipped files, portioning is done the standard way as above. This is the *file_name* option specifying the name of the zip files that shall contain the '%s' part used to generate the file names.

If a table is based on only one zip file containing several entries, this will be indicated by placing the '%s' part in the entry option value.

Note: If a table is based on several zipped files each containing several entries, only the first case is possible. Using sub-partitioning to make partitions on each entries is not supported yet.

Table Partitioning

With table partitioning, each partition is physically represented by a sub-table. Compared to standard partitioning, this brings the following features:

1. The partitions can be tables driven by different engines. This relieves the current existing limitation of the partition engine.

2. The partitions can be tables driven by engines not currently supporting partitioning.
3. Partition tables can be located on remote servers, enabling table sharding.
4. Like for TBL tables, the columns of the partition table do not necessarily match the columns of the sub-tables.

The way it is done is to create the partition table with a table type referring to other tables, PROXY, MYSQL ODBC or JDBC. Let us see how this is done on a simple example. Supposing we have created the following tables:

```
create table xt1 (
id int not null,
msg varchar(32))
engine=myisam;

create table xt2 (
id int not null,
msg varchar(32)); /* engine=innoDB */

create table xt3 (
id int not null,
msg varchar(32))
engine=connect table_type=CSV;
```

We can for instance create a partition table using these tables as physical partitions by:

```
create table t3 (
id int not null,
msg varchar(32))
engine=connect table_type=PROXY tabname='xt%s'
partition by range columns(id) (
partition `1` values less than(10),
partition `2` values less than(50),
partition `3` values less than(MAXVALUE));
```

Here the name of each partition sub-table will be made by replacing the '%s' part of the tabname option value by the partition name. Now if we do:

```
insert into t3 values
(4, 'four'),(7,'seven'),(10,'ten'),(40,'forty'),
(60,'sixty'),(81,'eighty one'),(72,'seventy two'),
(11,'eleven'),(1,'one'),(35,'thirty five'),(8,'eight');
```

The rows will be distributed in the different sub-tables according to the partition function. This can be seen by executing the query:

```
select partition_name, table_rows from
information_schema.partitions where table_name = 't3';
```

This query replies:

partition_name	table_rows
1	4
2	4
3	3

Query pruning is of course automatic, for instance:

```
explain partitions select * from t3 where id = 81;
```

This query replies:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	part5	3	ALL	<null>	<null>	<null>	<null>	22	Using where

When executing this select query, only sub-table xt3 will be used.

Indexing with Table Partitioning

Using the PROXY table type seems natural. However, in this current version, the issue is that PROXY (and ODBC) tables are not indexable. This is why, if you want the table to be indexed, you must use the MYSQL table type. The CREATE TABLE statement will be almost the same:

```
create table t4 (
id int key not null,
msg varchar(32))
engine=connect table_type=MYSQL tabname='xt%s'
partition by range columns(id) (
partition `1` values less than(10),
partition `2` values less than(50),
partition `3` values less than(MAXVALUE));
```

The column *id* is declared as a key, and the table type is now MYSQL. This makes Sub-tables accessed by calling a MariaDB server as MYSQL tables do. Note that this modifies only the way CONNECT sub-tables are accessed.

However, indexing just make the partitioned table use “remote indexing” the way FEDERATED tables do. This means that when sending the query to retrieve the table data, a where clause will be added to the query. For instance, let’s suppose you ask:

```
select * from t4 where id = 7;
```

The query sent to the server will be:

```
SELECT `id`, `msg` FROM `xt1` WHERE `id` = 7
```

On a query like this one, it does not change much because the where clause could have been added anyway by the cond_push function, but it does make a difference in case of joins. The main thing to understand is that real indexing is done by the called table and therefore that it should be indexed.

This also means that the xt1, xt2, and xt3 table indexes should be made separately because creating the t2 table as indexed does not make the indexes on the sub-tables.

Sharding with Table Partitioning

Using table partitioning can have one more advantage. Because the sub-tables can address a table located on another server, it is possible to shard a table on separate servers and hardware machines. This may be required to access as one table data already located on several remote machines, such as servers of a company branches. Or it can be just used to split a huge table for performance reason. For instance, supposing we have created the following tables:

```
create table rt1 (id int key not null, msg varchar(32))
engine=federated connection='mysql://root@host1/test/sales';

create table rt2 (id int key not null, msg varchar(32))
engine=federated connection='mysql://root@host2/test/sales';

create table rt3 (id int key not null, msg varchar(32))
engine=federated connection='mysql://root@host3/test/sales';
```

Creating the partition table accessing all these will be almost like what we did with the t4 table:

```
create table t5 (
id int key not null,
msg varchar(32))
engine=connect table_type=MYSQL tabname='rt%s'
partition by range columns(id) (
partition `1` values less than(10),
partition `2` values less than(50),
partition `3` values less than(MAXVALUE));
```

.

The only difference is the tabname option now referring to the rt1, rt2, and rt3 tables. However, even if it works, this is not the best way to do it. This is because accessing a table via the MySQL API is done twice per table. Once by CONNECT to access the FEDERATED table on the local server, then a second time by FEDERATED engine to access the remote table.

The CONNECT MYSQL table type being used anyway, you’d rather use it to directly access the remote tables. Indeed, the partition names can also be used to modify the connection URL’s. For instance, in the case shown above, the partition table can be created as:

```
create table t6 (
id int key not null,
msg varchar(32))
engine=connect table_type=MYSQL
option_list='connect=mysql://root@host%s/test/sales'
partition by range columns(id) (
partition `1` values less than(10),
partition `2` values less than(50),
partition `3` values less than(MAXVALUE));
```

Several things can be noted here:

1. As we have seen before, the partition engine currently loses the connection string. This is why it was specified as “connect” in the option list.
2. For each partition sub-tables, the “%s” part of the connection string has been replaced by the partition name.
3. It is not needed anymore to define the rt1, rt2, and rt3 tables (even it does not harm) and the FEDERATED engine is no more used to access the remote tables.

This is a simple case where the connection string is almost the same for all the sub-tables. But what if the sub-tables are accessed by very different connection strings? For instance:

```
For rt1: connection='mysql://root:tinono@127.0.0.1:3307/test/xt1'
For rt2: connection='mysql://foo:foopass@denver/dbemp/xt2'
For rt3: connection='mysql://root@huston :5505/test/tabx'
```

There are two solutions. The first one is to use the parts of the connection string to differentiate as partition names:

```
create table t7 (
id int key not null,
msg varchar(32))
engine=connect table_type=MYSQL
option_list='connect=mysql://%s'
partition by range columns(id) (
partition `root:tinono@127.0.0.1:3307/test/xt1` values less than(10),
partition `foo:foopass@denver/dbemp/xt2` values less than(50),
partition `root@huston :5505/test/tabx` values less than(MAXVALUE));
```

The second one, allowing avoiding too complicated partition names, is to create federated servers to access the remote tables (if they do not already exist, else just use them). For instance the first one could be:

```
create server `server_one` foreign data wrapper 'mysql'
options
(host '127.0.0.1',
database 'test',
user 'root',
password 'tinono',
port 3307);
```

Similarly, “server_two” and “server_three” would be created and the final partition table would be created as:

```
create table t8 (
id int key not null,
msg varchar(32))
engine=connect table_type=MYSQL
option_list='connect=server_%s'
partition by range columns(id) (
partition `one/xt1` values less than(10),
partition `two/xt2` values less than(50),
partition `three/tabx` values less than(MAXVALUE));
```

It would be even simpler if all remote tables had the same name on the remote databases, for instance if they all were named xt1, the connection string could be set as “server_%s/xt1” and the partition names would be just “one”, “two”, and “three”.

Sharding on a Special Column

The technique we have seen above with file partitioning is also available with table partitioning. Companies willing to use as one table data sharded on the company branch servers can, as we have seen, add to the table create definition a special column. For instance:

```
create table t9 (
id int not null,
msg varchar(32),
branch char(16) default 'main' special=PARTID,
index XID(id))
engine=connect table_type=MYSQL
option_list='connect=server_%s/sales'
partition by range columns(id) (
partition `main` values in('main'),
partition `east` values in('east'),
partition `west` values in('west'));
```

This example assumes that federated servers had been created named “server_main”, “server_east” and “server_west” and that all remote tables are named “sales”. Note also that in this example, the column id is no more a key.

Current Partition Limitations

Because the partition engine was written before some other engines were added to MariaDB, the way it works is sometime incompatible with these engines, in particular with CONNECT.

Update statement

With the sample tables above, you can do update statements such as:

```
update t2 set msg = 'quatre' where id = 4;
```

It works perfectly and is accepted by CONNECT. However, let us consider the statement:

```
update t2 set id = 41 where msg = 'four';
```

This statement is not accepted by CONNECT. The reason is that the column id being part of the partition function, changing its value may require the modified row to be moved to another partition. The way it is done by the partition engine is to delete the old row and to re-insert the new modified one. However, this is done in a way that is not currently compatible with CONNECT (remember that CONNECT supports UPDATE in a specific way, in particular for the table type MYSQL) This limitation could be temporary. Meanwhile the workaround is to manually do what is done above,

Deleting the row to modify and inserting the modified row:

```
delete from t2 where id = 4;
insert into t2 values(41, 'four');
```

Alter Table statement

For all CONNECT outward tables, the ALTER TABLE statement does not make any change in the table data. This is why ALTER TABLE should not be used; in particular to modify the partition definition, except of course to correct a wrong definition. Note that using ALTER TABLE to create a partition table in two steps because column options would be lost is valid as it applies to a table that is not yet partitioned.

As we have seen, it is also safe to use it to create or drop indexes. Otherwise, a simple rule of thumb is to avoid altering a table definition and better drop and re-create a table whose definition must be modified. Just remember that for outward CONNECT tables, dropping a table does not erase the data and that creating it does not modify existing data.

Rowid special column

Each partition being handled separately as one table, the ROWID special column returns the rank of the row in its partition, not in the whole table. This means that for partition tables ROWID and ROWNUM are equivalent.

5.3.7.10 CONNECT - Making the GetRest Library

To enable the REST feature with binary distributions of MariaDB, the function calling the cpprestsdk package is not included in CONNECT, thus allowing CONNECT normal operation when the cpprestsdk package is not installed. Therefore, it must be compiled separately as a library (so or dll) that will be loaded by CONNECT when needed.

This library will contain only one file shown here:

```
/************* Restget C++ Program Source Code File (.CPP) *************/
/* Adapted from the sample program of the Casablanca tutorial.         */
/* Copyright Olivier Bertrand 2019.                                    */
/***********************************************************************/
#include <cpprest/filestream.h>
#include <cpprest/http_client.h>

using namespace utility::conversions; // String conversions utilities
using namespace web;                  // Common features like URIs.
using namespace web::http;            // Common HTTP functionality
using namespace web::http::client;    // HTTP client features
using namespace concurrency::streams; // Asynchronous streams

typedef const char* PCSZ;

extern "C" int restGetFile(char* m, bool xt, PCSZ http, PCSZ uri, PCSZ fn);

/***********************************************************************/
/*  Make a local copy of the requested file.                           */
/***********************************************************************/
int restGetFile(char *m, bool xt, PCSZ http, PCSZ uri, PCSZ fn)
{
  int  rc = 0;
  auto fileStream = std::make_shared<ostream>();

  if (!http || !fn) {
  strcpy(m, "Missing http or filename");
  return 2;
  } // endif

 if (xt)
  fprintf(stderr, "restGetFile: fn=%s\n", fn);

  // Open stream to output file.
  pplx::task<void> requestTask = fstream::open_ostream(to_string_t(fn))
    .then([=](ostream outFile) {
      *fileStream= outFile;

   if (xt)
    fprintf(stderr, "Outfile isopen=%d\n", outFile.is_open());

      // Create http_client to send the request.
      http_client client(to_string_t(http));

      if (uri) {
        // Build request URI and start the request.
        uri_builder builder(to_string_t(uri));
        return client.request(methods::GET, builder.to_string());
      } else
        return client.request(methods::GET);
    })

    // Handle response headers arriving.
    .then([=](http_response response) {
   if (xt)
    fprintf(stderr, "Received response status code:%u\n",
                                response.status_code());

      // Write response body into the file.
      return response.body().read_to_end(fileStream->streambuf());
    })

    // Close the file stream.
    .then([=](size_t n) {
   if (xt)
    fprintf(stderr, "Return size=%zu\n", n);

      return fileStream->close();
    });

  // Wait for all the outstanding I/O to complete and handle any exceptions
  try {
  if (xt)
```

```
   fprintf(stderr, "Waiting\n");

  requestTask.wait();
  } catch (const std::exception &e) {
  if (xt)
   fprintf(stderr, "Error exception: %s\n", e.what());

  sprintf(m, "Error exception: %s", e.what());
  rc= 1;
  } // end try/catch

 if (xt)
  fprintf(stderr, "restget done: rc=%d\n", rc);

  return rc;
} // end of restGetFile
```

This file exists in the source of CONNECT as `restget.cpp`. If you have no access to the source, use your favorite editor to make it by copy/pasting from the above.

Then, on Linux, compile the GetRest.so library:

```
g++ -o GetRest.so -O3 -Wall -std=c++11 -fPIC -shared restget.cpp -lcpprest
```

Note: You can replace `-O3` by `-g` to make a debug version.

This library should be placed where it can be accessed. A good place is the directory where the `libcpprest.so` is, for instance `/usr/lib64`. You can move or copy it there.

On windows, using Visual Studio, make an empty win32 dll project named GetRest and add it the above file. Also add it the module definition file `restget.def`:

```
LIBRARY REST
EXPORTS
   restGetFile     @1
```

Important: This file must be specified in the property linker input page.

Once compiled, the release or debug versions can be copied in the `cpprestsdk` corresponding directories, bin or debug\bin.

That is all. It is a once-off job. Once done, it will work with all new MariaDB versions featuring CONNECT version 1.07.

Note: the xt tracing variable is true when connect_xtrace setting includes the value “MONGO” (512).

> Caution: If your server crashes when using this feature, this is likely because the GetRest lib is linked to the wrong cpprestsdk lib (this may only apply on Windows)
>
> A Release version of GetRest must be linked to the release version of the cpprestsdk lib (cpprest_2_10.dll) but if you make a Debug version of GetRest, make sure it is linked to the Debug version of cpprestsdk lib (cpprest_2_10d.dll)
>
> This may be automatic if you use Visual Studio to make the GetRest.dll.

5.3.7.11 CONNECT - Adding the REST Feature as a Library Called by an OEM Table

If you are using a version of MariaDB that does not support REST, this is how the REST feature can be added as a library called by an OEM table.

Before making the REST OEM module, the Microsoft Casablanca package must be installed as for compiling MariaDB from source.

Even if this module is to be used with a binary distribution, you need some CONNECT source files in order to successfully make it. It is made with four files existing in the version 1.06.0010 of CONNECT: tabrest.cpp, restget.cpp, tabrest.h and mini-global.h. It also needs the CONNECT header files that are included in tabrest.cpp and the ones they can include. This can be obtained by going to a recent download site of a version of MariaDB that includes the REST feature, downloading the MariaDB source file tar.gz and extracting from it the CONNECT sources files in a directory that will be added to the additional source directories if it is not the directory containing the above files.

On Windows, use a recent version of Visual Studio. Make a new empty DLL project and add the source files tabrest.cpp and restget.cpp. Visual studio should automatically generate all necessary connections to the cpprest SDK. Just edit the properties of the project to add the additional include directory (the one where the CONNECT source was downloaded) et the link to the ha_connect.lib of the binary version of MariaDB (in the same directory than ha_connect.dll in your binary distribution). Add the preprocessor definition XML_SUPPORT. Also set in the linker input page of the project property the Module definition File to the rest.def file (with its full path) also existing in the CONNECT source files. If you are making a debug configuration, make sure that in the C/C++ Code generation page the Runtime library line specifies Multi-threaded Debug DLL (/MDd) or your server will crash when using the feature.

This is not really simple but it is nothing compared with Linux! Someone having made an OEM module for its own application have written:

For whatever reason, g++ / ld on Linux are both extremely picky about what they will and won't consider a *"library"* for linking purposes. In order to get them to recognize and therefore find `ha_connect.so` as a "valid" linkable library, `ha_connect.so` must exist in a directory whose path is in `/etc/ld.so.conf` or `/etc/ld.so.conf.d/ha_connect.conf` *AND* its filename must begin with "lib".

On Fedora, you can make a link to ha_connect.so by:

```
$ sudo ln -s /..path to../ha_connect.so /usr/lib64/libconnect.so
```

This provides a library whose name begins with “lib”. It was made in /usr/lib64/ because it was the directory of the libcpprest.so Casablanca library. This solved the need of a file in /etc/ld.so.conf.d as this was already done for the cpprest library. Note that the -s parameter is a must, without it all sort of nasty errors are met when using the feature.

Then compile and install the OEM module with:

```
$ makdir oem
$ cd oem
$ makedir Release
$ make -f oemrest.mak
$ sudo cp rest.so /usr/local/mysql/lib/plugin
```

The oemrest.mak file:

```
#LINUX
CPP = g++
LD = g++
OD = ./Release/
SD = /home/olivier/MariaDB/server/storage/connect/
CD =/usr/lib64
# flags to compile object files that can be used in a dynamic library
CFLAGS= -Wall -c -O3 -std=c++11 -fPIC -fno-rtti -I$(SD) -DXML_SUPPORT
# Replace -03 by -g for debug
LDFLAGS = -L$(CD) -lcpprest -lconnect

# Flags to create a dynamic library.
DYNLINKFLAGS = -shared
# on some platforms, use '-G' instead.

# REST library's archive file
OEMREST = rest.so

SRCS_CPP = $(SD)tabrest.cpp $(SD)restget.cpp
OBJS_CPP = $(OD)tabrest.o $(OD)restget.o

# top-level rule
all: $(OEMREST)

$(OEMREST): $(OBJS_CPP)
  $(LD) $(OBJS_CPP) $(LDFLAGS) $(DYNLINKFLAGS) -o $@

#CPP Source files
$(OD)tabrest.o:   $(SD)tabrest.cpp   $(SD)mini-global.h $(SD)global.h $(SD)plgdbsem.h
$(SD)xtable.h $(SD)filamtxt.h $(SD)plgxml.h $(SD)tabdos.h  $(SD)tabfmt.h $(SD)tabjson.h
$(SD)tabrest.h $(SD)tabxml.h
  $(CPP) $(CFLAGS) -o $@ $(SD)$(*F).cpp
$(OD)restget.o:   $(SD)restget.cpp   $(SD)mini-global.h $(SD)global.h
  $(CPP) $(CFLAGS) -o $@ $(SD)$(*F).cpp

# clean everything
clean:
  $(RM) $(OBJS_CPP) $(OEMREST)
```

The SD and CD variables are the directories of the CONNECT source files and the one containing the libcpprest.so lib. They can be edited to match those on your machine OD is the directory that was made to contain the object files.

A very important flag is -fno-rtti. Without it you would be in big trouble.

The resulting module, for instance rest.so or rest.dll, must be placed in the plugin directory of the MariaDB server. Then, you will be able to use NoSQL tables simply replacing in the CREATE TABLE statement the TABLE_TYPE option =JSON or XML by TABLE_TYPE=OEM SUBTYPE=REST MODULE='rest.(so|dll)'. Actually, the module name, here supposedly 'rest', can be anything you like.

The file type is JSON by default. If not, it must be specified like this:

```
OPTION_LIST='Ftype=XML'
```

To be added to the create table statement. For instance:

```
CREATE  TABLE webw
ENGINE=CONNECT TABLE_TYPE=OEM MODULE='Rest.dll' SUBTYPE=REST
FILE_NAME='weatherdata.xml'
HTTP='https://samples.openweathermap.org/data/2.5/forecast?
q=London,us&mode=xml&appid=b6907d289e10d714a6e88b30761fae22'
OPTION_LIST='Ftype=XML,Depth=3,Rownode=weatherdata';
```

Note: this last example returns an XML file whose format was not recognized by old CONNECT versions. It is here the reason of the option 'Rownode=weatherdata'.

If you have trouble making the module, you can post an issue on JIRA.

5.3.7.12 CONNECT - Compiling JSON UDFs in a Separate Library

Although the JSON UDFs can be nicely included in the CONNECT library module, there are cases when you may need to have them in a separate library.

This is when CONNECT is compiled embedded, or if you want to test or use these UDFs with other MariaDB versions not including them.

To make it, you need to have access to the most recent MariaDB source code. Then, make a project containing these files:

1. jsonudf.cpp
2. json.cpp
3. value.cpp
4. osutil.c
5. plugutil.cpp
6. maputil.cpp
7. jsonutil.cpp

`jsonutil.cpp` is not distributed with the source code, you will have to make it from the following:

```
#include "my_global.h"
#include "mysqld.h"
#include "plugin.h"
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <errno.h>

#include "global.h"

extern "C" int GetTraceValue(void) { return 0; }
uint GetJsonGrpSize(void) { return 100; }

/***********************************************************************/
/*  These replace missing function of the (not used) DTVAL class.      */
/***********************************************************************/
typedef struct _datpar   *PDTP;
PDTP MakeDateFormat(PGLOBAL, PSZ, bool, bool, int) { return NULL; }
int ExtractDate(char*, PDTP, int, int val[6]) { return 0; }


#ifdef __WIN__
my_bool CloseFileHandle(HANDLE h)
{
 return !CloseHandle(h);
} /* end of CloseFileHandle */

#else  /* UNIX */
my_bool CloseFileHandle(HANDLE h)
{
 return (close(h)) ? TRUE : FALSE;
}  /* end of CloseFileHandle */

int GetLastError()
{
 return errno;
}  /* end of GetLastError */

#endif  // UNIX

/***********************************************************************/
/*  Program for sub-allocating one item in a storage area.             */
/*  Note: This function is equivalent to PlugSubAlloc except that in   */
/*  case of insufficient memory, it returns NULL instead of doing a    */
/*  long jump. The caller must test the return value for error.        */
/***********************************************************************/
void *PlgDBSubAlloc(PGLOBAL g, void *memp, size_t size)
{
  PPOOLHEADER pph;                       // Points on area header.

  if (!memp)   //  Allocation is to be done in the Sarea
    memp = g->Sarea;

  size = ((size + 7) / 8) * 8;  /* Round up size to multiple of 8 */
  pph = (PPOOLHEADER)memp;

  if ((uint)size > pph->FreeBlk) { /* Not enough memory left in pool */
    sprintf(g->Message,
     "Not enough memory in Work area for request of %d (used=%d free=%d)",
   (int)size, pph->To_Free, pph->FreeBlk);
    return NULL;
  } // endif size

  // Do the suballocation the simplest way
  memp = MakePtr(memp, pph->To_Free);   // Points to sub_allocated block
  pph->To_Free += size;                 // New offset of pool free block
  pph->FreeBlk -= size;                 // New size   of pool free block

  return (memp);
} // end of PlgDBSubAlloc
```

You can create the file by copy/paste from the above.

Set all the additional include directories to the MariaDB include directories used in plugin compiling plus the reference of the storage/connect directories, and compile like any other UDF giving any name to the made library module (I used `jsonudf.dll` on Windows).

Then you can create the functions using this name as the soname parameter.

There are some restrictions when using the UDFs this way:

- The connect_json_grp_size variable cannot be accessed. The group size is set and retrieved using the jsonset_grp_size and jsonget_grp_size functions (previously 100).
- In case of error, warnings are replaced by messages sent to stderr.
- No trace.

5.3.7.13 CONNECT System Variables

Contents

This page documents system variables related to the CONNECT storage engine. See Server System Variables for a complete list of system variables and instructions on setting them.

See also the Full list of MariaDB options, system and status variables.

`connect_class_path`

- **Description:** Java class path
- **Commandline:** `--connect-class-path=value`
- **Scope:** Global
- **Dynamic:**
- **Data Type:** `string`
- **Default Value:**

`connect_cond_push`

- **Description:** Enable condition pushdown
- **Commandline:** `--connect-cond-push={0|1}`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `ON`

`connect_conv_size`

- **Description:** The size of the VARCHAR created when converting from a TEXT type. See connect_type_conv.
- **Commandline:** `--connect-conv-size=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`

- **Default Value:**
  - >= MariaDB 10.4.8: `1024`
  - <= MariaDB 10.4.7: `8192`
- **Range:** `0` to `65500`

---

connect_default_depth

- **Description:** Default depth used by Json, XML and Mongo discovery.
- **Commandline:** `--connect-default-depth=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `5`
- **Range:** `-1` to `16`
- **Introduced:** MariaDB 10.5.7, MariaDB 10.4.16

---

connect_default_prec

- **Description:** Default precision used for doubles.
- **Commandline:** `--connect-default-prec=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `6`
- **Range:** `0` to `16`
- **Introduced:** MariaDB 10.5.9, MariaDB 10.4.18

---

connect_enable_mongo

- **Description:** Enable the Mongo table type.
- **Commandline:** `--connect-enable-mongo={0|1}`
- **Scope:** Global, Session
- **Dynamic:**
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB 10.3.2, MariaDB 10.2.9
- **Removed:** MariaDB 10.3.3

---

connect_exact_info

- **Description:** Whether the CONNECT engine should return an exact record number value to information queries. It is OFF by default because this information can take a very long time for large variable record length tables or for remote tables, especially if the remote server is not available. It can be set to ON when exact values are desired, for instance when querying the repartition of rows in a partition table.
- **Commandline:** `--connect-exact-info={0|1}`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

connect_force_bson

- **Description:** Force using BSON for JSON tables. Starting with these releases, the internal way JSON was parsed and handled was changed. The main advantage of the new way is to reduce the memory required to parse JSON (from 6 to 10 times the size of the JSON source to now only 2 to 4 times). However, this is in Beta mode and JSON tables are still handled using the old mode. To use the new mode, tables should be created with TABLE_TYPE=BSON, or by setting this session variable to 1 or ON. Then, all JSON tables will be handled as BSON. This is temporary until the new way replaces the old way by default.
- **Commandline:** `--connect-force-bson={0|1}`
- **Scope:** Global, Session

- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB 10.5.9, MariaDB 10.4.18

---

connect_indx_map

- **Description:** Enable file mapping for index files. To accelerate the indexing process, CONNECT makes an index structure in memory from the index file. This can be done by reading the index file or using it as if it was in memory by "file mapping". Set to 0 (file read, the default) or 1 (file mapping).
- **Commandline:** `--connect-indx-map=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

connect_java_wrapper

- **Description:** Java wrapper.
- **Commandline:** `--connect-java-wrapper=val`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `wrappers/JdbcInterface`

---

connect_json_all_path

- **Description:** Discovery to generate json path for all columns if ON (the default) or do not when the path is the column name.
- **Commandline:** `--connect-json-all-path={0|1}`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Data Type:** `boolean`
- **Default Value:** `ON`
- **Introduced:** MariaDB 10.5.7, MariaDB 10.4.16

---

connect_json_grp_size

- **Description:** Max number of rows for JSON aggregate functions.
- **Commandline:** `--connect-json-grp-size=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `50` (>= Connect 1.7.0003), `10` (<= Connect 1.7.0002)
- **Range:** `1` to `2147483647`

---

connect_json_null

- **Description:** Representation of JSON null values.
- **Commandline:** `--connect-json-null=value`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `<null>`

---

connect_jvm_path

- **Description:** Path to JVM library.

- **Commandline:** `--connect-jvm_path=value`
- **Scope:** Global
- **Dynamic:**
- **Data Type:** `string`
- **Default Value:**

---

connect_type_conv

- **Description:** Determines the handling of TEXT columns.
  - `NO` : The default until Connect 1.06.005, no conversion takes place, and a TYPE_ERROR is returned, resulting in a “not supported” message.
  - `YES` : The default from Connect 1.06.006. The column is internally converted to a column declared as VARCHAR(n), `n` being the value of connect_conv_size.
  - `FORCE` (>= Connect 1.06.006): Also convert ODBC blob columns to TYPE_STRING.
  - `SKIP` : No conversion. When the column declaration is provided via Discovery (meaning the CONNECT table is created without a column description), this column is not generated. Also applies to ODBC tables.
- **Commandline:** `--connect-type-conv=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `enum`
- **Valid Values:** `NO` , `YES` , `FORCE` or `SKIP`
- **Default Value:** `YES`

---

connect_use_tempfile

- **Description:**
  - `NO` : The first algorithm is always used. Because it can cause errors when updating variable record length tables, this value should be set only for testing.
  - `AUTO` : This is the default value. It leaves CONNECT to choose the algorithm to use. Currently it is equivalent to `NO` , except when updating variable record length tables (DOS, CSV or FMT) with file mapping forced to OFF.
  - `YES` : Using a temporary file is chosen with some exceptions. These are when file mapping is ON, for VEC tables and when deleting from DBF tables (soft delete). For variable record length tables, file mapping is forced to OFF.
  - `FORCE` : Like YES but forces file mapping to be OFF for all table types.
  - `TEST` : Reserved for CONNECT development.
- **Commandline:** `--connect-use-tempfile=#`
- **Scope:** Session
- **Dynamic:** Yes
- **Data Type:** `enum`
- **Default Value:** `AUTO`

---

connect_work_size

- **Description:** Size of the CONNECT work area used for memory allocation. Permits allocating a larger memory sub-allocation space when dealing with very large if sub-allocation fails. If the specified value is too big and memory allocation fails, the size of the work area remains but the variable value is not modified and should be reset.
- **Commandline:** `--connect-work-size=#`
- **Scope:** Global, Session (Session-only from CONNECT 1.03.005)
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `67108864`
- **Range:** `4194304` upwards, depending on the physical memory size

---

connect_xtrace

- **Description:** Console trace value. Set to `0` (no trace), or to other values if a console tracing is desired. Note that to test this handler, MariaDB should be executed with the --console parameter because CONNECT prints some error and trace messages on the console. In some Linux versions, this is re-routed into the error log file. Console tracing can be set on the command line or later by names or values. Valid values (from Connect 1.06.006) include:
  - `0` : No trace

    - `YES` or `1` : Basic trace
    - `MORE` or `2` : More tracing
    - `INDEX` or `4` : Index construction
    - `MEMORY` or `8` : Allocating and freeing memory
    - `SUBALLOC` or `16` : Sub-allocating in work area
    - `QUERY` or `32` : Constructed query sent to external server
    - `STMT` or `64` : Currently executing statement
    - `HANDLER` or `128` : Creating and dropping CONNECT handlers
    - `BLOCK` or `256` : Creating and dropping CONNECT objects
    - `MONGO` or `512` : Mongo and REST (from Connect 1.06.0010) tracing
- For example:
    - `set global connect_xtrace=0;` *No trace*
    - `set global connect_xtrace='YES';` *By name*
    - `set global connect_xtrace=1;` *By value*
    - `set global connect_xtrace='QUERY,STMT';` *By name*
    - `set global connect_xtrace=96;` *By value*
    - `set global connect_xtrace=1023;` *Trace all*
- **Commandline:** `--connect-xtrace=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `set`
- **Default Value:** `0`
- **Valid Values:** See description

5.3.7.14 JSON Sample Files

Contents

Expense.json

```
[
  {
    "WHO": "Joe",
    "WEEK": [
      {
        "NUMBER": 3,
        "EXPENSE": [
          {
            "WHAT": "Beer",
            "AMOUNT": 18.00
          },
          {
            "WHAT": "Food",
            "AMOUNT": 12.00
          },
          {
            "WHAT": "Food",
            "AMOUNT": 19.00
          },
          {
            "WHAT": "Car",
            "AMOUNT": 20.00
          }
        ]
      },
      {
        "NUMBER": 4,
        "EXPENSE": [
          {
            "WHAT": "Beer",
```

```
            "AMOUNT": 19.00
          },
          {
            "WHAT": "Beer",
            "AMOUNT": 16.00
          },
          {
            "WHAT": "Food",
            "AMOUNT": 17.00
          },
          {
            "WHAT": "Food",
            "AMOUNT": 17.00
          },
          {
            "WHAT": "Beer",
            "AMOUNT": 14.00
          }
        ]
      },
      {
        "NUMBER": 5,
        "EXPENSE": [
          {
            "WHAT": "Beer",
            "AMOUNT": 14.00
          },
          {
            "WHAT": "Food",
            "AMOUNT": 12.00
          }
        ]
      }
    ]
  },
  {
    "WHO": "Beth",
    "WEEK": [
      {
        "NUMBER": 3,
        "EXPENSE": [
          {
            "WHAT": "Beer",
            "AMOUNT": 16.00
          }
        ]
      },
      {
        "NUMBER": 4,
        "EXPENSE": [
          {
            "WHAT": "Food",
            "AMOUNT": 17.00
          },
          {
            "WHAT": "Beer",
            "AMOUNT": 15.00
          }
        ]
      },
      {
        "NUMBER": 5,
        "EXPENSE": [
          {
            "WHAT": "Food",
            "AMOUNT": 12.00
          },
          {
            "WHAT": "Beer",
            "AMOUNT": 20.00
          }
        ]
      }
    ]
  },
```

```
  {
    "WHO": "Janet",
    "WEEK": [
      {
        "NUMBER": 3,
        "EXPENSE": [
          {
            "WHAT": "Car",
            "AMOUNT": 19.00
          },
          {
            "WHAT": "Food",
            "AMOUNT": 18.00
          },
          {
            "WHAT": "Beer",
            "AMOUNT": 18.00
          }
        ]
      },
      {
        "NUMBER": 4,
        "EXPENSE": [
          {
            "WHAT": "Car",
            "AMOUNT": 17.00
          }
        ]
      },
      {
        "NUMBER": 5,
        "EXPENSE": [
          {
            "WHAT": "Beer",
            "AMOUNT": 14.00
          },
          {
            "WHAT": "Car",
            "AMOUNT": 12.00
          },
          {
            "WHAT": "Beer",
            "AMOUNT": 19.00
          },
          {
            "WHAT": "Food",
            "AMOUNT": 12.00
          }
        ]
      }
    ]
  }
]
```

OEM example

This is an example showing how an OEM table can be implemented. It is out of the scope of this document to explain how it works and to be a full guide on writing OEM tables for CONNECT.

tabfic.h

The header File tabfic.h:

```
// TABFIC.H     Olivier Bertrand    2008-2010
// External table type to read FIC files

#define TYPE_AM_FIC  (AMT)129

typedef class FICDEF *PFICDEF;
typedef class TDBFIC *PTDBFIC;
typedef class FICCOL *PFICCOL;

/* -------------------------- FIC classes -------------------------- */
```

```
/*                          FIC classes                                 */

/***********************************************************************/
/*  FIC: OEM table to read FIC files.                                  */
/***********************************************************************/

/***********************************************************************/
/*  This function is exported from the Tabfic.dll     */
/***********************************************************************/
extern "C" PTABDEF __stdcall GetFIC(PGLOBAL g, void *memp);

/***********************************************************************/
/*  FIC table definition class.                                        */
/***********************************************************************/
class FICDEF : public DOSDEF {        /* Logical table description */
  friend class TDBFIC;
 public:
  // Constructor
  FICDEF(void) {Pseudo = 3;}

  // Implementation
  virtual const char *GetType(void) {return "FIC";}

  // Methods
  virtual BOOL DefineAM(PGLOBAL g, LPCSTR am, int poff);
  virtual PTDB GetTable(PGLOBAL g, MODE m);

 protected:
  // No Members
}; // end of class FICDEF

/***********************************************************************/
/*  This is the class declaration for the FIC table.                   */
/***********************************************************************/
class TDBFIC : public TDBFIX {
  friend class FICCOL;
 public:
  // Constructor
  TDBFIC(PFICDEF tdp);

  // Implementation
  virtual AMT   GetAmType(void) {return TYPE_AM_FIC;}

  // Methods
  virtual void  ResetDB(void);
  virtual int   RowNumber(PGLOBAL g, BOOL b = FALSE);

  // Database routines
  virtual PCOL  MakeCol(PGLOBAL g, PCOLDEF cdp, PCOL cprec, int n);
  virtual BOOL  OpenDB(PGLOBAL g, PSQL sqlp);
  virtual int   ReadDB(PGLOBAL g);
  virtual int   WriteDB(PGLOBAL g);
  virtual int   DeleteDB(PGLOBAL g, int irc);

 protected:
  // Members
  int ReadMode;      // To read soft deleted lines
  int Rows;                          // Used for RowID
}; // end of class TDBFIC

/***********************************************************************/
/*  Class FICCOL: for Monetary columns.                                */
/***********************************************************************/
class FICCOL : public DOSCOL {
 public:
  // Constructors
  FICCOL(PGLOBAL g, PCOLDEF cdp, PTDB tdbp,
         PCOL cprec, int i, PSZ am = "FIC");

  // Implementation
  virtual int  GetAmType(void) {return TYPE_AM_FIC;}

  // Methods
  virtual void ReadColumn(PGLOBAL g);

 protected:
```

```
 protected:
  // Members
  char Fmt;       // The column format
}; // end of class FICCOL
```

tabfic.cpp

The source File tabfic.cpp:

```
/***********************************************************************/
/*  FIC: OEM table to read FIC files.                                  */
/***********************************************************************/
#if defined(WIN32)
#define WIN32_LEAN_AND_MEAN      // Exclude rarely-used stuff
#include <windows.h>
#endif   // WIN32
#include "global.h"
#include "plgdbsem.h"
#include "reldef.h"
#include "filamfix.h"
#include "tabfix.h"
#include "tabfic.h"

int TDB::Tnum;
int DTVAL::Shift;

/***********************************************************************/
/*  Initialize the CSORT static members.                               */
/***********************************************************************/
int    CSORT::Limit = 0;
double CSORT::Lg2 = log(2.0);
size_t CSORT::Cpn[1000] = {0};      /* Precalculated cmpnum values */

/* ------------- Implementation of the FIC subtype --------------- */

/***********************************************************************/
/*  This function is exported from the DLL.                            */
/***********************************************************************/
PTABDEF __stdcall GetFIC(PGLOBAL g, void *memp)
{
  return new(g, memp) FICDEF;
} // end of GetFIC

/* -------------- Implementation of the FIC classes -------------- */

/***********************************************************************/
/*  DefineAM: define specific AM block values from FIC file.           */
/***********************************************************************/
BOOL FICDEF::DefineAM(PGLOBAL g, LPCSTR am, int poff)
{
  ReadMode = GetIntCatInfo("Readmode", 0);

  // Indicate that we are a BIN format
  return DOSDEF::DefineAM(g, "BIN", poff);
} // end of DefineAM

/***********************************************************************/
/*  GetTable: makes a new TDB of the proper type.                      */
/***********************************************************************/
PTDB FICDEF::GetTable(PGLOBAL g, MODE m)
{
  return new(g) TDBFIC(this);
} // end of GetTable

/* ------------------------------------------------------------------ */

/***********************************************************************/
/*  Implementation of the TDBFIC class.                                */
/***********************************************************************/
TDBFIC::TDBFIC(PFICDEF tdp) : TDBFIX(tdp, NULL)
{
  ReadMode = tdp->ReadMode;
  Rows = 0;
} // end of TDBFIC constructor
```

```
/***********************************************************************/
/*  Allocate FIC column description block.                             */
/***********************************************************************/
PCOL TDBFIC::MakeCol(PGLOBAL g, PCOLDEF cdp, PCOL cprec, int n)
{
  PCOL colp;

  // BINCOL is alright except for the Monetary format
  if (cdp->GetFmt() && toupper(*cdp->GetFmt()) == 'M')
    colp = new(g) FICCOL(g, cdp, this, cprec, n);
  else
    colp = new(g) BINCOL(g, cdp, this, cprec, n);

  return colp;
} // end of MakeCol

/***********************************************************************/
/*  RowNumber: return the ordinal number of the current row.           */
/***********************************************************************/
int TDBFIC::RowNumber(PGLOBAL g, BOOL b)
{
  return (b) ? Txfp->GetRowID() : Rows;
} // end of RowNumber

/***********************************************************************/
/*  FIC Access Method reset table for re-opening.                      */
/***********************************************************************/
void TDBFIC::ResetDB(void)
{
  Rows = 0;
  TDBFIX::ResetDB();
} // end of ResetDB

/***********************************************************************/
/*  FIC Access Method opening routine.                                 */
/***********************************************************************/
BOOL TDBFIC::OpenDB(PGLOBAL g, PSQL sqlp)
{
  if (Use == USE_OPEN) {
    // Table already open, just replace it at its beginning.
    return TDBFIX::OpenDB(g);
    } // endif use

  if (Mode != MODE_READ) {
    // Currently FIC tables cannot be modified.
    strcpy(g->Message, "FIC tables are read only");
    return TRUE;
    } // endif Mode

  /*********************************************************************/
  /*  Physically open the FIC file.                                    */
  /*********************************************************************/
  if (TDBFIX::OpenDB(g))
    return TRUE;

  Use = USE_OPEN;
  return FALSE;
} // end of OpenDB

/***********************************************************************/
/*  ReadDB: Data Base read routine for FIC access method.              */
/***********************************************************************/
int TDBFIC::ReadDB(PGLOBAL g)
{
  int rc;

  /*********************************************************************/
  /*  Now start the reading process.                                   */
  /*********************************************************************/
  do {
    rc = TDBFIX::ReadDB(g);
    } while (rc == RC_OK && ((ReadMode == 0 && *To_Line == '*') ||
         (ReadMode == 2 && *To_Line != '*')));
```

```
  Rows++;
  return rc;
} // end of ReadDB

/*********************************************************************/
/*  WriteDB: Data Base write routine for FIC access methods.       */
/*********************************************************************/
int TDBFIC::WriteDB(PGLOBAL g)
{
  strcpy(g->Message, "FIC tables are read only");
  return RC_FX;
} // end of WriteDB

/*********************************************************************/
/*  Data Base delete line routine for FIC access methods.          */
/*********************************************************************/
int TDBFIC::DeleteDB(PGLOBAL g, int irc)
{
  strcpy(g->Message, "Delete not enabled for FIC tables");
  return RC_FX;
} // end of DeleteDB

// ---------------------- FICCOL functions --------------------------

/*********************************************************************/
/*  FICCOL public constructor.                                     */
/*********************************************************************/
FICCOL::FICCOL(PGLOBAL g, PCOLDEF cdp, PTDB tdbp, PCOL cprec, int i,
               PSZ am) : DOSCOL(g, cdp, tdbp, cprec, i, am)
{
  // Set additional FIC access method information for column.
  Fmt = toupper(*cdp->GetFmt());    // Column format
} // end of FICCOL constructor

/*********************************************************************/
/*  Handle the monetary value of this column. It is a big integer  */
/*  that represents the value multiplied by 1000.                  */
/*  In this function we translate it to a double float value.      */
/*********************************************************************/
void FICCOL::ReadColumn(PGLOBAL g)
{
  char   *p;
  int     rc;
  uint    n;
  double  fmon;
  PTDBFIC tdbp = (PTDBFIC)To_Tdb;

  /*******************************************************************/
  /*  If physical reading of the line was deferred, do it now.     */
  /*******************************************************************/
  if (!tdbp->IsRead())
    if ((rc = tdbp->ReadBuffer(g)) != RC_OK) {
      if (rc == RC_EF)
        sprintf(g->Message, MSG(INV_DEF_READ), rc);

      longjmp(g->jumper[g->jump_level], 11);
      } // endif

  p = tdbp->To_Line + Deplac;

  /*******************************************************************/
  /*  Set Value from the line field.                                 */
  /*******************************************************************/
  if (*(SHORT*)(p + 8) < 0) {
    n = ~*(SHORT*)(p + 8);
    fmon = (double)n;
    fmon *= 4294967296.0;
    n = ~*(int*)(p + 4);
    fmon += (double)n;
    fmon *= 4294967296.0;
    n = ~*(int*)p;
    fmon += (double)n;
    fmon++;
    fmon /= 1000000.0;
    fmon = -fmon;
```

```
  } else {
    fmon = ((double)*(USHORT*)(p + 8));
    fmon *= 4294967296.0;
    fmon += ((double)*(ULONG*)(p + 4));
    fmon *= 4294967296.0;
    fmon += ((double)*(ULONG*)p);
    fmon /= 1000000.0;
  } // enif neg

  Value->SetValue(fmon);
} // end of ReadColumn
```

tabfic.def

The file tabfic.def: (required only on Windows)

```
LIBRARY     TABFIC
DESCRIPTION 'FIC files'
EXPORTS
   GetFIC       @1
```

JSON UDFs in a separate library

Although the JSON UDF's can be nicely included in the CONNECT library module, there are cases when you may need to have them in a separate library.

This is when CONNECT is compiled embedded, or if you want to test or use these UDF's with other MariaDB versions not including them.

To make it, you need to have access to the last MariaDB source code. Then, make a project containing these files:

1. jsonudf.cpp
2. json.cpp
3. value.cpp
4. osutil.c
5. plugutil.c
6. maputil.cpp
7. jsonutil.cpp

jsonutil.cpp is not distributed with the source code, you will have to make it from the following:

```
#include "my_global.h"
#include "mysqld.h"
#include "plugin.h"
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <errno.h>

#include "global.h"

extern "C" int GetTraceValue(void) { return 0; }
uint GetJsonGrpSize(void) { return 100; }

/***********************************************************************/
/*  These replace missing function of the (not used) DTVAL class.      */
/***********************************************************************/
typedef struct _datpar   *PDTP;
PDTP MakeDateFormat(PGLOBAL, PSZ, bool, bool, int) { return NULL; }
int ExtractDate(char*, PDTP, int, int val[6]) { return 0; }


#ifdef __WIN__
my_bool CloseFileHandle(HANDLE h)
{
 return !CloseHandle(h);
} /* end of CloseFileHandle */

#else  /* UNIX */
my_bool CloseFileHandle(HANDLE h)
{
 return (close(h)) ? TRUE : FALSE;
}  /* end of CloseFileHandle */

int GetLastError()
{
 return errno;
}  /* end of GetLastError */

#endif  // UNIX

/***********************************************************************/
/*  Program for sub-allocating one item in a storage area.             */
/*  Note: This function is equivalent to PlugSubAlloc except that in   */
/*  case of insufficient memory, it returns NULL instead of doing a    */
/*  long jump. The caller must test the return value for error.        */
/***********************************************************************/
void *PlgDBSubAlloc(PGLOBAL g, void *memp, size_t size)
{
  PPOOLHEADER pph;                        // Points on area header.

  if (!memp)   //  Allocation is to be done in the Sarea
    memp = g->Sarea;

  size = ((size + 7) / 8) * 8;  /* Round up size to multiple of 8 */
  pph = (PPOOLHEADER)memp;

  if ((uint)size > pph->FreeBlk) { /* Not enough memory left in pool */
    sprintf(g->Message,
     "Not enough memory in Work area for request of %d (used=%d free=%d)",
   (int)size, pph->To_Free, pph->FreeBlk);
    return NULL;
  } // endif size

  // Do the suballocation the simplest way
  memp = MakePtr(memp, pph->To_Free);   // Points to sub_allocated block
  pph->To_Free += size;                 // New offset of pool free block
  pph->FreeBlk -= size;                 // New size   of pool free block

  return (memp);
} // end of PlgDBSubAlloc
```

You can create the file by copy/paste from the above.

Set all the additional include directories to the MariaDB include directories used in plugin compiling plus the reference of the storage/connect directories, and compile like any other UDF giving any name to the made library module (I used jsonudf.dll on Windows)

Then you can create the functions using this name as the soname parameter.

There are some restrictions when using the UDF's this way:

- The connect_json_grp_size variable cannot be accessed. The group size is set to 100.
- In case of error, warnings are replaced by messages sent to stderr.
- No trace.

5.3.8 CSV

The CSV storage engine

CSV Overview

Used to read and append to files stored in CSV (comma-separated-values) format.

Checking and Repairing CSV Tables

CSV tables support the CHECK TABLE and REPAIR TABLE statements.

5.3.8.1 CSV Overview

Contents

The CSV Storage Engine can read and append to files stored in CSV (comma-separated-values) format.

However, since MariaDB 10.0, a better storage engine is able to read and write such files: CONNECT.

The CSV storage engine and logging to tables

The CSV storage engine is the default storage engine when using logging of SQL queries to tables.

```
mysqld --log-output=table
```

CSV Storage Engine files

When you create a table using the CSV storage engine, three files are created:

- `<table_name>.frm`
- `<table_name>.CSV`
- `<table_name>.CSM`

The `.frm` file is the table format file.

The `.CSV` file is a plain text file. Data you enter into the table is stored as plain text in comma-separated-values format.

The `.CSM` file stores metadata about the table such as the state and the number of rows in the table.

Limitations

- CSV tables do not support indexing.
- CSV tables cannot be partitioned.
- Columns in CSV tables must be declared as NOT NULL.
- No transactions.
- The original CSV-format does not enable IETF-compatible parsing of embedded quote and comma characters. From MariaDB 10.1.8, it is possible to do so by setting the IETF_QUOTES option when creating a table.

Examples

Forgetting to add NOT NULL:

```
CREATE TABLE csv_test (x INT, y DATE, z CHAR(10)) ENGINE=CSV;
ERROR 1178 (42000): The storage engine for the table doesn't support nullable columns
```

Creating, inserting and selecting:

```
CREATE TABLE csv_test (
  x INT NOT NULL, y DATE NOT NULL, z CHAR(10) NOT NULL
  ) ENGINE=CSV;
```

```
INSERT INTO csv_test VALUES
    (1,CURDATE(),'one'),
    (2,CURDATE(),'two'),
    (3,CURDATE(),'three');
```

```
SELECT * FROM csv_test;
+---+------------+-------+
| x | y          | z     |
+---+------------+-------+
| 1 | 2011-11-16 | one   |
| 2 | 2011-11-16 | two   |
| 3 | 2011-11-16 | three |
+---+------------+-------+
```

Viewing in a text editor:

```
$ cat csv_test.CSV
1,"2011-11-16","one"
2,"2011-11-16","two"
3,"2011-11-16","three"
```

5.3.8.2 Checking and Repairing CSV Tables

CSV tables support the CHECK TABLE and REPAIR TABLE statements.

CHECK TABLE will mark the table as corrupt if it finds a problem, while REPAIR TABLE will restore rows until the first corrupted row, discarding the rest.

Examples

```
CREATE TABLE csv_test (
  x INT NOT NULL, y DATE NOT NULL, z CHAR(10) NOT NULL
  ) ENGINE=CSV;

INSERT INTO csv_test VALUES
    (1,CURDATE(),'one'),
    (2,CURDATE(),'two'),
    (3,CURDATE(),'three');
```

```
SELECT * FROM csv_test;
+---+------------+-------+
| x | y          | z     |
+---+------------+-------+
| 1 | 2013-07-08 | one   |
| 2 | 2013-07-08 | two   |
| 3 | 2013-07-08 | three |
+---+------------+-------+
```

Using an editor, the actual file will look as follows

```
$ cat csv_test.CSV
1,"2013-07-08","one"
2,"2013-07-08","two"
3,"2013-07-08","three"
```

Let's introduce some corruption with an unwanted quote in the 2nd row:

```
1,"2013-07-08","one"
2","2013-07-08","two"
3,"2013-07-08","three"
```

```
CHECK TABLE csv_test;
+---------------+-------+----------+----------+
| Table         | Op    | Msg_type | Msg_text |
+---------------+-------+----------+----------+
| test.csv_test | check | error    | Corrupt  |
+---------------+-------+----------+----------+
```

We can repair this, but all rows from the corrupt row onwards will be lost:

```
REPAIR TABLE csv_test;
+---------------+--------+----------+-----------------------------------------+
| Table         | Op     | Msg_type | Msg_text                                |
+---------------+--------+----------+-----------------------------------------+
| test.csv_test | repair | Warning  | Data truncated for column 'x' at row 2 |
| test.csv_test | repair | status   | OK                                      |
+---------------+--------+----------+-----------------------------------------+

SELECT * FROM csv_test;
+---+------------+-----+
| x | y          | z   |
+---+------------+-----+
| 1 | 2013-07-08 | one |
+---+------------+-----+
```

5.3.9 FederatedX

Information about the FederatedX Storage Engine

About FederatedX

Federated Storage Engine fork that uses uses libmysql to talk to the data source.

Differences Between FederatedX and Federated

Main differences between FederatedX and Federated.

5.3.9.1 About FederatedX

The FederatedX storage engine is a fork of MySQL's Federated storage engine, which is no longer being developed by Oracle. The original purpose of FederatedX was to keep this storage engine's development progressing-- to both add new features as well as fix old bugs.

Since MariaDB 10.0, the CONNECT storage engine also allows access to a remote database via MySQL or ODBC connection (table types: MYSQL, ODBC). However, in the current implementation there are several limitations.

Contents

What is the FederatedX storage engine?

The FederatedX Storage Engine is a storage engine that works with both MariaDB and MySQL. Where other storage engines are built as interfaces to lower-level file-based data stores, FederatedX uses libmysql to talk to the data source, the data source being a remote RDBMS. Currently, since FederatedX only uses libmysql, it can only talk to another MySQL RDBMS. The plan is of course to be able to use other RDBMS systems as a data source. There is an existing project Federated ODBC which was able to use PostgreSQL as a remote data source, and it is this type of functionality which will be brought to FederatedX in subsequent versions.

History

The history of FederatedX is derived from the History of Federated. Cisco needed a MySQL storage engine that would allow them to consolidate remote tables on some sort of routing device, being able to interact with these remote tables as if they were local to the device, but not actually on the device, since the routing device had only so much storage space. The first prototype of the Federated Storage Engine was developed by JD (need to check on this- Brian Aker can verify) using the HANDLER interface. Brian handed the code to Patrick Galbraith and explained how it needed to work, and with Brian and Monty's tutelage and Patrick had a working Federated Storage Engine with MySQL 5.0. Eventually, Federated was released to the public in a MySQL 5.0 release.

When MySQL 5.1 became the production release of MySQL, Federated had more features and enhancements added to it, namely:

- New Federated SERVER added to the parser. This was something Cisco needed that made it possible to change the connection parameters for numerous Federated tables at once without having to alter or re-create the Federated tables.
- Basic Transactional support-- for supporting remote transactional tables
- Various bugs that needed to be fixed from MySQL 5.0
- Plugin capability

In MariaDB 10.0.2 FederatedX got support for assisted table discovery.

Installing the Plugin

Although the plugin's shared library is distributed with MariaDB by default, the plugin is not actually installed by MariaDB by default. There are two methods that can be used to install the plugin with MariaDB.

The first method can be used to install the plugin without restarting the server. You can install the plugin dynamically by executing `INSTALL SONAME` or `INSTALL PLUGIN`. For example:

```
INSTALL SONAME 'ha_federatedx';
```

The second method can be used to tell the server to load the plugin when it starts up. The plugin can be installed this way by providing the --plugin-load or the --plugin-load-add options. This can be specified as a command-line argument to mariadbd or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
plugin_load_add = ha_federatedx
```

Uninstalling the Plugin

You can uninstall the plugin dynamically by executing `UNINSTALL SONAME` or `UNINSTALL PLUGIN`. For example:

```
UNINSTALL SONAME 'ha_federatedx';
```

If you installed the plugin by providing the --plugin-load or the --plugin-load-add options in a relevant server option group in an option file, then those options should be removed to prevent the plugin from being loaded the next time the server is restarted.

How FederatedX works

Every storage engine has to implement derived standard handler class API methods for a storage engine to work. FederatedX is no different in that regard. The big difference is that FederatedX needs to implement these handler methods in such as to construct SQL statements to run on the remote server and if there is a result set, process that result set into the internal handler format so that the result is returned to the user.

Internal workings of FederatedX

Normal database files are local and as such: You create a table called 'users', a file such as 'users.MYD' is created. A handler reads, inserts, deletes, updates data in this file. The data is stored in particular format, so to read, that data has to be parsed into fields, to write, fields have to be stored in this format to write to this data file.

With the FederatedX storage engine, there will be no local files for each table's data (such as .MYD). A foreign database will store the data that would normally be in this file. This will necessitate the use of MySQL client API to read, delete, update, insert this data. The data will have to be retrieve via an SQL call " `SELECT * FROM users` ". Then, to read this data, it will have to be retrieved via `mysql_fetch_row` one row at a time, then converted from the column in this select into the format that the handler expects.

The basic functionality of how FederatedX works is:

- The user issues an SQL statement against the local federatedX table. This statement is parsed into an item tree
- FederatedX uses the mysql handler API to implement the various methods required for a storage engine. It has access to the item tree for the SQL statement issued, as well as the Table object and each of its Field members. At
- With this information, FederatedX constructs an SQL statement
- The constructed SQL statement is sent to the Foreign data source through libmysql using the mysql client API
- The foreign database reads the SQL statement and sends the result back through the mysql client API to the origin
- If the original SQL statement has a result set from the foreign data source, the FederatedX storage engine iterates through the result set and converts each row and column to the internal handler format
- If the original SQL statement only returns the number of rows returned (affected_rows), that number is added to the table stats which results in the user seeing how many rows were affected.

FederatedX table creation

The create table will simply create the .frm file, and within the `CREATE TABLE` SQL statement, there SHALL be any of the following :

```
connection=scheme://username:password@hostname:port/database/tablename
connection=scheme://username@hostname/database/tablename
connection=scheme://username:password@hostname/database/tablename
connection=scheme://username:password@hostname/database/tablename
```

Or using the syntax introduced in MySQL versions 5.1 for a Federated server (SQL/MED Spec xxxx)

```
connection="connection_one"
connection="connection_one/table_foo"
```

An example of a connect string specifying all the connection parameters would be:

```
connection=mysql://username:password@hostname:port/database/tablename
```

Or, using a Federated server, first a server is created:

```
create server 'server_one' foreign data wrapper 'mysql' options
  (HOST '127.0.0.1',
  DATABASE 'db1',
  USER 'root',
  PASSWORD '',
  PORT 3306,
  SOCKET '',
  OWNER 'root');
```

Then the FederatedX table is created specifying the newly created Federated server:

```
CREATE TABLE federatedx.t1 (
  `id` int(20) NOT NULL,
  `name` varchar(64) NOT NULL default ''
  )
ENGINE="FEDERATED" DEFAULT CHARSET=latin1
CONNECTION='server_one';
```

(Note that in MariaDB, the original Federated storage engine is replaced with the new FederatedX storage engine. And for backward compatibility, the old name "FEDERATED" is used in create table. So in MariaDB, the engine type should be given as "FEDERATED" without an extra "X", not "FEDERATEDX").

The equivalent of above, if done specifying all the connection parameters

```
CONNECTION="mysql://root@127.0.0.1:3306/db1/t1"
```

You can also change the server to point to a new schema:

```
ALTER SERVER 'server_one' options(DATABASE 'db2');
```

All subsequent calls to any FederatedX table using the 'server_one' will now be against db2.t1! Guess what? You no longer have to perform an alter table in order to point one or more FederatedX tables to a new server!

This `connection="connection string"` is necessary for the handler to be able to connect to the foreign server, either by URL, or by server name.

Method calls

One way to see how the FederatedX storage engine works is to compile a debug build of MariaDB and turn on a trace log. Using a two column table, with one record, the following SQL statements shown below, can be analyzed for what internal methods they result in being called.

SELECT

If the query is for instance "`SELECT * FROM foo`", then the primary methods you would see with debug turned on would be first:

```
ha_federatedx::info
ha_federatedx::scan_time:
ha_federatedx::rnd_init: share->select_query SELECT * FROM foo
ha_federatedx::extra
```

Then for every row of data retrieved from the foreign database in the result set:

```
ha_federatedx::rnd_next
ha_federatedx::convert_row_to_internal_format
ha_federatedx::rnd_next
```

After all the rows of data that were retrieved, you would see:

```
ha_federatedx::rnd_end
ha_federatedx::extra
ha_federatedx::reset
```

INSERT

If the query was "
```
INSERT INTO foo (id, ts) VALUES (2, now());
```
", the trace would be:

```
ha_federatedx::write_row
ha_federatedx::reset
```

UPDATE

If the query was "
```
UPDATE foo SET ts = now() WHERE id = 1;
```
", the resultant trace would be:

```
ha_federatedx::index_init
ha_federatedx::index_read
ha_federatedx::index_read_idx
ha_federatedx::rnd_next
ha_federatedx::convert_row_to_internal_format
ha_federatedx::update_row

ha_federatedx::extra
ha_federatedx::extra
ha_federatedx::extra
ha_federatedx::external_lock
ha_federatedx::reset
```

FederatedX capabilities and limitations

- Tables MUST be created on the foreign server prior to any action on those tables via the handler, first version. IMPORTANT: IF you MUST use the FederatedX storage engine type on the REMOTE end, make sure that the table you connect to IS NOT a table pointing BACK to your ORIGINAL table! You know and have heard the screeching of audio feedback? You know putting two mirrors in front of each other how the reflection continues for eternity? Well, need I say more?!
- There is no way for the handler to know if the foreign database or table has changed. The reason for this is that this database has to work like a data file that would never be written to by anything other than the database. The integrity of the data in the local table could be breached if there was any change to the foreign database.
- Support for SELECT, INSERT, UPDATE, DELETE indexes.
- No ALTER TABLE, DROP TABLE or any other Data Definition Language calls.
- Prepared statements will not be used in the first implementation, it remains to to be seen whether the limited subset of the client API for the server supports this.
- This uses SELECT, INSERT, UPDATE, DELETE and not HANDLER for its implementation.
- This will not work with the query cache.
- FederatedX does not support GEOMETRY types. Such tables cannot be created explicitly, nor discovered.

How do you use FederatedX?

To use this handler, it's very simple. You must have two databases running, either both on the same host, or on different hosts.

First, on the foreign database you create a table, for example:

```
CREATE TABLE test_table (
  id     int(20) NOT NULL auto_increment,
  name   varchar(32) NOT NULL default '',
  other  int(20) NOT NULL default '0',
  PRIMARY KEY  (id),
  KEY name (name),
  KEY other_key (other))
DEFAULT CHARSET=latin1;
```

Then, on the server that will be connecting to the foreign host (client), you create a federated table without specifying the table structure:

```
CREATE TABLE test_table ENGINE=FEDERATED
  CONNECTION='mysql://root@127.0.0.1:9306/federatedx/test_federatedx';
```

Notice the "ENGINE" and "CONNECTION" fields? This is where you respectively set the engine type, "FEDERATED" and foreign host information, this being the database your 'client' database will connect to and use as the "data file". Obviously, the foreign database is running on port 9306, so you want to start up your other database so that it is indeed on port 9306, and your FederatedX database on a port other than that. In my setup, I use port 5554 for FederatedX, and port 5555 for the foreign database.

Alternatively (or if you're using MariaDB before version 10.0.2) you specify the federated table structure explicitly:

```
CREATE TABLE test_table (
  id     int(20) NOT NULL auto_increment,
  name   varchar(32) NOT NULL default '',
  other  int(20) NOT NULL default '0',
  PRIMARY KEY  (id),
  KEY name (name),
  KEY other_key (other))
ENGINE=FEDERATED
DEFAULT CHARSET=latin1
CONNECTION='mysql://root@127.0.0.1:9306/federatedx/test_federatedx';
```

In this case the table structure must match exactly the table on the foreign server.

How to see the storage engine in action

When developing this handler, I compiled the FederatedX database with debugging:

```
./configure --with-federatedx-storage-engine \
  --prefix=/home/mysql/mysql-build/federatedx/ --with-debug
```

Once compiled, I did a 'make install' (not for the purpose of installing the binary, but to install all the files the binary expects to see in the directory I specified in the build with

```
--prefix=/home/code-dev/maria
```

Then, I started the foreign server:

```
/usr/local/mysql/bin/mysqld_safe \
  --user=mysql --log=/tmp/mysqld.5555.log -P 5555
```

Then, I went back to the directory containing the newly compiled mysqld `<builddir>/sql/` , started up gdb:

```
gdb ./mysqld
```

Then, within the (gdb) prompt:

```
(gdb) run --gdb --port=5554 --socket=/tmp/mysqld.5554 --skip-innodb --debug
```

Next, I open several windows for each:

1. Tail the debug trace: tail -f /tmp/mysqld.trace|grep ha_fed
2. Tail the SQL calls to the foreign database: tail -f /tmp/mysqld.5555.log
3. A window with a client open to the federatedx server on port 5554
4. A window with a client open to the federatedx server on port 5555

I would create a table on the client to the foreign server on port 5555, and then to the FederatedX server on port 5554. At this point, I would run whatever queries I wanted to on the FederatedX server, just always remembering that whatever changes I wanted to make on the table, or if I created new tables, that I would have to do that on the foreign server.

Another thing to look for is 'show variables' to show you that you have support for FederatedX handler support:

```
show variables like '%federat%'
```

and:

```
show storage engines;
```

Both should display the federatedx storage handler.

How do I create a federated server?

A federated server is a way to have a foreign data source defined-- with all connection parameters-- so that you don't have to specify explicitly the connection parameters in a string.

For instance, if you wanted to connect to a table, t1, using this definition:

```
CREATE TABLE test_table ENGINE=FEDERATED
  CONNECTION='mysql://patg@192.168.1.123/first_db/t1';
```

You could instead create this with a server:

```
create server 'server_one' foreign data wrapper 'mysql' options
  (HOST '192.168.1.123',
  DATABASE 'first_db',
  USER 'patg',
  PASSWORD '',
  PORT 3306,
  SOCKET '',
  OWNER 'root');
```

You could now specify the server instead of the full URL in the connection string:

```
CREATE TABLE test_table ENGINE=FEDERATED
  CONNECTION='server_one/t1';
```

On the server where you create this `test_table` you will now have access to the tabel `t1` on the remote server found on 192.168.1.123.

How does FederatedX differ from the old Federated Engine?

FederatedX from a user point of view is the same for the most part. What is different with FederatedX and Federated is the following:

- Rewrite of the main Federated source code from one single ha_federated.cc file into three main abstracted components:
  - ha_federatedx.cc - Core implementation of FederatedX
  - federated_io.cc - Parent connection class to be over-ridden by derived classes for each RDBMS/client lib
  - federatated_io_<driver>.cc - derived federated_io class for a given RDBMS
  - federated_txn.cc - New support for using transactional engines on the foreign server using a connection poll
- Various bugs fixed (need to look at opened bugs for Federated)

Where can I get FederatedX

FederatedX is part of MariaDB 5.1 and later. MariaDB merged with the latest FederatedX when there is a need to get a bug fixed. You can get the latest code/follow/participate in the project from the FederatedX home page.

What are the plans for FederatedX?

- Support for other RDBMS vendors using ODBC
- Support for pushdown conditions
- Ability to limit result set sizes

5.3.9.2 Differences Between FederatedX and Federated

The main differences are:

New features in FederatedX

- Transactions (beta feature)

- Supports partitions (alpha feature)
- New class structure which allows developers to write connection classes for other RDBMSs without having to modify base classes for FederatedX

Different behavior

- FederatedX is statically compiled into MariaDB by default.
- When you create a table with FederatedX, the connection will be tested. The `CREATE` will fail if MariaDB can't connect to the remote host or if the remote table doesn't exist.

5.3.10 MEMORY Storage Engine

Contents

Contents of the MEMORY storage engine (previously known as HEAP) are stored in memory rather than on disk.

It is best-used for read-only caches of data from other tables, or for temporary work areas.

Since the data is stored in memory, it is highly vulnerable to power outages or hardware failure, and is unsuitable for permanent data storage. In fact, after a server restart, `MEMORY` tables will be recreated (because the definition file is stored on disk), but they will be empty. It is possible to re-populate them with a query using the `--init-file` server startup option.

Variable-length types like VARCHAR can be used in MEMORY tables. BLOB or TEXT columns are not supported for MEMORY tables.

Memory Usage

The maximum total size of MEMORY tables cannot exceed the max_heap_table_size system server variable. When a table is created this value applies to that table, and when the server is restarted this value applies to existing tables. Changing this value has no effect on existing tables. However, executing a `ALTER TABLE ... ENGINE=MEMORY` statement applies the current value of `max_heap_table_size` to the table. Also, it is possible to change the session value of `max_heap_table_size` before creating a table, to make sure that tables created by other sessions are not affected.

The `MAX_ROWS` table option provides a hint about the number of rows you plan to store in them. This is not a hard limit that cannot be exceeded, and does not allow to exceed `max_heap_table_size`. The storage engine uses max_heap_table_size and MAX_ROWS to calculate the maximum memory that could be allocated for the table.

Memory allocated to a MEMORY table is freed by running DROP TABLE or TRUNCATE TABLE, or rebuilding with ALTER TABLE tbl_name ENGINE = MEMORY. When rows are deleted, space is not automatically freed.

Index Type

The MEMORY storage engine permits indexes to be either B-tree or Hash. Hash is the default type for MEMORY. See Storage Engine index types for more on their characteristics.

A MEMORY table can have up to 64 indexes, 16 columns for each index and a maximum key length of 3072 bytes.

Replication

MEMORY tables are lost when a server restarts. In order to achieve this result in replication, the first time a primary uses a MEMORY table after a restart, it writes a DELETE statement for that table to the binary log, so that replicas are also emptied.

Example

The following example shows how to create a `MEMORY` table with a given maximum size, as described above.

```
SET max_heap_table_size = 1024*516;

CREATE TABLE t (a VARCHAR(10), b INT) ENGINE = MEMORY;

SET max_heap_table_size = @@max_heap_table_size;
```

5.3.11 MERGE

Contents

Description

The MERGE storage engine, also known as the MRG_MyISAM engine, is a collection of identical MyISAM tables that can be used as one. "Identical" means that all tables have identical column and index information. You cannot merge MyISAM tables in which the columns are listed in a different order, do not have exactly the same columns, or have the indexes in different order. However, any or all of the MyISAM tables can be compressed with myisampack. Columns names and indexes names can be different, as long as data types and NULL/NOT NULL clauses are the same. Differences in table options such as AVG_ROW_LENGTH, MAX_ROWS, or PACK_KEYS do not matter.

Each index in a MERGE table must match an index in underlying MyISAM tables, but the opposite is not true. Also, a MERGE table cannot have a PRIMARY KEY or UNIQUE indexes, because it cannot enforce uniqueness over all underlying tables.

The following options are meaningful for MERGE tables:

- `UNION` . This option specifies the list of the underlying MyISAM tables. The list is enclosed between parentheses and separated with commas.
- `INSERT_METHOD` . This options specifies whether, and how, INSERTs are allowed for the table. Allowed values are: `NO` (INSERTs are not allowed), `FIRST` (new rows will be written into the first table specified in the `UNION` list), `LAST` (new rows will be written into the last table specified in the `UNION` list). The default value is `NO` .

If you define a MERGE table with a definition which is different from the underlying MyISAM tables, or one of the underlying tables is not MyISAM, the CREATE TABLE statement will not return any error. But any statement which involves the table will produce an error like the following:

```
ERROR 1168 (HY000): Unable to open underlying table which is differently defined
  or of non-MyISAM type or doesn't exist
```

A `CHECK TABLE` will show more information about the problem.

The error is also produced if the table is properly define, but an underlying table's definition changes at some point in time.

If you try to insert a new row into a MERGE table with INSERT_METHOD=NO, you will get an error like the following:

```
ERROR 1036 (HY000): Table 'tbl_name' is read only
```

It is possible to build a MERGE table on MyISAM tables which have one or more virtual columns. MERGE itself does not support virtual columns, thus such columns will be seen as regular columns. The data types and sizes will still need to be identical, and they cannot be NOT NULL.

Examples

```
CREATE TABLE t1 (
    a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    message CHAR(20)) ENGINE=MyISAM;

CREATE TABLE t2 (
    a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    message CHAR(20)) ENGINE=MyISAM;


INSERT INTO t1 (message) VALUES ('Testing'),('table'),('t1');

INSERT INTO t2 (message) VALUES ('Testing'),('table'),('t2');

CREATE TABLE total (
    a INT NOT NULL AUTO_INCREMENT,
    message CHAR(20), INDEX(a))
    ENGINE=MERGE UNION=(t1,t2) INSERT_METHOD=LAST;

SELECT * FROM total;
+---+---------+
| a | message |
+---+---------+
| 1 | Testing |
| 2 | table   |
| 3 | t1      |
| 1 | Testing |
| 2 | table   |
| 3 | t2      |
+---+---------+
```

In the following example, we'll create three MyISAM tables, and then a MERGE table on them. However, one of them uses a different data type for the column b, so a SELECT will produce an error:

```
CREATE TABLE t1 (
  a INT,
  b INT
) ENGINE = MyISAM;

CREATE TABLE t2 (
  a INT,
  b INT
) ENGINE = MyISAM;

CREATE TABLE t3 (
  a INT,
  b TINYINT
) ENGINE = MyISAM;

CREATE TABLE t_mrg (
  a INT,
  b INT
) ENGINE = MERGE,UNION=(t1,t2,t3);

SELECT * FROM t_mrg;
ERROR 1168 (HY000): Unable to open underlying table which is differently defined
 or of non-MyISAM type or doesn't exist
```

To find out what's wrong, we'll use a CHECK TABLE:

```
CHECK TABLE t_mrg\G
*************************** 1. row ***************************
   Table: test.t_mrg
      Op: check
Msg_type: Error
Msg_text: Table 'test.t3' is differently defined or of non-MyISAM type or doesn't exist
*************************** 2. row ***************************
   Table: test.t_mrg
      Op: check
Msg_type: Error
Msg_text: Unable to open underlying table which is differently defined or of non-MyISAM type or
doesn't exist
*************************** 3. row ***************************
   Table: test.t_mrg
      Op: check
Msg_type: error
Msg_text: Corrupt
```

Now, we know that the problem is in `t3` 's definition.

5.3.12 Mroonga

Mroonga (formerly named Groonga Storage Engine) is a storage engine that provides fast CJK-ready full text searching using column store.

About Mroonga
Mroonga full text search storage engine.

Mroonga Overview
Basic Mroonga usage.

Mroonga Status Variables
Mroonga-related status variables.

Mroonga System Variables
Mroonga-related system variables.

Mroonga User-Defined Functions
Mroonga user-defined functions (UDFs).

Information Schema MROONGA_STATS Table
Mroonga activities statistics.

5.3.12.1 About Mroonga

Contents

Mroonga Version	Introduced	Maturity
7.07	MariaDB 10.2.11, MariaDB 10.1.29	Stable
5.04	MariaDB 10.1.6	Stable
5.02	MariaDB 10.0.18, MariaDB 10.1.5	Stable
5.0	MariaDB 10.0.17	Stable
4.06	MariaDB 10.0.15	Stable

Mroonga is a full text search storage engine based on Groonga, which is an open-source CJK-ready (Chinese, Japanese, and Korean) fulltext search engine using column base. See http://groonga.org for more.

With Mroonga, you can have a CJK-ready full text search feature, and it is faster than the MyISAM and InnoDB full text search for both updating and searching.

Mroonga also supports Groonga's fast geolocation search by using MariaDB's geolocation SQL syntax.

Mroonga currently only supports Linux x86_64 (Intel64/AMD64).

How to Install

Enable Mroonga with the following statement:

```
INSTALL SONAME 'ha_mroonga';
```

On Debian and Ubuntu mroonga engine will be installed with

```
sudo apt-get install mariadb-plugin-mroonga
```

See Plugin overview for details on installing and uninstalling plugins.

SHOW ENGINES can be used to check whether Mroonga is installed correctly:

```
SHOW ENGINES;
...
*************************** 8. row ***************************
      Engine: Mroonga
     Support: YES
     Comment: CJK-ready fulltext search, column store
Transactions: NO
          XA: NO
  Savepoints: NO
...
```

Once the plugin is installed, add a UDF (User-Defined Function) named "last_insert_grn_id", that returns the record ID assigned by groonga in INSERT, by the following SQL.

```
CREATE FUNCTION last_insert_grn_id RETURNS INTEGER SONAME 'ha_mroonga.so';
```

Limitations

- The maximum size of a single key is 4096 bytes.
- The maximum size of all keys is 4GB.
- The maximum number of records in a fulltext index is 268,435,455
- The maximum number of distinct terms in a fulltext index is 268,435,455
- The maximum size of a fulltext index is 256GB

Note that the maximum sizes are not hard limits, and may vary according to circumstance.

For more details, see http://mroonga.org/docs/reference/limitations.html.

Available Character Sets

Mroonga supports a limited number of character sets. These include:

- ASCII
- BINARY
- CP932
- EUCJPMS
- KOI8R
- LATIN1
- SJIS
- UJIS
- UTF8
- UTF8MB4

More Information

Further documentation for Mroonga can be found at http://mroonga.org/docs/

5.3.12.2 Mroonga Overview

Contents

Once Mroonga has been installed (see About Mroonga), its basic usage is similar to that of a regular fulltext index.

For example:

```
CREATE TABLE ft_mroonga(copy TEXT,FULLTEXT(copy)) ENGINE=Mroonga;

INSERT INTO ft_mroonga(copy) VALUES ('Once upon a time'),
    ('There was a wicked witch'), ('Who ate everybody up');

SELECT * FROM ft_mroonga WHERE MATCH(copy) AGAINST('wicked');
+--------------------------+
| copy                     |
+--------------------------+
| There was a wicked witch |
+--------------------------+
```

Score

Mroonga can also order by weighting. For example, first add another record:

```
INSERT INTO ft_mroonga(copy) VALUES ('She met a wicked, wicked witch');
```

Records can be returned by weighting, for example, the newly added record has two occurences of the word 'wicked' and a higher weighting:

```
SELECT *, MATCH(copy) AGAINST('wicked') AS score FROM ft_mroonga
   WHERE MATCH(copy) AGAINST('wicked') ORDER BY score DESC;
+--------------------------------+--------+
| copy                           | score  |
+--------------------------------+--------+
| She met a wicked, wicked witch | 299594 |
| There was a wicked witch       | 149797 |
+--------------------------------+--------+
```

Parser

Mroonga permits you to set a different parser for searching by specifying the parser in the `CREATE TABLE` statement as a comment or, in older versions, changing the value of the mroonga_default_parser system variable.

For example:

```
CREATE TABLE ft_mroonga(copy TEXT,FULLTEXT(copy) COMMENT 'parser "TokenDelimitNull"')
  ENGINE=Mroonga;,
```

or

```
SET GLOBAL mroonga_default_parser = 'TokenBigramSplitSymbol';
```

The following parser settings are available:

Setting	Description
off	No tokenizing is performed.
TokenBigram	Default value. Continuous alphabetical characters, numbers or symbols are treated as a token.
TokenBigramIgnoreBlank	Same as `TokenBigram` except that white spaces are ignored.
TokenBigramIgnoreBlankSplitSymbol	Same as `TokenBigramSplitSymbol` . except that white spaces are ignore.
TokenBigramIgnoreBlankSplitSymbolAlpha	Same as `TokenBigramSplitSymbolAlpha` except that white spaces are ignored.
TokenBigramIgnoreBlankSplitSymbolAlphaDigit	Same as `TokenBigramSplitSymbolAlphaDigit` except that white spaces are ignored.
TokenBigramSplitSymbol	Same as `TokenBigram` except that continuous symbols are not treated as a token, but tokenised in bigram.
TokenBigramSplitSymbolAlpha	Same as `TokenBigram` except that continuous alphabetical characters are not treated as a token, but tokenised in bigram.
TokenDelimit	Tokenises by splitting on white spaces.
TokenDelimitNull	Tokenises by splitting on null characters (\0).
TokenMecab	Tokenise using MeCab. Required Groonga to be buillt with MeCab support.
TokenTrigram	Tokenises in trigrams but continuous alphabetical characters, numbers or symbols are treated as a token.
TokenUnigram	Tokenises in unigrams but continuous alphabetical characters, numbers or symbols are treated as a token.

Examples

TokenBigram vs TokenBigramSplitSymbol

`TokenBigram` failing to match partial symbols which `TokenBigramSplitSymbol` matches, since `TokenBigramSplitSymbol` does not treat continuous symbols as a token.

```
DROP TABLE ft_mroonga;
CREATE TABLE ft_mroonga(copy TEXT,FULLTEXT(copy) COMMENT 'parser "TokenBigram"')
  ENGINE=Mroonga;
INSERT INTO ft_mroonga(copy) VALUES ('Once upon a time'),
  ('There was a wicked witch'),
  ('Who ate everybody up'),
  ('She met a wicked, wicked witch'),
  ('A really wicked, wicked witch!!?!');
SELECT * FROM ft_mroonga WHERE MATCH(copy) AGAINST('!?');
Empty set (0.00 sec)

DROP TABLE ft_mroonga;
CREATE TABLE ft_mroonga(copy TEXT,FULLTEXT(copy) COMMENT 'parser "TokenBigramSplitSymbol"')
  ENGINE=Mroonga;
INSERT INTO ft_mroonga(copy) VALUES ('Once upon a time'),
  ('There was a wicked witch'),
  ('Who ate everybody up'),
  ('She met a wicked, wicked witch'),
  ('A really wicked, wicked witch!!?!');
SELECT * FROM ft_mroonga WHERE MATCH(copy) AGAINST('!?');
+----------------------------------+
| copy                             |
+----------------------------------+
| A really wicked, wicked witch!!?! |
+----------------------------------+
```

TokenBigram vs TokenBigramSplitSymbolAlpha

```
DROP TABLE ft_mroonga;
CREATE TABLE ft_mroonga(copy TEXT,FULLTEXT(copy) COMMENT 'parser "TokenBigram"')
  ENGINE=Mroonga;
INSERT INTO ft_mroonga(copy) VALUES ('Once upon a time'),
  ('There was a wicked witch'),
  ('Who ate everybody up'),
  ('She met a wicked, wicked witch'),
  ('A really wicked, wicked witch!!?!');
SELECT * FROM ft_mroonga WHERE MATCH(copy) AGAINST('ick');
Empty set (0.00 sec)

DROP TABLE ft_mroonga;
CREATE TABLE ft_mroonga(copy TEXT,FULLTEXT(copy) COMMENT 'parser "TokenBigramSplitSymbolAlpha"')
  ENGINE=Mroonga;
INSERT INTO ft_mroonga(copy) VALUES ('Once upon a time'),
  ('There was a wicked witch'),
  ('Who ate everybody up'),
  ('She met a wicked, wicked witch'),
  ('A really wicked, wicked witch!!?!');
SELECT * FROM ft_mroonga WHERE MATCH(copy) AGAINST('ick');
+----------------------------------+
| copy                             |
+----------------------------------+
| There was a wicked witch         |
| She met a wicked, wicked witch   |
| A really wicked, wicked witch!!?! |
+----------------------------------+
```

5.3.12.3 Mroonga Status Variables

Contents

This page documents status variables related to the Mroonga storage engine. See Server Status Variables for a complete list of status variables that can be viewed with SHOW STATUS.

`Mroonga_count_skip`

- **Description:** Incremented each time the 'fast line count feature' is used. Can be used to check if the feature is working after enabling it.
- **Data Type:** `numeric`

`Mroonga_fast_order_limit`

- **Description:** Incremented each time the 'fast ORDER BY LIMIT feature' is used. Can be used to check if the feature is working after enabling it.
- **Data Type:** `numeric`

5.3.12.4 Mroonga System Variables

Contents

This page documents system variables related to the Mroonga storage engine. See Server System Variables for a complete list of system variables and instructions on setting them.

See also the Full list of MariaDB options, system and status variables.

`mroonga_action_on_fulltext_query_error`

- **Description:** Action to take when encountering a Mroonga fulltext error.
  - `ERROR` : Report an error without logging.
  - `ERROR_AND_LOG` : Report an error with logging (the default)
  - `IGNORE` : No logging or reporting - the error is ignored.
  - `IGNORE_AND_LOG` : Log the error without reporting it.
- **Commandline:** `--mroonga-action-on-fulltext-query-error=value`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `enum`
- **Default Value:** `ERROR_AND_LOG`

`mroonga_boolean_mode_syntax_flags`

- **Description:** Flags to customize syntax in BOOLEAN MODE searches. Available flags:
  - `DEFAULT` : (=SYNTAX_QUERY,ALLOW_LEADING_NOT)
  - `ALLOW_COLUMN` : Allows `COLUMN:...` syntax in query syntax, an incompatible change to the regular BOOLEAN MODE syntax. Permits multiple indexes in one `MATCH () AGAINST ()`. Can be used in other operations besides full-text search, such as equal, and prefix search. See Groonga query syntax for more details.
  - `ALLOW_LEADING_NOT` Permits using the `NOT_INCLUDED_KEYWORD` syntax in the query syntax.
  - `ALLOW_UPDATE` : Permits updating values with the `COLUMN:=NEW_VALUE` syntax in the query syntax.
  - `SYNTAX_QUERY` : Mroonga will use Groonga's query syntax, compatible with MariaDB's BOOLEAN MODE syntax. Unless `SYNTAX_SCRIPT` is specified, this mode is always in use.
  - `SYNTAX_SCRIPT` : Mroonga will use Groonga's script syntax, a JavaScript-like syntax. If both `SYNTAX_QUERY` and `SYNTAX_SCRIPT` are specified, `SYNTAX_SCRIPT` will take precedence..
- **Commandline:** `--mroonga-boolean-mode-syntax-flags=value`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `enum`
- **Default Value:** `DEFAULT`

`mroonga_database_path_prefix`

- **Description:** The database path prefix.
- **Commandline:** `--mroonga-database-path-prefix=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** (Empty)

---

mroonga_default_parser

- **Description:** The fulltext default parser, for example `TokenBigramSplitSymbolAlphaDigit` or `TokenBigram` (the default). See the list of options at Mroonga Overview:Parser. Deprecated since Mroonga 5.04, use mroonga_default_tokenizer instead.
- **Commandline:** `--mroonga-default-parser=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `TokenBigram`
- **Deprecated:** MariaDB 10.1.6, Mroonga 5.0.4

---

mroonga_default_tokenizer

- **Description:** The fulltext default parser, for example `TokenBigramSplitSymbolAlphaDigit` or `TokenBigram` (the default). See the list of options at Mroonga Overview:Parser.
- **Commandline:** `--mroonga-default-tokenizer=value`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `TokenBigram`
- **Introduced:** MariaDB 10.1.6, Mroonga 5.0.4

---

mroonga_default_wrapper_engine

- **Description:** The default engine for wrapper mode.
- **Commandline:** `--mroonga-default-wrapper-engine=value`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** (Empty)

---

mroonga_dry_write

- **Description:** If set to `on`, (`off` is default), data is not actually written to the Groonga database. Only really useful to change for benchmarking.
- **Commandline:** `--mroonga-dry-write[={0|1}]`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `off`

---

mroonga_enable_operations_recording

- **Description:** Whether recording operations for recovery to the Groonga database is enabled (default) or not. Requires reopening the database with FLUSH TABLES after changing the variable.
- **Commandline:** `--mroonga-enable-operations-recording={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `ON`
- **Introduced:** MariaDB 10.2.11, MariaDB 10.1.29

---

mroonga_enable_optimization

- **Description:** If set to `on` (the default), optimization is enabled. Only really useful to change for benchmarking.
- **Commandline:** `--mroonga-enable-optimization={0|1}`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `on`

---

mroonga_libgroonga_embedded

- **Description:** Whether libgroonga is embedded or not.
- **Commandline:** None
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `ON`
- **Introduced:** MariaDB 10.1.6

---

mroonga_libgroonga_support_lz4

- **Description:** Whether libgroonga supports lz4 or not.
- **Commandline:** None
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

mroonga_libgroonga_support_zlib

- **Description:** Whether libgroonga supports zlib or not.
- **Commandline:** None
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `ON`

---

mroonga_libgroonga_support_zstd

- **Description:** Whether libgroonga supports Zstandard or not.
- **Commandline:** None
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB 10.2.11, MariaDB 10.1.29

---

mroonga_libgroonga_version

- **Description:** Groonga library version.
- **Commandline:** None
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`

---

mroonga_lock_timeout

- **Description:** Lock timeout used in Groonga.
- **Commandline:** `<<code>>` --mroonga-lock-timeout=#</code>>

- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `900000`
- **Range:** `-1` to `2147483647`

`mroonga_log_file`

- **Description:** Name and path of the Mroonga log file.
- **Commandline:** `--mroonga-log-file=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `groonga.log`

`mroonga_log_level`

- **Description:** Mroonga log file output level, which determines what is logged. Valid levels include:
  - `NONE` No output.
  - `EMERG` : Only emergency error messages, such as database corruption.
  - `ALERT` : Alert messages, such as internal errors.
  - `CRIT` : Critical error messages, such as deadlocks.
  - `ERROR` : Errors, such as API errors.
  - `WARNING` : Warnings, such as invalid arguments.
  - `NOTICE` : Notices, such as a change in configuration or a status change.
  - `INFO` : Information messages, such as file system operations.
  - `DEBUG` : Debug messages, suggested for developers or testers.
  - `DUMP` : Dump messages.
- **Commandline:** `--mroonga-log-level=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `enum`
- **Default Value:** `NOTICE`

`mroonga_match_escalation_threshold`

- **Description:** The threshold to determine whether the match method is escalated. `-1` means never escalate.
- **Commandline:** `--mroonga-match-escalation-threshold=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `-1` to `9223372036854775807`

`mroonga_max_n_records_for_estimate`

- **Description:** The max number of records to estimate the number of matched records
- **Commandline:** `--mroonga-max-n-records-for-estimate=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1000`
- **Range:** `-1` to `2147483647`

`mroonga_query_log_file`

- **Description:** Query log file for Mroonga.
- **Commandline:** `--mroonga-query-log-file=filename`
- **Scope:** Global
- **Dynamic:** Yes

- **Data Type:** `string`
- **Default Value:** (Empty string)
- **Introduced:** MariaDB 10.2.11

`mroonga_vector_column_delimiter`

- **Description:** Delimiter to use when outputting a vector column. The default is a white space.
- **Commandline:** `--mroonga-vector-column-delimiter=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** (white space)

`mroonga_version`

- **Description:** Mroonga version
- **Commandline:** None
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`

5.3.12.5 Mroonga User-Defined Functions

Mroonga provides a number of User-defined functions (UDFs)

Creating Mroonga User-Defined Functions
How to install Mroonga's user-defined functions.

last_insert_grn_id
Returns the unique Groonga id of the last-inserted record.

mroonga_command
Pass a command to Groonga to execute.

mroonga_escape
Escaping a string.

mroonga_highlight_html
Highlights the specified keywords in the target text.

mroonga_normalize
Uses Groonga's normalizer to normalize text.

mroonga_snippet
A keyword with surrounding text, or the keyword in context.

mroonga_snippet_html
It provides a keyword with surrounding text, or the keyword in context.

5.3.12.5.1 Creating Mroonga User-Defined Functions

The Mroonga storage engine includes a number of user-defined functions that need to be created before they can be used. If these are not created already during Mroonga setup, you will need to do so yourself. The full list of available functions and the statements to create them are found in `share/mroonga/install.sql`, for example, as of Mroonga 7.07 (MariaDB 10.2.11 and MariaDB 10.1.29) running on Linux:

```
DROP FUNCTION IF EXISTS last_insert_grn_id;
CREATE FUNCTION last_insert_grn_id RETURNS INTEGER
  SONAME 'ha_mroonga.so';

DROP FUNCTION IF EXISTS mroonga_snippet;
CREATE FUNCTION mroonga_snippet RETURNS STRING
  SONAME 'ha_mroonga.so';

DROP FUNCTION IF EXISTS mroonga_command;
CREATE FUNCTION mroonga_command RETURNS STRING
  SONAME 'ha_mroonga.so';

DROP FUNCTION IF EXISTS mroonga_escape;
CREATE FUNCTION mroonga_escape RETURNS STRING
  SONAME 'ha_mroonga.so';

DROP FUNCTION IF EXISTS mroonga_snippet_html;
CREATE FUNCTION mroonga_snippet_html RETURNS STRING
  SONAME 'ha_mroonga.so';

DROP FUNCTION IF EXISTS mroonga_normalize;
CREATE FUNCTION mroonga_normalize RETURNS STRING
  SONAME 'ha_mroonga.so';

DROP FUNCTION IF EXISTS mroonga_highlight_html;
CREATE FUNCTION mroonga_highlight_html RETURNS STRING
  SONAME 'ha_mroonga.so';

DROP FUNCTION IF EXISTS mroonga_query_expand;
CREATE FUNCTION mroonga_query_expand RETURNS STRING
  SONAME 'ha_mroonga.so';
```

5.3.12.5.2 last_insert_grn_id

Syntax

```
last_insert_grn_id()
```

Contents

Description

`last_insert_grn_id` is a user-defined function (UDF) included with the Mroonga storage engine. It returns the unique Groonga id of the last-inserted record. See Creating Mroonga User-Defined Functions for details on creating this UDF if required.

Examples

```
SELECT last_insert_grn_id();
+----------------------+
| last_insert_grn_id() |
+----------------------+
|                    3 |
+----------------------+
```

5.3.12.5.3 mroonga_command

Contents

Syntax

```
mroonga_command (command)
```

Description

`mroonga_command` is a user-defined function (UDF) included with the Mroonga storage engine. It passes a command to Groonga for execution. See Creating Mroonga User-Defined Functions for details on creating this UDF if required.

- `command` - string, required parameter specifying the command to pass that will be executed by Groonga. See the Groonga reference for a list of commands.

Returns the result of the Groonga command.

Example

```
SELECT mroonga_command('status');
+-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------
----------------------+
| mroonga_command('status')
|
+-----------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------
----------------------+
|
{"alloc_count":593,"starttime":1512022368,"start_time":1512022368,"uptime":13510,"version":"7.0
.7","n_queries":0,"cache_hit_rate":0.0,"command_version":1,"default_command_version":1,"max_com
mand_version":3} |
```

5.3.12.5.4 mroonga_escape

Contents

Syntax

```
mroonga_escape (string [,special_characters])
```

- `string` - required parameter specifying the text you want to escape
- `special_characters` - optional parameter specifying the characters to escape

Description

`mroonga_escape` is a user-defined function (UDF) included with the Mroonga storage engine, used for escaping a string. See Creating Mroonga User-Defined Functions for details on creating this UDF if required.

If no `special_characters` parameter is provided, by default `+-<>*()":` are escaped.

Returns the escaped string.

Example

```
SELECT mroonga_escape("+-<>~*()\"\:");
'\\+\\-\\<\\>\\~\\*\\(\\)\\"\\:
```

5.3.12.5.5 mroonga_highlight_html

Syntax

```
mroonga_highlight_html(text[[, query AS query]])

mroonga_highlight_html(text[[, keyword1, ..., keywordN]])
```

Contents

Description

`mroonga_highlight_html` is a user-defined function (UDF) included with the Mroonga storage engine. It highlights the specified keywords in the target text. See Creating Mroonga User-Defined Functions for details on creating this UDF if required.

The optional parameter can either be one or more *keywords*, or a Groonga *query*.

The function highlights the specified keywords in the target text by surrounding each keyword with `<span class="keyword">...</span>`, and escaping special HTML characters such as `<` and `>`.

Returns highlighted HTML.

Examples

```
SELECT mroonga_highlight_html('<p>MariaDB includes the Mroonga storage engine</p>.')
  AS highlighted_html;
+-------------------------------------------------------------------+
| highlighted_html                                                  |
+-------------------------------------------------------------------+
| &lt;p&gt;MariaDB includes the Mroonga storage engine&lt;/p&gt;. |
+-------------------------------------------------------------------+
```

Highlighting the words `MariaDB` and `Mroonga` in a given text:

```
SELECT mroonga_highlight_html('MariaDB includes the Mroonga storage engine.', 'MariaDB',
'Mroonga')
  AS highlighted_html;
+-------------------------------------------------------------------------------------------------
------------+
| highlighted_html
|
+-------------------------------------------------------------------------------------------------
------------+
| <span class="keyword">MariaDB</span> includes the <span class="keyword">Mroonga</span>
storage engine. |
+-------------------------------------------------------------------------------------------------
------------+
```

The same outcome, formulated as a Groonga query:

```
SELECT mroonga_highlight_html('MariaDB includes the Mroonga storage engine.', 'MariaDB OR
Mroonga'
  AS query) AS highlighted_text;
+-------------------------------------------------------------------------------------------
------------+
| highlighted_text
|
+-------------------------------------------------------------------------------------------
------------+
| <span class="keyword">MariaDB</span> includes the <span class="keyword">Mroonga</span>
storage engine. |
+-------------------------------------------------------------------------------------------
------------+
```

5.3.12.5.6 mroonga_normalize

Syntax

```
mroonga_normalize(string[, normalizer_name])
```

Contents

Description

`mroonga_normalize` is a user-defined function (UDF) included with the Mroonga storage engine. It uses Groonga's normalizer to normalize text. See Creating Mroonga User-Defined Functions for details on creating this UDF if required.

Given a string, returns the normalized text.

See the Groonga Normalizer Reference for details on the Groonga normalizers. The default if no normalizer is provided is `NormalizerAuto`.

Examples

```
SELECT mroonga_normalize("ABいリットル");
+--------------------------------+
| mroonga_normalize("ABいリットル")   |
+--------------------------------+
| abいリットル                      |
+--------------------------------+
```

5.3.12.5.7 mroonga_snippet

Syntax

```
mroonga_snippet document,
                max_length,
                max_count,
                encoding,
                skip_leading_spaces,
                html_escape,
                snippet_prefix,
                snippet_suffix,
                word1, word1_prefix, word1_suffix
                ...
                [wordN wordN_prefix wordN_suffix]
```

Contents
1. Syntax
2. Description
3. Example

Description

`mroonga_snippet` is a user-defined function (UDF) included with the Mroonga storage engine. It provides a keyword with surrounding text, or the keyword in context. See Creating Mroonga User-Defined Functions for details on creating this UDF if required.

The required parameters include:

- `document` - Column name or string value.
- `max_length` - Maximum length of the snippet, in bytes.
- `max_count` - Maximum snippet elements (N word).
- `encoding` - Encoding of the document, for example `cp932_japanese_ci`
- `skip_leading_spaces` - `1` to skip leading spaces, `0` to not skip.
- `html_escape` = `1` to enable HTML espape, `0` to disable.
- `prefix` - Snippet start text.
- `suffix` - Snippet end text.

The optional parameters include:

- `wordN` - A word.
- `wordN_prefix` - wordN start text.
- `wordN_suffix` - wordN end text

It can be used in both storage and wrapper mode.

Returns the snippet string.

Example

5.3.12.5.8 mroonga_snippet_html

Description

`mroonga_snippet_html` is a user-defined function (UDF) included with the Mroonga storage engine. It provides a keyword with surrounding text, or the keyword in context. It is still considered experimental. See Creating Mroonga User-Defined Functions for details on creating this UDF if required.

1.1.1.2.9.1.1.29 Information Schema MROONGA_STATS Table

5.3.13 MyISAM

MyISAM was the default storage engine from MySQL 3.23 until it was replaced by InnoDB in MariaDB and MySQL 5.5. It's a light, non-transactional engine with great performance, is easy to copy between systems and has a small data footprint.

You're encouraged to rather use the Aria storage engine for new applications, which has even better performance and the goal of being crash-safe.

Until MariaDB 10.4, system tables used the MyISAM storage engine.

MyISAM Overview

Light, non-transactional storage engine.

MyISAM System Variables

MyISAM system variables.

MyISAM Storage Formats

The MyISAM storage engine supports three different table storage formats

MyISAM Clients and Utilities

Clients and utilities for working with MyISAM tables

MyISAM Index Storage Space

Regular MyISAM tables make use of B-tree indexes

MyISAM Log

Records all changes to MyISAM tables

Concurrent Inserts

Under some circumstances, MyISAM allows INSERTs and SELECTs to be executed concurrently.

Segmented Key Cache

Collection of structures for regular MyISAM key caches

There are 1 related questions.

5.3.13.1 MyISAM Overview

The MyISAM storage engine was the default storage engine from MySQL 3.23 until it was replaced as default by InnoDB in MariaDB and MySQL 5.5. Historically, MyISAM is a replacement for the older ISAM engine, removed in MySQL 4.1.

It's a light, non-transactional engine with great performance, is easy to copy between systems and has a small data footprint.

You're encouraged to rather use the Aria storage engine for new applications, which has even better performance in most cases and the goal of being crash-safe.

A MyISAM table is stored in three files on disk. There's a table definition file with an extension of `.frm`, a data file with the extension `.MYD`, and an index file with the extension `.MYI`.

MyISAM features

- Does not support transactions.
- Does not support foreign keys.
- Supports FULLTEXT indexes.
- Supports GIS data types.
- Storage limit of 256TB.
- Maximum of 64 indexes per table.
- Maximum of 32 columns per index.
- Maximum index length of 1000 bytes.
- Limit of $(2^{32})^2$ (1.844E+19) rows per table.
- Supports large files up to 63-bits in length where the underlying system supports this.
- All data is stored with the low byte first, so all files will still work if copied to other systems or other machines.
- The data file and the index file can be placed on different devices to improve speed.
- Supports table locking, not row locking.
- Supports a key buffer that is segmented in MariaDB.
- Supports concurrent inserts.
- Supports fixed length, dynamic and compressed formats - see MyISAM Storage Formats.
- Numeric index values are stored with the high byte first, which enables more efficient index compression.
- Data values are stored with the low byte first, making it mostly machine and operating system independent. The only exceptions are if a machine doesn't use two's-complement signed integers and the IEEE floating-point format.
- Can be copied between databases or systems with normal system tools, as long as the files are not open on either system. Use FLUSH_TABLES to ensure files are not in use.
- There are a number of tools for working with MyISAM tables. These include:
  - mariadb-check for checking or repairing
  - myisamchk for checking or repairing
  - myisampack for compressing
- It is possible to build a MERGE table on the top of one or more MyISAM tables.

5.3.13.2 MyISAM System Variables

Contents

This page documents system variables related to the MyISAM storage engine. For options, see MyISAM Options.

See Server System Variables for a complete list of system variables and instructions on setting them.

See also the Full list of MariaDB options, system and status variables.

`key_buffer_size`

- **Description:** Size of the buffer for the index blocks used by MyISAM tables and shared for all threads. See Optimizing key_buffer_size for more on selecting the best value.
- **Commandline:** `--key-buffer-size=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `134217728`
- **Range:** `8` upwards (upper limit determined by operating system per process limit)

`key_cache_age_threshold`

- **Description:** The lower the setting, the more quickly buffers move from the hot key cache sublist to the warm sublist.
- **Commandline:** `--key-cache-age-threshold=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `300`
- **Range:** `100` to `4294967295`

`key_cache_block_size`

- **Description:** MyISAM key cache block size in bytes .
- **Commandline:** `--key-cache-block-size=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1024`
- **Range:** `512` to `16384`

`key_cache_division_limit`

- **Description:** Percentage to use for the warm key cache buffer list (the remainder is allocated between the hot and cold caches).
- **Commandline:** `--key-cache-division-limit=#`
- **Scope:** Global

- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `100`
- **Range:** `1` to `100`

---

key_cache_file_hash_size

- **Description:** Number of hash buckets for open and changed files. If you have many MyISAM files open you should increase this for faster flushing of changes. A good value is probably 1/10th of the number of possible open MyISAM files.
- **Commandline:** `--key-cache-file-hash-size=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `512`
- **Range:** `128` to `16384`

---

key_cache_segments

- **Description:** The number of segments in a key cache. See Segmented Key Cache.
- **Commandline:** `--key-cache-segments=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Type:** numeric
- **Default Value:** `0` *(non-segmented)*
- **Range:** `0` to `64`

---

myisam_block_size

- **Description:** Block size to be used for MyISAM index pages.
- **Commandline:** `--myisam-block-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `1024`

---

myisam_data_pointer_size

- **Description:** Size in bytes of the default pointer, used in a MyISAM CREATE TABLE with no MAX_ROWS option.
- **Commandline:** `--myisam-data-pointer-size=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `6`
- **Range:** `2` to `7`

---

myisam_max_extra_sort_file_size

- **Description:** Removed in MySQL 5.0.6, was used as a way to force long character keys in large tables to use the key cache method.
- **Removed:** MySQL 5.0.6

---

myisam_max_sort_file_size

- **Description:** Maximum size in bytes of the temporary file used while recreating a MyISAM index. If the this size is exceeded, the slower process of using the key cache is done instead.
- **Commandline:** `--myisam-max-sort-file-size=#`
- **Scope:** Global
- **Dynamic:** Yes

- **Data Type:** `numeric`
- **Default Value - 32 bit:** `2147483648`
- **Default Value - 64 bit:** `9223372036854775807`

---

myisam_mmap_size

- **Description:** Maximum memory in bytes that can be used for memory mapping compressed MyISAM files. Too high a value may result in swapping if there are many compressed MyISAM tables.
- **Commandline:** `--myisam-mmap-size=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value - 32 bit:** `4294967295`
- **Default Value - 64 bit:** `18446744073709547520`
- **Range - 32-bit:** `7` to `4294967295`
- **Range - 64-bit:** `7` to `18446744073709547520`

---

myisam_recover_options

- **Description:** MyISAM recovery mode. Multiple options can be selected, comma-delimited. Using no argument is equivalent to specifying `DEFAULT`, while specifying "" is equivalent to `OFF`. If enabled each time the server opens a MyISAM table, it checks whether it has been marked as crashed, or wasn't closed properly. If so, mysqld will run a check and then attempt to repair the table, writing to the error log beforehand.
  - **OFF**: No recovery.
  - **BACKUP**: If the data file is changed while recovering, saves a backup of the .MYD data file. t.MYD will be saved as t.MYD-datetime.BAK.
  - **BACKUP_ALL**: Same as `BACKUP` but also backs up the .MYI index file. t.MYI will be saved as t.MYI-datetime.BAK.
  - **DEFAULT**: Recovers without backing up, forcing, or quick checking.
  - **FORCE**: Runs the recovery even if it determines that more than one row from the data file will be lost.
  - **QUICK**: Does not check rows in the table if there are no delete blocks.
- **Commandline:** `--myisam-recover-options[=name]`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `enumeration`
- **Default Value:**
  - `BACKUP,QUICK` (>= MariaDB 10.2.4)
  - `DEFAULT` (<= MariaDB 10.2.3)
  - `OFF`
- **Valid Values:** `OFF`, `DEFAULT`, `BACKUP`, `BACKUP_ALL`, `FORCE` or `QUICK`

---

myisam_repair_threads

- **Description:** If set to more than `1`, the default, MyISAM table indexes each have their own thread during repair and sorting. Increasing from the default will usually result in faster repair, but will use more CPU and memory.
- **Commandline:** `--myisam-repair-threads=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1`
- **Range - 32-bit:** `1` to `4294967295`
- **Range - 64-bit:** `1` to `18446744073709547520`

---

myisam_sort_buffer_size

- **Description:** Size in bytes of the buffer allocated when creating or sorting indexes on a MyISAM table.
- **Commandline:** `--myisam-sort-buffer-size=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `134217720` (128MB)

- **Range:** `4096` to `18446744073709547520`

`myisam_stats_method`

- **Description:** Determines how NULLs are treated for MyISAM index statistics purposes. If set to `nulls_equal`, the default, all NULL index values are treated as a single group. This is usually fine, but if you have large numbers of NULLs the average group size is slanted higher, and the optimizer may miss using the index for ref accesses when it would be useful. If set to `nulls_unequal`, the opposite approach is taken, with each NULL forming its own group of one. Conversely, the average group size is slanted lower, and the optimizer may use the index for ref accesses when not suitable. Setting to `nulls_ignored` ignores NULLs altogether from index group calculations. See also Index Statistics, aria_stats_method, innodb_stats_method.
- **Commandline:** `--myisam-stats-method=name`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `enumeration`
- **Default Value:** `nulls_equal`
- **Valid Values:** `nulls_equal`, `nulls_unequal`, `nulls_ignored`

`myisam_use_mmap`

- **Description:** If set to `1` (0 is default), memory mapping will be used to reading and writing MyISAM tables.
- **Commandline:** `--myisam-use-mmap`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

5.3.13.3 MyISAM Storage Formats

Contents

The MyISAM storage engine supports three different table storage formats.

These are FIXED, DYNAMIC and COMPRESSED. FIXED and DYNAMIC can be set with the ROW FORMAT option in the CREATE TABLE statement, or will be chosen automatically depending on the columns the table contains. COMPRESSED can only be set via the myisampack tool.

The SHOW TABLE STATUS statement can be used to see the storage format used by a table. Note that `COMPRESSED` tables are reported as `DYNAMIC` in that context.

Fixed-length

Fixed-length (or static) tables contain records of a fixed-length. Each column is the same length for all records, regardless of the actual contents. It is the default format if a table has no BLOB, TEXT, VARCHAR or VARBINARY fields, and no ROW FORMAT is provided. You can also specify a fixed table with ROW_FORMAT=FIXED in the table definition.

> Tables containing BLOB or TEXT fields cannot be FIXED, as by design these are both dynamic fields. However, no error or warning will be raised if you specify FIXED.

Fixed-length tables have a number of characteristics

- fast, since MariaDB will always know where a record begins
- easy to repair: myisamchk is always able to recover all rows, except for the last one if it is not entirely written
- easy to cache
- take up more space than dynamic or compressed tables, as the maximum amount of storage space will be allocated to each record.
- reconstructing after a crash is uncomplicated due to the fixed positions
- no fragmentation or need to re-organize, unless records have been deleted and you want to free the space up.

Dynamic

Dynamic tables contain records of a variable length. It is the default format if a table has any BLOB, TEXT, VARCHAR or VARBINARY fields, and no ROW FORMAT is provided. You can also specify a DYNAMIC table with ROW_FORMAT=DYNAMIC in the table definition. If the table contains BLOB or TEXT columns, its format is always DYNAMIC, and the ROW FORMAT option is ignored.

Dynamic tables have a number of characteristics

- Each row contains a header indicating the length of the row.
- Rows tend to become fragmented easily. UPDATING a record to be longer will likely ensure it is stored in different places on the disk. Use OPTIMIZE TABLE when the fragmentation is too high.
- All string columns with a length of four or more are dynamic.
- They require much less space than fixed-length tables.
- Restoring after a crash is more complicated than with FIXED tables. Some fragments may be lost.

If a DYNAMIC table has some frequently-accessed fixed-length columns, it could be a good idea to move them into a separate FIXED table to avoid fragmentation.

Compressed

Compressed tables are a read-only format, created with the myisampack tool. This can be done while the server is running, but external lock must not be disabled. myisamchk is used to uncompress them.

Compressed tables have a number of characteristics:

- while the data is read-only, DDL statements such as DROP TABLE and TRUNCATE TABLE will still function.
- take much less space than fixed or dynamic tables. Each data has usually a 40-70% compression ratio
- rows are compressed separately, reducing access overhead.
- row headers will be from one to three bytes.
- rows can be compressed with different compression types, including
    - prefix space compression
    - suffix space compression
    - columns with small sets of values are converted to ENUM
    - numeric zeros are stored with only one bit
    - integer columns will be reduced to the smallest int type that can hold the contents

1.3.8 MyISAM Clients and Utilities

5.3.13.5 MyISAM Index Storage Space

Regular MyISAM tables make use of B-tree indexes.

String indexes are space-compressed, which reduces the size of VARCHARs that don't use the full length, or a string that has trailing spaces. String indexes also make use of prefix-compression, where strings with identical prefixes are compressed.

Numeric indexes can also be prefix-compressed compressed if the PACK_KEYS=1 option is used. Regardless, the high byte is always stored first, which allows a reduced index size.

In the worst case, with no strings being space-compressed, the total index storage space will be (index_length+4)/0.67 per index.

5.3.13.6 MyISAM Log

The MyISAM log records all changes to MyISAM tables. It is not enabled by default. To enable it, start the server with the --log-isam option, for example:

```
--log-isam=myisam.log
```

The *isam* instead of *myisam* above is not a typo - it's a legacy from when the predecessor to the MyISAM format, called ISAM. The option can be used without specifying a filename, in which case the default, *myisam.log* is used.

To process the contents of the log file, use the myisamlog utility.

5.3.13.8 Segmented Key Cache

Contents

About Segmented Key Cache

A segmented key cache is a collection of structures for regular MyISAM key caches called key cache segments. Segmented key caches mitigate one of the major problems of the simple key cache: thread contention for key cache lock (mutex). With regular key caches, every call of a key cache interface function must acquire this lock. So threads compete for this lock even in the case when they have acquired shared locks for the file and the pages they want to read from are in the key cache buffers.

When working with a segmented key cache any key cache interface function that needs only one page has to acquire the key cache lock only for the segment the page is assigned to. This makes the chances for threads not having to compete for the same key cache lock better.

Any page from a file can be placed into a buffer of only one segment. The number of the segment is calculated from the file number and the position of the page in the file, and it's always the same for the page. Pages are evenly distributed among segments.

The idea and the original code of the segmented key cache was provided by Fredrik Nylander from Stardoll.com. The code was extensively reworked, improved, and eventually merged into MariaDB by Igor Babaev from Monty Program (now MariaDB Corporation).

You can find some benchmark results comparing various settings on the Segmented Key Cache Performance page.

Segmented Key Cache Syntax

New global variable: key_cache_segments. This variable sets the number of segments in a key cache. Valid values for this variable are whole numbers between `0` and `64`. If the number of segments is set to a number greater than `64` the number of segments will be truncated to 64 and a warning will be issued.

A value of `0` means the key cache is a regular (i.e. non-segmented) key cache. This is the default. If `key_cache_segments` is `1` (or higher) then the new key cache segmentation code is used. In practice there is no practical use of a single-segment segmented key cache except for testing purposes, and setting `key_cache_segments = 1` should be slower than any other option and should not be used in production.

Other global variables used when working with regular key caches also apply to segmented key caches: key_buffer_size, key_cache_age_threshold, key_cache_block_size, and key_cache_division_limit.

Segmented Key Cache Statistics

Statistics about the key cache can be found by looking at the KEY_CACHES table in the INFORMATION_SCHEMA database. Columns in this table are:

Column Name	Description
`KEY_CACHE_NAME`	The name of the key cache
`SEGMENTS`	total number of segments (set to `NULL` for regular key caches)
`SEGMENT_NUMBER`	segment number (set to `NULL` for any regular key caches and for rows containing aggregation statistics for segmented key caches)
`FULL_SIZE`	memory for cache buffers/auxiliary structures
`BLOCK_SIZE`	size of the blocks
`USED_BLOCKS`	number of currently used blocks
`UNUSED_BLOCKS`	number of currently unused blocks
`DIRTY_BLOCKS`	number of currently dirty blocks

READ_REQUESTS	number of read requests
READS	number of actual reads from files into buffers
WRITE_REQUESTS	number of write requests
WRITES	number of actual writes from buffers into files

5.3.14 MyRocks

MyRocks is a storage engine that adds the RocksDB database to MariaDB. RocksDB is an LSM database with a great compression ratio that is optimized for flash storage.

MyRocks Version	Introduced	Maturity
MyRocks 1.0	MariaDB 10.3.7, MariaDB 10.2.16	Stable
MyRocks 1.0	MariaDB 10.3.5, MariaDB 10.2.14	Gamma
MyRocks 1.0	MariaDB 10.3.4, MariaDB 10.2.13	Beta
MyRocks 1.0	MariaDB 10.2.5	Alpha

About MyRocks for MariaDB

Enables greater compression than InnoDB, and less write amplification.

Getting Started with MyRocks

Installing and getting started with MyRocks.

Building MyRocks in MariaDB

MariaDB compile process for MyRocks.

Loading Data Into MyRocks

MyRocks has ways to load data much faster than normal INSERTs

MyRocks Status Variables

MyRocks-related status variables.

MyRocks System Variables

MyRocks server system variables.

MyRocks Transactional Isolation

TODO: MyRocks uses snapshot isolation Support do READ-COMMITTED and REPEAT...

MyRocks and Replication

Details about how MyRocks works with replication.

MyRocks and Group Commit with Binary log

MyRocks supports group commit with the binary log

Optimizer Statistics in MyRocks

How MyRocks provides statistics to the query optimizer

Differences Between MyRocks Variants

Differences between Facebook's, MariaDB's and Percona Server's MyRocks.

MyRocks and Bloom Filters

Bloom filters are used to reduce read amplification.

MyRocks and CHECK TABLE

MyRocks supports the CHECK TABLE command.

MyRocks and Data Compression

MyRocks supports several compression algorithms.

MyRocks and Index-Only Scans

MyRocks and index-only scans on secondary indexes.

MyRocks and START TRANSACTION WITH CONSISTENT SNAPSHOT

FB/MySQL has added new syntax which returns the binlog coordinates pointing at the snapshot.

MyRocks Column Families

MyRocks stores data in column families, which are similar to tablespaces.

MyRocks in MariaDB 10.2 vs MariaDB 10.3

MyRocks storage engine in MariaDB 10.2 and MariaDB 10.3.

MyRocks Performance Troubleshooting

MyRocks exposes its performance metrics through several interfaces.

There are 1 related questions.

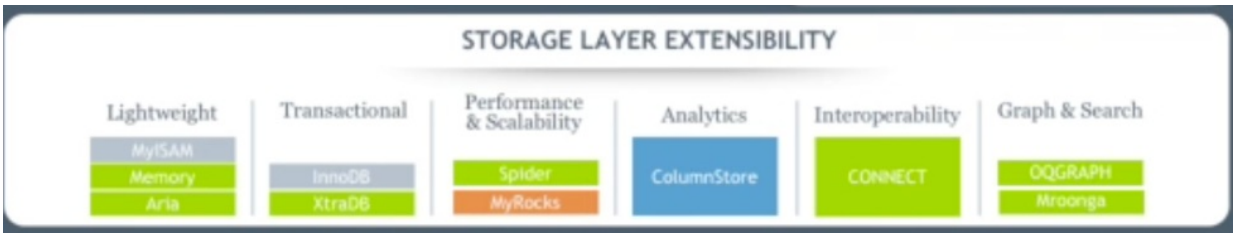
5.3.14.1 About MyRocks for MariaDB

Contents

About MyRocks for MariaDB

MyRocks is an open source storage engine that was originally developed by Facebook.

MyRocks has been extended by the MariaDB engineering team to be a pluggable storage engine that you use in your MariaDB solutions. It works seamlessly with MariaDB features. This openness in the storage layer allows you to use the right storage engine to optimize your usage requirements, which provides optimum performance. Community contributions are one of MariaDB's greatest advantages over other databases. Under the lead of our developer Sergey Petrunia, MyRocks in MariaDB is occasionally being merged with upstream MyRocks from Facebook.
See more at: https://mariadb.com/resources/blog/facebook-myrocks-mariadb#sthash.ZlEr7kNq.dpuf



MyRocks, typically, gives greater performance for web scale type applications. It can be an ideal storage engine solution when you have workloads that require greater compression and IO efficiency. It uses a Log Structured Merge (LSM) architecture, which has advantages over B-Tree algorithms, to provide efficient data ingestion, like read-free replication slaves, or fast bulk data loading. MyRocks distinguishing features include:

- compaction filter
- merge operator
- backup
- column families
- bulk loading
- persistent cache

For more MyRocks features see: https://github.com/facebook/rocksdb/wiki/Features-Not-in-LevelDB

Benefits

On production workloads, MyRocks was tested to prove that it provides:

Greater Space Efficiency

- 2x more compression
  MyRocks has 2x better compression compared to compressed InnoDB, 3-4x better compression compared to uncompressed InnoDB, meaning you use less space.

Greater Writing Efficiency

- 2x lower write rates to storage
  MyRocks has a 10x less write amplification compared to InnoDB, giving you better endurance of flash storage and improving overall throughput.

Faster Data Loading

- faster database loads
  MyRocks writes data directly onto the bottommost level, which avoids all compaction overheads when you enable faster data loading for a session.

Faster Replication

- No random reads for updating secondary keys, except for unique indexes. The Read-Free Replication option does away with random reads when updating primary keys, regardless of uniqueness, with a row-based binary logging format.

http://myrocks.io https://mariadb.com/resources/blog/facebook-myrocks-mariadb

Requirements and Limitations

- MyRocks is included from MariaDB 10.2.5.
- MyRocks is available in the MariaDB Server packages for Linux and Windows.
- Maria DB optimistic parallel replication may not be supported.
- MyRocks is not available for 32-bit platforms
- Galera Cluster is tightly integrated into InnoDB storage engine (it also supports Percona's XtraDB which is a modified version of InnoDB). Galera Cluster does not work with any other storage engines, including MyRocks (or TokuDB for example).

MyRocks builds are available on platforms that support a sufficiently modern compiler, for example:

- Ubuntu Trusty, Xenial, (amd64 and ppc64el)
- Ubuntu Yakkety (amd64)
- Debian Jessie, stable (amd64, ppc64el)
- Debian Stretch, Sid (testing and unstable) (amd64)
- CentOS/RHEL 7 (amd64)
- Centos/RHEL 7.3 (amd64)
- Fedora 24 and 25 (amd64)
- OpenSUSE 42 (amd64)
- Windows 64 (zip and MSI)

5.3.14.2 Getting Started with MyRocks

MariaDB starting with 10.2.5
The MyRocks storage engine was first released in MariaDB 10.2.5.

MyRocks is a storage engine that adds the RocksDB database to MariaDB. RocksDB is an LSM database with a great compression ratio that is optimized for flash storage.

The storage engine must be installed before it can be used.

Contents

Installing the Plugin's Package

The MyRocks storage engine's shared library is included in MariaDB packages as the `ha_rocksdb.so` or `ha_rocksdb.dll` shared library on systems where it can be built. The plugin was first included in MariaDB 10.2.5.

Installing on Linux

The MyRocks storage engine is included in binary tarballs on Linux.

Installing with a Package Manager

The MyRocks storage engine can also be installed via a package manager on Linux. In order to do so, your system needs to be configured to install from one of the MariaDB repositories.

You can configure your package manager to install it from MariaDB Corporation's MariaDB Package Repository by using the MariaDB Package Repository setup script.

You can also configure your package manager to install it from MariaDB Foundation's MariaDB Repository by using the MariaDB Repository Configuration Tool.

Installing with yum/dnf

On RHEL, CentOS, Fedora, and other similar Linux distributions, it is highly recommended to install the relevant RPM package from MariaDB's repository using `yum` or `dnf`. Starting with RHEL 8 and Fedora 22, `yum` has been replaced by `dnf`, which is the next major version of `yum`. However, `yum` commands still work on many systems that use `dnf`. For example:

```
sudo yum install MariaDB-rocksdb-engine
```

Installing with apt-get

On Debian, Ubuntu, and other similar Linux distributions, it is highly recommended to install the relevant DEB package from MariaDB's repository using `apt-get`. For example:

```
sudo apt-get install mariadb-plugin-rocksdb
```

Installing with zypper

On SLES, OpenSUSE, and other similar Linux distributions, it is highly recommended to install the relevant RPM package from MariaDB's repository using `zypper`. For example:

```
sudo zypper install MariaDB-rocksdb-engine
```

Installing on Windows

The MyRocks storage engine is included in MSI and ZIP packages on Windows.

Installing the Plugin

Once the shared library is in place, the plugin is not actually installed by MariaDB by default. There are two methods that

can be used to install the plugin with MariaDB.

The first method can be used to install the plugin without restarting the server. You can install the plugin dynamically by executing `INSTALL SONAME` or `INSTALL PLUGIN`. For example:

```
INSTALL SONAME 'ha_rocksdb';
```

The second method can be used to tell the server to load the plugin when it starts up. The plugin can be installed this way by providing the `--plugin-load` or the `--plugin-load-add` options. This can be specified as a command-line argument to `mysqld` or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
plugin_load_add = ha_rocksdb
```

Note: When installed with a package manager, an option file that contains the `--plugin-load-add` option may also be installed. The RPM package installs it as `/etc/my.cnf.d/rocksdb.cnf`, and the DEB package installs it as `/etc/mysql/mariadb.conf.d/rocksdb.cnf`

Uninstalling the Plugin

You can uninstall the plugin dynamically by executing `UNINSTALL SONAME` or `UNINSTALL PLUGIN`. For example:

```
UNINSTALL SONAME 'ha_rocksdb';
```

If you installed the plugin by providing the `--plugin-load` or the `--plugin-load-add` options in a relevant server option group in an option file, then those options should be removed to prevent the plugin from being loaded the next time the server is restarted.

Verifying the Installation

After installing MyRocks you will see RocksDB in the list of plugins:

```
SHOW PLUGINS;
+-------------------------------+----------+--------------------+---------------+---------+
| Name                          | Status   | Type               | Library       | License |
+-------------------------------+----------+--------------------+---------------+---------+
...
| ROCKSDB                       | ACTIVE   | STORAGE ENGINE     | ha_rocksdb.so | GPL     |
| ROCKSDB_CFSTATS               | ACTIVE   | INFORMATION SCHEMA | ha_rocksdb.so | GPL     |
| ROCKSDB_DBSTATS               | ACTIVE   | INFORMATION SCHEMA | ha_rocksdb.so | GPL     |
| ROCKSDB_PERF_CONTEXT          | ACTIVE   | INFORMATION SCHEMA | ha_rocksdb.so | GPL     |
| ROCKSDB_PERF_CONTEXT_GLOBAL   | ACTIVE   | INFORMATION SCHEMA | ha_rocksdb.so | GPL     |
| ROCKSDB_CF_OPTIONS            | ACTIVE   | INFORMATION SCHEMA | ha_rocksdb.so | GPL     |
| ROCKSDB_COMPACTION_STATS      | ACTIVE   | INFORMATION SCHEMA | ha_rocksdb.so | GPL     |
| ROCKSDB_GLOBAL_INFO           | ACTIVE   | INFORMATION SCHEMA | ha_rocksdb.so | GPL     |
| ROCKSDB_DDL                   | ACTIVE   | INFORMATION SCHEMA | ha_rocksdb.so | GPL     |
| ROCKSDB_INDEX_FILE_MAP        | ACTIVE   | INFORMATION SCHEMA | ha_rocksdb.so | GPL     |
| ROCKSDB_LOCKS                 | ACTIVE   | INFORMATION SCHEMA | ha_rocksdb.so | GPL     |
| ROCKSDB_TRX                   | ACTIVE   | INFORMATION SCHEMA | ha_rocksdb.so | GPL     |
...
+-------------------------------+----------+--------------------+---------------+---------+
```

Compression

Supported compression types are listed in the rocksdb_supported_compression_types variable. For example:

```
SHOW VARIABLES LIKE 'rocksdb_supported_compression_types';
+-----------------------------------+-------------+
| Variable_name                     | Value       |
+-----------------------------------+-------------+
| rocksdb_supported_compression_types | Snappy,Zlib |
+-----------------------------------+-------------+
```

See MyRocks and Data Compression for more.

System and Status Variables

All MyRocks system variables and status variables are prefaced with "rocksdb", so you can query them with, for example:

```
SHOW VARIABLES LIKE 'rocksdb%';
SHOW STATUS LIKE 'rocksdb%';
```

5.3.14.3 Building MyRocks in MariaDB

Contents

This page describes how to get MyRocks in MariaDB when compiling MariaDB from source.

(See https://github.com/facebook/mysql-5.6/wiki/Build-Steps for instructions how to build the upstream)

Build Process and Requirements

MariaDB compile process will compile MyRocks into `ha_rocksdb.so` by default if the platform supports it (That is, no WITH_ROCKSDB switch is necessary).

Platform requirements:

- A 64-bit platform (due to some 32 bit compilers having difficulties with RocksDB)
- git installed (or git submodules fetched somehow)
- A sufficiently recent compiler:
  - gcc >= 4.8, or
  - clang >= 3.3, or
  - MS Visual Studio 2015 or newer

Building on Ubuntu 16.04

The steps were checked on a fresh install of Ubuntu 16.04.2 LTS Xenial.

```
sudo apt-get update
sudo apt-get -y install g++ cmake libbz2-dev libaio-dev bison zlib1g-dev libsnappy-dev
sudo apt-get -y install libgflags-dev libreadline6-dev libncurses5-dev libssl-dev liblz4-dev gdb
;
```

```
git clone https://github.com/MariaDB/server.git mariadb-10.2
cd mariadb-10.2
git checkout 10.2
git submodule init
git submodule update
cmake .
make -j10
```

This should produce `storage/rocksdb/ha_rocksdb.so` which is MyRocks storage engine in the loadable form.

Starting MyRocks

MyRocks does not require any special way to initialize the data directory. Minimal my.cnf file:

```
cat > ~/my1.cnf <<EOF
[mysqld]

datadir=../mysql-test/var/install.db
plugin-dir=../storage/rocksdb
language=./share/english
socket=/tmp/mysql.sock
port=3307

plugin-load=ha_rocksdb
default-storage-engine=rocksdb
EOF
```

Run the server like this

```
(cd mysql-test; ./mtr alias)
cp -r mysql-test/var/install.db ~/data1
cd ../sql
./mysqld --defaults-file=~/my1.cnf
```

Compression libraries. Supported compression libraries are listed in rocksdb_supported_compression_types. Compiling like the above, I get:

```
Snappy,Zlib,LZ4,LZ4HC
```

5.3.14.4 Loading Data Into MyRocks

Being a write-optimized storage engine, MyRocks has special ways to load data much faster than normal INSERTs would.

See

- http://myrocks.io/docs/getting-started/; the section about "Migrating from InnoDB to MyRocks in production" has some clues.
- https://github.com/facebook/mysql-5.6/wiki/Data-Loading covers the topic in greater detail.

Note When one loads data with rocksdb_bulk_load=1 and the data conflicts with the data already in the database, one may get non-trivial errors, for example:

```
ERROR 1105 (HY000): [./.rocksdb/test.t1_PRIMARY_2_0.bulk_load.tmp] bulk load error:
  Invalid argument: External file requires flush
```

5.3.14.5 MyRocks Status Variables

Contents

21. Rocksdb_block_cachecompressed_hit
22. Rocksdb_block_cachecompressed_miss
23. Rocksdb_bloom_filter_full_positive
24. Rocksdb_bloom_filter_full_true_positive
25. Rocksdb_bloom_filter_prefix_checked
26. Rocksdb_bloom_filter_prefix_useful
27. Rocksdb_bloom_filter_useful
28. Rocksdb_bytes_read
29. Rocksdb_bytes_written
30. Rocksdb_compact_read_bytes
31. Rocksdb_compact_write_bytes
32. Rocksdb_compaction_key_drop_new
33. Rocksdb_compaction_key_drop_obsolete
34. Rocksdb_compaction_key_drop_user
35. Rocksdb_covered_secondary_key_lookups
36. Rocksdb_flush_write_bytes
37. Rocksdb_get_hit_l0
38. Rocksdb_get_hit_l1
39. Rocksdb_get_hit_l2_and_up
40. Rocksdb_getupdatessince_calls
41. Rocksdb_iter_bytes_read
42. Rocksdb_l0_num_files_stall_micros
43. Rocksdb_l0_slowdown_micros
44. Rocksdb_manual_compactions_processed
45. Rocksdb_manual_compactions_running
46. Rocksdb_memtable_compaction_micros
47. Rocksdb_memtable_hit
48. Rocksdb_memtable_miss
49. Rocksdb_memtable_total
50. Rocksdb_memtable_unflushed
51. Rocksdb_no_file_closes
52. Rocksdb_no_file_errors
53. Rocksdb_no_file_opens
54. Rocksdb_num_iterators
55. Rocksdb_number_block_not_compressed
56. Rocksdb_number_db_next
57. Rocksdb_number_db_next_found
58. Rocksdb_number_db_prev
59. Rocksdb_number_db_prev_found
60. Rocksdb_number_db_seek
61. Rocksdb_number_db_seek_found
62. Rocksdb_number_deletes_filtered
63. Rocksdb_number_keys_read
64. Rocksdb_number_keys_updated
65. Rocksdb_number_keys_written
66. Rocksdb_number_merge_failures
67. Rocksdb_number_multiget_bytes_read
68. Rocksdb_number_multiget_get
69. Rocksdb_number_multiget_keys_read
70. Rocksdb_number_reseeks_iteration
71. Rocksdb_number_sst_entry_delete
72. Rocksdb_number_sst_entry_merge
73. Rocksdb_number_sst_entry_other
74. Rocksdb_number_sst_entry_put
75. Rocksdb_number_sst_entry_singledelete
76. Rocksdb_number_superversion_acquires
77. Rocksdb_number_superversion_cleanups
78. Rocksdb_number_superversion_releases
79. Rocksdb_queries_point
80. Rocksdb_queries_range
81. Rocksdb_row_lock_deadlocks
82. Rocksdb_row_lock_wait_timeouts
83. Rocksdb_rows_deleted
84. Rocksdb_rows_deleted_blind
85. Rocksdb_rows_expired
86. Rocksdb_rows_filtered
87. Rocksdb_rows_inserted
88. Rocksdb_rows_read

89. Rocksdb_rows_updated
90. Rocksdb_snapshot_conflict_errors
91. Rocksdb_stall_l0_file_count_limit_slowdowns
92. Rocksdb_stall_l0_file_count_limit_stops
93. Rocksdb_stall_locked_l0_file_count_limit_slowdowns
94. Rocksdb_stall_locked_l0_file_count_limit_stops
95. Rocksdb_stall_memtable_limit_slowdowns
96. Rocksdb_stall_memtable_limit_stops
97. Rocksdb_stall_micros
98. Rocksdb_stall_pending_compaction_limit_slowdowns
99. Rocksdb_stall_pending_compaction_limit_stops
100. Rocksdb_stall_total_slowdowns
101. Rocksdb_stall_total_stops
102. Rocksdb_system_rows_deleted
103. Rocksdb_system_rows_inserted
104. Rocksdb_system_rows_read
105. Rocksdb_system_rows_updated
106. Rocksdb_wal_bytes
107. Rocksdb_wal_group_syncs
108. Rocksdb_wal_synced
109. Rocksdb_write_other
110. Rocksdb_write_self
111. Rocksdb_write_timedout
112. Rocksdb_write_wal

This page documents status variables related to the MyRocks storage engine. See Server Status Variables for a complete list of status variables that can be viewed with SHOW STATUS.

See also the Full list of MariaDB options, system and status variables.

`Rocksdb_block_cache_add`

- **Description:** Number of blocks added to the Block Cache.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Rocksdb_block_cache_add_failures`

- **Description:** Number of failures when adding blocks to Block Cache.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.15, MariaDB 10.3.7

`Rocksdb_block_cache_bytes_read`

- **Description:** Bytes read from Block Cache.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.15, MariaDB 10.3.7

`Rocksdb_block_cache_bytes_write`

- **Description:** Bytes written to Block Cache.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.15, MariaDB 10.3.7

`Rocksdb_block_cache_data_add`

- **Description:** Number of data blocks added to the Block Cache.
- **Scope:** Global, Session
- **Data Type:** `numeric`

- **Introduced:** MariaDB 10.2.15, MariaDB 10.3.7

Rocksdb_block_cache_data_bytes_insert

- **Description:** Bytes added to the Block Cache.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.15, MariaDB 10.3.7

Rocksdb_block_cache_data_hit

- **Description:** Number of hits when accessing the data block from the Block Cache.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_block_cache_data_miss

- **Description:** Number of misses when accessing the data block from the Block Cache.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_block_cache_filter_add

- **Description:** Number of bloom filter blocks added to the Block Cache.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.15, MariaDB 10.3.7

Rocksdb_block_cache_filter_bytes_evict

- **Description:** Bytes of bloom filter blocks evicted from the Block Cache.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.15, MariaDB 10.3.7

Rocksdb_block_cache_filter_bytes_insert

- **Description:** Bytes of bloom filter blocks added to the Block Cache.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.15, MariaDB 10.3.7

Rocksdb_block_cache_filter_hit

- **Description:** Number of hits when accessing the filter block from the Block Cache.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_block_cache_filter_miss

- **Description:** Number of misses when accessing the filter block from the Block Cache.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_block_cache_hit

- **Description:** Total number of hits for the Block Cache.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Rocksdb_block_cache_index_add`

- **Description:** Number of index blocks added to Block Cache index.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.15, MariaDB 10.3.7

`Rocksdb_block_cache_index_bytes_evict`

- **Description:** Bytes of index blocks evicted from the Block Cache.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.15, MariaDB 10.3.7

`Rocksdb_block_cache_index_bytes_insert`

- **Description:** Bytes of index blocks added to the Block Cache.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.15, MariaDB 10.3.7

`Rocksdb_block_cache_index_hit`

- **Description:** Number of hits for the Block Cache index.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Rocksdb_block_cache_index_miss`

- **Description:** Number of misses for the Block Cache index.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Rocksdb_block_cache_miss`

- **Description:** Total number of misses for the Block Cache.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Rocksdb_block_cachecompressed_hit`

- **Description:** Number of hits for the compressed Block Cache.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Rocksdb_block_cachecompressed_miss`

- **Description:** Number of misses for the compressed Block Cache.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_bloom_filter_full_positive

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.18, MariaDB 10.3.10

Rocksdb_bloom_filter_full_true_positive

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.18, MariaDB 10.3.10

Rocksdb_bloom_filter_prefix_checked

- **Description:** Number of times the Bloom Filter checked before creating an iterator on a file.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_bloom_filter_prefix_useful

- **Description:** Number of times the Bloom Filter check used to avoid creating an iterator on a file.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_bloom_filter_useful

- **Description:** Number of times the Bloom Filter used instead of reading form file.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_bytes_read

- **Description:** Total number of uncompressed bytes read from memtables, cache or table files.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_bytes_written

- **Description:** Total number of uncompressed bytes written.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_compact_read_bytes

- **Description:** Number of bytes read during compaction.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_compact_write_bytes

- **Description:** Number of bytes written during compaction.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_compaction_key_drop_new

- **Description:** Number of keys dropped during compaction due their being overwritten by new values.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_compaction_key_drop_obsolete

- **Description:** Number of keys dropped during compaction due to their being obsolete.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_compaction_key_drop_user

- **Description:** Number of keys dropped during compaction due to user compaction.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_covered_secondary_key_lookups

- **Description:** Incremented when avoiding reading a record via a keyread. This indicates lookups that were performed via a secondary index containing a field that is only a prefix of the VARCHAR column, and that could return all requested fields directly from the secondary index.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_flush_write_bytes

- **Description:** Number of bytes written during flush.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_get_hit_l0

- **Description:** Number of times reads got data from the L0 compaction layer.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.15, MariaDB 10.3.7

Rocksdb_get_hit_l1

- **Description:** Number of times reads got data from the L1 compaction layer.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.15, MariaDB 10.3.7

Rocksdb_get_hit_l2_and_up

- **Description:** Number of times reads got data from the L2 and up compaction layer.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.15, MariaDB 10.3.7

Rocksdb_getupdatessince_calls

- **Description:** Number of calls to the `GetUpdatesSince` function. You may find this useful when monitoring refreshes of the transaction log.

- **Scope:** Global, Session
- **Data Type:** `numeric`

`Rocksdb_iter_bytes_read`

- **Description:** Total uncompressed bytes read from an iterator, including the size of both key and value.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.15, MariaDB 10.3.7

`Rocksdb_l0_num_files_stall_micros`

- **Description:** Shows how long in microseconds throttled due to too mnay files in L0.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Removed:** MariaDB 10.3.1, MariaDB 10.2.8

`Rocksdb_l0_slowdown_micros`

- **Description:** Total time spent waiting in microseconds while performing L0-L1 compactions.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Removed:** MariaDB 10.3.1, MariaDB 10.2.8

`Rocksdb_manual_compactions_processed`

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.18, MariaDB 10.3.10

`Rocksdb_manual_compactions_running`

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.18, MariaDB 10.3.10

`Rocksdb_memtable_compaction_micros`

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Removed:** MariaDB 10.3.1, MariaDB 10.2.8

`Rocksdb_memtable_hit`

- **Description:** Number of memtable hits.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Rocksdb_memtable_miss`

- **Description:** Number of memtable misses.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Rocksdb_memtable_total`

- **Description:** Memory used, in bytes, of all memtables.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Rocksdb_memtable_unflushed`

- **Description:** Memory used, in bytes, of all unflushed memtables.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Rocksdb_no_file_closes`

- **Description:** Number of times files were closed.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Rocksdb_no_file_errors`

- **Description:** Number of errors encountered while trying to read data from an SST file.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Rocksdb_no_file_opens`

- **Description:** Number of times files were opened.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Rocksdb_num_iterators`

- **Description:** Number of iterators currently open.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Rocksdb_number_block_not_compressed`

- **Description:** Number of uncompressed blocks.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Rocksdb_number_db_next`

- **Description:** Number of `next` calls.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.15, MariaDB 10.3.7

`Rocksdb_number_db_next_found`

- **Description:** Number of `next` calls that returned data.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.15, MariaDB 10.3.7

Rocksdb_number_db_prev

- **Description:** Number of `prev` calls.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.15, MariaDB 10.3.7

Rocksdb_number_db_prev_found

- **Description:** Number of `prev` calls that returned data.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.15, MariaDB 10.3.7

Rocksdb_number_db_seek

- **Description:** Number of `seek` calls.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.15, MariaDB 10.3.7

Rocksdb_number_db_seek_found

- **Description:** Number of `seek` calls that returned data.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.15, MariaDB 10.3.7

Rocksdb_number_deletes_filtered

- **Description:** Number of deleted records were not written to storage due to a nonexistent key.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_number_keys_read

- **Description:** Number of keys have been read.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_number_keys_updated

- **Description:** Number of keys have been updated.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_number_keys_written

- **Description:** Number of keys have been written.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_number_merge_failures

- **Description:** Number of failures encountered while performing merge operator actions.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_number_multiget_bytes_read

- **Description:** Number of bytes read during RocksDB `MultiGet()` calls.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_number_multiget_get

- **Description:** Number of RocksDB `MultiGet()` requests made.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_number_multiget_keys_read

- **Description:** Number of keys read through RocksDB `MultiGet()` calls.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_number_reseeks_iteration

- **Description:** Number of reseeks that have occurred inside an iteration that skipped over a large number of keys with the same user key.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_number_sst_entry_delete

- **Description:** Number of delete markers written.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_number_sst_entry_merge

- **Description:** Number of merge keys written.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_number_sst_entry_other

- **Description:** Number of keys written that are not delete, merge or put keys.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_number_sst_entry_put

- **Description:** Number of put keys written.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_number_sst_entry_singledelete

- **Description:** Number of single-delete keys written.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_number_superversion_acquires

- **Description:** Number of times the superversion structure acquired. This is useful when tracking files for the database.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_number_superversion_cleanups

- **Description:** Number of times the superversion structure performed cleanups.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_number_superversion_releases

- **Description:** Number of times the superversion structure released.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_queries_point

- **Description:** Number of single-row queries.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_queries_range

- **Description:** Number of multi-row queries.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_row_lock_deadlocks

- **Description:** Number of deadlocks.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.15, MariaDB 10.3.7

Rocksdb_row_lock_wait_timeouts

- **Description:** Number of row lock wait timeouts.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.15, MariaDB 10.3.7

Rocksdb_rows_deleted

- **Description:** Number of rows deleted.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_rows_deleted_blind

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_rows_expired

- **Description:** Number of expired rows.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_rows_filtered

- **Description:** Number of TTL filtered rows.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.15, MariaDB 10.3.7

Rocksdb_rows_inserted

- **Description:** Number of rows inserted.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_rows_read

- **Description:** Number of rows read.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_rows_updated

- **Description:** Number of rows updated.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_snapshot_conflict_errors

- **Description:** Number of snapshot conflict errors that have occurred during transactions that forced a rollback.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_stall_l0_file_count_limit_slowdowns

- **Description:** Write slowdowns due to L0 being near to full.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_stall_l0_file_count_limit_stops

- **Description:** Write stops due to L0 being to full.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_stall_locked_l0_file_count_limit_slowdowns

- **Description:** Write slowdowns due to L0 being near to full and L0 compaction in progress.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_stall_locked_l0_file_count_limit_stops

- **Description:** Write stops due to L0 being full and L0 compaction in progress.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_stall_memtable_limit_slowdowns

- **Description:** Write slowdowns due to approaching maximum permitted number of memtables.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.10, MariaDB 10.3.3

Rocksdb_stall_memtable_limit_stops

- **Description:** * **Description:** Write stops due to reaching maximum permitted number of memtables.
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.2.10, MariaDB 10.3.3

Rocksdb_stall_micros

- **Description:** Time in microseconds that the writer had to wait for the compaction or flush to complete.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_stall_pending_compaction_limit_slowdowns

- **Description:** Write slowdowns due to nearing the limit for the maximum number of pending compaction bytes.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_stall_pending_compaction_limit_stops

- **Description:** Write stops due to reaching the limit for the maximum number of pending compaction bytes.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_stall_total_slowdowns

- **Description:** Total number of write slowdowns.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_stall_total_stops

- **Description:** Total number of write stops.
- **Scope:** Global, Session
- **Data Type:** `numeric`

Rocksdb_system_rows_deleted

- **Description:** Number of rows deleted from system tables.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Rocksdb_system_rows_inserted`

- **Description:** Number of rows inserted into system tables.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Rocksdb_system_rows_read`

- **Description:** Number of rows read from system tables.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Rocksdb_system_rows_updated`

- **Description:** Number of rows updated for system tables.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Rocksdb_wal_bytes`

- **Description:** Number of bytes written to WAL.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Rocksdb_wal_group_syncs`

- **Description:** Number of group commit WAL file syncs have occurred. This is provided by MyRocks and is not a view of a RocksDB counter. Increased in `rocksdb_flush_wal()` when doing the `rdb->FlushWAL()` call.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Rocksdb_wal_synced`

- **Description:** Number of syncs made on RocksDB WAL file.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Rocksdb_write_other`

- **Description:** Number of writes processed by a thread other than the requesting thread.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Rocksdb_write_self`

- **Description:** Number of writes processed by requesting thread.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Rocksdb_write_timedout`

- **Description:** Number of writes that timed out.
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Rocksdb_write_wal`

- **Description:** Number of write calls that requested WAL.
- **Scope:** Global, Session
- **Data Type:** `numeric`

5.3.14.6 MyRocks System Variables

Contents

56. rocksdb_enable_ttl_read_filtering
57. rocksdb_enable_write_thread_adaptive_yield
58. rocksdb_error_if_exists
59. rocksdb_error_on_suboptimal_collation
60. rocksdb_flush_log_at_trx_commit
61. rocksdb_flush_memtable_on_analyze
62. rocksdb_force_compute_memtable_stats
63. rocksdb_force_compute_memtable_stats_cachetime
64. rocksdb_force_flush_memtable_and_lzero_now
65. rocksdb_force_flush_memtable_now
66. rocksdb_force_index_records_in_range
67. rocksdb_git_hash
68. rocksdb_hash_index_allow_collision
69. rocksdb_ignore_unknown_options
70. rocksdb_index_type
71. rocksdb_info_log_level
72. rocksdb_io_write_timeout
73. rocksdb_is_fd_close_on_exec
74. rocksdb_keep_log_file_num
75. rocksdb_large_prefix
76. rocksdb_lock_scanned_rows
77. rocksdb_lock_wait_timeout
78. rocksdb_log_dir
79. rocksdb_log_file_time_to_roll
80. rocksdb_manifest_preallocation_size
81. rocksdb_manual_compaction_threads
82. rocksdb_manual_wal_flush
83. rocksdb_master_skip_tx_api
84. rocksdb_max_background_compactions
85. rocksdb_max_background_flushes
86. rocksdb_max_background_jobs
87. rocksdb_max_latest_deadlocks
88. rocksdb_max_log_file_size
89. rocksdb_max_manifest_file_size
90. rocksdb_max_manual_compactions
91. rocksdb_max_open_files
92. rocksdb_max_row_locks
93. rocksdb_max_subcompactions
94. rocksdb_max_total_wal_size
95. rocksdb_merge_buf_size
96. rocksdb_merge_combine_read_size
97. rocksdb_merge_tmp_file_removal_delay_ms
98. rocksdb_new_table_reader_for_compaction_inputs
99. rocksdb_no_block_cache
100. rocksdb_override_cf_options
101. rocksdb_paranoid_checks
102. rocksdb_pause_background_work
103. rocksdb_perf_context_level
104. rocksdb_persistent_cache_path
105. rocksdb_persistent_cache_size_mb
106. rocksdb_pin_l0_filter_and_index_blocks_in_cache
107. rocksdb_print_snapshot_conflict_queries
108. rocksdb_rate_limiter_bytes_per_sec
109. rocksdb_read_free_rpl_tables
110. rocksdb_records_in_range
111. rocksdb_remove_mariabackup_checkpoint
112. rocksdb_reset_stats
113. rocksdb_rollback_on_timeout
114. rocksdb_seconds_between_stat_computes
115. rocksdb_signal_drop_index_thread
116. rocksdb_sim_cache_size
117. rocksdb_skip_bloom_filter_on_read
118. rocksdb_skip_fill_cache
119. rocksdb_skip_unique_check_tables
120. rocksdb_sst_mgr_rate_bytes_per_sec
121. rocksdb_stats_dump_period_sec
122. rocksdb_stats_level
123. rocksdb_stats_recalc_rate

This page documents system variables related to the MyRocks storage engine. See Server System Variables for a complete list of system variables and instructions on setting them.

See also the Full list of MariaDB options, system and status variables.

`rocksdb_access_hint_on_compaction_start`

- **Description:** DBOptions::access_hint_on_compaction_start for RocksDB. Specifies the file access pattern, applied to all input files, once a compaction starts.
- **Commandline:** `--rocksdb-access-hint-on-compaction-start=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `1`
- **Range:** `0` to `3`

`rocksdb_advise_random_on_open`

- **Description:** DBOptions::advise_random_on_open for RocksDB.
- **Commandline:** `--rocksdb-advise-random-on-open={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `ON`

`rocksdb_allow_concurrent_memtable_write`

- **Description:** DBOptions::allow_concurrent_memtable_write for RocksDB.
- **Commandline:** `--rocksdb-allow-concurrent-memtable-write={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

rocksdb_allow_mmap_reads

- **Description:** DBOptions::allow_mmap_reads for RocksDB
- **Commandline:** `--rocksdb-allow-mmap-reads={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

rocksdb_allow_mmap_writes

- **Description:** DBOptions::allow_mmap_writes for RocksDB
- **Commandline:** `--rocksdb-allow-mmap-writes={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

rocksdb_allow_to_start_after_corruption

- **Description:** Allow server still to start successfully even if RocksDB corruption is detected.
- **Commandline:** `--rocksdb-allow-to-start-after-corruption={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB 10.3.7, MariaDB 10.2.15

---

rocksdb_background_sync

- **Description:** Turns on background syncs for RocksDB
- **Commandline:** `--rocksdb-background-sync={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Removed:** MariaDB 10.3.1, MariaDB 10.2.8

---

rocksdb_base_background_compactions

- **Description:** DBOptions::base_background_compactions for RocksDB
- **Commandline:** `--rocksdb-base-background-compactions=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `1`
- **Range:** `-1` to `64`
- **Removed:** MariaDB 10.3.1, MariaDB 10.2.8

---

rocksdb_blind_delete_primary_key

- **Description:** Deleting rows by primary key lookup, without reading rows (Blind Deletes). Blind delete is disabled if the table has secondary key.
- **Commandline:** `--rocksdb-blind-delete-primary-key={0|1}`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

rocksdb_block_cache_size

- **Description:** Block_cache size for RocksDB (block size 1024)
- **Commandline:** `--rocksdb-block-cache-size=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `536870912`
- **Range:** `1024` to `9223372036854775807`

To see the statistics of block cache usage, check `SHOW ENGINE ROCKSDB STATUS` output (search for lines starting with `rocksdb.block.cache`).

One can check the size of data of the block cache in `DB_BLOCK_CACHE_USAGE` column of the `INFORMATION_SCHEMA.ROCKSDB_DBSTATS` table.

---

rocksdb_block_restart_interval

- **Description:** BlockBasedTableOptions::block_restart_interval for RocksDB
- **Commandline:** `--rocksdb-block-restart-interval=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `16`
- **Range:** `1` to `2147483647`

---

rocksdb_block_size

- **Description:** BlockBasedTableOptions::block_size for RocksDB
- **Commandline:** `--rocksdb-block-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `4096`
- **Range:** `1` to `18446744073709551615`

---

rocksdb_block_size_deviation

- **Description:** BlockBasedTableOptions::block_size_deviation for RocksDB
- **Commandline:** `--rocksdb-block-size-deviation=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `10`
- **Range:** `0` to `2147483647`

---

rocksdb_bulk_load

- **Description:** Use bulk-load mode for inserts. This disables unique_checks and enables rocksdb_commit_in_the_middle.
- **Commandline:** `--rocksdb-bulk-load={0|1}`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

rocksdb_bulk_load_allow_sk

- **Description:** Allow bulk loading of sk keys during bulk-load. Can be changed only when bulk load is disabled.
- **Commandline:** `--rocksdb-bulk-load_allow_sk={0|1}`
- **Scope:** Global, Session

- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB 10.3.10, MariaDB 10.2.18

---

rocksdb_bulk_load_allow_unsorted

- **Description:** Allow unsorted input during bulk-load. Can be changed only when bulk load is disabled.
- **Commandline:** `--rocksdb-bulk-load_allow_unsorted={0|1}`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

rocksdb_bulk_load_size

- **Description:** Maximum number of records in a batch for bulk-load mode.
- **Commandline:** `--rocksdb-bulk-load-size=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1000`
- **Range:** `1` to `1073741824`

---

rocksdb_bytes_per_sync

- **Description:** DBOptions::bytes_per_sync for RocksDB.
- **Commandline:** `--rocksdb-bytes-per-sync=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `18446744073709551615`

---

rocksdb_cache_dump

- **Description:** Include RocksDB block cache content in core dump.
- **Commandline:** `--rocksdb-cache-dump={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `ON`
- **Introduced:** MariaDB 10.4.7, MariaDB 10.3.17, MariaDB 10.2.26

---

rocksdb_cache_high_pri_pool_ratio

- **Description:** Specify the size of block cache high-pri pool.
- **Commandline:** `--rocksdb-cache-high-pri-pool-ratio=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `double`
- **Default Value:** `0.000000`
- **Range:** `0` to `1`
- **Introduced:** MariaDB 10.4.7, MariaDB 10.3.17, MariaDB 10.2.26

---

rocksdb_cache_index_and_filter_blocks

- **Description:** BlockBasedTableOptions::cache_index_and_filter_blocks for RocksDB.
- **Commandline:** `--rocksdb-cache-index-and-filter-blocks={0|1}`

- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `ON`

---

rocksdb_cache_index_and_filter_with_high_priority

- **Description:** cache_index_and_filter_blocks_with_high_priority for RocksDB.
- **Commandline:** `--rocksdb-cache-index-and-filter-with-high-priority={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `ON`
- **Introduced:** MariaDB 10.4.7, MariaDB 10.3.17, MariaDB 10.2.26

---

rocksdb_checksums_pct

- **Description:** Percentage of rows to be checksummed.
- **Commandline:** `--rocksdb-checksums-pct=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `100`
- **Range:** `0` to `100`

---

rocksdb_collect_sst_properties

- **Description:** Enables collecting SST file properties on each flush.
- **Commandline:** `--rocksdb-collect-sst-properties={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `ON`

---

rocksdb_commit_in_the_middle

- **Description:** Commit rows implicitly every rocksdb_bulk_load_size, on bulk load/insert, update and delete.
- **Commandline:** `--rocksdb-commit-in-the-middle={0|1}`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

rocksdb_commit_time_batch_for_recovery

- **Description:** TransactionOptions::commit_time_batch_for_recovery for RocksDB.
- **Commandline:** `--rocksdb-commit-time-batch-for-recovery={0|1}`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB 10.3.10, MariaDB 10.2.18

---

rocksdb_compact_cf

- **Description:** Compact column family.
- **Commandline:** `--rocksdb-compact-cf=value`
- **Scope:** Global
- **Dynamic:** Yes

- **Data Type:** `string`
- **Default Value:** (Empty)

`rocksdb_compaction_readahead_size`

- **Description:** DBOptions::compaction_readahead_size for RocksDB.
- **Commandline:** `--rocksdb-compaction-readahead-size=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `18446744073709551615`

`rocksdb_compaction_sequential_deletes`

- **Description:** RocksDB will trigger compaction for the file if it has more than this number sequential deletes per window.
- **Commandline:** `--rocksdb-compaction-sequential-deletes=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `2000000`

`rocksdb_compaction_sequential_deletes_count_sd`

- **Description:** Counting SingleDelete as rocksdb_compaction_sequential_deletes.
- **Commandline:** `--rocksdb-compaction-sequential-deletes-count-sd={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

`rocksdb_compaction_sequential_deletes_file_size`

- **Description:** Minimum file size required for compaction_sequential_deletes.
- **Commandline:** `--rocksdb-compaction-sequential-deletes-file-size=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `-1` to `9223372036854775807`

`rocksdb_compaction_sequential_deletes_window`

- **Description:** Size of the window for counting rocksdb_compaction_sequential_deletes.
- **Commandline:** `--rocksdb-compaction-sequential-deletes-window=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `2000000`

`rocksdb_concurrent_prepare`

- **Description:** DBOptions::concurrent_prepare for RocksDB.
- **Commandline:** `--rocksdb-coconcurrent-prepare={0|1}`
- **Scope:** Global
- **Dynamic:** No

- **Data Type:** `boolean`
- **Default Value:** `1`
- **Removed:** MariaDB 10.3.7, MariaDB 10.2.15

---

rocksdb_create_checkpoint

- **Description:** Checkpoint directory.
- **Commandline:** `--rocksdb-create-checkpoint=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** (Empty)

---

rocksdb_create_if_missing

- **Description:** DBOptions::create_if_missing for RocksDB.
- **Commandline:** `--rocksdb-create-if-missing={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `ON`

---

rocksdb_create_missing_column_families

- **Description:** DBOptions::create_missing_column_families for RocksDB.
- **Commandline:** `--rocksdb-create-missing-column-families={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

rocksdb_datadir

- **Description:** RocksDB data directory.
- **Commandline:** `--rocksdb-datadir[=value]`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** `./#rocksdb`

---

rocksdb_db_write_buffer_size

- **Description:** DBOptions::db_write_buffer_size for RocksDB.
- **Commandline:** `--rocksdb-db-write-buffer-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `18446744073709551615`

---

rocksdb_deadlock_detect

- **Description:** Enables deadlock detection.
- **Commandline:** `--rocksdb-deadlock-detect={0|1}`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

rocksdb_deadlock_detect_depth

- **Description:** Number of transactions deadlock detection will traverse through before assuming deadlock.
- **Commandline:** `--rocksdb-deadlock-detect-depth=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `50`
- **Range:** `2` to `18446744073709551615`

---

rocksdb_debug_manual_compaction_delay

- **Description:** For debugging purposes only. Sleeping specified seconds for simulating long running compactions.
- **Commandline:** `--rocksdb-debug_manual_compaction_delay=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `4294967295`
- **Introduced:** MariaDB 10.3.10, MariaDB 10.2.18

---

rocksdb_debug_optimizer_no_zero_cardinality

- **Description:** If cardinality is zero, override it with some value.
- **Commandline:** `--rocksdb-debug-optimizer-no-zero-cardinality={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `ON`

---

rocksdb_debug_ttl_ignore_pk

- **Description:** For debugging purposes only. If true, compaction filtering will not occur on PK TTL data. This variable is a no-op in non-debug builds.
- **Commandline:** `--rocksdb-debug-ttl-ignore-pk={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

rocksdb_debug_ttl_read_filter_ts

- **Description:** For debugging purposes only. Overrides the TTL read filtering time to time + debug_ttl_read_filter_ts. A value of 0 denotes that the variable is not set. This variable is a no-op in non-debug builds.
- **Commandline:** `--rocksdb-debug-ttl-read-filter-ts=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `-3600` to `3600`
- **Introduced:** MariaDB 10.3.1, MariaDB 10.2.8

---

rocksdb_debug_ttl_rec_ts

- **Description:** For debugging purposes only. Overrides the TTL of records to now() + debug_ttl_rec_ts. The value can be +/- to simulate a record inserted in the past vs a record inserted in the 'future'. A value of 0 denotes that the variable is not set. This variable is a no-op in non-debug builds.
- **Commandline:** `--rocksdb-debug-ttl-read-filter-ts=#`
- **Scope:** Global
- **Dynamic:** Yes

- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `-3600` to `3600`
- **Introduced:** MariaDB 10.3.1, MariaDB 10.2.8

---

rocksdb_debug_ttl_snapshot_ts

- **Description:** For debugging purposes only. Sets the snapshot during compaction to now() + debug_set_ttl_snapshot_ts. The value can be positive or negative to simulate a snapshot in the past vs a snapshot created in the 'future'. A value of 0 denotes that the variable is not set. This variable is a no-op in non-debug builds.
- **Commandline:** `--rocksdb-debug-ttl-snapshot-ts=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `-3600` to `3600`
- **Introduced:** MariaDB 10.3.1, MariaDB 10.2.8

---

rocksdb_default_cf_options

- **Description:** Default cf options for RocksDB.
- **Commandline:** `--rocksdb-default-cf-options=value`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** (Empty)

---

rocksdb_delayed_write_rate

- **Description:** DBOptions::delayed_write_rate.
- **Commandline:** `--rocksdb-delayed-write-rate=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0` (Previously `16777216`)
- **Range:** `0` to `18446744073709551615`

---

rocksdb_delete_cf

- **Description:** Delete column family.
- **Commandline:** `--rocksdb-delete-cf=val`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** (Empty string)
- **Introduced:** MariaDB 10.4.7, MariaDB 10.3.17, MariaDB 10.2.26

---

rocksdb_delete_obsolete_files_period_micros

- **Description:** DBOptions::delete_obsolete_files_period_micros for RocksDB.
- **Commandline:** `--rocksdb-delete-obsolete-files-period-micros=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `21600000000`
- **Range:** `0` to `9223372036854775807`

---

rocksdb_enable_2pc

- **Description:** Enable two phase commit for MyRocks. When set, MyRocks will keep its data consistent with the binary log (in other words, the server will be a crash-safe master). The consistency is achieved by doing two-phase XA commit with the binary log.
- **Commandline:** `--rocksdb-enable-2pc={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `ON`

---

rocksdb_enable_bulk_load_api

- **Description:** Enables using SstFileWriter for bulk loading.
- **Commandline:** `--rocksdb-enable-bulk-load-api={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `ON`

---

rocksdb_enable_insert_with_update_caching

- **Description:** Whether to enable optimization where we cache the read from a failed insertion attempt in INSERT ON DUPLICATE KEY UPDATE.
- **Commandline:** `--rocksdb-enable-insert-with-update-caching={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `ON`
- **Introduced:** MariaDB 10.4.7, MariaDB 10.3.17, MariaDB 10.2.26

---

rocksdb_enable_thread_tracking

- **Description:** DBOptions::enable_thread_tracking for RocksDB.
- **Commandline:** `--rocksdb-enable-thread-tracking={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

rocksdb_enable_ttl

- **Description:** Enable expired TTL records to be dropped during compaction.
- **Commandline:** `--rocksdb-enable-ttl={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `ON`
- **Introduced:** MariaDB 10.3.1, MariaDB 10.2.8

---

rocksdb_enable_ttl_read_filtering

- **Description:** For tables with TTL, expired records are skipped/filtered out during processing and in query results. Disabling this will allow these records to be seen, but as a result rows may disappear in the middle of transactions as they are dropped during compaction. Use with caution.
- **Commandline:** `--rocksdb-enable-ttl-read-filtering={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `ON`
- **Introduced:** MariaDB 10.3.1, MariaDB 10.2.8

---

rocksdb_enable_write_thread_adaptive_yield

- **Description:** DBOptions::enable_write_thread_adaptive_yield for RocksDB.
- **Commandline:** `--rocksdb-enable-write-thread-adaptive-yield={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

rocksdb_error_if_exists

- **Description:** DBOptions::error_if_exists for RocksDBB.
- **Commandline:** `--rocksdb-error-if-exists={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

rocksdb_error_on_suboptimal_collation

- **Description:** Raise an error instead of warning if a sub-optimal collation is used.
- **Commandline:** `--rocksdb-error-on-suboptimal-collation={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `ON`
- **Introduced:** MariaDB 10.3.10, MariaDB 10.2.18

---

rocksdb_flush_log_at_trx_commit

- **Description:** Sync on transaction commit. Similar to innodb_flush_log_at_trx_commit. One can check the flushing by examining the rocksdb_wal_synced and rocksdb_wal_bytes status variables.
  - **1**: Always sync on commit (the default).
  - **0**: Never sync.
  - **2**: Sync based on a timer controlled via rocksdb-background-sync.
- **Commandline:** `--rocksdb-flush-log-at-trx-commit=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1`
- **Range:** `0` to `2`

---

rocksdb_flush_memtable_on_analyze

- **Description:** Forces memtable flush on ANALZYE table to get accurate cardinality.
- **Commandline:** `--rocksdb-flush-memtable-on-analyze={0|1}`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `ON`
- **Removed:** MariaDB 10.3.7, MariaDB 10.2.15

---

rocksdb_force_compute_memtable_stats

- **Description:** Force to always compute memtable stats.
- **Commandline:** `--rocksdb-force-compute-memtable-stats={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `ON`

rocksdb_force_compute_memtable_stats_cachetime

- **Description:** Time in usecs to cache memtable estimates.
- **Commandline:** `--rocksdb-force-compute-memtable-stats-cachetime=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `60000000`
- **Range:** `0` to `2147483647`

rocksdb_force_flush_memtable_and_lzero_now

- **Description:** Acts similar to force_flush_memtable_now, but also compacts all L0 files.
- **Commandline:** `--rocksdb-force-flush-memtable-and-lzero-now={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB 10.3.1, MariaDB 10.2.8

rocksdb_force_flush_memtable_now

- **Description:** Forces memstore flush which may block all write requests so be careful.
- **Commandline:** `--rocksdb-force-flush-memtable-now={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

rocksdb_force_index_records_in_range

- **Description:** Used to override the result of records_in_range() when FORCE INDEX is used.
- **Commandline:** `--rocksdb-force-index-records-in-range=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `2147483647`

rocksdb_git_hash

- **Description:** Git revision of the RocksDB library used by MyRocks.
- **Commandline:** `--rocksdb-git-hash=value=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** As per git revision.

rocksdb_hash_index_allow_collision

- **Description:** BlockBasedTableOptions::hash_index_allow_collision for RocksDB.
- **Commandline:** `--rocksdb-hash-index-allow-collision={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `ON`

rocksdb_ignore_unknown_options

- **Description:** Enable ignoring unknown options passed to RocksDB.
- **Commandline:** `--rocksdb-ignore-unknown-options={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `ON`
- **Introduced:** MariaDB 10.3.7, MariaDB 10.2.15

---

rocksdb_index_type

- **Description:** BlockBasedTableOptions::index_type for RocksDB.
- **Commandline:** `--rocksdb-index-type=value`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `enum`
- **Default Value:** `kBinarySearch`
- **Valid Values:** `kBinarySearch`, `kHashSearch`

---

rocksdb_info_log_level

- **Description:** Filter level for info logs to be written mysqld error log. Valid values include 'debug_level', 'info_level', 'warn_level', 'error_level' and 'fatal_level'.
- **Commandline:** `--rocksdb-info-log-level=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `enum`
- **Default Value:** `error_level`
- **Valid Values:** `error_level`, `debug_level`, `info_level`, `warn_level`, `fatal_level`

---

rocksdb_io_write_timeout

- **Description:** Timeout for experimental I/O watchdog.
- **Commandline:** `--rocksdb-io-write-timeout=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Valid Values:** `0` to `4294967295`
- **Introduced:** MariaDB 10.3.1, MariaDB 10.2.8

---

rocksdb_is_fd_close_on_exec

- **Description:** DBOptions::is_fd_close_on_exec for RocksDB.
- **Commandline:** `--rocksdb-is-fd-close-on-exec={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `ON`

---

rocksdb_keep_log_file_num

- **Description:** DBOptions::keep_log_file_num for RocksDB.
- **Commandline:** `--rocksdb-keep-log-file-num=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `1000`
- **Range:** `0` to `18446744073709551615`

rocksdb_large_prefix

- **Description:** Support large index prefix length of 3072 bytes. If off, the maximum index prefix length is 767.
- **Commandline:** `--rocksdb-large_prefix={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

rocksdb_lock_scanned_rows

- **Description:** Take and hold locks on rows that are scanned but not updated.
- **Commandline:** `--rocksdb-lock-scanned-rows={0|1}`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

rocksdb_lock_wait_timeout

- **Description:** Number of seconds to wait for lock.
- **Commandline:** `--rocksdb-lock-wait-timeout=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1`
- **Range:** `1` to `1073741824`

rocksdb_log_dir

- **Description:** DBOptions::log_dir for RocksDB. Where the log files are stored. An empty value implies `rocksdb_datadir` is used as the directory.
- **Commandline:** `--rocksdb-log-dir=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** (Empty)
- **Introduced:** MariaDB 10.9.1

rocksdb_log_file_time_to_roll

- **Description:** DBOptions::log_file_time_to_roll for RocksDB.
- **Commandline:** `--rocksdb-log-file-time-to_roll=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `18446744073709551615`

rocksdb_manifest_preallocation_size

- **Description:** DBOptions::manifest_preallocation_size for RocksDB.
- **Commandline:** `--rocksdb-manifest-preallocation-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `4194304`
- **Range:** `0` to `18446744073709551615`

rocksdb_manual_compaction_threads

- **Description:** How many rocksdb threads to run for manual compactions.
- **Commandline:** `--rocksdb-manual-compation-threads=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `128`
- **Introduced:** MariaDB 10.3.10, MariaDB 10.2.18

---

rocksdb_manual_wal_flush

- **Description:** DBOptions::manual_wal_flush for RocksDB.
- **Commandline:** `--rocksdb-manual-wal-flush={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `ON`

---

rocksdb_master_skip_tx_api

- **Description:** Skipping holding any lock on row access. Not effective on slave.
- **Commandline:** `--rocksdb-master-skip-tx-api={0|1}`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

rocksdb_max_background_compactions

- **Description:** DBOptions::max_background_compactions for RocksDB.
- **Commandline:** `--rocksdb-max-background-compactions=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1`
- **Range:** `1` to `64`
- **Removed:** MariaDB 10.3.1, MariaDB 10.2.8

---

rocksdb_max_background_flushes

- **Description:** DBOptions::max_background_flushes for RocksDB.
- **Commandline:** `--rocksdb-max-background-flushes=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `1`
- **Range:** `1` to `64`
- **Removed:** MariaDB 10.3.1, MariaDB 10.2.8

---

rocksdb_max_background_jobs

- **Description:** DBOptions::max_background_jobs for RocksDB.
- **Commandline:** `--rocksdb-max-background-jobs=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `2`
- **Range:** `-1` to `64`

- **Introduced:** MariaDB 10.3.1, MariaDB 10.2.8

rocksdb_max_latest_deadlocks

- **Description:** Maximum number of recent deadlocks to store.
- **Commandline:** `--rocksdb-max-latest-deadlocks=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `5`
- **Range:** `0` to `4294967295`

rocksdb_max_log_file_size

- **Description:** DBOptions::max_log_file_size for RocksDB.
- **Commandline:** `--rocksdb-max-log-file-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `18446744073709551615`

rocksdb_max_manifest_file_size

- **Description:** DBOptions::max_manifest_file_size for RocksDB.
- **Commandline:** `--rocksdb-manifest-log-file-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `1073741824`
- **Range:** `0` to `18446744073709551615`

rocksdb_max_manual_compactions

- **Description:** Maximum number of pending + ongoing number of manual compactions..
- **Commandline:** `--rocksdb-manual_compactions=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `10`
- **Range:** `0` to `4294967295`
- **Introduced:** MariaDB 10.3.10, MariaDB 10.2.18

rocksdb_max_open_files

- **Description:** DBOptions::max_open_files for RocksDB.
- **Commandline:** `--rocksdb-max-open-files=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `-2`
- **Range:** `-2` to `2147483647`

rocksdb_max_row_locks

- **Description:** Maximum number of locks a transaction can have.
- **Commandline:** `--rocksdb-max-row-locks=#`
- **Scope:** Global, Session
- **Dynamic:** Yes

- **Data Type:** `numeric`
- **Default Value:** `1048576`
- **Range:**
  - `1` to `1073741824` (>= MariaDB 10.3.10, MariaDB 10.2.18)
  - `1` to `1048576` (<= MariaDB 10.3.9, MariaDB 10.2.17)

---

rocksdb_max_subcompactions

- **Description:** DBOptions::max_subcompactions for RocksDB.
- **Commandline:** `--rocksdb-max-subcompactions=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `1`
- **Range:** `1` to `64`

---

rocksdb_max_total_wal_size

- **Description:** DBOptions::max_total_wal_size for RocksDB. The maximum size limit for write-ahead-log files. Once this limit is reached, RocksDB forces the flushing of memtables.
- **Commandline:** `--rocksdb-max-total-wal-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `9223372036854775807`

---

rocksdb_merge_buf_size

- **Description:** Size to allocate for merge sort buffers written out to disk during inplace index creation.
- **Commandline:** `--rocksdb-merge-buf-size=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `67108864`
- **Range:** `100` to `18446744073709551615`

---

rocksdb_merge_combine_read_size

- **Description:** Size that we have to work with during combine (reading from disk) phase of external sort during fast index creation.
- **Commandline:** `--rocksdb-merge-combine-read-size=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1073741824`
- **Range:** `100` to `18446744073709551615`

---

rocksdb_merge_tmp_file_removal_delay_ms

- **Description:** Fast index creation creates a large tmp file on disk during index creation. Removing this large file all at once when index creation is complete can cause trim stalls on Flash. This variable specifies a duration to sleep (in milliseconds) between calling chsize() to truncate the file in chunks. The chunk size is the same as merge_buf_size.
- **Commandline:** `--rocksdb-merge-tmp-file-removal-delay-ms=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `18446744073709551615`

---

rocksdb_new_table_reader_for_compaction_inputs

- **Description:** DBOptions::new_table_reader_for_compaction_inputs for RocksDB.
- **Commandline:** `--rocksdb-new-table-reader-for-compaction-inputs={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

rocksdb_no_block_cache

- **Description:** BlockBasedTableOptions::no_block_cache for RocksDB.
- **Commandline:** `--rocksdb-no-block-cache={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

rocksdb_override_cf_options

- **Description:** Option overrides per cf for RocksDB. Note that the `rocksdb-override-cf-options` syntax is quite strict, and any typos will result in a parse error, and the MyRocks plugin will not be loaded. Depending on your configuration, the server may still start. If it does start, you can use this command to check if the plugin is loaded: `select * from information_schema.plugins where plugin_name='ROCKSDB'` (note that you need the "ROCKSDB" plugin. Other auxiliary plugins like "ROCKSDB_TRX" might still get loaded). Another way is to detect the error is check the error log.
- **Commandline:** `--rocksdb-override-cf-options=value`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** (Empty)

---

rocksdb_paranoid_checks

- **Description:** DBOptions::paranoid_checks for RocksDB.
- **Commandline:** `--rocksdb-paranoid-checks={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `ON`

---

rocksdb_pause_background_work

- **Description:** Disable all rocksdb background operations.
- **Commandline:** `--rocksdb-pause-background-work={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

rocksdb_perf_context_level

- **Description:** Perf Context Level for rocksdb internal timer stat collection.
- **Commandline:** `--rocksdb-perf-context-level=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `5`

---

rocksdb_persistent_cache_path

- **Description:** Path for BlockBasedTableOptions::persistent_cache for RocksDB.
- **Commandline:** `--rocksdb-persistent-cache-path=value`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** (Empty)

---

rocksdb_persistent_cache_size_mb

- **Description:** Size of cache in MB for BlockBasedTableOptions::persistent_cache for RocksDB.
- **Commandline:** `--rocksdb-persistent-cache-size-mb=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `18446744073709551615`

---

rocksdb_pin_l0_filter_and_index_blocks_in_cache

- **Description:** pin_l0_filter_and_index_blocks_in_cache for RocksDB.
- **Commandline:** `--rocksdb-pin-l0-filter-and-index-blocks-in-cache={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `ON`

---

rocksdb_print_snapshot_conflict_queries

- **Description:** Logging queries that got snapshot conflict errors into *.err log.
- **Commandline:** `--rocksdb-print-snapshot-conflict-queries={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

rocksdb_rate_limiter_bytes_per_sec

- **Description:** DBOptions::rate_limiter bytes_per_sec for RocksDB.
- **Commandline:** `--rocksdb-rate-limiter-bytes-per-sec=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `9223372036854775807`

---

rocksdb_read_free_rpl_tables

- **Description:** List of tables that will use read-free replication on the slave (i.e. not lookup a row during replication).
- **Commandline:** `--rocksdb-read-free-rpl-tables=value`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** (Empty)
- **Removed:** MariaDB 10.4.7, MariaDB 10.3.17, MariaDB 10.2.26

---

rocksdb_records_in_range

- **Description:** Used to override the result of records_in_range(). Set to a positive number to override.
- **Commandline:** `--rocksdb-records-in-range=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `2147483647`

---

rocksdb_remove_mariabackup_checkpoint

- **Description:** Remove mariabackup checkpoint.
- **Commandline:** `--rocksdb-remove-mariabackup-checkpoint={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB 10.3.8, MariaDB 10.2.16

---

rocksdb_reset_stats

- **Description:** Reset the RocksDB internal statistics without restarting the DB.
- **Commandline:** `--rocksdb-reset-stats={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB 10.3.1, MariaDB 10.2.8

---

rocksdb_rollback_on_timeout

- **Description:** Whether to roll back the complete transaction or a single statement on lock wait timeout (a single statement by default).
- **Commandline:** `--rocksdb-rollback-on-timeout={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB 10.4.7, MariaDB 10.3.17, MariaDB 10.2.26

---

rocksdb_seconds_between_stat_computes

- **Description:** Sets a number of seconds to wait between optimizer stats recomputation. Only changed indexes will be refreshed.
- **Commandline:** `--rocksdb-seconds-between-stat-computes=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `3600`
- **Range:** `0` to `4294967295`

---

rocksdb_signal_drop_index_thread

- **Description:** Wake up drop index thread.
- **Commandline:** `--rocksdb-signal-drop-index-thread={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

rocksdb_sim_cache_size

- **Description:** Simulated cache size for RocksDB.
- **Commandline:** `--rocksdb-sim-cache-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `9223372036854775807`

---

rocksdb_skip_bloom_filter_on_read

- **Description:** Skip using bloom filter for reads.
- **Commandline:** `--rocksdb-skip-bloom-filter-on-read={0|1}`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

rocksdb_skip_fill_cache

- **Description:** Skip filling block cache on read requests.
- **Commandline:** `--rocksdb-skip-fill-cache={0|1}`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

rocksdb_skip_unique_check_tables

- **Description:** Skip unique constraint checking for the specified tables.
- **Commandline:** `--rocksdb-skip-unique-check-tables=value`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `.*`

---

rocksdb_sst_mgr_rate_bytes_per_sec

- **Description:** DBOptions::sst_file_manager rate_bytes_per_sec for RocksDB
- **Commandline:** `--rocksdb-sst-mgr-rate-bytes-per-sec=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `18446744073709551615`

---

rocksdb_stats_dump_period_sec

- **Description:** DBOptions::stats_dump_period_sec for RocksDB.
- **Commandline:** `--rocksdb-stats-dump-period-sec=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `600`
- **Range:** `0` to `2147483647`

---

rocksdb_stats_level

- **Description:** Statistics Level for RocksDB. Default is 0 (kExceptHistogramOrTimers).
- **Commandline:** `--rocksdb-stats-level=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `4`
- **Introduced:** MariaDB 10.4.7, MariaDB 10.3.17, MariaDB 10.2.26

---

rocksdb_stats_recalc_rate

- **Description:** The number of indexes per second to recalculate statistics for. 0 to disable background recalculation.
- **Commandline:** `--rocksdb-stats-recalc-rate=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `4294967295`
- **Introduced:** MariaDB 10.3.10 MariaDB 10.2.18

---

rocksdb_store_row_debug_checksums

- **Description:** Include checksums when writing index/table records.
- **Commandline:** `--rocksdb-store-row-debug-checksums={0|1}`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

rocksdb_strict_collation_check

- **Description:** Enforce case sensitive collation for MyRocks indexes.
- **Commandline:** `--rocksdb-strict-collation-check={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `ON`

---

rocksdb_strict_collation_exceptions

- **Description:** List of tables (using regex) that are excluded from the case sensitive collation enforcement.
- **Commandline:** `--rocksdb-strict-collation-exceptions=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** (Empty)

---

rocksdb_supported_compression_types

- **Description:** Compression algorithms supported by RocksDB. Note that RocksDB does not make use of MariaDB 10.7 compression-plugins.
- **Commandline:** `--rocksdb-supported-compression-types=value`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** `Snappy,Zlib,ZSTDNotFinal`

---

rocksdb_table_cache_numshardbits

- **Description:** DBOptions::table_cache_numshardbits for RocksDB.
- **Commandline:** `--rocksdb-table-cache-numshardbits=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `6`
- **Range:** `0` to `19`

---

rocksdb_table_stats_sampling_pct

- **Description:** Percentage of entries to sample when collecting statistics about table properties. Specify either 0 to sample everything or percentage [1..100]. By default 10% of entries are sampled.
- **Commandline:** `--rocksdb-table-stats-sampling-pct=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `10`
- **Range:** `0` to `100`

---

rocksdb_tmpdir

- **Description:** Directory for temporary files during DDL operations.
- **Commandline:** `--rocksdb-tmpdir[=value]`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** (Empty)

---

rocksdb_trace_sst_api

- **Description:** Generate trace output in the log for each call to the SstFileWriter.
- **Commandline:** `--rocksdb-trace-sst-api={0|1}`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

rocksdb_two_write_queues

- **Description:** DBOptions::two_write_queues for RocksDB.
- **Commandline:** `--rocksdb-two-write-queues={0|1}`
- **Scope:** Global,
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `ON`
- **Introduced:** MariaDB 10.3.7, MariaDB 10.2.15

---

rocksdb_unsafe_for_binlog

- **Description:** Allowing statement based binary logging which may break consistency.
- **Commandline:** `--rocksdb-unsafe-for-binlog={0|1}`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

rocksdb_update_cf_options

- **Description:** Option updates per column family for RocksDB.

- **Commandline:** `--rocksdb-update-cf-options=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `varchar`
- **Default Value:** (Empty)
- **Introduced:** MariaDB 10.3.1, MariaDB 10.2.8

---

rocksdb_use_adaptive_mutex

- **Description:** DBOptions::use_adaptive_mutex for RocksDB.
- **Commandline:** `--rocksdb-use-adaptive-mutex={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

rocksdb_use_clock_cache

- **Description:** Use ClockCache instead of default LRUCache for RocksDB.
- **Commandline:** `--rocksdb-use-clock-cache={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

rocksdb_use_direct_io_for_flush_and_compaction

- **Description:** DBOptions::use_direct_io_for_flush_and_compaction for RocksDB.
- **Commandline:** `--rocksdb-use-direct-io-for-flush-and-compaction={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB 10.3.1, MariaDB 10.2.8

---

rocksdb_use_direct_reads

- **Description:** DBOptions::use_direct_reads for RocksDB.
- **Commandline:** `--rocksdb-use-direct-reads={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

rocksdb_use_direct_writes

- **Description:** DBOptions::use_direct_writes for RocksDB.
- **Commandline:** `--rocksdb-use-direct-reads={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Removed:** MariaDB 10.3.1, MariaDB 10.2.8

---

rocksdb_use_fsync

- **Description:** DBOptions::use_fsync for RocksDB.
- **Commandline:** `--rocksdb-use-fsync={0|1}`
- **Scope:** Global

- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

rocksdb_validate_tables

- **Description:** Verify all .frm files match all RocksDB tables (0 means no verification, 1 means verify and fail on error, and 2 means verify but continue.
- **Commandline:** `--rocksdb-validate-tables=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `1`
- **Range:** `0` to `2`

---

rocksdb_verify_row_debug_checksums

- **Description:** Verify checksums when reading index/table records.
- **Commandline:** `--rocksdb-verify-row-debug-checksums={0|1}`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

rocksdb_wal_bytes_per_sync

- **Description:** DBOptions::wal_bytes_per_sync for RocksDB.
- **Commandline:** `--rocksdb-wal-bytes-per-sync=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `18446744073709551615`

---

rocksdb_wal_dir

- **Description:** DBOptions::wal_dir for RocksDB. Directory where the write-ahead-log files are stored.
- **Commandline:** `--rocksdb-wal-dir=value`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** (Empty)

---

rocksdb_wal_recovery_mode

- **Description:** DBOptions::wal_recovery_mode for RocksDB. Default is kAbsoluteConsistency. Records that are not yet committed are stored in the Write-Ahead-Log (WAL). If the server is not cleanly shut down, the recovery mode will determine the WAL recovery behavior.
  - **1**: kAbsoluteConsistency. Will not start if any corrupted entries (including incomplete writes) are detected (the default).
  - **0**: kTolerateCorruptedTailRecords. Ignores any errors found at the end of the WAL
  - **2**: kPointInTimeRecovery. Replay of the WAL is halted after finding an error. The system will be recovered to the latest consistent point-in-time. Data from a replica can used to replay past the point-in-time.
  - **3**: kSkipAnyCorruptedRecords. A risky option where any corrupted entries are skipped while subsequent healthy WAL entries are applied.
- **Commandline:** `--rocksdb-wal-recovery-mode=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1`

- **Range:** `0` to `3`

---

rocksdb_wal_size_limit_mb

- **Description:** DBOptions::WAL_size_limit_MB for RocksDB. Write-ahead-log files are moved to an archive directory once their memtables are flushed. This variable specifies the largest size, in MB, that the archive can be.
- **Commandline:** `--rocksdb-wal-size-limit-mb=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `9223372036854775807`

---

rocksdb_wal_ttl_seconds

- **Description:** DBOptions::WAL_ttl_seconds for RocksDB. Oldest time, in seconds, that a write-ahead-log file should exist.
- **Commandline:** `--rocksdb-wal-ttl-seconds=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `9223372036854775807`

---

rocksdb_whole_key_filtering

- **Description:** BlockBasedTableOptions::whole_key_filtering for RocksDB. If set (the default), the bloomfilter to use the whole key (rather than only the prefix) for filtering is enabled. Lookups should use the whole key for matching to make best use of this setting.
- **Commandline:** `--rocksdb-whole-key-filtering={0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `ON`

---

rocksdb_write_batch_max_bytes

- **Description:** Maximum size of write batch in bytes. 0 means no limit.
- **Commandline:** `--rocksdb-write-batch-max-bytes=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `18446744073709551615`

---

rocksdb_write_disable_wal

- **Description:** WriteOptions::disableWAL for RocksDB.
- **Commandline:** `--rocksdb-write-disable-wal={0|1}`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

rocksdb_write_ignore_missing_column_families

- **Description:** WriteOptions::ignore_missing_column_families for RocksDB.
- **Commandline:** `--rocksdb-write-ignore-missing-column-families={0|1}`
- **Scope:** Global, Session

- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

`rocksdb_write_policy`

- **Description:** DBOptions::write_policy for RocksDB.
- **Commandline:** `--rocksdb-write-policy=val`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `enum`
- **Default Value:** `write_committed`
- **Valid Values:** `write_committed`, `write_prepared`, `write_unprepared`
- **Introduced:** MariaDB 10.3.10, MariaDB 10.2.18

5.3.14.7 MyRocks Transactional Isolation

TODO:

- MyRocks uses snapshot isolation
- Support do READ-COMMITTED and REPEATABLE-READ
- SERIALIZABLE is not supported
    - There is no "Gap Locking" which makes Statement Based Replication unsafe (See MyRocks and Replication).

5.3.14.8 MyRocks and Replication

Contents

Details about how MyRocks works with replication.

MyRocks and Statement-Based Replication

Statement-based replication (SBR) works as follows: SQL statements are executed on the master (possibly concurrently). They are written into the binlog (this fixes their ordering, "a serialization"). The slave then reads the binlog and executes the statements in their binlog order.

In order to prevent data drift, serial execution of statements on the slave must have the same effect as concurrent execution of these statements on the master. In other words, transaction isolation on the master must be close to `SERIALIZABLE` transaction isolation level (This is not a strict mathematical proof but shows the idea).

InnoDB achieves this by (almost) supporting `SERIALIZABLE` transactional isolation level. It does so by supporting "Gap Locks". MyRocks doesn't support `SERIALIZABLE` isolation, and it doesn't support gap locks.

Because of that, generally one cannot use MyRocks and statement-based replication.

Updating a MyRocks table while having SBR on, will result in an error as follow:

```
ERROR 4056 (HY000): Can't execute updates on master with binlog_format != ROW.
```

Can One Still Use SBR with MyRocks?

Yes. In many cases, database applications run a restricted set of SQL statements, and it's possible to prove that lack of Gap Lock support is not a problem and data skew will not occur.

In that case, one can set `@@rocksdb_unsafe_for_binlog=1` and MyRocks will work with SBR. The user is however responsible for making sure their queries are not causing a data skew.

Read-Free Slave

Differences From Upstream MyRocks

MyRocks upstream (that is, Facebook's MySQL branch) has a number of unique replication enhancements. These are available in upstream's version of MyRocks but not in MariaDB's version of MyRocks.

- Read-Free Replication (see https://github.com/facebook/mysql-5.6/wiki/Read-Free-Replication) TODO
- `<<unique_check_lag_threshold>>` . This is FB/MySQL-5.6 feature where unique checks are disabled if replication lag exceeds a certain threshold.
- `<<slave_gtid_info=OPTIMIZED>>` . This is said to be:

```
<<quote>>
"Whether SQL threads update mysql.slave_gtid_info table. If this value "
"is OPTIMIZED, updating the table is done inside storage engines to "
"avoid MySQL layer's performance overhead",
<</quote>>
```

5.3.14.9 MyRocks and Group Commit with Binary log

Contents

MyRocks supports group commit with the binary log (MDEV-11934).

Counter Values to Watch

(The following is only necessary if you are studying MyRocks internals)

MariaDB's group commit counters are:

Binlog_commits - how many transactions were written to the binary log

Binlog_group_commits - how many group commits happened. (e.g. if each group had two transactions, this will be twice as small as Binlog_commits)

On the RocksDB side, there is one relevant counter: Rocksdb_wal_synced - How many times RocksDB's WAL file was synced. (TODO: this is after group commit happened, right?)

On the Value of rocksdb_wal_group_syncs

FB/MySQL-5.6 has a rocksdb_wal_group_syncs counter (The counter is provided by MyRocks, it is not a view of a RocksDB counter). It is increased in rocksdb_flush_wal() when doing the rdb->FlushWAL() call.

rocksdb_flush_wal() is called by MySQL's Group Commit when it wants to make the effect of several rocksdb_prepare() calls persistent.

So, the value of rocksdb_wal_group_syncs in FB/MySQL-5.6 is similar to Binlog_group_commits in MariaDB.

MariaDB doesn't have that call, each rocksdb_prepare() call takes care of being persistent on its own.

Because of that, rocksdb_wal_group_syncs is zero for MariaDB. (Currently, it is only incremented when the binlog is rotated).

Examples

So for a workload with concurrency=50, n_queries=10K, one gets

- Binlog_commits=10K
- Binlog_group_commits=794
- Rocksdb_wal_synced=8362

This is on a RAM disk

For a workload with concurrency=50, n_queries=10K, rotating laptop hdd, one gets

- Binlog_commits= 10K
- Binlog_group_commits=1403
- Rocksdb_wal_synced=400

The test took 38 seconds, Number of syncs was 1400+400=1800, which gives 45 syncs/sec which looks normal for this slow rotating desktop hdd.

Note that the WAL was synced fewer times than there were binlog commit groups (?)

5.3.14.10 Optimizer Statistics in MyRocks

Contents

This article describes how MyRocks storage engine provides statistics to the query optimizer.

There are three kinds of statistics:

- Table statistics (number of rows in the table, average row size)
- Index cardinality (how distinct values are in the index)
- records-in-range estimates (how many rows are in a certain range "const1 < tbl.key < const2".

How MyRocks computes statistics

MyRocks (actually RocksDB) uses LSM files which are written once and never updated. When an LSM file is written, MyRocks will compute index cardinalities and number-of-rows for the data in the file. (The file generally has rows, index records and/or tombstones for multiple tables/indexes).

For performance reasons, statistics are computed based on a fraction of rows in the LSM file. The percentage of rows used is controlled by rocksdb_table_stats_sampling_pct; the default value is 10%.

Before the data is dumped into LSM file, it is stored in the MemTable. MemTable doesn't allow computing index cardinalities, but it can provide an approximate number of rows in the table. Use of MemTable data for statistics is controlled by rocksdb_force_compute_memtable_stats; the default value is `ON` .

Are index statistics predictable?

Those who create/run MTR tests, need to know whether EXPLAIN output is deterministic. For MyRocks tables, the answer is NO (just like for InnoDB).

Statistics are computed using sampling and GetApproximateMemTableStats() which means that the #rows column in the EXPLAIN output may vary slightly.

Records-in-range estimates

MyRocks uses RocksDB's GetApproximateSizes() call to produce an estimate for the number of rows in the certain range. The data in MemTable is also taken into account by issuing a GetApproximateMemTableStats call.

ANALYZE TABLE

ANALYZE TABLE will possibly flush the MemTable (depending on the rocksdb_flush_memtable_on_analyze and rocksdb_pause_background_work settings).

After that, it will re-read statistics from the SST files and re-compute the summary numbers (TODO: and if the data was already on disk, the result should not be different from the one we had before ANALYZE?)

Debugging helper variables

There are a few variables that will cause MyRocks to report certain pre-defined estimate numbers to the optimizer:

- @@rocksdb_records_in_range - if not 0, report that any range has this many rows
- @@rocksdb_force_index_records_in_range - if not 0, and FORCE INDEX hint is used, report that any range has this

many rows.
- @@rocksdb_debug_optimizer_n_rows - if not 0, report that any MyRocks table has this many rows.

5.3.14.11 Differences Between MyRocks Variants

MyRocks is available in

- Facebook's (FB) MySQL branch (originally based on MySQL 5.6)
- MariaDB (from 10.2 and 10.3)
- Percona Server from 5.7

This page lists differences between these variants.

This is a work in progress. The contents are not final

Contents

RocksDB Data Location

FB and Percona store RocksdDB files in $datadir/ `.rocksdb` . MariaDB puts them in $datadir/ `#rocksdb` . This is more friendly for packaging and OS scripts.

Compression Algorithms

- FB's branch doesn't provide binaries. One needs to compile it with appropriate compression libraries.
- In MariaDB, available compression algorithms can be seen in the rocksdb_supported_compression_types variable. From MariaDB 10.7, algorithms can be installed as a plugin. In earlier versions, the set of supported compression algorithms depends on the platform.
  - On Ubuntu 16.04 (current LTS) it is `Snappy,Zlib,LZ4,LZ4HC` .
  - On CentOS 7.4 it is `Snappy,Zlib` .
  - In the bintar tarball it is `Snappy,Zlib` .
- Percona Server supports: `Zlib, ZSTD, LZ4 (the default), LZ4HC` . Unsupported algorithms: `Snappy, BZip2, XPress` .

RocksDB Version Information

- FB's branch provides the `rocksdb_git_hash` *status* variable.
- MariaDB provides the `@@rocksdb_git_hash` *system* variable.
- Percona Server doesn't provide either.

RocksDB Version

- Facebook's branch uses RocksDB 5.10.0 (the version number can be found in `include/rocksdb/version.h`)

```
commit ba295cda29daee3ffe58549542804efdfd969784
Author: Andrew Kryczka <andrewkr@fb.com>
Date:   Fri Jan 12 11:03:55 2018 -0800
```

- MariaDB currently uses 5.8.0

```
commit 9a970c81af9807071bd690f4c808c5045866291a
Author: Yi Wu <yiwu@fb.com>
Date:   Wed Sep 13 17:21:35 2017 -0700
```

- Percona Server uses 5.8.0

```
commit ab0542f5ec6e7c7e405267eaa2e2a603a77d570b
Author: Maysam Yabandeh <myabandeh@fb.com>
Date:   Fri Sep 29 07:55:22 2017 -0700
```

Binlog Position in information_schema.rocksdb_global_info

- FB branch provides information_schema.rocksdb_global_info type=BINLOG, NAME={FILE, POS, GTID}.
- Percona Server doesn't provide it.
- MariaDB doesn't provide it.

One use of that information is to take the output of `myrocks_hotbackup` and make it a new master.

Gap Lock Detector

- FB branch has a "Gap Lock Detector" feature. It is at the SQL layer. It can be controlled with `gap_lock_XXX` variables and is disabled by default (gap-lock-raise-error=false, gap-lock-write-lock=false).
- Percona Server has gap lock checking ON but doesn't seem to have any way to control it? Queries that use Gap Lock on MyRocks fail with an error like this:

```
insert into tbl2 select * from tbl1;
ERROR 1105 (HY000): Using Gap Lock without full unique key in multi-table or multi-statement
transactions
is not allowed. You need to either rewrite queries to use all unique key columns in WHERE equal
conditions,
or rewrite to single-table, single-statement transaction.  Query: insert into tbl2 select *
from tbl1
```

- MariaDB doesn't include the Gap Lock Detector.

Generated Columns

- Both MariaDB and Percona Server support generated columns, but neither one supports them for the MyRocks storage engine (attempts to create a table will produce an error).
- Invisible columns in MariaDB 10.3 are supported (as they are an SQL layer feature).

rpl_skip_tx_api

Facebook's branch has a performance feature for replication slaves, `rpl_skip_tx_api`. It is not available in MariaDB or in Percona Server.

Details

The above comparison was made using

- FB/MySQL 5.6.35
- Percona Server 5.7.20-19-log
- MariaDB 10.2.13 (MyRocks is beta)

5.3.14.12 MyRocks and Bloom Filters

Contents

Bloom filters are used to reduce read amplification. Bloom filters can be set on a per-column family basis (see myrocks-column-families).

Bloom Filter Parameters

- How many bits to use
- whole_key_filtering=true/false
- Whether the bloom filter is for the entire key or for the prefix. In case of a prefix, you need to look at the index definition and compute the desired prefix length.

Computing Prefix Length

- It's 4 bytes for `index_nr`
- Then, for fixed-size columns (integer, date[time], decimal) it is key_length as shown by `EXPLAIN`. For VARCHAR columns, determining the length is tricky (It depends on the values stored in the table. Note that MyRocks encodes VARCHARs with "Variable-Length Space-Padded Encoding" format).

Configuring Bloom Filter

To enable 10-bit bloom filter for 8-byte prefix length for column family "cf1", put this into my.cnf:

```
rocksdb_override_cf_options='cf1={block_based_table_factory=
{filter_policy=bloomfilter:10:false;whole_key_filtering=0;};prefix_extractor=capped:8};'
```

and restart the server.

Check if the column family actually uses the bloom filter:

```
select *
from information_schema.rocksdb_cf_options
where
  cf_name='cf1' and
  option_type IN ('TABLE_FACTORY::FILTER_POLICY','PREFIX_EXTRACTOR');
```

```
+---------+-------------------------------+----------------------------+
| CF_NAME | OPTION_TYPE                   | VALUE                      |
+---------+-------------------------------+----------------------------+
| cf1     | PREFIX_EXTRACTOR              | rocksdb.CappedPrefix.8     |
| cf1     | TABLE_FACTORY::FILTER_POLICY | rocksdb.BuiltinBloomFilter |
+---------+-------------------------------+----------------------------+
```

Checking if Bloom Filter is Useful

Watch these status variables:

```
show status like '%bloom%';
+---------------------------------------+-------+
| Variable_name                         | Value |
+---------------------------------------+-------+
| Rocksdb_bloom_filter_prefix_checked | 1     |
| Rocksdb_bloom_filter_prefix_useful  | 0     |
| Rocksdb_bloom_filter_useful         | 0     |
+---------------------------------------+-------+
```

Other useful variables are:

- `rocksdb_force_flush_memtable_now` - bloom filter is only used when reading data from disk. If you are doing testing, flush the data to disk first.

- `rocksdb_skip_bloom_filter_on_read` - skip using the bloom filter (default is FALSE).

5.3.14.13 MyRocks and CHECK TABLE

MyRocks supports the CHECK TABLE command.

The command will do a number of checks to verify that the table data is self-consistent.

The details about the errors are printed into the error log. If log_warnings > 2, the error log will also have some informational messages which can help with troubleshooting.

Besides this, RocksDB has its own (low-level) log in `#rocksdb/LOG` file.

5.3.14.14 MyRocks and Data Compression

Contents

MyRocks supports several compression algorithms.

Supported Compression Algorithms

Supported compression algorithms can be checked like so:

```
show variables like 'rocksdb%compress%';
+-----------------------------------+-----------------------------------+
| Variable_name                     | Value                             |
+-----------------------------------+-----------------------------------+
| rocksdb_supported_compression_types | Snappy,Zlib,LZ4,LZ4HC,ZSTDNotFinal |
+-----------------------------------+-----------------------------------+
```

Another way to make the check is to look into `#rocksdb/LOG` file in the data directory. It should have lines like:

```
2019/04/12-14:08:23.869919 7f839188b540 Compression algorithms supported:
2019/04/12-14:08:23.869920 7f839188b540         kZSTDNotFinalCompression supported: 1
2019/04/12-14:08:23.869922 7f839188b540         kZSTD supported: 1
2019/04/12-14:08:23.869923 7f839188b540         kXpressCompression supported: 0
2019/04/12-14:08:23.869924 7f839188b540         kLZ4HCCompression supported: 1
2019/04/12-14:08:23.869924 7f839188b540         kLZ4Compression supported: 1
2019/04/12-14:08:23.869925 7f839188b540         kBZip2Compression supported: 0
2019/04/12-14:08:23.869926 7f839188b540         kZlibCompression supported: 1
2019/04/12-14:08:23.869927 7f839188b540         kSnappyCompression supported: 1
```

Compression Settings

Compression is set on a per-Column Family basis. See MyRocks Column Families.

Checking Compression Settings

To check current compression settings for a column family one can use a query like so:

```
select * from information_schema.rocksdb_cf_options
where option_type like '%ompression%' and cf_name='default';
```

The output will be like:

```
+---------+-----------------------------------------+---------------------------+
| CF_NAME | OPTION_TYPE                             | VALUE                     |
+---------+-----------------------------------------+---------------------------+
| default | COMPRESSION_TYPE                        | kSnappyCompression        |
| default | COMPRESSION_PER_LEVEL                   | NUL                       |
| default | COMPRESSION_OPTS                        | -14:32767:0               |
| default | BOTTOMMOST_COMPRESSION                  | kDisableCompressionOption |
| default | TABLE_FACTORY::VERIFY_COMPRESSION       | 0                         |
| default | TABLE_FACTORY::ENABLE_INDEX_COMPRESSION | 1                         |
+---------+-----------------------------------------+---------------------------+
```

Current column family settings will be used for the new SST files.

Modifying Compression Settings

Compression settings are not dynamic parameters, one cannot change them by setting rocksdb_update_cf_options.

The procedure to change compression settings is as follows:

- Edit `my.cnf` to set rocksdb_override_cf_options.

Example:

```
rocksdb-override-cf-options='cf1={compression=kZSTD;bottommost_compression=kZSTD;}'
```

- Restart the server.

The data will not be re-compressed immediately. However, all new SST files will use the new compression settings, so as data gets inserted/updated the column family will gradually start using the new option.

Caveat: Syntax Errors

Please note that `rocksdb-override-cf-options` syntax is quite strict. Any typos will result in the parse error, and MyRocks plugin will not be loaded. Depending on your configuration, the server may still start. If it does start, you can use this command to check if the plugin is loaded:

```
select * from information_schema.plugins where plugin_name='ROCKSDB'
```

(note that you need the "ROCKSDB" plugin. Other auxiliary plugins like "ROCKSDB_TRX" might still get loaded).

Another way is to detect the error is check the error log. When option parsing fails, it will contain messages like so:

```
2019-04-16 11:07:57 140283675678016 [Warning] Invalid cf config for cf1 in override options
(options: cf1={compression=kLZ4Compression;bottommost_compression=kZSTDCompression;})
2019-04-16 11:07:57 140283675678016 [ERROR] RocksDB: Failed to initialize CF options map.
2019-04-16 11:07:57 140283675678016 [ERROR] Plugin 'ROCKSDB' init function returned error.
2019-04-16 11:07:57 140283675678016 [ERROR] Plugin 'ROCKSDB' registration as a STORAGE ENGINE
failed.
```

Checking How the Data is Compressed

A query to check what compression is used in the SST files that store the data for a given table (test.t1):

```
select
  SP.sst_name, SP.compression_algo
from
  information_schema.rocksdb_sst_props SP,
  information_schema.rocksdb_ddl D,
  information_schema.rocksdb_index_file_map IFM
where
  D.table_schema='test' and D.table_name='t1' and
  D.index_number= IFM.index_number and
  IFM.sst_name=SP.sst_name;
```

Example output:

```
+------------+------------------+
| sst_name   | compression_algo |
+------------+------------------+
| 000028.sst | Snappy           |
| 000028.sst | Snappy           |
| 000026.sst | Snappy           |
| 000026.sst | Snappy           |
+------------+------------------+
```

5.3.14.15 MyRocks and Index-Only Scans

Contents

This article is about MyRocks and index-only scans on secondary indexes. It applies to MariaDB's MyRocks, Facebook's MyRocks, and other variants.

Secondary Keys Only

The primary key in MyRocks is always the clustered key, that is, the index record is THE table record and so it's not possible to do "index only" because there isn't anything that is not in the primary key's (Key,Value) pair.

Secondary keys may or may not support index-only scans, depending on the datatypes of the columns that the query is trying to read.

Background: Mem-Comparable Keys

MyRocks indexes store "mem-comparable keys" (that is, the key values are compared with `memcmp`). For some datatypes, it is easily possible to convert between the column value and its mem-comparable form, while for others the conversion is one-way.

For example, in case-insensitive collations capital and regular letters are considered identical, i.e. 'c' ='C'. For some datatypes, MyRocks stores some extra data which allows it to restore the original value back. (For the `latin1_general_ci` collation and character 'c', for example, it will store one bit which says whether the original value was a small 'c' or a capital letter 'C'). This doesn't work for all datatypes, though.

Index-Only Support for Various Datatypes

Index-only scans are supported for numeric and date/time datatypes. For CHAR and VAR[CHAR], it depends on which collation is used, see below for details.

Index-only scans are currently not supported for less frequently used datatypes, like

- `BIT(n)`
- `SET(...)`
- `ENUM(...)` *It is actually possible to add support for those, feel free to write a patch or at least make a case why a particular datatype is important*

Index-Only Support for Various Collations

As far as Index-only support is concerned, MyRocks distinguishes three kinds of collations:

- 1. Binary (Reversible) Collations

These are `binary`, `latin1_bin`, and `utf8_bin`.

For these collations, it is possible to convert a value back from its mem-comparable form. Hence, one can restore the

original value back from its index record, and index-only scans are supported.

- 2. Restorable Collations

These are collations where one can store some extra information which helps to restore the original value.

Criteria (from storage/rocksdb/rdb_datadic.cc, rdb_is_collation_supported()) are:

- The charset should use 1-byte characters (so, unicode-based collations are not included)
- strxfrm(1 byte) = {one 1-byte weight value always}
- no binary sorting
- PAD attribute

The examples are: `latin1_general_ci` , `latin1_general_cs` , `latin1_swedish_ci` , etc.

Index-only scans are supported for these collations.

- 3. All Other Collations

For these collations, there is no known way to restore the value from its mem-comparable form, and so index-only scans are not supported.

MyRocks needs to fetch the clustered PK record to get the field value.

Covering Secondary Key Lookups for VARCHARs

TODO: there is also this optimization:

https://github.com/facebook/mysql-5.6/issues/303 https://github.com/facebook/mysql-5.6/commit/f349c95848e92b5b27b44f0e57194100eb0997e7

document it.

5.3.14.16 MyRocks and START TRANSACTION WITH CONSISTENT SNAPSHOT

FB/MySQL has added new syntax:

```
START TRANSACTION WITH CONSISTENT ROCKSDB|INNODB SNAPSHOT;
```

The statement returns the binlog coordinates pointing at the snapshot.

MariaDB (and Percona Server) support extension to the regular

```
START TRANSACTION WITH CONSISTENT SNAPSHOT;
```

syntax as documented in Enhancements for START TRANSACTION WITH CONSISTENT SNAPSHOT.

After issuing the statement, one can examine the binlog_snapshot_file and binlog_snapshot_position status variables to see the binlog position that corresponds to the snapshot.

5.3.14.17 MyRocks Column Families

MyRocks stores data in column families. These are similar to tablespaces. By default, the data is stored in the `default` column family.

One can specify which column family the data goes to by using index comments:

```
INDEX index_name(col1, col2, ...) COMMENT 'column_family_name'
```

Contents

If the column name starts with `rev:`, the column family is reverse-ordered.

Reasons for Column Families

Storage parameters like

- Bloom filter settings
- Compression settings
- Whether the data is stored in reverse order

are specified on a per-column family basis.

Creating a Column Family

When creating a table or index, you can specify the name of the column family for it. If the column family doesn't exist, it will be automatically created.

Dropping a Column Family

There is currently no way to drop a column family. RocksDB supports this internally but MyRocks doesn't provide any way to do it.

Setting Column Family Parameters

Use these variables:

- rocksdb_default_cf_options - a my.cnf parameter specifying default options for all column families.
- rocksdb_override_cf_options - a my.cnf parameter specifying per-column family option overrides.
- rocksdb_update_cf_options - a dynamically-settable variable which allows to change parameters online. Not all parameters can be changed.

rocksdb_override_cf_options

This parameter allows one to override column family options for specific column families. Here is an example of how to set option1=value1 and option2=value for column family cf1, and option3=value3 for column family cf3:

```
rocksdb_override_cf_options='cf1={option1=value1;option2=value2};cf2={option3=value3}'
```

One can check the contents of `INFORMATION_SCHEMA.ROCKSDB_CF_OPTIONS` to see what options are available.

Options that are frequently configured are:

- Data compression. See myrocks-and-data-compression.
- Bloom Filters. See myrocks-and-bloom-filters.

Examining Column Family Parameters

See the INFORMATION_SCHEMA.ROCKSDB_CF_OPTIONS table.

5.3.14.18 MyRocks in MariaDB 10.2 vs MariaDB 10.3

MyRocks storage engine itself is identical in MariaDB 10.2 and MariaDB 10.3.

MariaDB 10.3 has a feature that should be interesting for MyRocks users. It is the gtid_pos_auto_engines option (MDEV-

12179). This is a performance feature for replication slaves that use multiple transactional storage engines.

For further information, see mysql.gtid_slave_pos table.

5.3.14.19 MyRocks Performance Troubleshooting

Contents

MyRocks exposes its performance metrics through several interfaces:

- Status variables
- SHOW ENGINE ROCKSDB STATUS
- RocksDB's perf context

the contents slightly overlap, but each source has its own unique information, so be sure to check all three.

Status Variables

Check the output of

```
SHOW STATUS like 'Rocksdb%'
```

See MyRocks Status Variables for more information.

SHOW ENGINE ROCKSDB STATUS

This produces a lot of information.

One particularly interesting part is compaction statistics. It shows the amount of data on each SST level and other details:

```
*************************** 4. row ***************************
  Type: CF_COMPACTION
  Name: default
Status:
** Compaction Stats [default] **
Level    Files   Size     Score Read(GB)  Rn(GB) Rnp1(GB) Write(GB) Wnew(GB) Moved(GB) W-Amp
Rd(MB/s) Wr(MB/s) Comp(sec) Comp(cnt) Avg(sec) KeyIn KeyDrop
----------------------------------------------------------------------------------------------------
----------------------------------------------------------
  L0      3/0   30.16 MB   1.0      0.0     0.0      0.0      11.9     11.9       0.0   1.0
0.0     76.6       159       632    0.251       0       0
  L1      5/0   247.54 MB   1.0      0.7     0.2      0.5       0.5      0.0      11.6   2.6
58.5     44.1        12         4    2.926     30M    10M
  L2    112/0    2.41 GB   1.0      0.6     0.0      0.6       0.5     -0.1      11.4  43.4
55.2     45.9        11         1   10.827     21M  3588K
  L3    466/0    8.91 GB   0.4      0.0     0.0      0.0       0.0      0.0       8.9   0.0
0.0      0.0         0         0    0.000       0       0
 Sum    586/0   11.59 GB   0.0      1.3     0.2      1.0      12.8     11.8      32.0   1.1
7.1     72.6       181       637    0.284     52M    13M
 Int      0/0    0.00 KB   0.0      0.9     0.1      0.8       0.8      0.0       0.1  20.5
48.4     45.3        19         6    3.133     33M  3588K
```

Performance Context

RocksDB has an internal mechanism called "perf context". The counter values are exposed through two tables:

- INFORMATION_SCHEMA.ROCKSDB_PERF_CONTEXT_GLOBAL - global counters
- INFORMATION_SCHEMA.ROCKSDB_PERF_CONTEXT - Per-table/partition counters

By default statistics are NOT collected. One needs to set rocksdb_perf_context_level to some value (e.g. 3) to enable collection.

5.3.15 OQGRAPH

The Open Query GRAPH computation engine, or OQGRAPH as the engine itself is called, allows you to handle hierarchies (tree structures) and complex graphs (nodes having many connections in several directions).

OQGRAPH Version	Introduced	Maturity
3.0	MariaDB 10.0.25	Gamma
3.0	MariaDB 10.0.7	Beta
2.0	MariaDB 5.2.1	

Installing OQGRAPH

Installing OQGRAPH.

OQGRAPH Overview

Overview of the OQGRAPH storage engine.

OQGRAPH Examples

OQGRAPH examples.

Compiling OQGRAPH

How to compile OQGRAPH.

Building OQGRAPH Under Windows

OQGRAPH build instructions for Windows.

OQGRAPH System and Status Variables

List and description of OQGRAPH system and status variables.

There are 5 related questions.

5.3.15.1 Installing OQGRAPH

Contents

The Open Query GRAPH computation engine, or OQGRAPH as the engine itself is called, allows you to handle hierarchies (tree structures) and complex graphs (nodes having many connections in several directions).

Installation

The OQGRAPH storage engine exists as a separate package in the repositories for MariaDB 10.0.7 and later. On Ubuntu and Debian the package is called `mariadb-oqgraph-engine-10.0` or `mariadb-plugin-oqgraph`. On Red Hat, CentOS, and Fedora the package is called `MariaDB-oqgraph-engine`. To install the plugin, first install the appropriate package and then install the plugin using the INSTALL SONAME or INSTALL PLUGIN commands.

Debian and Ubuntu

On Debian and Ubuntu, install the package as follows:

```
sudo apt-get install mariadb-plugin-oqgraph
```

or (for MariaDB 10.0)

```
sudo apt-get install mariadb-oqgraph-engine-10.0
```

Fedora/Red Hat/CentOS

Note that OQGRAPH v3 requires libjudy, which is not in the official Red Hat/Fedora repositories. This needs to be installed first, for example:

```
wget http://dl.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm
rpm -Uvh epel-release-6-8.noarch.rpm
```

Then install the package, as follows:

```
sudo yum install MariaDB-oqgraph-engine
```

Installing the Plugin

On either system you can then launch the `mysql` command-line client and install the plugin in MariaDB as follows:

```
INSTALL SONAME 'ha_oqgraph';
```

5.3.15.2 OQGRAPH Overview

Contents

The Open Query GRAPH computation engine, or OQGRAPH as the engine itself is called, allows you to handle hierarchies (tree structures) and complex graphs (nodes having many connections in several directions).

OQGRAPH is unlike other storage engines, consisting of an entirely different architecture to a regular storage engine such as Aria, MyISAM or InnoDB.

It is intended to be used for retrieving hierarchical information, such as those used for graphs, routes or social relationships, in plain SQL.

Installing

See Installing OQGRAPH. Note that if the query cache is enabled, OQGRAPH will not use it.

Creating a Table

> The following documentation is based upon MariaDB 10.0.7 and OQGRAPH v3.

Example with origin and destination nodes only

To create an OQGRAPH v3 table, a backing table must first be created. This backing table will store the actual data, and will be used for all INSERTs, UPDATEs and so on. It must be a regular table, not a view. Here's a simple example to start with:

```
CREATE TABLE oq_backing (
  origid INT UNSIGNED NOT NULL,
  destid INT UNSIGNED NOT NULL,
  PRIMARY KEY (origid, destid),
  KEY (destid)
);
```

Some data can be inserted into the backing table to test with later:

```
INSERT INTO oq_backing(origid, destid)
 VALUES (1,2), (2,3), (3,4), (4,5), (2,6), (5,6);
```

Now the read-only OQGRAPH table is created. The CREATE statement must match the format below - any difference will

result in an error.

```
CREATE TABLE oq_graph (
  latch VARCHAR(32) NULL,
  origid BIGINT UNSIGNED NULL,
  destid BIGINT UNSIGNED NULL,
  weight DOUBLE NULL,
  seq BIGINT UNSIGNED NULL,
  linkid BIGINT UNSIGNED NULL,
  KEY (latch, origid, destid) USING HASH,
  KEY (latch, destid, origid) USING HASH
)
ENGINE=OQGRAPH
data_table='oq_backing' origid='origid' destid='destid';
```

An older format (prior to MariaDB 10.0.7) has the latch field as a SMALLINT rather than a VARCHAR. The format is still valid, but gives an error by default:

```
CREATE TABLE oq_old (
  latch SMALLINT UNSIGNED NULL,
  origid BIGINT UNSIGNED NULL,
  destid BIGINT UNSIGNED NULL,
  weight DOUBLE NULL,
  seq BIGINT UNSIGNED NULL,
  linkid BIGINT UNSIGNED NULL,
  KEY (latch, origid, destid) USING HASH,
  KEY (latch, destid, origid) USING HASH
)
ENGINE=OQGRAPH
data_table='oq_backing' origid='origid' destid='destid';

ERROR 1005 (HY000): Can't create table `test`.`oq_old` (errno: 140 "Wrong create options")
```

The old, deprecated format can still be used if the value of the oqgraph_allow_create_integer_latch system variable is changed from its default, `FALSE`, to `TRUE`.

```
SET GLOBAL oqgraph_allow_create_integer_latch=1;

CREATE TABLE oq_old (
  latch SMALLINT UNSIGNED NULL,
  origid BIGINT UNSIGNED NULL,
  destid BIGINT UNSIGNED NULL,
  weight DOUBLE NULL,
  seq BIGINT UNSIGNED NULL,
  linkid BIGINT UNSIGNED NULL,
  KEY (latch, origid, destid) USING HASH,
  KEY (latch, destid, origid) USING HASH
)
ENGINE=OQGRAPH
data_table='oq_backing' origid='origid' destid='destid';
Query OK, 0 rows affected, 1 warning (0.19 sec)

SHOW WARNINGS;
+---------+------+-----------------------------------------------------------------------------
-----------------------------------------------------+
| Level   | Code | Message
|
+---------+------+-----------------------------------------------------------------------------
-----------------------------------------------------+
| Warning | 1287 | 'latch SMALLINT UNSIGNED NULL' is deprecated and will be removed in a future
release. Please use 'latch VARCHAR(32) NULL' instead |
+---------+------+-----------------------------------------------------------------------------
-----------------------------------------------------+
```

Data is only inserted into the backing table, not the OQGRAPH table.

Now, having created the `oq_graph` table linked to a backing table, it is now possible to query the `oq_graph` table directly. The `weight` field, since it was not specified in this example, defaults to `1`.

```
SELECT * FROM oq_graph;
+-------+--------+--------+--------+------+--------+
| latch | origid | destid | weight | seq  | linkid |
+-------+--------+--------+--------+------+--------+
|  NULL |      1 |      2 |      1 | NULL |   NULL |
|  NULL |      2 |      3 |      1 | NULL |   NULL |
|  NULL |      2 |      6 |      1 | NULL |   NULL |
|  NULL |      3 |      4 |      1 | NULL |   NULL |
|  NULL |      4 |      5 |      1 | NULL |   NULL |
|  NULL |      5 |      6 |      1 | NULL |   NULL |
+-------+--------+--------+--------+------+--------+
```

The data here represents one-directional starting and ending nodes. So node 2 has paths to node 3 and node 6, while node 6 has no paths to any other node.

Manipulating Weight

There are three fields which can be manipulated: `origid`, `destid` (the example above uses these two), as well as `weight`. To create a backing table with a `weight` field as well, the following syntax can be used:

```
CREATE TABLE oq2_backing (
  origid INT UNSIGNED NOT NULL,
  destid INT UNSIGNED NOT NULL,
  weight DOUBLE NOT NULL,
  PRIMARY KEY (origid, destid),
  KEY (destid)
);
```

```
INSERT INTO oq2_backing(origid, destid, weight)
 VALUES (1,2,1), (2,3,1), (3,4,3), (4,5,1), (2,6,10), (5,6,2);
```

```
CREATE TABLE oq2_graph (
  latch VARCHAR(32) NULL,
  origid BIGINT UNSIGNED NULL,
  destid BIGINT UNSIGNED NULL,
  weight DOUBLE NULL,
  seq BIGINT UNSIGNED NULL,
  linkid BIGINT UNSIGNED NULL,
  KEY (latch, origid, destid) USING HASH,
  KEY (latch, destid, origid) USING HASH
)
ENGINE=OQGRAPH
data_table='oq2_backing' origid='origid' destid='destid' weight='weight';
```

```
SELECT * FROM oq2_graph;
+-------+--------+--------+--------+------+--------+
| latch | origid | destid | weight | seq  | linkid |
+-------+--------+--------+--------+------+--------+
| NULL  |      1 |      2 |      1 | NULL |   NULL |
| NULL  |      2 |      3 |      1 | NULL |   NULL |
| NULL  |      2 |      6 |     10 | NULL |   NULL |
| NULL  |      3 |      4 |      3 | NULL |   NULL |
| NULL  |      4 |      5 |      1 | NULL |   NULL |
| NULL  |      5 |      6 |      2 | NULL |   NULL |
+-------+--------+--------+--------+------+--------+
```

See OQGRAPH Examples for OQGRAPH usage examples.

5.3.15.3 OQGRAPH Examples

Contents

Creating a Table with origid, destid Only

```
CREATE TABLE oq_backing (
  origid INT UNSIGNED NOT NULL,
  destid INT UNSIGNED NOT NULL,
  PRIMARY KEY (origid, destid),
  KEY (destid)
);
```

Some data can be inserted into the backing table to test with later:

```
INSERT INTO oq_backing(origid, destid)
 VALUES (1,2), (2,3), (3,4), (4,5), (2,6), (5,6);
```

Now the read-only OQGRAPH table is created.

From MariaDB 10.1.2 onwards you can use the following syntax:

```
CREATE TABLE oq_graph
ENGINE=OQGRAPH
data_table='oq_backing' origid='origid' destid='destid';
```

Prior to MariaDB 10.1.2, the CREATE statement must match the format below - any difference will result in an error.

```
CREATE TABLE oq_graph (
  latch VARCHAR(32) NULL,
  origid BIGINT UNSIGNED NULL,
  destid BIGINT UNSIGNED NULL,
  weight DOUBLE NULL,
  seq BIGINT UNSIGNED NULL,
  linkid BIGINT UNSIGNED NULL,
  KEY (latch, origid, destid) USING HASH,
  KEY (latch, destid, origid) USING HASH
)
ENGINE=OQGRAPH
data_table='oq_backing' origid='origid' destid='destid';
```

Creating a Table with Weight

For the examples on this page, we'll create a second OQGRAPH table and backing table, this time with `weight` as well.

```
CREATE TABLE oq2_backing (
  origid INT UNSIGNED NOT NULL,
  destid INT UNSIGNED NOT NULL,
  weight DOUBLE NOT NULL,
  PRIMARY KEY (origid, destid),
  KEY (destid)
);
```

```
INSERT INTO oq2_backing(origid, destid, weight)
 VALUES (1,2,1), (2,3,1), (3,4,3), (4,5,1), (2,6,10), (5,6,2);
```

```
CREATE TABLE oq2_graph (
  latch VARCHAR(32) NULL,
  origid BIGINT UNSIGNED NULL,
  destid BIGINT UNSIGNED NULL,
  weight DOUBLE NULL,
  seq BIGINT UNSIGNED NULL,
  linkid BIGINT UNSIGNED NULL,
  KEY (latch, origid, destid) USING HASH,
  KEY (latch, destid, origid) USING HASH
)
ENGINE=OQGRAPH
data_table='oq2_backing' origid='origid' destid='destid' weight='weight';
```

Shortest Path

A `latch` value of `'dijkstras'` and an `origid` and `destid` is used for finding the shortest path between two nodes, for example:

```
SELECT * FROM oq_graph WHERE latch='breadth_first' AND origid=1 AND destid=6;
+----------+--------+--------+--------+------+--------+
| latch    | origid | destid | weight | seq  | linkid |
+----------+--------+--------+--------+------+--------+
| dijkstras|      1 |      6 |   NULL |    0 |      1 |
| dijkstras|      1 |      6 |      1 |    1 |      2 |
| dijkstras|      1 |      6 |      1 |    2 |      6 |
+----------+--------+--------+--------+------+--------+
```

Note that nodes are uni-directional, so there is no path from node 6 to node 1:

```
SELECT * FROM oq_graph WHERE latch='dijkstras' AND origid=6 AND destid=1;
Empty set (0.00 sec)
```

Using the GROUP_CONCAT function can produce more readable results, for example:

```
SELECT GROUP_CONCAT(linkid ORDER BY seq) AS path FROM oq_graph
 WHERE latch='dijkstras' AND origid=1 AND destid=6;
+-------+
| path  |
+-------+
| 1,2,6 |
+-------+
```

Using the table `oq2_graph`, the shortest path is different:

```
SELECT GROUP_CONCAT(linkid ORDER BY seq) AS path FROM oq2_graph
 WHERE latch='dijkstras' AND origid=1 AND destid=6;
+-------------+
| path        |
+-------------+
| 1,2,3,4,5,6 |
+-------------+
```

The reason is the weight between nodes 2 and 6 is `10` in `oq_graph2`, so the shortest path taking into account `weight` is now across more nodes.

Possible Destinations

```
SELECT GROUP_CONCAT(linkid) AS dests FROM oq_graph WHERE latch='dijkstras' AND origid=2;
+-----------+
| dests     |
+-----------+
| 5,4,6,3,2 |
+-----------+
```

Note that this returns all possible destinations along the path, not just immediate links.

Leaf Nodes

MariaDB 10.3.3

Support for the `leaves` latch value was introduced in MariaDB 10.3.3.

A `latch` value of `'leaves'` and either `origid` or `destid` is used for finding leaf nodes at the beginning or end of a graph.

```
INSERT INTO oq_backing(origid, destid)
 VALUES (1,2), (2,3), (3,5), (4,5), (5,6), (6,7), (6,8), (2,8);
```

For example, to find all reachable nodes from `origid` that only have incoming edges:

```
SELECT * FROM oq_graph WHERE latch='leaves' AND origid=2;
+--------+--------+--------+--------+------+--------+
| latch  | origid | destid | weight | seq  | linkid |
+--------+--------+--------+--------+------+--------+
| leaves |      2 |   NULL |      4 |    2 |      7 |
| leaves |      2 |   NULL |      1 |    1 |      8 |
+--------+--------+--------+--------+------+--------+
```

And to find all nodes from which a path can be found to `destid` that only have outgoing edges:

```
SELECT * FROM oq_graph WHERE latch='leaves' AND destid=5;
+--------+--------+--------+--------+------+--------+
| latch  | origid | destid | weight | seq  | linkid |
+--------+--------+--------+--------+------+--------+
| leaves |   NULL |      5 |      3 |    2 |      1 |
| leaves |   NULL |      5 |      1 |    1 |      4 |
+--------+--------+--------+--------+------+--------+
```

Summary of Implemented Latch Commands

Latch	Alternative	additional where clause fields	Graph operation
NULL	(unspecified)	(none)	List original data
(empty string)	0	(none extra)	List all vertices in linkid column
(empty string)	0	origid	List all first hop vertices from origid in linkid column
dijkstras	1	origid, destid	Find shortest path using Dijkstras algorithm between origid and destid, with traversed vertex ids in linkid column
dijkstras	1	origid	Find all vertices reachable from origid, listed in linkid column, and report sum of weights of vertices on path to given vertex in weight
dijkstras	1	destid	Find all vertices from which a path can be found to destid, listed in linkid column, and report sum of weights of vertices on path to given vertex in weight
breadth_first	2	origid	List vertices reachable from origid in linkid column
breadth_first	2	destid	List vertices from which a path can be found to destid in linkid column
breadth_first	2	origid, destid	Find shortest path between origid and destid, report in linkid column
leaves	4	origid	List vertices reachable from origid, that only have incoming edges (from MariaDB 10.3.3)
leaves	4	destid	List vertices from which a path can be found to destid, that only have outgoing edges (from MariaDB 10.3.3)
leaves	4	origid, destid	Not supported, will return an empty result

Note: the use of integer latch commands is deprecated and may be phased out in a future release. Currently, numeric values in the strings are interpreted as aliases, and use of an integer column can be optionally allowed, for the latch commands column.

The use of integer latches is controlled using the oqgraph_allow_create_integer_latch system variable.

5.3.15.4 Compiling OQGRAPH

To compile OQGraph v2 you need to have the boost library with the versions not earlier than 1.40 and not later than 1.55 and gcc version not earlier than 4.5.

> MariaDB starting with 10.0.7
> OQGraph v3 compiles fine with the newer boost libraries, but it additionally needs the Judy library installed.

When all build prerequisites are met, OQGraph is enabled and compiled automatically. To enable or disable OQGRAPH explicitly, see the generic plugin build instructions.

Finding Out Why OQGRAPH Didn't Compile

If OQGRAPH gets compiled properly, there should be a file like:

```
storage/oqgraph/ha_oqgraph.so
```

If this is not the case, then you can find out if there is any modules missing that are required by OQGRAPH by doing:

```
cmake . -LAH | grep -i oqgraph
```

5.3.15.5 Building OQGRAPH Under Windows

OQGRAPH v3 can be built on Windows.

> MariaDB starting with 10.0.11
> This has been tested using Windows 7, Microsoft Visual Studio Express 2010 (32-bit), Microsoft Windows 64-bit Platform SDK 7.1 (64-bit), the Boost library >= 1.55 and Judy 1.0.5. Probably other recent versions of Boost, Judy or MSVC may work but these combinations have not been tested.

- Download the source package for Boost 1.55 from the Boost project website, http://www.boost.org
- Download the source package for Judy 1.05 via http://judy.sourceforge.net/
- Follow the documented instructions for building under Windows from the command line: Building_MariaDB_on_Windows
  - Ensure that the following variable is set to CMAKE: `JUDY_ROOT=path\to\judy\unzipped`
  - See also comments in `storage/oqgraph/cmake/FindJudy.cmake`

5.3.15.6 OQGRAPH System and Status Variables

Contents

This page documents system and status variables related to the OQGRAPH storage engine. See Server Status Variables and Server System Variables for complete list of all system and status variables.

System Variables

`oqgraph_allow_create_integer_latch`

- **Description:** Created when the OQGRAPH storage engine is installed, if set to `1` (`0` is default), permits the `latch` field to be an integer (see OQGRAPH Overview).
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `0`

Status Variables

`Oqgraph_boost_version`

- **Description:** OQGRAPH boost version.
- **Scope:** Global, Session
- **Data Type:** `string`

`Oqgraph_compat_mode`

- **Description:** Whether or not legacy tables with integer latches are supported.
- **Scope:** Global, Session
- **Data Type:** `string`

`Oqgraph_verbose_debug`

- **Description:** Whether or not verbose debugging is enabled. If it is, performance may be adversely impacted
- **Scope:** Global, Session
- **Data Type:** `string`

5.3.16 S3 Storage Engine

MariaDB starting with 10.5

The S3 storage engine has been available since MariaDB 10.5.4.

S3 is a read-only storage engine that stores its data in Amazon S3.

Using the S3 Storage Engine

Using the S3 storage engine.

Testing the Connections to S3

Steps to help verify where an S3 problem could be.

S3 Storage Engine Internals

The S3 storage engine is based on the Aria code.

aria_s3_copy

Copies an Aria table to and from S3.

S3 Storage Engine Status Variables

S3 Storage Engine-related status variables.

S3 Storage Engine System Variables

S3 Storage Engine-related system variables.

There are 4 related questions.

5.3.16.1 Using the S3 Storage Engine

MariaDB starting with 10.5

The S3 storage engine has been available since MariaDB 10.5.4.

Contents

The S3 storage engine is read only and allows one to archive MariaDB tables in Amazon S3, or any third-party public or private cloud that implements S3 API (of which there are many), but still have them accessible for reading in MariaDB.

Installing the Plugin

As of MariaDB 10.5.7, the S3 storage engine is currently gamma maturity, so the following step can be omitted.

On earlier releases, when it was alpha maturity, it will not load by default on a stable release of the server due to the default value of the plugin_maturity variable. Set to `alpha` (or below) in your config file to permit installation of the plugin:

```
[mysqld]
plugin-maturity = alpha
```

and restart the server.

Now install the plugin library, for example:

```
INSTALL SONAME 'ha_s3';
```

If the library is not available, for example:

```
INSTALL SONAME 'ha_s3';
ERROR 1126 (HY000): Can't open shared library '/var/lib/mysql/lib64/mysql/plugin/ha_s3.so'
  (errno: 13, cannot open shared object file: No such file or directory)
```

you may need to install a separate package for the S3 storage engine, for example:

```
shell> yum install MariaDB-s3-engine
```

Moving Data to S3

To move data from an existing table to S3, one can run:

```
ALTER TABLE old_table ENGINE=S3 COMPRESSION_ALGORITHM=zlib
```

To get data back to a 'normal' table one can do:

```
ALTER TABLE s3_table ENGINE=INNODB
```

New Options for ALTER TABLE

- `S3_BLOCK_SIZE` : Set to 4M as default. This is the block size for all index and data pages stored in S3.
- `COMPRESSION_ALGORITHM` : Set to 'none' as default. Which compression algorithm to use for block stored in S3. Options are: `none` or `zlib` .

ALTER TABLE can be used on S3 tables as normal to add columns or change column definitions.

mysqld Startup Options for S3

To be able to use S3 for storage one ***must*** define how to access S3 and where data are stored in S3:

- **s3_access_key**: The AWS access key to access your data
- **s3_secret_key**: The AWS secret key to access your data
- **s3_bucket**: The AWS bucket where your data should be stored. All MariaDB table data is stored in this bucket.
- **s3_region**: The AWS region where your data should be stored.

If you are using an S3 service that is using HTTP to connect (like https://min.io/) you also need the set the following variables:

- **s3_port**: Port number to connect to (0 means use default)
- **s3_use_http**: If true, force use of HTTP protocol

If you are going to use a primary-replica setup, you should look at the following variables:

- **s3_replicate_alter_as_create_select**: When converting an S3 table to local table, log all rows in binary log. Defaults to `TRUE` . This allows the replica to replicate `CREATE TABLE .. SELECT FROM s3_table` even it the replica doesn't have access to the original `s3_table` .
- **s3_slave_ignore_updates**: Should be set if primary and replica share the same S3 instance. This tells the replica that it can ignore any updates to the S3 tables as they are already applied on the primary. Defaults to `FALSE` .

The above defaults assume that the primary and replica don't share the same S3 instance.

Other, less critical options, are:

- **s3_host_name**: Hostname for the S3 service. "s3.amazonaws.com", Amazon S3 service, by default.
- **s3_protocol_version**: Protocol used to communication with S3. One of "Auto", "Amazon" or "Original" where "Auto" is the default. If you get errors like "8 Access Denied" when you are connecting to another service provider, then try to change this option. The reason for this variable is that Amazon has changed some parts of the S3 protocol since they originally introduced it but other service providers are still using the original protocol.
- **s3_block_size**: Set to 4M as default. This is the default block size for a table, if not specified in CREATE TABLE.
- **s3_pagecache_buffer_size**: Default 128M. The size of the buffer used for data and index blocks for S3 tables. Increase this to get better index handling (for all reads and multiple writes) to as much as you can afford.

Last some options you probably don't have to ever touch:

- **s3_pagecache_age_threshold** : Default 300: This characterizes the number of hits a hot block has to be untouched until it is considered aged enough to be downgraded to a warm block. This specifies the percentage ratio of that number of hits to the total number of blocks in the page cache.
- **s3_pagecache_division_limit**: Default 100. The minimum percentage of warm blocks in key cache.
- **s3_pagecache_file_hash_size**: Default 512. Number of hash buckets for open files. If you have a lot of S3 files open you should increase this for faster flush of changes. A good value is probably 1/10 of number of possible open S3 files.
- **s3_debug**: Default 0. Generates a trace file from libmarias3 on stderr (mysqld.err) for debugging the S3 protocol.

Typical my.cnf Entry for connecting to Amazon S3 service

```
[mariadb]
s3=ON
s3-bucket=mariadb
s3-access-key=xxxx
s3-secret-key=xxx
s3-region=eu-north-1
s3-host-name=s3.amazonaws.com
# The following is useful if you want to use minio as a S3 server. (https://min.io/)
#s3-port=9000
#s3-use-http=ON

# Primary and replica share same S3 tables.
s3-slave-ignore-updates=1

[aria_s3_copy]
s3-bucket=mariadb
s3-access-key=xxxx
s3-secret-key=xxx
s3-region=eu-north-1
s3-host-name=s3.amazonaws.com
# The following is useful if you want to use minio as a S3 server. (https://min.io/)
#s3-port=9000
#s3-use-http=ON
```

Typical my.cnf entry for connecting to a minio S3 server

```
[mariadb]
s3=ON
s3-host-name="127.0.0.1"
s3-bucket=storage-engine
s3-access-key=minio
s3-secret-key=minioadmin
s3-port=9000
s3-use-http=ON

[aria_s3_copy]
s3=ON
s3-host-name="127.0.0.1"
s3-bucket=storage-engine
s3-access-key=minio
s3-secret-key=minioadmin
s3-port=9000
s3-use-http=ON
```

Typical Usage Case for S3 Tables

The typical use case would be that there exists tables that after some time would become fairly inactive, but are still important so that they can not be removed. In that case, an option is to move such a table to an archiving service, which is accessible through an S3 API.

Notice that S3 means the Cloud Object Storage API defined by Amazon AWS. Often the whole of Amazon's Cloud Object Storage is referred to as S3. In the context of the S3 archive storage engine, it refers to the API itself that defines how to store objects in a cloud service, being it Amazon's or someone else's. OpenStack for example provides an S3 API for storing objects.

The main benefit of storing things in an S3 compatible storage is that the cost of storage is much cheaper than many other alternatives. Many S3 implementations also provide reliable long-term storage.

Operations Allowed on S3 Tables

- ALTER TABLE S3 supports all types, keys and other options that are supported by the Aria engine. One can also perform ALTER TABLE on an S3 table to add or modify columns etc.
- DROP TABLE
- SELECT Any SELECT operations you can perform on a normal table should work with an S3 table.
- SHOW TABLES will show all tables that exist in the current defined S3 location.
- S3 tables can be part of partitions. See Discovery below.

Discovery

The S3 storage engine supports full MariaDB discovery. This means that if you have the S3 storage engine enabled and properly configured, the table stored in S3 will automatically be discovered when it's accessed with SHOW TABLES, SELECT or any other operation that tries to access it. In the case of SELECT, the .frm file from S3 will be copied to the local storage to speed up future accesses.

When an S3 table is opened for the first time (it's not in the table cache) and there is a local .frm file, the S3 engine will check if it's still relevant, and if not, update or delete the .frm file.

This means that if the table definition changes on S3 and it's in the local cache, one has to execute `FLUSH TABLES` to get MariaDB to notice the change and update the .frm file.

If partitioning S3 tables are used, the partition definitions will also be stored on S3 storage and will be discovered by other servers.

Discovery of S3 tables is not done for tables in the mysql databases to make mysqld boot faster and more securely.

Replication

S3 works with replication. One can use replication in two different scenarios:

- The primary and replica share the same S3 storage. In this case the primary will make all changes to the S3 data and the replica will ignore any changes in the replication stream to S3 data . This scenario is achieved by setting s3_slave_ignore_updates to 1.
- The primary and replica don't share the same S3 storage or the replica uses another storage engine for the S3 tables. This scenario is achieved by setting s3_slave_ignore_updates to 0.

aria_s3_copy

aria_s3_copy is an external tool that one can use to copy Aria tables to and from S3. Use `aria_s3_copy --help` to get the options of how to use it.

mariadb-dump

- mariadb-dump will by default ignore S3 tables. If `mariadb-dump` is run with the `--copy-s3-tables` option, the resulting file will contain a CREATE statement for a similar Aria table, followed by the table data and ending with an `ALTER TABLE xxx ENGINE=S3` .

ANALYZE TABLE

As of MariaDB 10.5.14, ANALYZE TABLE is supported for S3 tables. As the S3 tables are read-only, a normal `ANALYZE TABLE` will not do anything. However using `ANALYZE TABLE table_name PERSISTENT FOR...` will now work.

CHECK TABLE

As of MariaDB 10.5.14, CHECK TABLE will work. As S3 tables are read only it is very unlikely that they can become corrupted. The only known way an S3 table could be corrupted if either the original table copied to S3 was corrupted or the process of copying the original table to S3 was somehow interrupted.

Current Limitations

- mysql-test-run doesn't by default test the S3 engine as we can't embed AWS keys into mysql-test-run.
- Replicas should not access S3 tables while they are ALTERed! This is because there is no locking implemented to S3 between servers. However, after a table (either the original S3 table or the partitioned S3 table) is changed on the primary, the replica will notice this on the next access and update its local definition.

Limitations in ALTER .. PARTITION

All ALTER PARTITION operations are supported on S3 partitioning tables except:

- REBUILD PARTITION
- TRUNCATE PARTITION
- REORGANIZE PARTITION

Performance Considerations

Depending on your connection speed to your S3 provider, there can be some notable slowdowns in some operations.

Discovery

As S3 is supporting discovery (automatically making tables available that are in S3) this can cause some small performance problems if the S3 engine is enabled. Partitioning S3 tables also support discovery.

- CREATE TABLE is a bit slower as the S3 engine has to check if the to-be-created table is already S3.
- Queries on information_schema tables are slower as S3 has to check if there is new tables in S3.
- DROP of non existing tables are slower as S3 has to check if the table is in S3.

There are no performance degradation's when accessing existing tables on the server. Accessing the S3 table the first time will copy the .frm file from S3 to the local disk, speeding up future accesses to the table.

Caching

- Accessing a table on S3 can take some time , especially if you are using big packets (s3_block_size). However the second access to the same data should be fast as it's then cached in the S3 page cache.

Things to Try to Increase Performance

If you have performance problems with the S3 engine, here are some things you can try:

- Decreasing s3_block_size. This can be done both globally and per table.
- Use COMPRESSION_ALGORITHM=zlib when creating the table. This will decrease the amount of data transferred from S3 to the local cache.
- Increasing the size of the s3 page cache: s3_pagecache_buffer_size

Try also to execute the query twice to check if the problem is that the data was not properly cached. When data is cached locally the performance should be excellent.

Future Development Ideas

- Store aws keys and region in the mysql.servers table (as Spider and FederatedX). This will allow one to have different tables on different S3 servers.
- Store s3 bucket, access_key and secret key in a cache to better be able to better to reuse connections. This would save some memory and make some S3 accesses a bit faster as we could reuse old connections.

Troubleshooting S3 on SELinux

If you get errors such as:

```
ERROR 3 (HY000): Got error from put_object(bubu/produkt/frm): 5 Couldn't connect to server
```

one reason could be that your system doesn't allow MariaDB to connect to ports other than 3306. To procedure to enable other ports is the following:

Search for the ports allowed for MariaDB:

```
$ sudo semanage port -l | grep mysqd_port_t
mysqld_port_t                  tcp    1186, 3306, 63132-63164
```

Say you want to allow MariaDB to connect to port 32768:

```
$ sudo semanage port -a -t mysqld_port_t -p tcp 32768
```

You can verify that the new port, 32768, is now allowed for MariaDB:

```
$ sudo semanage port -l | grep mysqd_port_t
mysqld_port_t                  tcp    32768,1186, 3306, 63132-63164
```

5.3.16.2 Testing the Connections to S3

MariaDB starting with 10.5

The S3 storage engine has been available since MariaDB 10.5.4.

Contents

If you can't get the S3 storage engine to work, here are some steps to help verify where the problem could be.

S3 Connection Variables

In most cases the problem is to correctly set the S3 connection variables.

The variables are:

- **s3_access_key**: The AWS access key to access your data
- **s3_secret_key**: The AWS secret key to access your data
- **s3_bucket**: The AWS bucket where your data should be stored. All MariaDB table data is stored in this bucket.
- **s3_region**: The AWS region where your data should be stored.
- **s3_host_name**: Hostname for the S3 service.
- **s3_protocol_version**: Protocol used to communicate with S3. One of "Amazon" or "Original"

There are several ways to ensure you get them right:

Using aria_s3_copy to Test the Connection

aria_s3_copy is a tool that allows you to copy aria tables to and from S3. It's useful for testing the connection as it allows you to specify all s3 options on the command line.

Execute the following sql commands to create a trivial sql table:

```
use test;
create table s3_test (a int) engine=aria row_format=page transactional=0;
insert into s3_test values (1),(2);
flush tables s3_test;
```

Now you can use the aria_s3_copy tool to copy this to S3 from your shell/the command line:

```
shell> cd mariadb-data-directory/test
shell> aria_s3_copy --op=to --verbose --force --**other*options* s3_test.frm

Copying frm file s3_test.frm
Copying aria table: test.s3_test to s3
Creating aria table information test/s3_test/aria
Copying index information test/s3_test/index
Copying data information test/s3_test/data
```

As you can see from the above, aria_s3_copy is using the current directory as the database name.

You can also set the aria_s3_copy options in your my.cnf file to avoid some typing.

Using mysql-test-run to Test the Connection and the S3 Storage Engine

One can use the MariaDB test system to run all default S3 test against your S3 storage.

To do that you have to locate the `mysql-test` directory in your system and `cd` to it.

The config file for the S3 test system can be found at `suite/s3/my.cnf`. To enable testing you have to edit this file and add the s3 connection options to the end of the file. It should look something like this after editing:

```
!include include/default_mysqld.cnf
!include include/default_client.cnf

[mysqld.1]
s3=ON
#s3-host-name=s3.amazonaws.com
#s3-protocol-version=Amazon
s3-bucket=MariaDB
s3-access-key=
s3-secret-key=
s3-region=
```

You must give values for `s3-access-key`, `s3-secret-key` and `s3-region` that reflects your S3 provider. The `s3-bucket` name is defined by your administrator.

If you are not using Amazon Web Services as your S3 provider you must also specify `s3-hostname` and possibly change `s3-protocol-version` to "Original".

Now you can test the configuration:

```
shell> cd **mysql-test** directory
shell> ./mysql-test-run --suite=s3
...
s3.no_s3                                 [ pass ]      5
s3.alter                                 [ pass ]  11073
s3.arguments                             [ pass ]   2667
s3.basic                                 [ pass ]   2757
s3.discovery                             [ pass ]   7851
s3.select                                [ pass ]   1325
s3.unsupported                           [ pass ]    363
```

Note that there may be more tests in your output as we are constantly adding more tests to S3 when needed.

What to Do Next

When you got the connection to work, you should add the options to your global my.cnf file. Now you can start testing S3 from your mysql command client by converting some existing table to S3 with ALTER TABLE ... ENGINE=S3.

5.3.16.3 S3 Storage Engine Internals

MariaDB starting with 10.5

The S3 storage engine has been available since MariaDB 10.5.4.

Contents

The S3 storage engine is based on the Aria code. Internally the S3 storage inherits from the Aria code, with hooks that change reads, so that instead of reading data from the local disk it reads things from S3.

The S3 engine uses it's own page cache, modified to be able to handle reading blocks from S3 (of size `s3_block_size`). Internally the S3 page cache uses pages of aria-block-size for splitting the blocks read from S3.

ALTER TABLE

ALTER TABLE will first create a local table in the normal Aria on disk format and then move both index and data to S3 in buckets of S3_BLOCK_SIZE. The .frm file is also copied to S3 for discovery to support discovery for other MariaDB servers. One can also use ALTER TABLE to change the structure of an S3 table.

Partitioning Tables

Starting from MariaDB 10.5.3, S3 tables can also be used with Partitioning tables. All ALTER PARTITION operations are supported except:

- REBUILD PARTITION
- TRUNCATE PARTITION
- REORGANIZE PARTITION

Big Reads

One of the properties of many S3 implementations is that they favor large reads. It's said that 4M gives the best performance, which is why the default value for `S3_BLOCK_SIZE` is 4M.

Compression

If compression (`COMPRESSION_ALGORITHM=zlib`) is used, then all index blocks and data blocks are compressed. The `.frm` file and Aria definition header (first page/pages in the index file) are not compressed as these are used by discovery/open.

If compression is used, then the local block size is `S3_BLOCK_SIZE` , but the block stored in S3 will be the size of the compressed block.

Typical compression we have seen is in the range of 80% saved space.

Structure Stored on S3

The table will be copied in S3 into the following locations:

```
frm file (for discovery):
s3_bucket/database/table/frm

First index block (contains description of the Aria file):
s3_bucket/database/table/aria

Rest of the index file:
s3_bucket/database/table/index/block_number

Data file:
s3_bucket/database/table/data/block_number
```

block_number is a 6-digit decimal number, prefixed with 0 (Can be larger than 6 numbers, the prefix is just for nice output)

Using the awsctl Python Tool to Examine Data

Installing awsctl on Linux

```
# install python-pip (on an OpenSuse distribution)
# use the appropriate command for your distribution
zypper install python-pip
pip install --upgrade pip

# the following installs awscli tools in ~/.local/bin
pip install --upgrade --user awscli
export PATH=~/.local/bin:$PATH

# configure your aws credentials
aws configure
```

Using the awsctl Tool

One can use the `aws` python tool to see how things are stored on S3:

```
shell> aws s3 ls --recursive s3://mariadb-bucket/
2019-05-10 17:46:48       8192 foo/test1/aria
2019-05-10 17:46:49    3227648 foo/test1/data/000001
2019-05-10 17:46:48        942 foo/test1/frm
2019-05-10 17:46:48    1015808 foo/test1/index/000001
```

To delete an obsolete table `foo.test1` one can do:

```
shell> ~/.local/bin/aws s3 rm --recursive s3://mariadb-bucket/foo/test1
delete: s3://mariadb-bucket/foo/test1/aria
delete: s3://mariadb-bucket/foo/test1/data/000001
delete: s3://mariadb-bucket/foo/test1/frm
delete: s3://mariadb-bucket/foo/test1/index/000001
```

5.3.16.4 aria_s3_copy

MariaDB starting with 10.5

The S3 storage engine has been available since MariaDB 10.5.4.

`aria_s3_copy` is a tool for copying an Aria table to and from S3.

The Aria table must be non transactional and have ROW_FORMAT=PAGE.

For `aria_s3_copy` to work reliably, the table should not be changed by the MariaDB server during the copy, and one should have first performed FLUSH TABLES to ensure that the table is properly closed.

Example of properly created Aria table:

```
create table test1 (a int) transactional=0 row_format=PAGE engine=aria;
```

Note that ALTER TABLE table_name ENGINE=S3 will work for any kind of table. This internally converts the table to an Aria table and then moves it to S3 storage.

Main Arguments

Option	Description
-?, --help	Display this help and exit.
-k, --s3-access-key=name	AWS access key ID
-r, --s3-region=name	AWS region
-K, --s3-secret-key=name	AWS secret access key ID
-b, --s3-bucket=name	AWS prefix for tables
-h, --s3-host-name=name	Host name to S3 provider
-c, --compress	Use compression
-o, --op=name	Operation to execute. One of 'from_s3', 'to_s3' or 'delete_from_s3'
-d, --database=name	Database for copied table (second prefix). If not given, the directory of the table file is used
-B, --s3-block-size=#	Block size for data/index blocks in s3
-L, --s3-protocol-version=name	Protocol used to communication with S3. One of "Amazon" or "Original".
-f, --force	Force copy even if target exists
-v, --verbose	Write more information
-V, --version	Print version and exit.
-#, --debug[=name]	Output debug log. Often this is 'd:t:o,filename'.
--s3-debug	Output debug log from marias3 to stdout

Typical Configuration in a my.cnf File

```
[aria_s3_copy]
s3-bucket=mariadb
s3-access-key=xxxx
s3-secret-key=xxx
s3-region=eu-north-1
#s3-host-name=s3.amazonaws.com
#s3-protocol-version=Amazon
verbose=1
op=to
```

Example Usage

The following code will copy an existing Aria table named `test1` to S3. If the `--database` option is not given, then the directory name where the table files exist will be used as the database.

```
shell> aria_s3_copy --force --op=to --database=foo --compress --verbose --s3_block_size=4M test1
Delete of aria table: foo.test1
Delete of index information foo/test1/index
Delete of data information foo/test1/data
Delete of base information and frm
Copying frm file test1.frm
Copying aria table: foo.test1 to s3
Creating aria table information foo/test1/aria
Copying index information foo/test1/index
.
Copying data information foo/test1/data
.
```

When using `--verbose`, `aria_s3_copy` will write a dot for each #/79 part of the file copied.

5.3.16.5 S3 Storage Engine Status Variables

MariaDB starting with 10.5

The S3 storage engine has been available since MariaDB 10.5.4.

Contents

This page documents status variables related to the S3 storage engine. See Server Status Variables for a complete list of status variables that can be viewed with SHOW STATUS.

See also the Full list of MariaDB options, system and status variables.

`S3_pagecache_blocks_not_flushed`

- **Description:**
- **Scope:**
- **Data Type:**
- **Introduced:** MariaDB 10.5

`S3_pagecache_blocks_unused`

- **Description:**
- **Scope:**
- **Data Type:**
- **Introduced:** MariaDB 10.5

`S3_pagecache_blocks_used`

- **Description:**
- **Scope:**
- **Data Type:**
- **Introduced:** MariaDB 10.5

`S3_pagecache_reads`

- **Description:**
- **Scope:**
- **Data Type:**
- **Introduced:** MariaDB 10.5

5.3.16.6 S3 Storage Engine System Variables

MariaDB starting with 10.5

The S3 storage engine has been available since MariaDB 10.5.4.

Contents

This page documents system variables related to the S3 storage engine.

See Server System Variables for a complete list of system variables and instructions on setting system variables.

Also see the Full list of MariaDB options, system and status variables

Variables

`s3_access_key`

- **Description:** The AWS access key to access your data. See mysqld startup options for S3.
- **Commandline:** `--s3-access-key=val`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** String
- **Default Value:** (Empty)
- **Introduced:** MariaDB 10.5.4

`s3_block_size`

- **Description:** The default block size for a table, if not specified in CREATE TABLE. Set to 4M as default. See mysqld startup options for S3.
- **Commandline:** `--s3-block-size=#`

- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** Numeric
- **Default Value:** `4194304`
- **Range:** `4194304` to `16777216`
- **Introduced:** MariaDB 10.5.4

---

s3_bucket

- **Description:** The AWS bucket where your data should be stored. All MariaDB table data is stored in this bucket. See mysqld startup options for S3.
- **Commandline:** `--s3-bucket=val`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** String
- **Default Value:** `MariaDB`
- **Introduced:** MariaDB 10.5.4

---

s3_debug

- **Description:** Generates a trace file from libmarias3 on stderr (mysqld.err) for debugging the S3 protocol.
- **Commandline:** `--s3-debug{=0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** Boolean
- **Valid Values:** 0 or 1
- **Default Value:** `0`
- **Introduced:** MariaDB 10.5.4

---

s3_host_name

- **Description:** Hostname for the S3 service. "s3.amazonaws.com", Amazon S3 service, by default
- **Commandline:** `--s3-host-name=val`
- **Scope:** Globa;
- **Dynamic:** No
- **Data Type:** String
- **Default Value:** `s3.amazonaws.com`
- **Introduced:** MariaDB 10.5.4

---

s3_pagecache_age_threshold

- **Description:** This characterizes the number of hits a hot block has to be untouched until it is considered aged enough to be downgraded to a warm block. This specifies the percentage ratio of that number of hits to the total number of blocks in the page cache.
- **Commandline:** `--s3-pagecache-age-threshold=val`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** Numeric
- **Default Value:** `300`
- **Range:** `100` to `18446744073709551615`
- **Introduced:** MariaDB 10.5.4

---

s3_pagecache_buffer_size

- **Description:** The size of the buffer used for index blocks for S3 tables. Increase this to get better index handling (for all reads and multiple writes) to as much as you can afford. Size can be adjusted in blocks of `8192` .
- **Commandline:** `--s3-pagecache-buffer-size=val`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** Numeric

- **Default Value:** `134217728` (128M)
- **Range:** `33554432` to `18446744073709551615`
- **Introduced:** MariaDB 10.5.4

---

s3_pagecache_division_limit

- **Description:** The minimum percentage of warm blocks in key cache.
- **Commandline:** `--s3-pagecache-division-limit=val`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** Numeric
- **Default Value:** `100`
- **Range:** `1` to `100`
- **Introduced:** MariaDB 10.5.4

---

s3_pagecache_file_hash_size

- **Description:** Number of hash buckets for open files. Default 512. If you have a lot of S3 files open you should increase this for faster flush of changes. A good value is probably 1/10 of number of possible open S3 files.
- **Commandline:** `--s3-pagecache-file-hash-size=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** Numeric
- **Default Value:** `512`
- **Range:** `32` to `16384`
- **Introduced:** MariaDB 10.5.4

---

s3_port

- **Description:** The TCP port number on the S3 host to connect to. A values of 0 means determine automatically.
- **Commandline:** `--s3-port=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** Numeric
- **Default Value:** `0`
- **Range:** `0` to `65535`
- **Introduced:** MariaDB 10.5.7

---

s3_protocol_version

- **Description:** Protocol used to communication with S3. One of "Auto", "Amazon" or "Original" where "Auto" is the default. If you get errors like "8 Access Denied" when you are connecting to another service provider, then try to change this option. The reason for this variable is that Amazon has changed some parts of the S3 protocol since they originally introduced it but other service providers are still using the original protocol.
- **Commandline:** `--s3-protocol-version=val`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** Enum
- **Valid Values:** `Auto`, `Amazon` or `Original`
- **Default Value:** `Auto`
- **Introduced:** MariaDB 10.5.4

---

s3_region

- **Description:** The AWS region where your data should be stored. See mysqld startup options for S3.
- **Commandline:** `--s3-region=val`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** String
- **Default Value:** (Empty)

- **Introduced:** MariaDB 10.5.4

`s3_replicate_alter_as_create_select`

- **Description:** When converting S3 table to local table, log all rows in binary log. This allows the slave to replicate `CREATE TABLE .. SELECT FROM s3_table` even it the slave doesn't have access to the original `s3_table`.
- **Commandline:** `--s3-replicate-alter-as-create-select{=0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** Boolean
- **Default Value:** `1`
- **Introduced:** MariaDB 10.5.4

`s3_secret_key`

- **Description:** The AWS secret key to access your data. See mysqld startup options for S3.
- **Commandline:** `--s3-secret-key=val`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** String
- **Default Value:** (Empty)
- **Introduced:** MariaDB 10.5.4

`s3_slave_ignore_updates`

- **Description:** Should be set if master and slave share the same S3 instance. This tells the slave that it can ignore any updates to the S3 tables as they are already applied on the master.
- **Commandline:** `--s3-slave-ignore-updates{=0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** Boolean
- **Default Value:** `0`
- **Introduced:** MariaDB 10.5.4

`s3_use_http`

- **Description:** If enabled, HTTP will be used instead of HTTPS.
- **Commandline:** `--s3-use-http{=0|1}`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** Boolean
- **Default Value:** `0`
- **Introduced:** MariaDB 10.5.7

5.3.17 Sequence Storage Engine

Contents

This article is about the Sequence storage engine. For details about sequence objects, see Sequences.

A **Sequence** engine allows the creation of ascending or descending sequences of numbers (positive integers) with a given starting value, ending value and increment.

It creates completely virtual, ephemeral tables automatically when you need them. There is no way to create a Sequence table explicitly. Nor are they ever written to disk or create `.frm` files. They are read-only, transactional, and support XA.

Installing

The Sequence engine is installed by default, and SHOW ENGINES will list the Sequence storage engine as supported:

```
SHOW ENGINES\G
...
*************************** 5. row ***************************
      Engine: MyISAM
     Support: YES
     Comment: MyISAM storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 6. row ***************************
      Engine: SEQUENCE
     Support: YES
     Comment: Generated tables filled with sequential values
Transactions: YES
          XA: YES
  Savepoints: YES
*************************** 7. row ***************************
      Engine: MRG_MyISAM
     Support: YES
     Comment: Collection of identical MyISAM tables
Transactions: NO
          XA: NO
  Savepoints: NO

...
```

Usage and Examples

To use a Sequence table, you simply select from it, as in

```
SELECT * FROM seq_1_to_5;
+-----+
| seq |
+-----+
|   1 |
|   2 |
|   3 |
|   4 |
|   5 |
+-----+
```

To use a sequence in a statement, you select from the table named by a pattern **seq_** `FROM` **_to_** `TO` or **seq_** `FROM` **_to_** `TO` **_step_** `STEP` .

In the case of an odd step, the sequence will commence with the `FROM` , and end at the final result before `TO` .

```
SELECT * FROM seq_1_to_15_step_3;
+-----+
| seq |
+-----+
|   1 |
|   4 |
|   7 |
|  10 |
|  13 |
+-----+
```

A sequence can go backwards too. In this case the final value will always be the `TO` value, so that a descending sequence has the same values as an ascending sequence:

```
SELECT * FROM seq_5_to_1_step_2;
+-----+
| seq |
+-----+
|   5 |
|   3 |
|   1 |
+-----+
```

```
SELECT * FROM seq_15_to_1_step_3;
+-----+
| seq |
+-----+
|  13 |
|  10 |
|   7 |
|   4 |
|   1 |
+-----+
```

```
SELECT * FROM seq_15_to_2_step_3;
+-----+
| seq |
+-----+
|  14 |
|  11 |
|   8 |
|   5 |
|   2 |
+-----+
```

This engine is particularly useful with joins and subqueries. For example, this query finds all prime numbers below 50:

```
SELECT seq FROM seq_2_to_50 s1 WHERE 0 NOT IN
     (SELECT s1.seq % s2.seq FROM seq_2_to_50 s2 WHERE s2.seq <= sqrt(s1.seq));
+-----+
| seq |
+-----+
|   2 |
|   3 |
|   5 |
|   7 |
|  11 |
|  13 |
|  17 |
|  19 |
|  23 |
|  29 |
|  31 |
|  37 |
|  41 |
|  43 |
|  47 |
+-----+
```

And almost (without 2, the only even prime number) the same result with joins:

```
SELECT s1.seq FROM seq_2_to_50 s1 JOIN seq_2_to_50 s2
  WHERE s1.seq > s2.seq AND s1.seq % s2.seq <> 0
  GROUP BY s1.seq HAVING s1.seq - COUNT(*) = 2;
+-----+
| seq |
+-----+
|   3 |
|   5 |
|   7 |
|  11 |
|  13 |
|  17 |
|  19 |
|  23 |
|  29 |
|  31 |
|  37 |
|  41 |
|  43 |
|  47 |
+-----+
```

Sequence tables can also be useful in date calculations. For example, to find the day of the week that a particular date has fallen on over a 40 year period (perhaps for birthday planning ahead!):

```
SELECT DAYNAME('1980-12-05' + INTERVAL (seq) YEAR) day,
    '1980-12-05' + INTERVAL (seq) YEAR date FROM seq_0_to_40;
+-----------+------------+
| day       | date       |
+-----------+------------+
| Friday    | 1980-12-05 |
| Saturday  | 1981-12-05 |
| Sunday    | 1982-12-05 |
...
| Friday    | 2014-12-05 |
| Saturday  | 2015-12-05 |
| Monday    | 2016-12-05 |
| Tuesday   | 2017-12-05 |
| Wednesday | 2018-12-05 |
| Thursday  | 2019-12-05 |
| Saturday  | 2020-12-05 |
+-----------+------------+
```

Although Sequence tables can only directly make use of positive integers, they can indirectly be used to return negative results by making use of the CAST statement. For example:

```
SELECT CAST(seq AS INT) - 5 x FROM seq_5_to_1;
+----+
| x  |
+----+
|  0 |
| -1 |
| -2 |
| -3 |
| -4 |
+----+
```

CAST is required to avoid a `BIGINT UNSIGNED value is out of range` error.

Sequence tables, while virtual, are still tables, so they must be in a database. This means that a default database must be selected (for example, via the USE command) to be able to query a Sequence table. The information_schema database cannot be used as the default for a Sequence table.

Table Name Conflicts

If the SEQUENCE storage engine is installed, it is not possible to create a table with a name which follows the SEQUENCE pattern:

```
CREATE TABLE seq_1_to_100 (col INT) ENGINE = InnoDB;
ERROR 1050 (42S01): Table 'seq_1_to_100' already exists
```

However, a SEQUENCE table can be converted to another engine and the new table can be referred in any statement:

```
ALTER TABLE seq_1_to_100 ENGINE = BLACKHOLE;

SELECT * FROM seq_1_to_100;
Empty set (0.00 sec)
```

While a SEQUENCE table cannot be dropped, it is possible to drop the converted table. The SEQUENCE table with the same name will still exist:

```
DROP TABLE seq_1_to_100;

SELECT COUNT(*) FROM seq_1_to_100;
+----------+
| COUNT(*) |
+----------+
|      100 |
+----------+
1 row in set (0.00 sec)
```

A temporary table with a SEQUENCE-like name can always be created and used:

```
CREATE TEMPORARY TABLE seq_1_to_100 (col INT) ENGINE = InnoDB;

SELECT * FROM seq_1_to_100;
Empty set (0.00 sec)
```

5.3.18 SphinxSE

SphinxSE is a storage engine that talks to searchd (Sphinx daemon) to enable text searching.

About SphinxSE

About the Sphinx Storage Engine

Installing Sphinx

Installing the Sphinx daemon

Configuring Sphinx

Before you can get Sphinx working with the Sphinx Storage Engine on MariaDB...

Installing and Testing SphinxSE with MariaDB

How to install and test SphinxSE

Sphinx Status Variables

Sphinx status variables.

There are 3 related questions.

5.3.18.1 About SphinxSE

Contents

The Sphinx storage engine (SphinxSE) is a storage engine that talks to searchd (the Sphinx daemon) to enable text searching. Sphinx and SphinxSE is used as a faster and more customizable alternative to MariaDB's built-in full-text search.

Sphinx does not depend on MariaDB, and can run independently, but SphinxSE provides a convenient interface to the underlying Sphinx daemon.

Versions of SphinxSE in MariaDB

SphinxSE Version	Introduced	Maturity
SphinxSE 2.2.6	MariaDB 10.0.15	Stable
SphinxSE 2.1.9	MariaDB 10.0.14	Stable
SphinxSE 2.0.4	MariaDB 5.5	
SphinxSE 0.99	MariaDB 5.2 and MariaDB 5.3	

Enabling SphinxSE in MariaDB

The Sphinx storage engine is included in the source, binaries, and packages of MariaDB. SphinxSE is built as a dynamically loadable .so plugin. To use it, you need to perform a one-time install:

```
INSTALL SONAME 'ha_sphinx';
```

MariaDB until 10.0

In Debian/Ubuntu packages SphinxSE is statically compiled into the MariaDB server, there is no need to use the `INSTALL SONAME` statement.

Once installed, SphinxSE will show up in the list of installed storage engines:

```
SHOW ENGINES;
+------------+---------+----------------------------------------------+--------------+------+--
----------+
| Engine     | Support | Comment                                      | Transactions | XA   |
Savepoints |
+------------+---------+----------------------------------------------+--------------+------+--
----------+
...
| SPHINX     | YES     | Sphinx storage engine 0.9.9                  | NO           | NO   | NO
|
...
+------------+---------+----------------------------------------------+--------------+------+--
----------+
```

This is a one-time step and will not need to be performed again.

Note: SphinxSE is just the storage engine part of Sphinx. You will have to install Sphinx itself in order to make use of SphinxSE in MariaDB.

Despite the name, SphinxSE does not actually store any data itself. It is actually a built-in client which allows MariaDB to talk to Sphinx, run search queries, and obtain search results. All indexing and searching happen outside MariaDB.

Some SphinxSE applications include:

- easier porting of MariaDB/MySQL FTS applications to Sphinx

- allowing Sphinx use with programming languages for which native APIs are not available yet
- optimizations when additional Sphinx result set processing on the MariaDB side is required (eg. JOINs with original document tables, additional MariaDB-side filtering, and etc...)

Using SphinxSE

Basic Usage

To search via SphinxSE, you would need to create a special `ENGINE=SPHINX` "search table", and then `SELECT` from it with full text query put into the `WHERE` clause for query column.

Here is an example create statement and search query:

```
CREATE TABLE t1
(
    id          BIGINT UNSIGNED NOT NULL,
    weight      INTEGER NOT NULL,
    query       VARCHAR(3072) NOT NULL,
    group_id    INTEGER,
    INDEX(query)
) ENGINE=SPHINX CONNECTION="sphinx://127.0.0.1:9312/test1";

SELECT * FROM t1 WHERE query='test it;mode=any';
```

The first three columns of the search table must have a type of `BIGINT` for the 1st column (document id), `INTEGER` or `BIGINT` for the 2nd column (match weight), and `VARCHAR` or `TEXT` for the 3rd column (your query), respectively. This mapping is fixed; you cannot omit any of these three required columns, or move them around, or change types. Also, the query column must be indexed; all the others must be kept unindexed. Column names are ignored so you can use arbitrary ones.

Additional columns must be either `INTEGER`, `TIMESTAMP`, `BIGINT`, `VARCHAR`, or `FLOAT`. They will be bound to the attributes provided in the Sphinx result set by name, so their names must match the attribute names specified in `sphinx.conf`. If there's no such attribute name in the Sphinx search results, the additional columns will have `NULL` values.

Special "virtual" attribute names can also be bound to SphinxSE columns. `_sph_` needs to be used instead of `@` for that. For instance, to obtain the values of '`@groupby`', '`@count`', or '`@distinct`' virtual attributes, use '`_sph_groupby`', '`_sph_count`' or '`_sph_distinct`' column names, respectively.

The `CONNECTION` string parameter is used to specify the default `searchd` host, port, and indexes for queries issued using this table. If no connection string is specified in `CREATE TABLE`, index name '`*`' (ie. search all indexes) and '`127.0.0.1:9312`' are assumed. The connection string syntax is as follows:

```
CONNECTION="sphinx://HOST:PORT/INDEXNAME"
```

You can change the default connection string later like so:

```
ALTER TABLE t1 CONNECTION="sphinx://NEWHOST:NEWPORT/NEWINDEXNAME";
```

You can also override all these parameters per-query.

> **Note:** To use Linux sockets you can modify the searchd section of the Sphinx configuration file, setting the listen parameter to a socket file. Instruct SphinxSE about the socket using CONNECTION="unix:*unix/domain/socket[:index]".*

Search Options

As seen in the example above, both query text and search options should be put into the '`WHERE`' clause of the search query column (i.e. the 3rd column); the options are separated by semicolons ('`;`') and separate names from values using an equals sign ('`=`'). Any number of options can be specified. Available options are:

- query - query text;
- mode - matching mode. Must be one of "all", "any", "phrase", "boolean", or "extended". Default is "all";
- sort - match sorting mode. Must be one of "relevance", "attr_desc", "attr_asc", "time_segments", or "extended". In all modes besides "relevance" attribute name (or sorting clause for "extended") is also required after a colon:

```
... WHERE query='test;sort=attr_asc:group_id';
... WHERE query='test;sort=extended:@weight desc, group_id asc';
```

- offset - offset into result set, default is 0;
- limit - amount of matches to retrieve from result set, default is 20;
- index - names of the indexes to search:

```
... WHERE query='test;index=test1;';
... WHERE query='test;index=test1,test2,test3;';
```

- minid, maxid - min and max document ID to match;
- weights - comma-separated list of weights to be assigned to Sphinx full-text fields:

```
... WHERE query='test;weights=1,2,3;';
```

- filter, !filter - comma-separated attribute name and a set of values to match:

```
# only include groups 1, 5 and 19
... WHERE query='test;filter=group_id,1,5,19;';

# exclude groups 3 and 11
... WHERE query='test;!filter=group_id,3,11;';
```

- range, !range - comma-separated attribute name, min and max value to match:

```
# include groups from 3 to 7, inclusive
... WHERE query='test;range=group_id,3,7;';

# exclude groups from 5 to 25
... WHERE query='test;!range=group_id,5,25;';
```

- maxmatches - per-query max matches value:

```
... WHERE query='test;maxmatches=2000;';
```

- groupby - group-by function and attribute:

```
... WHERE query='test;groupby=day:published_ts;';
... WHERE query='test;groupby=attr:group_id;';
```

- groupsort - group-by sorting clause:

```
... WHERE query='test;groupsort=@count desc;';
```

- indexweights - comma-separated list of index names and weights to use when searching through several indexes:

```
... WHERE query='test;indexweights=idx_exact,2,idx_stemmed,1;';
```

- comment - a string to mark this query in query log (mapping to $comment parameter in Query() API call):

```
... WHERE query='test;comment=marker001;';
```

- select - a string with expressions to compute (mapping to SetSelect() API call):

```
... WHERE query='test;select=2*a+3*b as myexpr;';
```

Note: It is much more efficient to allow Sphinx to perform sorting, filtering, and slicing of the result set than to raise max matches count and use ' `WHERE` ', ' `ORDER BY` ', and ' `LIMIT` ' clauses on the MariaDB side. This is for two reasons:

1. Sphinx does a number of optimizations and performs better than MariaDB/MySQL on these tasks.
2. Less data would need to be packed by `searchd` , and transferred and unpacked by SphinxSE.

SHOW ENGINE SPHINX STATUS

Starting with version 0.9.9-rc1, additional query info besides the result set can be retrieved with the ' `SHOW ENGINE SPHINX STATUS` ' statement:

```
SHOW ENGINE SPHINX STATUS;
+--------+-------+----------------------------------------------------+
| Type   | Name  | Status                                             |
+--------+-------+----------------------------------------------------+
| SPHINX | stats | total: 25, total found: 25, time: 126, words: 2 |
| SPHINX | words | sphinx:591:1256 soft:11076:15945                |
+--------+-------+----------------------------------------------------+
```

This information can also be accessed through status variables. Note that this method does not require super-user privileges.

```
SHOW STATUS LIKE 'sphinx_%';
+--------------------+---------------------------------+
| Variable_name      | Value                           |
+--------------------+---------------------------------+
| sphinx_total       | 25                              |
| sphinx_total_found | 25                              |
| sphinx_time        | 126                             |
| sphinx_word_count  | 2                               |
| sphinx_words       | sphinx:591:1256 soft:11076:15945 |
+--------------------+---------------------------------+
```

JOINs with SphinxSE

You can perform `JOIN` s on a SphinxSE search table and tables using other engines. Here's an example with "documents" from example.sql:

```
SELECT content, date_added FROM test.documents docs
    JOIN t1 ON (docs.id=t1.id)
    WHERE query="one document;mode=any";
+-----------------------------------+---------------------+
| content                           | docdate             |
+-----------------------------------+---------------------+
| this is my test document number two | 2006-06-17 14:04:28 |
| this is my test document number one | 2006-06-17 14:04:28 |
+-----------------------------------+---------------------+

SHOW ENGINE SPHINX STATUS;
+--------+-------+-----------------------------------------------+
| Type   | Name  | Status                                        |
+--------+-------+-----------------------------------------------+
| SPHINX | stats | total: 2, total found: 2, time: 0, words: 2 |
| SPHINX | words | one:1:2 document:2:2                          |
+--------+-------+-----------------------------------------------+
```

Building snippets (excerpts) via MariaDB

Starting with version 0.9.9-rc2, SphinxSE also includes a UDF function that lets you create snippets through MariaDB. The functionality is fully similar to the BuildExcerprts API call but is accessible through MariaDB+SphinxSE.

MariaDB until 5.5

The binary that provides the UDF is named sphinx.so and is automatically built and installed to the proper location along with SphinxSE itself. Register the UDF using the following statement:

```
CREATE FUNCTION sphinx_snippets RETURNS STRING SONAME 'sphinx.so';
```

MariaDB until 10.0

The UDF is packed together with the storage engine, in the same binary named ha_sphinx.so. Register the UDF using the following statement:

```
CREATE FUNCTION sphinx_snippets RETURNS STRING SONAME 'ha_sphinx.so';
```

The function name must be ' `sphinx_snippets` ', you can not use an arbitrary name. Function arguments are as follows:

```
Prototype: function sphinx_snippets ( document, index, words, [options] );
```

Document and words arguments can be either strings or table columns. Options must be specified like this: <code>'value' AS option_name</code>. For a list of supported options, refer to the BuildExcerprts() API call. The only UDF-specific additional option is named 'sphinx' and lets you specify searchd location (host and port).

Usage examples:

```
SELECT sphinx_snippets('hello world doc', 'main', 'world',
    'sphinx://192.168.1.1/' AS sphinx, true AS exact_phrase,
    '[b]' AS before_match, '[/b]' AS after_match)
FROM documents;

SELECT title, sphinx_snippets(text, 'index', 'mysql php') AS text
    FROM sphinx, documents
    WHERE query='mysql php' AND sphinx.id=documents.id;
```

More Information

More information on Sphinx and SphinxSE is available on the Sphinx website.

5.3.18.2 Installing Sphinx

In order to use the Sphinx Storage Engine, it is necessary to install the Sphinx daemon.

Many Linux distributions have Sphinx in their repositories. These can be used to install Sphinx instead of following the instructions below, but these are usually quite old versions and don't all include API's for easy integration. Ubuntu users can use the updated repository at https://launchpad.net/~builds/+archive/sphinxsearch-rel21 (see instructions below). Alternatively, download from http://sphinxsearch.com/downloads/release/

Debian and Ubuntu

Ubuntu users can make use of the repository, as follows:

```
sudo add-apt-repository ppa:builds/sphinxsearch-rel21
sudo apt-get update
sudo apt-get install sphinxsearch
```

Alternatively, install as follows:

- The Sphinx package and daemon are named `sphinxsearch` .
- `sudo apt-get install unixodbc libpq5 mariadb-client`
- `sudo dpkg -i sphinxsearch*.deb`
- Configure Sphinx as required
- You may need to check `/etc/default/sphinxsearch` to see that `START=yes`
- Start with `sudo service sphinxsearch start` (and stop with `sudo service sphinxsearch stop`)

Red Hat and CentOS

- The package name is `sphinx` and the daemon `searchd` .
- `sudo yum install postgresql-libs unixODBC`
- `sudo rpm -Uhv sphinx*.rpm`
- Configure Sphinx as required
- `service searchd start`

Windows

- Unzip and extract the downloaded zip file
- Move the extracted directory to `C:\Sphinx`
- Configure Sphinx as required

- Install as a service:
  - `C:\Sphinx\bin> C:\Sphinx\bin\searchd --install --config C:\Sphinx\sphinx.conf.in --servicename SphinxSearch`

Once Sphinx has been installed, it will need to be configured.

Full instructions, including details on compiling Sphinx yourself, are available at http://sphinxsearch.com/docs/current.html .

5.3.18.3 Configuring Sphinx

Before you can get Sphinx working with the Sphinx Storage Engine on MariaDB, you need to configure it.

- The default configuration file is called `sphinx.conf` , usually located in `/etc/sphinxsearch` (Debian/Ubuntu), `/etc/sphinx/sphinx.conf.` (Red Hat/CentOS) or `C:\Sphinx\sphinx.conf` (Windows).

If it doesn't already exist, you can use the sample configuration file, `sphinx.conf.dist` . There is also sample data supplied that we can use for testing. Load the sample data (which creates two tables, `documents` and `tags` in the `test` database), for example:

`mysql -u test < /usr/local/sphinx/etc/example.sql` (Red Hat, CentOS) `mysql -u test < /usr/share/doc/sphinxsearch/example-conf/example.sql` (Debian/Ubuntu)

The sample configuration file documents the available options. You will need to make at least a few changes. A MariaDB user with permission to access the database must be created. For example:

```
CREATE USER 'sphinx'@localhost
  IDENTIFIED BY 'sphinx_password';
GRANT SELECT on test.* to 'sphinx'@localhost;
```

Add these details to the `mysql` section of the config file:

```
sql_host = localhost
sql_user = sphinx
sql_pass = sphinx_password
sql_db   = test
sql_port = 3306
```

On Windows, the `path` and `pid` lines will need to be changed to reflect a valid path, usually as follows:

```
path = C:\Sphinx\docsidx
...
pid_file = C:\Sphinx\sphinx.pid
```

The query in the configuration files is the query that will be used for building the index. In the sample data, this is:

```
sql_query = \
  SELECT id, group_id, UNIX_TIMESTAMP(date_added) AS date_added, title, content \
  FROM documents
```

5.3.18.4 Installing and Testing SphinxSE with MariaDB

To use SphinxSE with MariaDB you need to first download and install Sphinx.

Complete Sphinx documentation is available on the Sphinx website.

Tips for Installing Sphinx

libexpat

One library we know you will need on Linux before you can install Sphinx is `libexpat` . If it is not installed, you will get the warning `checking for libexpat... not found` . On Suse the package is called `libexpat-devel` , on Ubuntu the package is called `libexpat1-dev` .

MariaDB detection

If you run into problems with MariaDB not being detected, use the following options:

```
--with-mysql            compile with MySQL support (default is enabled)
--with-mysql-includes   path to MySQL header files
--with-mysql-libs       path to MySQL libraries
```

The above will tell the `configure` script where your MySQL/MariaDB installation is.

Testing Sphinx

After installing Sphinx, you can check that things are working in MariaDB by doing the following:

```
cd installation-dir/mysql-test
./mysql-test-run --suite=sphinx
```

If the above test doesn't pass, check the logs in the `'var'` directory. If there is a problem with the sphinx installation, the reason can probably be found in the log file at: `var/log/sphinx.sphinx/searchd/sphinx.log` .

3.3.7.25 Sphinx Status Variables

5.3.19 Spider

The Spider storage engine supports partitioning and xa transactions, and allows tables of different MariaDB instances to be handled as if they were on the same instance.

Versions of Spider in MariaDB

From MariaDB 10.9.2, the Spider version number matches the server version.

Spider Version	Introduced	Maturity
Spider 3.3.15	MariaDB 10.5.7, MariaDB 10.4.6	Stable
Spider 3.3.15	MariaDB 10.5.4	Gamma
Spider 3.3.14	MariaDB 10.4.3, MariaDB 10.3.13	Stable
Spider 3.3.13	MariaDB 10.3.7	Stable
Spider 3.3.13	MariaDB 10.3.3	Gamma
Spider 3.2.37	MariaDB 10.1.10, MariaDB 10.0.23	Gamma
Spider 3.2.21	MariaDB 10.1.5, MariaDB 10.0.18	Gamma
Spider 3.2.18	MariaDB 10.0.17	Gamma
Spider 3.2.11	MariaDB 10.0.14	Gamma
Spider 3.2.4	MariaDB 10.0.12	Gamma
Spider 3.2	MariaDB 10.0.11	Gamma
Spider 3.0	MariaDB 10.0.4	Beta

Spider Documentation

See the spider-2.0-doc repository for complete, older, documentation.

Presentation for new sharding features in Spider 3.3.

Spider Storage Engine Overview

Storage engine with sharding features.

Spider Installation

Setting up Spider.

Spider Storage Engine Core Concepts

Key Spider concepts

Spider Use Cases

Basic working examples for Spider

Spider Cluster Management

Spider cluster management.

Spider Feature Matrix

Matrix of Spider features

Spider System Variables

System variables for the Spider storage engine.

Spider Table Parameters

Spider table parameters available in the CREATE TABLE ... COMMENT clause

Spider Status Variables

Spider server status variables.

Spider Functions

User-defined functions available with the Spider storage engine.

Spider mysql Database Tables

System tables related to the Spider storage engine.

Information Schema SPIDER_ALLOC_MEM Table

Information about Spider's memory usage.

Information Schema SPIDER_WRAPPER_PROTOCOLS Table

Installed along with the Spider storage engine.

Spider Differences Between SpiderForMySQL and MariaDB

Spider differences between MySQL and MariaDB

Spider Case Studies

List of clients using Spider

Spider Benchmarks

Benchmarks for Spider

Spider FAQ

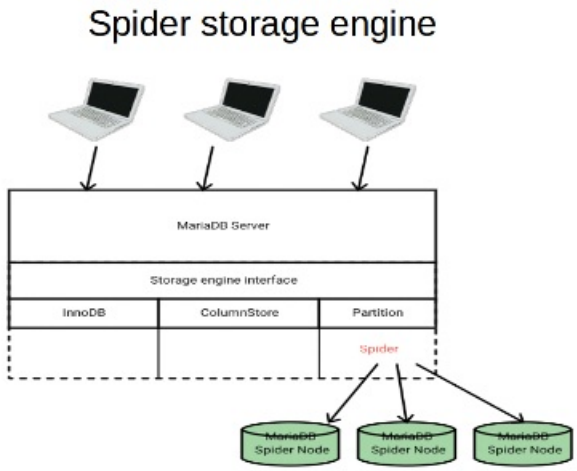
Frequently-asked questions about the Spider storage engine

There are 5 related questions.

5.3.19.1 Spider Storage Engine Overview

Contents

About



The Spider storage engine is a storage engine with built-in sharding features. It supports partitioning and xa transactions, and allows tables of different MariaDB instances to be handled as if they were on the same instance. It refers to one possible implementation of ISO/IEC 9075-9:2008 SQL/MED.

When a table is created with the Spider storage engine, the table links to the table on a remote server. The remote table can be of any storage engine. The table link is concretely achieved by the establishment of the connection from a local MariaDB server to a remote MariaDB server. The link is shared for all tables that are part of a the same transaction.

Spider Versions in MariaDB

Spider Version	Introduced	Maturity
Spider 3.3.15	MariaDB 10.5.7, MariaDB 10.4.6	Stable
Spider 3.3.15	MariaDB 10.5.4	Gamma
Spider 3.3.14	MariaDB 10.4.3, MariaDB 10.3.13	Stable
Spider 3.3.13	MariaDB 10.3.7	Stable
Spider 3.3.13	MariaDB 10.3.3	Gamma
Spider 3.2.37	MariaDB 10.1.10, MariaDB 10.0.23	Gamma
Spider 3.2.21	MariaDB 10.1.5, MariaDB 10.0.18	Gamma
Spider 3.2.18	MariaDB 10.0.17	Gamma
Spider 3.2.11	MariaDB 10.0.14	Gamma
Spider 3.2.4	MariaDB 10.0.12	Gamma
Spider 3.2	MariaDB 10.0.11	Gamma
Spider 3.0	MariaDB 10.0.4	Beta

Some Server Variables to Set When Using Spider

MariaDB starting with 10.3.4

If you are using Spider with replication, you can expand the list of transaction errors to be retried by setting slave_transaction_retry_errors to the following to avoid network problems:

- 1158: Got an error reading communication packets
- 1159: Got timeout reading communication packets
- 1160: Got an error writing communication packets
- 1161: Got timeout writing communication packets
- 1429: Unable to connect to foreign data source
- 2013: Lost connection to MySQL server during query
- 12701: Remote MySQL server has gone away

Do this as follows in your my.cnf file:

```
slave_transaction_retry_errors="1158,1159,1160,1161,1429,2013,12701"
```

From MariaDB 10.4.5, the above is included the default.

Usage

Basic Usage

To create a table in the Spider storage engine format, the COMMENT and/or CONNECTION clauses of the CREATE TABLE statement are used to pass connection information about the remote server.

For example, the following table exists on a remote server (in this example, the remote node was created with the MySQL Sandbox tool, an easy way to test with multiple installations)::

```
node1 >CREATE TABLE s(
  id INT NOT NULL AUTO_INCREMENT,
  code VARCHAR(10),
  PRIMARY KEY(id));
```

On the local server, a Spider table can be created as follows:

```
CREATE TABLE s(
  id INT NOT NULL AUTO_INCREMENT,
  code VARCHAR(10),
  PRIMARY KEY(id)
)
ENGINE=SPIDER
COMMENT='host "127.0.0.1", user "msandbox", password "msandbox", port "8607"';
```

Records can now be inserted on the local server, and they will be stored on the remote server:

```
INSERT INTO s(code) VALUES ('a');

node1 > SELECT * FROM s;
+----+------+
| id | code |
+----+------+
|  1 | a    |
+----+------+
```

MariaDB starting with 10.8.1

Alternative to specifying the data node information in the COMMENT, certain information (server, database, table) can also be specified using Table Options, like so:

```
CREATE SERVER srv FOREIGN DATA WRAPPER mysql OPTIONS(
  HOST '127.0.0.1',
  USER 'msandbox',
  PASSWORD 'msandbox',
  PORT 8607);

CREATE TABLE s(
  id INT NOT NULL AUTO_INCREMENT,
  code VARCHAR(10),
  PRIMARY KEY(id)
)
ENGINE=SPIDER REMOTE_SERVER="srv" REMOTE_DATABASE="db" REMOTE_TABLE="s";
```

Further Examples

Preparing 10M record table using the sysbench utility

```
/usr/local/skysql/sysbench/bin/sysbench --test=oltp  --db-driver=mysql  --mysql-table-
engine=innodb --mysql-user=skysql --mysql-password=skyvodka --mysql-host=192.168.0.202 --mysql-
port=5054 --oltp-table-size=10000000  --mysql-db=test  prepare
```

Make a first read only benchmark to check the initial single node performance.

```
/usr/local/skysql/sysbench/bin/sysbench --test=oltp  --db-driver=mysql  --mysql-table-
engine=innodb --mysql-user=skysql --mysql-password=skyvodka --mysql-host=192.168.0.202 --mysql-
port=5054 --mysql-db=test --oltp-table-size=10000000 --num-threads=4 --max-requests=100000 --
oltp-read-only=on run
```

```
sysbench 0.4.12:  multi-threaded system evaluation benchmark

Running the test with following options:
Number of threads: 4

Doing OLTP test.
Running mixed OLTP test
Doing read-only test
Using Special distribution (12 iterations,  1 pct of values are returned in 75 pct cases)
Using "BEGIN" for starting transactions
Using auto_inc on the id column
Maximum number of requests for OLTP test is limited to 100000
Threads started!
Done.

OLTP test statistics:
    queries performed:
        read:                            1400196
        write:                           0
        other:                           200028
        total:                           1600224
    transactions:                        100014 (1095.83 per sec.)
    deadlocks:                           0      (0.00 per sec.)
    read/write requests:                 1400196 (15341.58 per sec.)
    other operations:                    200028 (2191.65 per sec.)

Test execution summary:
    total time:                          91.2681s
    total number of events:              100014
    total time taken by event execution: 364.3693
    per-request statistics:
         min:                                  1.85ms
         avg:                                  3.64ms
         max:                                 30.70ms
         approx.  95 percentile:               4.66ms

Threads fairness:
    events (avg/stddev):           25003.5000/84.78
    execution time (avg/stddev):   91.0923/0.00
```

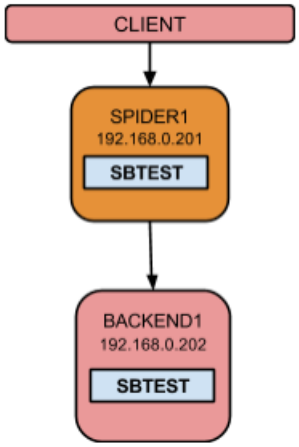
Define an easy way to access the nodes from the MariaDB or MySQL client.

```
alias backend1='/usr/local/skysql/mysql-client/bin/mysql  --user=skysql --password=skyvodka --
host=192.168.0.202 --port=5054'
alias backend2='/usr/local/skysql/mysql-client/bin/mysql  --user=skysql --password=skyvodka --
host=192.168.0.203 --port=5054'
alias spider1='/usr/local/skysql/mysql-client/bin/mysql  --user=skysql --password=skyvodka --
host=192.168.0.201 --port=5054'
```

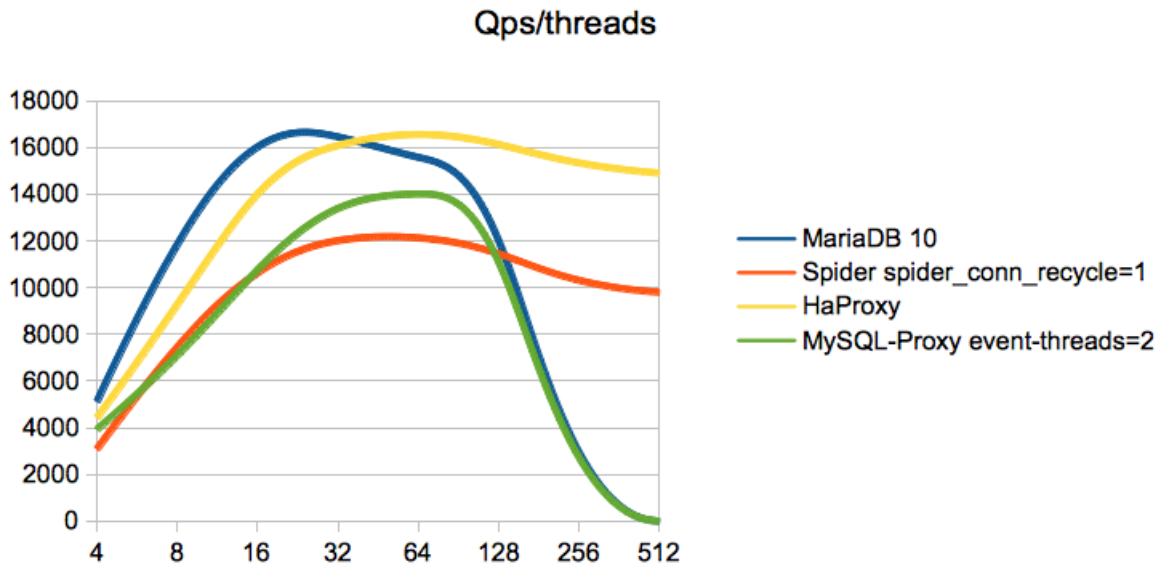
Create the empty tables to hold the data and repeat for all available backend nodes.

```
backend1 << EOF
CREATE DATABASE backend;
CREATE TABLE backend.sbtest (
  id int(10) unsigned NOT NULL AUTO_INCREMENT,
  k int(10) unsigned NOT NULL DEFAULT '0',
  c char(120) NOT NULL DEFAULT '',
  pad char(60) NOT NULL DEFAULT '',
  PRIMARY KEY (id),
  KEY k (k)
) ENGINE=InnoDB;
EOF

backend2 << EOF
CREATE DATABASE backend;
CREATE TABLE backend.sbtest (
  id int(10) unsigned NOT NULL AUTO_INCREMENT,
  k int(10) unsigned NOT NULL DEFAULT '0',
  c char(120) NOT NULL DEFAULT '',
  pad char(60) NOT NULL DEFAULT '',
  PRIMARY KEY (id),
  KEY k (k)
) ENGINE=InnoDB;
EOF
```

Federation Setup

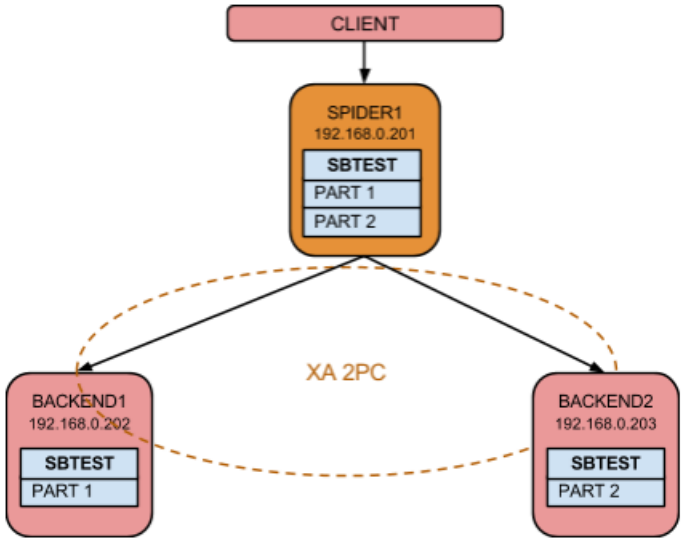



```
spider1 << EOF
CREATE SERVER backend
  FOREIGN DATA WRAPPER mysql
OPTIONS(
  HOST '192.168.0.202',
  DATABASE 'test',
  USER 'skysql',
  PASSWORD 'skyvodka',
  PORT 5054
);

CREATE  TABLE test.sbtest
(
  id int(10) unsigned NOT NULL AUTO_INCREMENT,
  k int(10) unsigned NOT NULL DEFAULT '0',
  c char(120) NOT NULL DEFAULT '',
  pad char(60) NOT NULL DEFAULT '',
  PRIMARY KEY (id),
  KEY k (k)
) ENGINE=spider COMMENT='wrapper "mysql",srv "backend"';
SELECT * FROM test.sbtest LIMIT 10;
EOF
```



Without connection pool or MariaDB thread pool, HaProxy and Spider have been protecting the tcp socket overflow without specific TCP tuning. In reality with a well tuned TCP stack or thread pool the curve should not decrease so abruptly to 0. Refer to the MariaDB Thread Pool to explore this feature.

Sharding Setup



Create the spider table on the Spider Node

```
#spider1 << EOF
CREATE SERVER backend1
  FOREIGN DATA WRAPPER mysql
OPTIONS(
  HOST '192.168.0.202',
  DATABASE 'backend',
  USER 'skysql',
  PASSWORD 'skyvodka',
  PORT 5054
);
CREATE SERVER backend2
  FOREIGN DATA WRAPPER mysql
OPTIONS(
  HOST '192.168.0.203',
  DATABASE 'backend',
  USER 'skysql',
  PASSWORD 'skyvodka',
  PORT 5054
);
CREATE DATABASE IF NOT EXISTS backend;
CREATE  TABLE backend.sbtest
(
  id int(10) unsigned NOT NULL AUTO_INCREMENT,
  k int(10) unsigned NOT NULL DEFAULT '0',
  c char(120) NOT NULL DEFAULT '',
  pad char(60) NOT NULL DEFAULT '',
  PRIMARY KEY (id),
  KEY k (k)
) ENGINE=spider COMMENT='wrapper "mysql", table "sbtest"'
 PARTITION BY KEY (id)
(
 PARTITION pt1 COMMENT = 'srv "backend1"',
 PARTITION pt2 COMMENT = 'srv "backend2"'
) ;
EOF
```

Copy the data from the original sysbench table to the spider table

```
#/usr/local/skysql/mariadb/bin/mysqldump  --user=skysql --password=skyvodka --
host=192.168.0.202 --port=5054 --no-create-info test sbtest | spider1 backend

#backend2 -e"select count(*) from backend.sbtest;"
+----------+
| count(*) |
+----------+
|  3793316 |
+----------+
#backend1 -e"select count(*) from backend.sbtest;"
+----------+
| count(*) |
+----------+
|  6206684 |
+----------+
```

We observe a common issue with partitioning is a non uniform distribution of data between the backends. based on the partition key hashing algorithm.

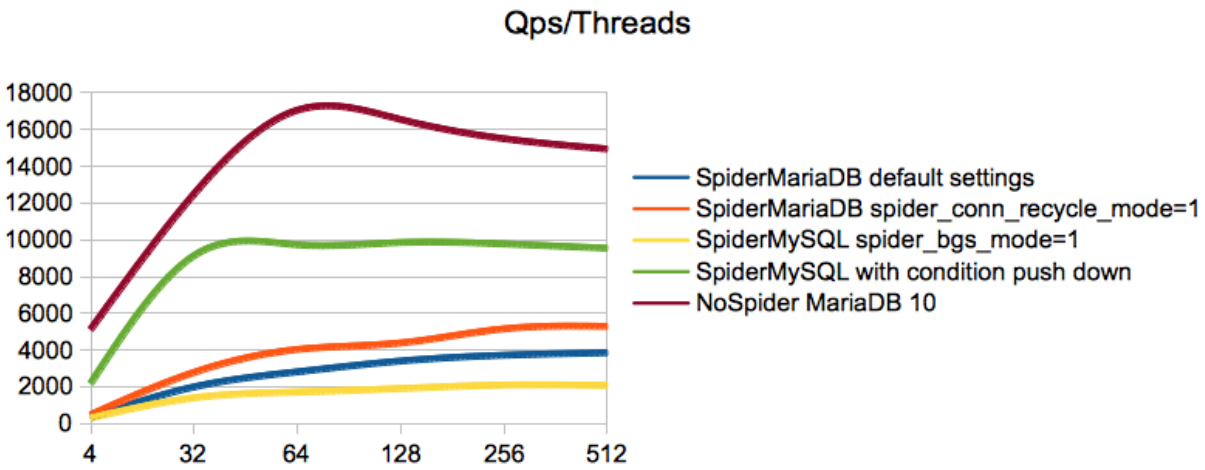
Rerun the Benchmark with less queries

```
#/usr/local/skysql/sysbench/bin/sysbench --test=oltp  --db-driver=mysql  --mysql-table-
engine=innodb --mysql-user=skysql --mysql-password=skyvodka --mysql-host=192.168.0.201 --mysql-
port=5054 --mysql-db=backend --mysql-engine-trx=yes --oltp-table-size=10000000 --num-threads=4
--max-requests=100 --oltp-read-only=on run
```

```
OLTP test statistics:
    queries performed:
        read:                            1414
        write:                           0
        other:                           202
        total:                           1616
    transactions:                        101    (22.95 per sec.)
    deadlocks:                           0      (0.00 per sec.)
    read/write requests:                 1414   (321.30 per sec.)
    other operations:                    202    (45.90 per sec.)

Test execution summary:
    total time:                          4.4009s
    total number of events:              101
    total time taken by event execution: 17.2960
    per-request statistics:
         min:                                  114.48ms
         avg:                                  171.25ms
         max:                                  200.98ms
         approx.  95 percentile:               195.12ms

Threads fairness:
    events (avg/stddev):           25.2500/0.43
    execution time (avg/stddev):   4.3240/0.04
```

The response time decreases to 0.04. This is expected because the query latency is increased from multiple network round trips and condition push down is not implemented yet. Sysbench doing a lot of range queries. Just consider for now that this range query can be a badly optimized query.

We need to increase the concurrency to get better throughput.



Background Setup

We have no background search available in MariaDB. It won't be available before MariaDB 10.2, but the next table definition mainly enables improving the performance of a single complex query plan with background search that can be found via the upstream spiral binaries MariaDB branch.

We have 4 cores per backend and 2 backends .

On `backend1`

```
#backend1 << EOF
CREATE DATABASE bsbackend1;
CREATE DATABASE bsbackend2;
CREATE DATABASE bsbackend3;
CREATE DATABASE bsbackend4;
CREATE TABLE bsbackend1.sbtest (
  id int(10) unsigned NOT NULL AUTO_INCREMENT,
  k int(10) unsigned NOT NULL DEFAULT '0',
  c char(120) NOT NULL DEFAULT '',
  pad char(60) NOT NULL DEFAULT '',
  PRIMARY KEY (id),
  KEY k (k)
) ENGINE=InnoDB;
CREATE TABLE bsbackend2.sbtest (
  id int(10) unsigned NOT NULL AUTO_INCREMENT,
  k int(10) unsigned NOT NULL DEFAULT '0',
  c char(120) NOT NULL DEFAULT '',
  pad char(60) NOT NULL DEFAULT '',
  PRIMARY KEY (id),
  KEY k (k)
) ENGINE=InnoDB;
CREATE TABLE bsbackend3.sbtest (
  id int(10) unsigned NOT NULL AUTO_INCREMENT,
  k int(10) unsigned NOT NULL DEFAULT '0',
  c char(120) NOT NULL DEFAULT '',
  pad char(60) NOT NULL DEFAULT '',
  PRIMARY KEY (id),
  KEY k (k)
) ENGINE=InnoDB;
CREATE TABLE bsbackend4.sbtest (
  id int(10) unsigned NOT NULL AUTO_INCREMENT,
  k int(10) unsigned NOT NULL DEFAULT '0',
  c char(120) NOT NULL DEFAULT '',
  pad char(60) NOT NULL DEFAULT '',
  PRIMARY KEY (id),
  KEY k (k)
) ENGINE=InnoDB;
EOF
```

On `backend2`

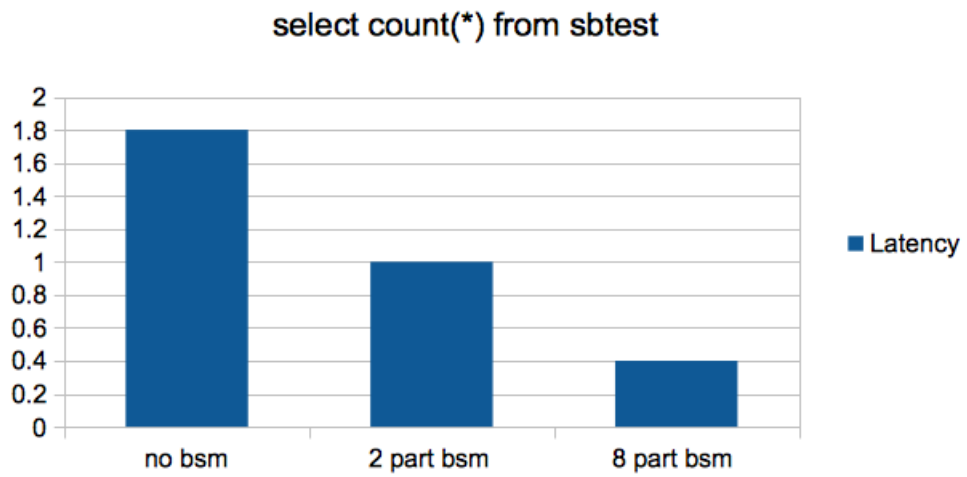
```
#backend2 << EOF
CREATE DATABASE bsbackend5;
CREATE DATABASE bsbackend6;
CREATE DATABASE bsbackend7;
CREATE DATABASE bsbackend8;
CREATE TABLE bsbackend5.sbtest (
  id int(10) unsigned NOT NULL AUTO_INCREMENT,
  k int(10) unsigned NOT NULL DEFAULT '0',
  c char(120) NOT NULL DEFAULT '',
  pad char(60) NOT NULL DEFAULT '',
  PRIMARY KEY (id),
  KEY k (k)
) ENGINE=InnoDB;
CREATE TABLE bsbackend6.sbtest (
  id int(10) unsigned NOT NULL AUTO_INCREMENT,
  k int(10) unsigned NOT NULL DEFAULT '0',
  c char(120) NOT NULL DEFAULT '',
  pad char(60) NOT NULL DEFAULT '',
  PRIMARY KEY (id),
  KEY k (k)
) ENGINE=InnoDB;
CREATE TABLE bsbackend7.sbtest (
  id int(10) unsigned NOT NULL AUTO_INCREMENT,
  k int(10) unsigned NOT NULL DEFAULT '0',
  c char(120) NOT NULL DEFAULT '',
  pad char(60) NOT NULL DEFAULT '',
  PRIMARY KEY (id),
  KEY k (k)
) ENGINE=InnoDB;
CREATE TABLE bsbackend8.sbtest (
  id int(10) unsigned NOT NULL AUTO_INCREMENT,
  k int(10) unsigned NOT NULL DEFAULT '0',
  c char(120) NOT NULL DEFAULT '',
  pad char(60) NOT NULL DEFAULT '',
  PRIMARY KEY (id),
  KEY k (k)
) ENGINE=InnoDB;
EOF
```

On `Spider Node`

```
#spider2 << EOF
CREATE SERVER bsbackend1 FOREIGN DATA WRAPPER mysql OPTIONS( HOST '192.168.0.202', DATABASE
'bsbackend1',USER 'skysql', PASSWORD 'skyvodka',PORT 5054);
CREATE SERVER bsbackend2 FOREIGN DATA WRAPPER mysql OPTIONS( HOST '192.168.0.202', DATABASE
'bsbackend2',USER 'skysql', PASSWORD 'skyvodka',PORT 5054);
CREATE SERVER bsbackend3 FOREIGN DATA WRAPPER mysql OPTIONS( HOST '192.168.0.202', DATABASE
'bsbackend3',USER 'skysql', PASSWORD 'skyvodka',PORT 5054);
CREATE SERVER bsbackend4 FOREIGN DATA WRAPPER mysql OPTIONS( HOST '192.168.0.202', DATABASE
'bsbackend4',USER 'skysql', PASSWORD 'skyvodka',PORT 5054);
CREATE SERVER bsbackend5 FOREIGN DATA WRAPPER mysql OPTIONS( HOST '192.168.0.203', DATABASE
'bsbackend5',USER 'skysql', PASSWORD 'skyvodka',PORT 5054);
CREATE SERVER bsbackend6 FOREIGN DATA WRAPPER mysql OPTIONS( HOST '192.168.0.203', DATABASE
'bsbackend6',USER 'skysql', PASSWORD 'skyvodka',PORT 5054);
CREATE SERVER bsbackend7 FOREIGN DATA WRAPPER mysql OPTIONS( HOST '192.168.0.203', DATABASE
'bsbackend7',USER 'skysql', PASSWORD 'skyvodka',PORT 5054);
CREATE SERVER bsbackend8 FOREIGN DATA WRAPPER mysql OPTIONS( HOST '192.168.0.203', DATABASE
'bsbackend8',USER 'skysql', PASSWORD 'skyvodka',PORT 5054);

CREATE DATABASE IF NOT EXISTS bsbackend;
CREATE  TABLE bsbackend.sbtest
(
  id int(10) unsigned NOT NULL AUTO_INCREMENT,
  k int(10) unsigned NOT NULL DEFAULT '0',
  c char(120) NOT NULL DEFAULT '',
  pad char(60) NOT NULL DEFAULT '',
  PRIMARY KEY (id),
  KEY k (k)
) ENGINE=spider COMMENT='wrapper "mysql", table "sbtest"'
 PARTITION BY KEY (id)
(
 PARTITION pt1 COMMENT = 'srv "bsbackend1"',
 PARTITION pt2 COMMENT = 'srv "bsbackend2"',
 PARTITION pt3 COMMENT = 'srv "bsbackend3"',
 PARTITION pt4 COMMENT = 'srv "bsbackend4"',
 PARTITION pt5 COMMENT = 'srv "bsbackend5"',
 PARTITION pt6 COMMENT = 'srv "bsbackend6"',
 PARTITION pt7 COMMENT = 'srv "bsbackend7"',
 PARTITION pt8 COMMENT = 'srv "bsbackend8"'
) ;
EOF
INSERT INTO  bsbackend.sbtest SELECT * FROM backend.sbtest;
```

Now test the following query :

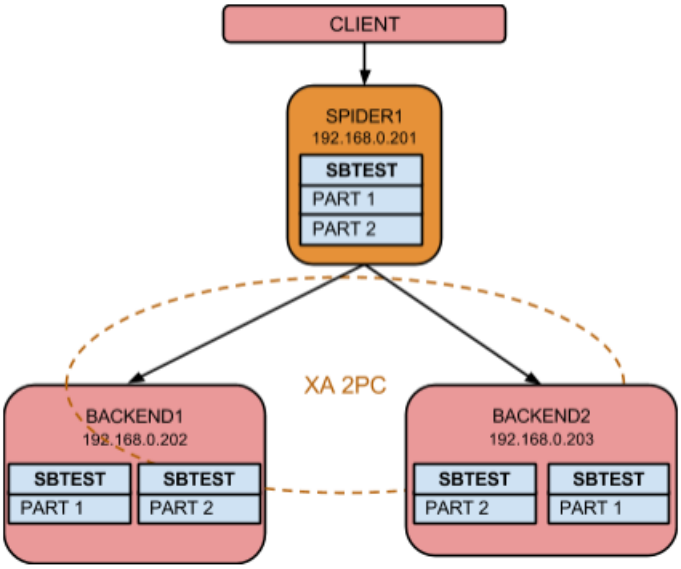
```
select count(*) from sbtest;
+----------+
| count(*) |
+----------+
| 10000001 |
+----------+
1 row in set (8,38 sec)

set spider_casual_read=1;
set spider_bgs_mode=2;

select count(*) from sbtest;
+----------+
| count(*) |
+----------+
| 10000001 |
+----------+
1 row in set (4,25 sec)

mysql> select sum(k) from sbtest;
+--------+
| sum(k) |
+--------+
|      0 |
+--------+
1 row in set (5,67 sec)

mysql> set spider_casual_read=0;
mysql> select sum(k) from sbtest;
+--------+
| sum(k) |
+--------+
|      0 |
+--------+
1 row in set (12,56 sec)
```



High Availability Setup

MariaDB starting with 10.7.5

Spider's high availability feature has been deprecated (MDEV-28479), and will be deleted. Please use other high availability solutions like replication or galera-cluster.


CLIENT
SPIDER1
192.168.0.201
SBTEST
PART 1
PART 2
XA 2PC
BACKEND1
192.168.0.202
SBTEST
PART 1
SBTEST
PART 2
BACKEND2
192.168.0.203
SBTEST
PART 2
SBTEST
PART 1

```
#backend1 -e "CREATE DATABASE backend_rpl"
#backend2 -e "CREATE DATABASE backend_rpl"

#/usr/local/skysql/mariadb/bin/mysqldump  --user=skysql --password=skyvodka --
host=192.168.0.202 --port=5054  backend sbtest | backend1 backend_rpl
#/usr/local/skysql/mariadb/bin/mysqldump  --user=skysql --password=skyvodka --
host=192.168.0.203 --port=5054  backend sbtest | backend2 backend_rpl

#spider1 << EOF
DROP TABLE backend.sbtest;
CREATE SERVER backend1_rpl
  FOREIGN DATA WRAPPER mysql
OPTIONS(
  HOST '192.168.0.202',
  DATABASE 'backend_rpl',
  USER 'skysql',
  PASSWORD 'skyvodka',
  PORT 5054
);
CREATE SERVER backend2_rpl
  FOREIGN DATA WRAPPER mysql
OPTIONS(
  HOST '192.168.0.203',
  DATABASE 'backend_rpl',
  USER 'skysql',
  PASSWORD 'skyvodka',
  PORT 5054
);
CREATE  TABLE backend.sbtest
(
  id int(10) unsigned NOT NULL AUTO_INCREMENT,
  k int(10) unsigned NOT NULL DEFAULT '0',
  c char(120) NOT NULL DEFAULT '',
  pad char(60) NOT NULL DEFAULT '',
  PRIMARY KEY (id),
  KEY k (k)
) ENGINE=spider COMMENT='wrapper "mysql", table "sbtest"'
 PARTITION BY KEY (id)
(
 PARTITION pt1 COMMENT = 'srv "backend1 backend2_rpl"',
 PARTITION pt2 COMMENT = 'srv "backend2 backend1_rpl"'
) ;
INSERT INTO backend.sbtest select 10000001, 0, '' ,'replicas test';
EOF
#backend1 -e "SELECT * FROM backend.sbtest WHERE id=10000001";
+----------+---+---+---------------+
| id       | k | c | pad           |
+----------+---+---+---------------+
| 10000001 | 0 |   | replicas test |
+----------+---+---+---------------+
# backend2 -e "SELECT * FROM backend.sbtest where id=10000001";
# backend2 -e "SELECT * FROM backend_rpl.sbtest where id=10000001";
+----------+---+---+---------------+
| id       | k | c | pad           |
+----------+---+---+---------------+
| 10000001 | 0 |   | replicas test |
+----------+---+---+---------------+
```

What is happening if we stop one backend?

```
#spider1 -e "SELECT * FROM backend.sbtest where id=10000001";
ERROR 1429 (HY000) at line 1: Unable to connect to foreign data source: backend1
```

Let's fix this with spider monitoring. Note that msi is the list of spider nodes @@server_id variable participating in the quorum.

```
#spider1 << EOF
DROP  TABLE backend.sbtest;
CREATE  TABLE backend.sbtest
(
  id int(10) unsigned NOT NULL AUTO_INCREMENT,
  k int(10) unsigned NOT NULL DEFAULT '0',
  c char(120) NOT NULL DEFAULT '',
  pad char(60) NOT NULL DEFAULT '',
  PRIMARY KEY (id),
  KEY k (k)
) ENGINE=spider COMMENT='wrapper "mysql", table "sbtest"'
 PARTITION BY KEY (id)
(
 PARTITION pt1 COMMENT = 'srv "backend1 backend2_rpl",  mbk "2", mkd "2", msi "5054",
link_status "0 0"',
 PARTITION pt2 COMMENT = 'srv "backend2 backend1_rpl",  mbk "2", mkd "2", msi "5054",
link_status "0 0" '
) ;

CREATE SERVER mon
  FOREIGN DATA WRAPPER mysql
OPTIONS(
  HOST '192.168.0.201’,
  DATABASE 'backend',
  USER 'skysql',
  PASSWORD 'skyvodka',
  PORT 5054
);
INSERT INTO `mysql`.`spider_link_mon_servers` VALUES
('%','%','%',5054,'mon',NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,0,NULL,NULL);
SELECT spider_flush_table_mon_cache();
EOF
```

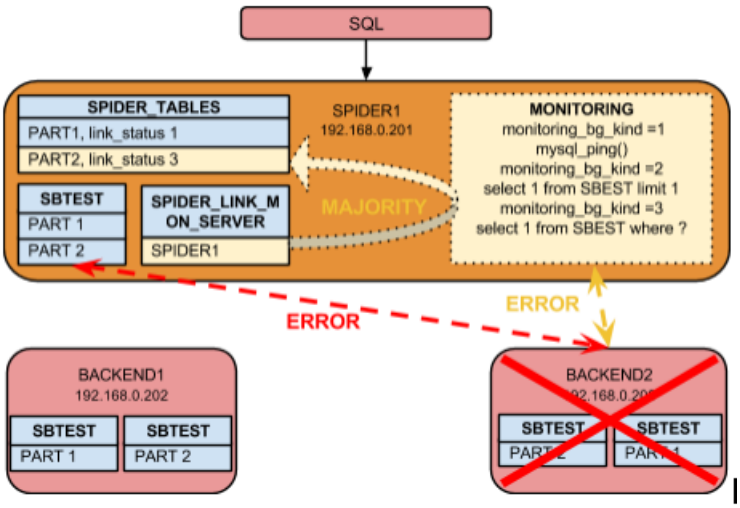
Monitoring should be setup between Spider nodes participating in the cluster. We only have one `Spider Node` and spider_link_mon_servers represent the inter-connection of all Spider nodes in our setup.

This simple setup does not bring HA in case the `Spider Node` is not available. In a production setup the number of `Spider Nodes` in the spider_link_mon_servers table should be at least 3 to get a majority consensus.

```
#spider1 -e "SELECT * FROM backend.sbtest WHERE id=10000001"
+----------+---+---+---------------+
| id       | k | c | pad           |
+----------+---+---+---------------+
| 10000001 | 0 |   | replicas test |
+----------+---+---+---------------+
```

Checking the state of the nodes:

```
#spider1 -e "SELECT db_name, table_name,server  FROM mysql.spider_tables WHERE link_status=3"
+---------+-------------+----------+
| db_name | table_name   | server   |
+---------+-------------+----------+
| backend | sbtest#P#pt1 | backend1 |
+---------+-------------+----------+
```



No change has been made to cluster, so let's create a divergence:

```
# spider1 -e "INSERT INTO backend.sbtest select 10000003, 0, '' ,'replicas test';"
# backend1 -e "SELECT * FROM backend.sbtest WHERE id=10000003"
# backend2 -e "SELECT * FROM backend_rpl.sbtest WHERE id=10000003"
+----------+---+---+---------------+
| id       | k | c | pad           |
+----------+---+---+---------------+
| 10000003 | 0 |   | replicas test |
+----------+---+---+---------------+
```

Reintroducing the failed backend1 in the cluster:

```
#spider1 << EOF
ALTER TABLE backend.sbtest
ENGINE=spider COMMENT='wrapper "mysql", table "sbtest"'
 PARTITION BY KEY (id)
(
 PARTITION pt1 COMMENT = 'srv "backend1 backend2_rpl"  mbk "2", mkd "2", msi "5054", link_status
"2 0"',
 PARTITION pt2 COMMENT = 'srv "backend2 backend1_rpl"  mbk "2", mkd "2", msi "5054", link_status
"0 2" '
) ;
select spider_copy_tables('backend.sbtest#P#pt1','0','1');
select spider_copy_tables('backend.sbtest#P#pt2','1','0');
ALTER TABLE backend.sbtest
ENGINE=spider COMMENT='wrapper "mysql", table "sbtest"'
 PARTITION BY KEY (id)
(
 PARTITION pt1 COMMENT = 'srv "backend1 backend2_rpl"  mbk "2", mkd "2", msi "5054", link_status
"1 0"',
 PARTITION pt2 COMMENT = 'srv "backend2 backend1_rpl"  mbk "2", mkd "2", msi "5054", link_status
"0 1" '
) ;
EOF
```

5.3.19.2 Spider Installation

The Spider storage engine supports partitioning and XA transactions, and allows tables of different MariaDB instances to be handled as if they were on the same instance.

To use Spider, you need two or more instances of MariaDB, typically running on separate hosts. The Spider node is the MariaDB server that receives queries from your application. It then processes these queries, connecting to one or more data nodes. The data nodes are the MariaDB servers that actually store the table data.

In order for this to work, you need to configure the data nodes to accept queries from the Spider node and you need to configure the Spider node to use the data nodes as remote storage.

You don't need to install any additional packages to use it, but it does require some configuration.

Contents

Configuring Data Nodes

Spider deployments use data nodes to store the actual table data. In order for a MariaDB server to operate as a data node for Spider, you need to create a table or tables on which to store the data and configure the server to accept client connections from the Spider node.

For instance, first create the table:

```
CREATE TABLE test.spider_example (
   id INT PRIMARY KEY AUTO_INCREMENT,
   name VARCHAR(50)
) ENGINE=InnoDB;
```

Next, create a user for the Spider node and set a password for that user. For the sake of the example, assume the Spider node is at the IP address 192.168.1.1:

```
CREATE USER spider@192.168.1.1;

SET PASSWORD FOR spider@192.168.1.1 = PASSWORD('passwd');
```

Then grant the `spider` user privileges on the example table.

```
GRANT ALL ON test.spider_example TO spider@192.168.1.1;
```

The data node is now ready for use. You can test it by attempting to connect the MariaDB client to the data from the Spider node. For instance, assuming the data node is at the IP address 192.168.1.5, SSH into the Spider node then try to establish a client connection.

```
$ mysql -u spider -p -h 192.168.1.5 test -e "SHOW TABLES;"

+----------------+
| Tables_in_test |
+----------------+
| spider_example |
+----------------+
```

Install Spider on Spider Node

The Spider storage engine must be installed on the Spider node. The Spider node is the MariaDB server that receives queries for the table, (in this case `test.spider_example`). It then uses the Spider storage engine to connect to the tables on the data nodes to retrieve data and return the result-set.

To install the Spider storage engine, complete the installation process shown below.

Step 1: Install Spider Package (Debian/Ubuntu)

On Debian and Ubuntu, the Spider storage engine is installed via a separate `mariadb-plugin-spider` package. To install the package via APT, execute the following command:

```
$ sudo apt install mariadb-plugin-spider
```

On other Linux distributions, the Spider storage engine is installed with MariaDB Server.

Step 2a: Load the Spider Plugin (MariaDB 10.4 and Later)

With MariaDB 10.4 and later, the Spider storage engine can be loaded as a normal plugin, and Spider automatically creates its dependencies. There are two primary ways to load the plugin.

The plugin can be loaded dynamically without a server restart by executing `INSTALL SONAME` or `INSTALL PLUGIN`:

```
INSTALL SONAME "ha_spider";
```

Alternatively, the plugin can be loaded by adding `plugin_load_add=ha_spider` to a configuration file:

```
<<quote>>
[mariadb]
...
plugin_load_add = "ha_spider"
<</quote>>
```

If the plugin is loaded in a configuration file, then the server will load the plugin after the server has been restarted.

Loading the plugin also creates a series of new tables in the `mysql` database, including:

table name	added version
spider_xa	MariaDB 10.0.4
spider_xa_member	MariaDB 10.0.4
spider_xa_failed_log	MariaDB 10.0.5
spider_tables	MariaDB 10.0.4
spider_link_mon_servers	MariaDB 10.0.4
spider_link_failed_log	MariaDB 10.0.4
spider_table_position_for_recovery	MariaDB 10.3.3
spider_table_sts	MariaDB 10.3.3
spider_table_crd	MariaDB 10.3.3

Step 2b: Load the Spider Plugin (MariaDB 10.3 and Before)

With MariaDB 10.3 and before, the Spider storage engine can be loaded by executing the included `install_spider.sql` script:

```
$ mysql --user root --password < /usr/share/mysql/install_spider.sql
```

Running this configuration script also creates a series of new tables in the `mysql` database, including:

table name	added version
spider_xa	MariaDB 10.0.4
spider_xa_member	MariaDB 10.0.4
spider_xa_failed_log	MariaDB 10.0.5
spider_tables	MariaDB 10.0.4
spider_link_mon_servers	MariaDB 10.0.4
spider_link_failed_log	MariaDB 10.0.4
spider_table_position_for_recovery	MariaDB 10.3.3
spider_table_sts	MariaDB 10.3.3
spider_table_crd	MariaDB 10.3.3

Step 3: Verify Loading of the Spider Plugin

You can verify that the Spider plugin has been loaded by querying the `information_schema.ENGINES` table:

```
SELECT ENGINE, SUPPORT
FROM information_schema.ENGINES
WHERE ENGINE = 'SPIDER';

+--------------------+---------+
| ENGINE             | SUPPORT |
+--------------------+---------+
| SPIDER             | YES     |
+--------------------+---------+
```

If the Spider plugin is not loaded, then the query will not return any results.

Configuring Spider Nodes

With the data node or data nodes configured, you can set up the Spider node to use them. The Spider node is the MariaDB server that receives queries for the table, (in this case `test.spider_example`). It then uses the Spider storage engine to connect to the tables on the data nodes to retrieve data and return the result-set.

Configure the Server

In order to connect the Spider node to the data nodes, you need to issue a `CREATE SERVER` statement for each data node. You can then use the server definition in creating the Spider table.

```
CREATE SERVER dataNode1 FOREIGN DATA WRAPPER mysql
OPTIONS (
   HOST '192.168.1.5',
   DATABASE 'test',
   USER 'spider',
   PASSWORD 'passwd',
   PORT 3306);
```

In the event that you need to modify or replace this server after setting up the Spider table, remember to issue a `FLUSH` statement to update the server definition.

```
FLUSH TABLES;
```

Create the Table

With the data nodes set up and the Spider node configured for use, you can create the Spider table. The Spider table must have the same column definitions as the InnoDB tables on the data nodes. Spider is configured through table system variables passed to the `COMMENT` option.

```
CREATE TABLE test.spider_example (
   id INT PRIMARY KEY AUTO_INCREMENT,
   name VARCHAR(50)
) ENGINE=Spider
COMMENT='wrapper "mysql", srv "dataNode1", table "spider_example"';
```

This configures Spider to use the server `dataNode1`, (defined above), as a remote table. Any data you write to this table is actually stored on the MariaDB server at 192.168.1.5.

5.3.19.3 Spider Storage Engine Core Concepts

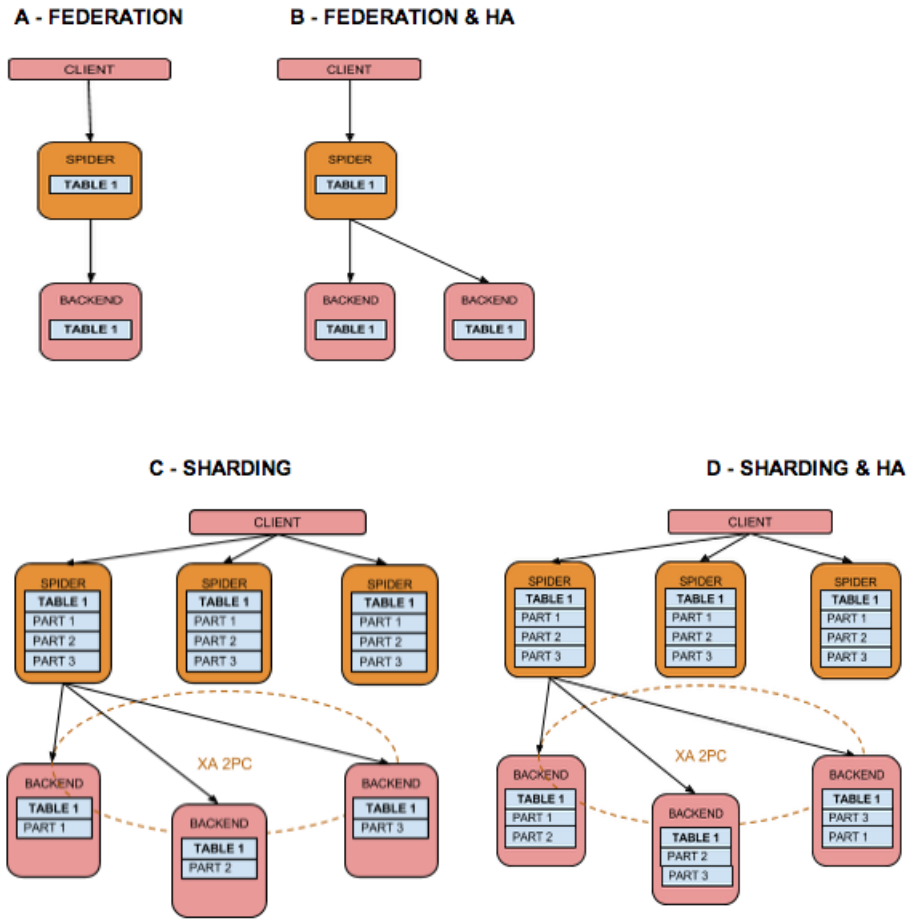
Contents

A typical Spider deployment has a shared-nothing clustered architecture. The system works with any inexpensive hardware, and with a minimum of specific requirements for hardware or software. It consists of a set of computers, with one or more MariaDB processes known as nodes.

The nodes that store the data will be designed as `Backend Nodes`, and can be any MariaDB, MySQL, Oracle server instances using any storage engine available inside the backend.

The `Spider Proxy Nodes` are instances running at least MariaDB 10. `Spider Proxy Nodes` are used to declare per table attachment to the backend nodes. In addition `Spider Proxy Nodes` can be setup to enable the tables to be split and mirrored to multiple `Backend Nodes`.

Spider Common Usage

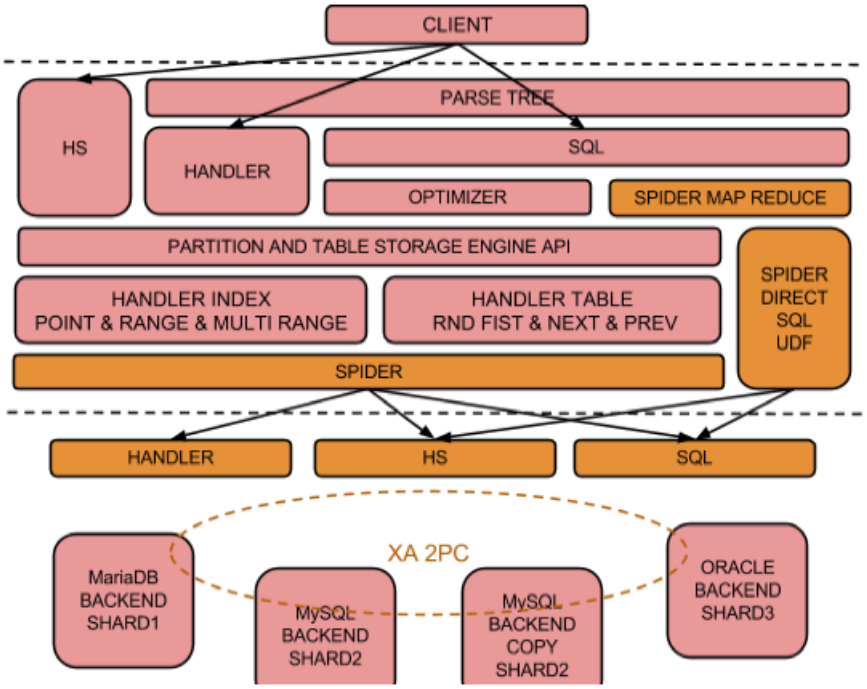


In the default high availability setup #Spider Nodes# produce SQL errors when a backend server is not responding. Per table monitoring can be setup to enable availability in case of unresponsive backends `monotoring_bg_kind=1` or `monotoring_bg_kind=2`. The Monitoring Spider Nodes will be inter-connected with usage of the system table `mysql.link_mon_servers` to manage network partitioning. Better known as split brain, an even number of `Spider Monitor Nodes` should be setup to allow a consensus based on the majority. Rather a single separated shared `Monitoring Node` instance or a minimum set of 3 `Spider Nodes`. More information can be found here.

Spider Storage Engine Federation

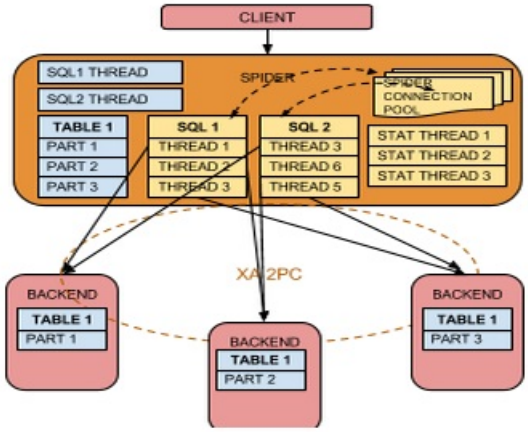
Spider is a pluggable Storage Engine, acting as a proxy between the optimizer and the remote backends. When the optimizer requests multiple calls to the storage engine, Spider enforces consistency using the 2 phase commit protocol to the backends and by creating transactions on the backends to preserve atomic operations for a single SQL execution. Preserving atomic operation during execution is used at multiple levels in the architecture,. For the regular optimizer plan, it refers to `multiple split reads` and for concurrent partition scans, it will refer to `semi transactions`.

Costly queries can be more efficient when it is possible to fully push down part of the execution plan on each backend and reduce the result afterwards. Spider enables such execution with some direct execution shortcuts.



Spider Threading Model

Spider uses the per partitions and per table model to concurrently access the remote backend nodes. For memory workload that property can be used to define multiple partitions on a single remote backend node to better adapt the concurrency to available CPUs in the hardware.



Spider maintains an internal dictionary of Table and Index statistics based on separated threads. The statistics are pulled per default on a time line basis and refer to `crd` for cardinality and `sts` for table status.

Spider Memory Model

Spider stores resultsets into memory, but spider_quick_mode=3 stores resultsets into internal temporary tables if the resultsets are larger than quick_table_size.

5.3.19.4 Spider Use Cases

Contents

Introduction

This article will cover simple working examples for some standard use cases for Spider. The example will be illustrated using a sales opportunities table to be consistent throughout. In some cases the actual examples will be contrived but are used to illustrate the varying syntax options.

Basic setup

Have 3 or more servers available and Install MariaDB on each of these servers:

- **spider** server which will act as the front end server hosting the spider storage engine.
- **backend1** which will act as a backed server storing data
- **backend2** which will act as a second backend server storing data

Follow the instructions here to enable the Spider storage engine on the spider server:

```
mysql  -u root -p < /usr/share/mysql/install_spider.sql
```

Enable use of non root connections

When using the General Query Log, non-root users may encounter issues when querying Spider tables. Explicitly setting the `spider_internal_sql_log_off` system variable causes the Spider node to execute `SET sql_log_off` statements on the data nodes to enable or disable the General Query Log. When this is done, queries issued by users without the `SUPER` privilege raise an error.

To avoid this, don't explicitly set the `spider_internal_sql_log_off` system variable.

Create accounts for spider to connect with on backend servers

Spider needs a remote connection to the backend server to actually perform the remote query. So this should be setup on each backend server. In this case 172.21.21.2 is the ip address of the spider node limiting access to just that server.

```
backend1> mysql
grant all on test.* to spider@'172.21.21.2' identified by 'spider';
backend2> mysql
grant all on test.* to spider@'172.21.21.2' identified by 'spider';
```

Now verify that these connections can be used from the spider node (here 172.21.21.3 = backend1 and 172.21.21.4 = backend2):

```
spider> mysql -u spider -p -h 172.21.21.3 test
spider> mysql -u spider -p -h 172.21.21.4 test
```

Create table on backend servers

The table definition should be created in the test database on both backend1 and backend2 servers:

```
create table opportunities (
id int,
accountName varchar(20),
name varchar(128),
owner varchar(7),
amount decimal(10,2),
closeDate date,
stageName varchar(11),
primary key (id),
key (accountName)
) engine=InnoDB;
```

Create server entries on spider server

While the connection information can also be specified inline in the comment, it is cleaner to define a server object representing each remote backend server connection:

```
create server backend1 foreign data wrapper mysql options
(host '172.21.21.3', database 'test', user 'spider', password 'spider', port 3306);
create server backend2 foreign data wrapper mysql options
(host '172.21.21.4', database 'test', user 'spider', password 'spider', port 3306);
```

Unable to Connect Errors

Bear in mind, if you ever need to remove, recreate or otherwise modify the server definition for any reason, you need to also execute a `FLUSH TABLES` statement. Otherwise, Spider continues to use the old server definition, which can result in queries raising the error

```
Error 1429: Unable to connect to foreign data source
```

If you encounter this error when querying Spider tables, issue a `FLUSH TABLES` statement to update the server definitions.

```
FLUSH TABLES;
```

Use case 1: remote table

In this case, a spider table is created to allow remote access to the opportunities table hosted on backend1. This then allows for queries and remote dml into the backend1 server from the spider server:

```
create table opportunities (
id int,
accountName varchar(20),
name varchar(128),
owner varchar(7),
amount decimal(10,2),
closeDate date,
stageName varchar(11),
primary key (id),
key (accountName)
) engine=spider comment='wrapper "mysql", srv "backend1" , table "opportunities"';
```

Use case 2: sharding by hash

In this case a spider table is created to distribute data across backend1 and backend2 by hashing the id column. Since the id column is an incrementing numeric value the hashing will ensure even distribution across the 2 nodes.

```
create table opportunities (
id int,
accountName varchar(20),
name varchar(128),
owner varchar(7),
amount decimal(10,2),
closeDate date,
stageName varchar(11),
primary key (id),
key (accountName)
) engine=spider COMMENT='wrapper "mysql", table "opportunities"'
 PARTITION BY HASH (id)
(
 PARTITION pt1 COMMENT = 'srv "backend1"',
 PARTITION pt2 COMMENT = 'srv "backend2"'
) ;
```

Use case 3: sharding by range

In this case a spider table is created to distribute data across backend1 and backend2 based on the first letter of the accountName field. All accountNames that start with the letter L and prior will be stored in backend1 and all other values stored in backend2. Note that the accountName column must be added to the primary key which is a requirement of MariaDB partitioning:

```
create table opportunities (
id int,
accountName varchar(20),
name varchar(128),
owner varchar(7),
amount decimal(10,2),
closeDate date,
stageName varchar(11),
primary key (id, accountName),
key(accountName)
) engine=spider COMMENT='wrapper "mysql", table "opportunities"'
 PARTITION BY range columns (accountName)
(
 PARTITION pt1 values less than ('M') COMMENT = 'srv "backend1"',
 PARTITION pt2 values less than (maxvalue) COMMENT = 'srv "backend2"'
) ;
```

Use case 4: sharding by list

In this case a spider table is created to distribute data across backend1 and backend2 based on specific values in the owner field. Bill, Bob, and Chris will be stored in backend1 and Maria and Olivier stored in backend2. Note that the owner column must be added to the primary key which is a requirement of MariaDB partitioning:

```
create table opportunities (
id int,
accountName varchar(20),
name varchar(128),
owner varchar(7),
amount decimal(10,2),
closeDate date,
stageName varchar(11),
primary key (id, owner),
key(accountName)
) engine=spider COMMENT='wrapper "mysql", table "opportunities"'
 PARTITION BY list columns (owner)
(
 PARTITION pt1 values in ('Bill', 'Bob', 'Chris') COMMENT = 'srv "backend1"',
 PARTITION pt2 values in ('Maria', 'Olivier') COMMENT = 'srv "backend2"'
) ;
```

With MariaDB 10.2 the following partition clause can be used to specify a default partition for all other values, however this must be a distinct partition / shard:

```
PARTITION partition_name DEFAULT
```

5.3.19.5 Spider Cluster Management

Contents

Direct SQL

Direct SQL is a way to map reduced execution on remote backends and store the results in a local table. This can either be sequential, using the UDF function spider_direct_sql, or concurrently, using spider_bg_direct_sql.

```
spider1 backend << EOF
CREATE TEMPORARY TABLE res
(
  id int(10) unsigned NOT NULL,
  k int(10) unsigned NOT NULL DEFAULT '0',
  c char(120) NOT NULL DEFAULT '',
  pad char(60) NOT NULL DEFAULT ''
) ENGINE=MEMORY;

SELECT spider_direct_sql(
'SELECT * FROM sbtest s  WHERE s.id IN(10,12,13)',
  'res',
  concat('host "', host, '", port "', port, '", user "', username, '", password "', password,
'", database "', tgt_db_name, '"')
) a
FROM
  mysql.spider_tables
WHERE
  db_name = 'backend' and table_name like 'sbtest#P#pt%';

SELECT * FROM res;
EOF
```

Or if you are using a SERVER:

```
SELECT spider_direct_sql(
  'SELECT * FROM sbtest s  WHERE s.id IN(10,12,13)',
  'res',
  concat('server "', server, '"')
) a
  FROM mysql.spider_tables
  WHERE db_name = 'backend' and table_name like 'sbtest#P#pt%' ;
```

The default for spider_bg_direct_sql is to access concurrently all backends. If you have multiple partitions store inside a single backend, you still can increase parallelism affecting different channels to each partitions.

```
CREATE TEMPORARY TABLE res
(
  id int(10) unsigned NOT NULL ,
  col_microsec DATETIME(6) default NOW(8),
  db varchar(20)
) ENGINE=MEMORY;

SELECT spider_bg_direct_sql( 'SELECT count(*) ,min(NOW(6)),min(DATABASE())) FROM sbtest',
'res',  concat('srv "', server,'" cch ',@rn:=@rn+1  ) ) a  FROM    mysql.spider_tables,(SELECT
@rn:=1) t2  WHERE    db_name = 'bsbackend' and table_name like 'sbtest#P#pt%';
```

Direct Handler Socket

Check that Handler Socket is running on the backend nodes

```
:~# backend2 -e "show variables like 'handler%'"
+------------------------------+---------------+
| Variable_name                | Value         |
+------------------------------+---------------+
| handlersocket_accept_balance | 0             |
| handlersocket_address        | 192.168.0.201 |
| handlersocket_backlog        | 32768         |
| handlersocket_epoll          | 1             |
| handlersocket_plain_secret   |               |
| handlersocket_plain_secret_wr|               |
| handlersocket_port           | 20500         |
| handlersocket_port_wr        | 20501         |
| handlersocket_rcvbuf         | 0             |
| handlersocket_readsize       | 0             |
| handlersocket_sndbuf         | 0             |
| handlersocket_threads        | 4             |
| handlersocket_threads_wr     | 1             |
| handlersocket_timeout        | 300           |
| handlersocket_verbose        | 10            |
| handlersocket_wrlock_timeout | 12            |
+------------------------------+---------------+
```

```
spider1 backend << EOF
CREATE TEMPORARY TABLE res
(
  id int(10) unsigned NOT NULL,
  k int(10) unsigned NOT NULL DEFAULT '0',
  c char(120) NOT NULL DEFAULT '',
  pad char(60) NOT NULL DEFAULT ''
) ENGINE=MEMORY;

SELECT spider_direct_sql('1\t=\t1\t2\t100000\t0','res', 'host "192.168.0.202", table "sbtest",
database "test", port "20500", access_mode "1"');
```

Inter Nodes Copy Table

The UDF function spider_copy_tables is available for copying table data from the source link ID to the destination link ID list without stopping your service for copying

```
spider_copy_tables(Spider table name, source link ID, destination link ID list[, parameters])
```

- `Returns 1` if copying data succeeded.
- `Returns 0` if copying data failed.

If the Spider table is partitioned, you must set "Spider table name" with a part name such as "table_name#P#part_name".

You can check the table name and the link ID with the part name using the following SQL:

```
SELECT table_name FROM mysql.spider_tables;
```

Resharding

General Log

To capture all queries sent to remote backends on a `Spider Node` :

```
SET GLOBAL general_log=ON;
SET GLOBAL spider_general_log=ON;
SET GLOBAL spider_log_result_errors=1;
SET GLOBAL spider_log_result_error_with_sql=3;
```

Compiling in Debug Mode

Compile MariaDB in debug mode

```
#cmake -DBUILD_CONFIG=mysql_release -DCMAKE_BUILD_TYPE=Debug  -DWITH_FAST_MUTEXES=ON  -
DWITH_VALGRIND=ON
```

Run MariaDB the following way to have a detailed command trace file

```
mysqld --
debug=S:T:t:r:p:n:L:i:F:f:D:d,info,error,query,qcache,my,exit,general,where:O,/tmp/mysqld.trace
```

Or with Valgrind to get a complete stack trace on a crash.

```
valgrind /usr/local/mysql/bin/mysqld --basedir=/usr/local/mysql --datadir=/data/inetbase/mysql
--plugin-dir=/usr/local/mysql/lib/plugin --user=mysql --log-
error=/data/inetbase/mysql/lucifer.err --open-files-limit=65000 --pid-
file=/var/run/mysqld/mysqld.pid --socket=/var/run/mysqld/mysqld.sock --port=3306
```

Report the issue in MariaDB JIRA (see Reporting Bugs) or to the MariaDB Corporation support center.

Compiling in Static

Available since version 3.1.14

To activate spider as a static plugin change "MODULE_ONLY" to "MANDATORY" in storage/spider/CMakeList.txt before compiling

Note that Spider UDF functions will not work with such settings.

Status Variables

A number of new status variables have been introduced:

- Spider_direct_aggregate
- Spider_direct_order_limit
- Spider_mon_table_cache_version
- Spider_mon_table_cache_version_req

Information Schema Tables

- A new Information Schema table is installed - SPIDER_ALLOC_MEM.

```
+-------------------+---------------------+------+-----+---------+-------+
| Field             | Type                | Null | Key | Default | Extra |
+-------------------+---------------------+------+-----+---------+-------+
| ID                | int(10) unsigned    | NO   |     | 0       |       |
| FUNC_NAME         | varchar(64)         | YES  |     | NULL    |       |
| FILE_NAME         | varchar(64)         | YES  |     | NULL    |       |
| LINE_NO           | int(10) unsigned    | YES  |     | NULL    |       |
| TOTAL_ALLOC_MEM   | bigint(20) unsigned | YES  |     | NULL    |       |
| CURRENT_ALLOC_MEM | bigint(20)          | YES  |     | NULL    |       |
| ALLOC_MEM_COUNT   | bigint(20) unsigned | YES  |     | NULL    |       |
| FREE_MEM_COUNT    | bigint(20) unsigned | YES  |     | NULL    |       |
+-------------------+---------------------+------+-----+---------+-------+
```

Performance Schema

The Performance schema is commonly used to troubleshoot issues that consume time inside your workload. The Performance schema should not be activated for servers that are experimenting constant heavy load, but most of time it is acceptable to lose 5% to 20% additional CPU to keep track of server internals execution.

To activate the performance schema, use the performance_schema system variable and add the following to the server section of the MariaDB configuration file.

```
performance_schema=on
```

Activate the Spider probes to be monitored.

```
UPDATE performance_schema.setup_instruments SET
  ENABLED='YES', TIMED='yes' WHERE NAME LIKE '%spider%';
```

Run your queries ...

And check the performance metrics. Remove specific Spider metrics to have a more global view.

```
SELECT * FROM performance_schema.events_waits_summary_global_by_event_name
  WHERE COUNT_STAR<>0 AND EVENT_NAME LIKE '%spider%'
  ORDER BY SUM_TIMER_WAIT DESC LIMIT 10;
```

5.3.19.6 Spider Feature Matrix

Not complete yet - still being updated

F(*) Federation only , P(*)partioning only . Spider column is for SpiderForMySQL found on the Spider web slte.

Feature	Spider	10.0
Clustering and High Availability		
Commit, Rollback transactions on multiple backend	Yes	Yes
Multiplexing to a number of replicas using xa protocol 2PC	Yes	Yes
Split brain resolution based on a majority decision, failed node is remove from the list of replicas	Yes	Yes
Enable a failed backend to re enter the cluster transparently	No	No
Synchronize DDL to backend, table modification, schema changes	No	No
Synchronize DDL to other Spider	No	No
GTID tracking per table on XA error	No	Yes
Transparent partitioning	No	No
Covered by generic SQL test case	Yes	Yes
Heterogenous Backends		
MariaDB and MySQL database backend	Yes	Yes
Oracle database backend, if build from source against the client library 'ORACLE_HOME'	Yes	Yes
Local table attachment	Yes	Yes
Performance		
Index Condition Pushdown	No	No
Engine Condition Pushdown	Yes	Yes
Concurrent backend scan	Yes	No
Concurrent partition scan	Yes	No
Batched key access	Yes	Yes
Block hash join	No	Yes
HANDLER backend propagation	Yes	Yes
HANDLER backend translation from SQL	Yes	Yes
HANDLER OPEN cache per connection	No	Yes
HANDLER use prepared statement	No	Yes
HANDLER_SOCKET protocol backend propagation	Yes	Yes
HANDLER_SOCKET backend translation from SQL	No	No
Map reduce for `ORDER BY` ... LIMIT	Yes	Yes
Map reduce for `MAX & MIN & SUM`	Yes	Yes

Map reduce for some `GROUP BY`	Yes	Yes
Batch multiple WRITES in auto commit to reduce network round trip	Yes	Yes
Relaxing backend consistency	Yes	Yes
Execution Control		
Configuration at table and partition level, settings can change per data collection	Yes	Yes
Configurable empty result set on errors. For API that does not have transactions replay	Yes	Yes
Query Cache tuning per table of the on remote backend	Yes	Yes
Index Hint per table imposed on remote backend	Yes	Yes
SSL connections to remote backend connections	Yes	Yes
Table definition discovery from remote backend	Yes	F(*)
Direct SQL execution to backend via UDF	Yes	Yes
Table re synchronization between backends via UDF	Yes	Yes
Maintain Index and Table Statistics of remote backends	Yes	Yes
Can use Independent Index and Table Statistics	No	Yes
Maintain local or remote table increments	Yes	Yes
LOAD DATA INFILE translate to bulk inserting	Yes	Yes
Performance Schema Probes	Yes	Yes
Load Balance Reads to replicate weight control	Yes	Yes
Fine tuning tcp timeout, connections retry	Yes	Yes

5.3.19.7 Spider System Variables

The following variables are available when the Spider storage engine has been installed.

See Server System Variables for a complete list of system variables and instructions on setting them.

See also the Full list of MariaDB options, system and status variables.

MariaDB starting with 10.4.31

Starting from MariaDB 10.4.31, MariaDB 10.5.22, MariaDB 10.6.15, MariaDB 10.9.8, MariaDB 10.10.6, MariaDB 10.11.5, MariaDB 11.0.3, all spider system variables with the value -1 for deferring to table parameter values follow the correct overriding mechanism: table parameter (if set) overrides system variables (if not -1) overrides actual variable default. As a side effect, all such system variables in all versions have the same default value as the table param default value.

Before this change, a non-minus-one system variable value would override the table parameter value. That is, if both the system variable value and the table parameter value were set to be non-minus-one, the system variable value would prevail. For MariaDB 10.7+ where the system variable default values were the same as table param default instead of -1, this means that if the system variable were not set, but a table param is set to a non-default value, the default would override the non-default value.

Contents

22. spider_crd_bg_mode
23. spider_crd_interval
24. spider_crd_mode
25. spider_crd_sync
26. spider_crd_type
27. spider_crd_weight
28. spider_delete_all_rows_type
29. spider_direct_dup_insert
30. spider_direct_order_limit
31. spider_dry_access
32. spider_error_read_mode
33. spider_error_write_mode
34. spider_first_read
35. spider_force_commit
36. spider_general_log
37. spider_ignore_comments
38. spider_index_hint_pushdown
39. spider_init_sql_alloc_size
40. spider_internal_limit
41. spider_internal_offset
42. spider_internal_optimize
43. spider_internal_optimize_local
44. spider_internal_sql_log_off
45. spider_internal_unlock
46. spider_internal_xa
47. spider_internal_xa_id_type
48. spider_internal_xa_snapshot
49. spider_load_crd_at_startup
50. spider_load_sts_at_startup
51. spider_local_lock_table
52. spider_lock_exchange
53. spider_log_result_error_with_sql
54. spider_log_result_errors
55. spider_low_mem_read
56. spider_max_connections
57. spider_max_order
58. spider_multi_split_read
59. spider_net_read_timeout
60. spider_net_write_timeout
61. spider_ping_interval_at_trx_start
62. spider_quick_mode
63. spider_quick_page_byte
64. spider_quick_page_size
65. spider_read_only_mode
66. spider_remote_access_charset
67. spider_remote_autocommit
68. spider_remote_default_database
69. spider_remote_sql_log_off
70. spider_remote_time_zone
71. spider_remote_trx_isolation
72. spider_remote_wait_timeout
73. spider_reset_sql_alloc
74. spider_same_server_link
75. spider_second_read
76. spider_select_column_mode
77. spider_selupd_lock_mode
78. spider_semi_split_read
79. spider_semi_split_read_limit
80. spider_semi_table_lock
81. spider_semi_table_lock_connection
82. spider_semi_trx
83. spider_semi_trx_isolation
84. spider_skip_default_condition
85. spider_skip_parallel_search
86. spider_slave_trx_isolation
87. spider_split_read
88. spider_store_last_crd
89. spider_store_last_sts

89. spider_store_last_sts
90. spider_strict_group_by
91. spider_sts_bg_mode
92. spider_sts_interval
93. spider_sts_mode
94. spider_sts_sync
95. spider_support_xa
96. spider_suppress_comment_ignored_warning
97. spider_sync_autocommit
98. spider_sync_sql_mode
99. spider_sync_time_zone
100. spider_sync_trx_isolation
101. spider_table_crd_thread_count
102. spider_table_init_error_interval
103. spider_table_sts_thread_count
104. spider_udf_ct_bulk_insert_interval
105. spider_udf_ct_bulk_insert_rows
106. spider_udf_ds_bulk_insert_rows
107. spider_udf_ds_table_loop_mode
108. spider_udf_ds_use_real_table
109. spider_udf_table_lock_mutex_count
110. spider_udf_table_mon_mutex_count
111. spider_use_all_conns_snapshot
112. spider_use_cond_other_than_pk_for_update
113. spider_use_consistent_snapshot
114. spider_use_default_database
115. spider_use_flash_logs
116. spider_use_handler
117. spider_use_pushdown_udf
118. spider_use_snapshot_with_flush_tables
119. spider_use_table_charset
120. spider_version
121. spider_wait_timeout
122. spider_xa_register_mode

spider_auto_increment_mode

- **Description:** The auto increment mode.
  - `-1` Falls back to the default value, if the table parameter is not set.
  - `0` Normal Mode. Uses a counter that Spider gets from the remote backend server with an exclusive lock for the auto-increment value. This mode is slow. Use Quick Mode (`2`), if you use Spider tables with the table partitioning feature and the auto-increment column is the first column of the index. Before MariaDB 10.3, this value works as "1" for partitioned Spider tables.
  - `1` Quick Mode. Uses an internal Spider counter for the auto-increment value. This mode is fast, but it is possible for duplicates to occur when updating the same table from multiple Spider proxies.
  - `2` Set Zero Mode. The auto-increment value is given by the remote backend. Sets the column to `0` , even if you set the value to the auto-increment column in your statement. If you use the table with the table partitioning feature, it sets to zero after choosing an inserted partition.
  - `3` When the auto-increment column is set to `NULL` , the value is given by the remote backend server. If you set the auto-increment column to `0` ,the value is given by the local server. Set spider_reset_auto_incremnet to `2` or `3` if you want to use an auto-increment column on the remote server.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `0`
- **Default Table Value:** `0`
- **Range:** `-1` to `3`
- **DSN Parameter Name:** `aim`

spider_bgs_first_read

- **Description:** Number of first read records to use when performing a concurrent background search. To start a range scan on the remote backend, the storage engine first needs to send the first record. Fetching a second record in the same query can save a network round trip stopping the plan if the backend has a single record. The first and second reads are used to warm up for background search. When not using spider_split_read and spider_semi_split_read, the third read fetches the remaining data source in a single fetch.

    - `-1` Falls back to the default value, if the table parameter is not set.
    - `0` Records are usually retrieved.
    - `1` and greater: Number of records.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `2`
- **Default Table Value:** `2`
- **Range:** `-1` to `9223372036854775807`
- **DSN Parameter Name:** `bfr`

---

spider_bgs_mode

- **Description:** Background search mode. This enables the use of a thread per data server connection if the query is not shard-based and must be distributed across shards. The partitioning plugin scans partitions one after the other to optimize memory usage. Because the shards are external, reading all shards can be performed in parallel when the plan prunes multiple partitions.
    - `-1` Falls back to the default value, if the table parameter is not set.
    - `0` Disables background search.
    - `1` Uses background search when searching without locks
    - `2` Uses background search when searching without locks or with shared locks.
    - `3` Uses background search regardless of locks.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `0`
- **Default Table Value:** `0`
- **Range:** `-1` to `3`
- **DSN Parameter Name:** `bmd`

---

spider_bgs_second_read

- **Description:** Number of second read records on the backend server when using background search. When the first records are found from spider_bgs_first_read, the engine continues scanning a range adding a `LIMIT` of spider_bgs_first_read and spider_bgs_second_read.
    - `-1` Falls back to the default value, if the table parameter is not set.
    - `0` Records are usually retrieved.
    - `1` and greater: Number of records.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Default Session Value:** `100`
- **Default Table Value:** `100`
- **Range:** `-1` to `9223372036854775807`
- **DSN Parameter Name:** `bsr`

---

spider_bka_engine

- **Description:** Storage engine used with temporary tables when the spider_bka_mode system variable is set to `1`. Defaults to the value of the table parameter, which is MEMORY by default.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Session Value:** `""`
- **Default Table Value:** `Memory`
- **DSN Parameter Name:** `bke`

---

spider_bka_mode

- **Description:** Internal action to perform when multi-split reads are disabled. If the spider_multi_split_read system variable is set to `0`, Spider uses this variable to determine how to handle statements when the optimizer resolves range retrieval to multiple conditions.

    - `-1` Falls back to the default value, if the table parameter is not set.
    - `0` Uses "union all".
    - `1` Uses a temporary table, if it is judged acceptable.
    - `2` Uses a temporary table, if it is judged acceptable and avoids replication delay.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `1`
- **Default Table Value:** `1`
- **Range:** `-1` to `2`
- **DSN Parameter Name:** `bkm`

`spider_bka_table_name_type`

- **Description:** The type of temporary table name for bka.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `1`
- **Range:** `-1` to `1`

`spider_block_size`

- **Description:** Size of memory block used in MariaDB. Can usually be left unchanged.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `16384`
- **Range:** `0` to `4294967295`
- **DSN Parameter Name:** `bsz`

`spider_buffer_size`

- **Description:** Buffer size. `-1` , the default, will use the table parameter.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `16000`
- **Default Table Value:** `16000`
- **Range:** `-1` to `2147483647`
- **Introduced:** MariaDB 10.5.4

`spider_bulk_size`

- **Description:** Size in bytes of the buffer when multiple grouping multiple `INSERT` statements in a batch, (that is, bulk inserts).
    - `-1` The table parameter is adopted.
    - `0` or greater: Size of the buffer.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `16000`
- **Default Table Value:** `16000`
- **Range:** `-1` to `2147483647`
- **DSN Parameter Name:** `bsz`

`spider_bulk_update_mode`

- **Description:** Bulk update and delete mode. Note: If you use a non-default value for the spider_bgs_mode or

spider_split_read system variables, Spider sets this variable to 2 .
    - -1 Falls back to the default value, if the table parameter is not set.
    - 0 Sends UPDATE and DELETE statements one by one.
    - 1 Collects multiple UPDATE and DELETE statements, then sends the collected statements one by one.
    - 2 Collects multiple UPDATE and DELETE statements and sends them together.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** numeric
- **Default Session Value:** 0
- **Default Table Value:** 0
- **Range:** -1 to 2
- **DSN Parameter Name:** bum

---

spider_bulk_update_size

- **Description:** Size in bytes for UPDATE and DELETE queries when generating bulk updates.
    - -1 The table parameter is adopted.
    - 0 or greater: Size of buffer.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** numeric
- **Default Session Value:** 16000
- **Default Table Value:** 16000
- **Range:** -1 to 2147483647
- **DSN Parameter Name:** bus

---

spider_casual_read

- **Description:** Casual Reads enables all isolation levels, (such as repeatable reads) to work with transactions on multiple backends. With auto-commit queries, you can relax read consistency and run on multiple connections to the backends. This enables parallel queries across partitions, even if multiple shards are stored on the same physical server.
    - -1 Use table parameter.
    - 0 Use casual read.
    - 1 Choose connection channel automatically.
    - 2 to 63 Number of connection channels.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** numeric
- **Default Session Value:** 0
- **Default Table Value:** 0
- **Range:** -1 to 63
- **DSN Parameter Name:** ##

---

spider_conn_recycle_mode

- **Description:** Connection recycle mode.
    - 0 Disconnect.
    - 1 Recycle by all sessions.
    - 2 Recycle in the same session.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** numeric
- **Range:** 0 to 2
- **Default Session Value:** 0

---

spider_conn_recycle_strict

- **Description:** Whether to force the creation of new connections.
    - 1 Don't force.

    - `0` Force new connection
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `0`
- **Range:** `0` to `1`

---

spider_conn_wait_timeout

- **Description:** Max waiting time in seconds for Spider to get a remote connection.
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `10`
- **Range:** `0` to `1000`
- **Introduced:** MariaDB 10.3.3

---

spider_connect_error_interval

- **Description:** Return same error code until interval passes if connection is failed
- **Scope:** Global,
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1`
- **Range:** `0` to `4294967295`

---

spider_connect_mutex

- **Description:** Whether to serialize remote servers connections (use mutex at connecting). Use this parameter if you get an error or slowdown due to too many connection processes.
    - `0` Not serialized.
    - `1` : Serialized.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Session Value:** `0`

---

spider_connect_retry_count

- **Description:** Number of times to retry connections that fail due to too many connection processes.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `1000`
- **Range:** `0` to `2147483647`

---

spider_connect_retry_interval

- **Description:** Interval in microseconds for connection failure due to too many connection processes.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `1000`
- **Range:** `-1` to `9223372036854775807`

---

spider_connect_timeout

- **Description:** Timeout in seconds to declare remote backend unresponsive when opening a connection. Change for high-latency networks.

    - `-1` The table parameter is adopted.
    - `0` Less than 1.
    - `1` and greater: Number of seconds.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `6`
- **Default Table Value:** `0`
- **Range:** `-1` to `2147483647`
- **DSN Parameter Name:** `cto`

`spider_crd_bg_mode`

- **Description:** Indexes cardinality statistics in the background. Disable when the spider_crd_mode system variable is set to `3` or when the spider_crd_interval variable is set to `0`.
    - `-1` Falls back to the default value, if the table parameter is not set.
    - `0` Disables background confirmation.
    - `2` Enables background confirmation.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `2`
- **Default Table Value:** `1`
- **Range:** `-1` to `2`
- **DSN Parameter Name:** `cbm`

`spider_crd_interval`

- **Description:** Time interval in seconds of index cardinality statistics. Set to `0` to always get the latest information from remote servers.
    - `-1` The table parameter is adopted.
    - `1` or more: Interval in seconds table state confirmation.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `51`
- **Default Table Value:** `51`
- **Range:** `-1` to `2147483647`
- **DSN Parameter Name:** `civ`

`spider_crd_mode`

- **Description:** Mode for index cardinality statistics. By default, uses `SHOW` at the table-level.
    - `-1`, `0` Uses the table parameter.
    - `1` Uses the `SHOW` command.
    - `2` Uses the Information Schema.
    - `3` Uses the `EXPLAIN` command.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `1`
- **Default Table Value:** `1`
- **Range:** `-1` to `3`
- **DSN Parameter Name:** `cmd`
- **Deprecated:** MariaDB 10.7.4, MariaDB 10.8.3

`spider_crd_sync`

- **Description:** Synchronize index cardinality statistics in partitioned tables.
    - `-1` Falls back to the default value, if the table parameter is not set.

    - `0` Disables synchronization.
    - `1` Uses table state synchronization when opening a table, but afterwards performs no synchronization.
    - `2` Enables synchronization.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `0`
- **Default Table Value:** `0`
- **Range:** `-1` to `2`
- **DSN Parameter Name:** `csy`

---

spider_crd_type

- **Description:** Type of cardinality calculation. Only effective when the spider_crd_mode system variable is set to use `SHOW` (`1`) or to use the Information Schema (`2`).
    - `-1` Falls back to the default value, if the table parameter is not set.
    - `0` Uses the value of the spider_crd_weight system variable, as a fixed value.
    - `1` Uses the value of the spider_crd_weight system variable, as an addition value.
    - `2` Uses the value of the spider_crd_weight system variable, as a multiplication value.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `2`
- **Default Table Value:** `2`
- **Range:** `-1` to `2`
- **DSN Parameter Name:** `ctp`
- **Deprecated:** MariaDB 10.7.4, MariaDB 10.8.3

---

spider_crd_weight

- **Description:** Weight coefficient used to calculate effectiveness of index from the cardinality of column. For more information, see the description for the spider_crd_type system variable.
    - `-1` Falls back to the default value, if the table parameter is not set.
    - `0` or greater: Weight.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `2`
- **Default Table Value:** `2`
- **Range:** `-1` to `2147483647`
- **DSN Parameter Name:** `cwg`
- **Deprecated:** MariaDB 10.7.4, MariaDB 10.8.3

---

spider_delete_all_rows_type

- **Description:** The type of delete_all_rows.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `1`
- **Range:** `-1` to `1`

---

spider_direct_dup_insert

- **Description:** Manages duplicate key check for REPLACE, INSERT IGNORE and LOAD DATA LOCAL INFILE to remote servers. This can save on network roundtrips if the key always maps to a single partition. For bulk operations, records are checked for duplicate key errors one by one on the remote server, unless you set it to avoid duplicate checks on local servers (`1`).
    - `-1` Falls back to the default value, if the table parameter is not set.
    - `0` Performs duplicate checks on the local server.

    - `1` Avoids duplicate checks on the local server.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `0`
- **Default Table Value:** `0`
- **Range:** `-1` to `1`
- **DSN Parameter Name:** `ddi`

`spider_direct_order_limit`

- **Description:** Pushes `ORDER BY` and `LIMIT` operations to the remote server.
    - `-1` Falls back to the default value, if the table parameter is not set.
    - `0` Uses local execution.
    - `1` Uses push down execution.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `2`
- **Default Table Value:** `9223372036854775807`
- **Range:** `-1` to `9223372036854775807`
- **DSN Parameter Name:** `dol`

`spider_dry_access`

- **Description:** Simulates an empty result-set. No queries are sent to the backend. Use for performance tuning.
    - `0` Normal access.
    - `1` All access from Spider to data node is suppressed.
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`

`spider_error_read_mode`

- **Description:** Sends an empty result-set when reading a backend server raises an error. Useful with applications that don't implement transaction replays.
    - `-1` Falls back to the default value, if the table parameter is not set.
    - `0` Returns an error.
    - `1` Returns an empty result.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `0`
- **Default Table Value:** `0`
- **Range:** `-1` to `1`
- **DSN Parameter Name:** `erm`

`spider_error_write_mode`

- **Description:** Sends an empty result-set when writing to a backend server raises an error. Useful with applications that don't implement transaction replays.
    - `-1` Falls back to the default value, if the table parameter is not set.
    - `0` Returns an error.
    - `1` Returns an empty result-set on error.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `0`
- **Default Table Value:** `0`

- **Range:** `-1` to `1`
- **DSN Parameter Name:** `ewm`

`spider_first_read`

- **Description:** Number of first read records to start a range scan on the backend server. Spider needs to send the first record. Fetching the second record saves network round-trips, stopping the plan if the backend has a single record. First read and second read are used to warm up for background searches, third reads without using the spider_split_read and spider_semi_split_read system variables fetches the remaining data source in a single last fetch.
  - `-1` Use the table parameter.
  - `0` Usually retrieves records.
  - `1` and greater: Sets the number of first read records.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `0`
- **Default Table Value:** `2`
- **Range:** `-1` to `9223372036854775807`
- **DSN Parameter Name:** `frd`

`spider_force_commit`

- **Description:** Behavior when error occurs on `XA PREPARE`, `XA COMMIT`, and `XA ROLLBACK` statements.
  - `0` Returns the error.
  - `1` Returns the error when the `xid` doesn't exist, otherwise it continues processing the XA transaction.
  - `2` Continues processing the XA transaction, disregarding all errors.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `0`
- **Range:** `0` to `2`

`spider_general_log`

- **Description:** Whether Spider logs all commands to the General Log. Spider logs error codes according to the spider_log_result_errors system variable.
  - `OFF` Logs no commands.
  - `ON` Logs commands to the General Log.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Session Value:** `OFF`

`spider_ignore_comments`

- **Description:** Whether to unconditionally ignore COMMENT and CONNECTION strings without checking whether table options are specified.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB 11.3.0

`spider_index_hint_pushdown`

- **Description:** Whether to use pushdown index hints, like `force_index`.
  - `0` Do not use pushdown hints
  - `1` Use pushdown hints
- **Scope:** Global, Session

- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Session Value:** `OFF`
- **Introduced:** MariaDB 10.3.3

---

spider_init_sql_alloc_size

- **Description:** Initial size of the local SQL buffer.
  - `-1` Falls back to the default value, if the table parameter is not set.
  - `0` or greater: Size of the buffer.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `1024`
- **Default Table Value:** `1024`
- **DSN Parameter Name:** `isa`
- **Range:** `-1` to `2147483647`
- **Deprecated:** MariaDB 10.7.5, MariaDB 10.8.4, MariaDB 10.9.2

---

spider_internal_limit

- **Description:** Limits the number of records when acquired from a remote server.
  - `-1` The table parameter is adopted.
  - `0` or greater: Records limit.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `9223372036854775807`
- **Default Table Value:** `9223372036854775807`
- **Range:** `-1` to `9223372036854775807`
- **DSN Parameter Name:** `ilm`
- **Deprecated:** MariaDB 10.7.4, MariaDB 10.8.3

---

spider_internal_offset

- **Description:** Skip records when acquired from the remote server.
  - `-1` Falls back to the default value, if the table parameter is not set.
  - `0` or more : Number of records to skip.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `0`
- **Default Table Value:** `0`
- **Range:** `-1` to `9223372036854775807`
- **DSN Parameter Name:** `ios`
- **Deprecated:** MariaDB 10.7.4, MariaDB 10.8.3

---

spider_internal_optimize

- **Description:** Whether to perform push down operations for OPTIMIZE TABLE statements.
  - `-1` Falls back to the default value, if the table parameter is not set.
  - `0` Transmitted.
  - `1` Not transmitted.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `0`
- **Default Table Value:** `0`
- **Range:** `-1` to `1`
- **DSN Parameter Name:** `iom`

spider_internal_optimize_local

- **Description:** Whether to transmit to remote servers when OPTIMIZE TABLE statements are executed on the local server.
  - `-1` Falls back to the default value, if the table parameter is not set.
  - `0` Not transmitted.
  - `1` Transmitted.
- **Default Session Value:** `0`
- **Default Table Value:** `0`
- **Range:** `-1` to `1`
- **DSN Parameter Name:** `iol`

spider_internal_sql_log_off

- **Description:** Whether to log SQL statements sent to the remote server in the General Query Log.
  - Explicitly setting this system variable to either `ON` or `OFF` causes the Spider node to send a `SET sql_log_off` statement to each of the data nodes using the `SUPER` privilege.
  - `-1` Don't know or does not matter; don't send 'SET SQL_LOG_OFF' statement
  - `0` Send 'SET SQL_LOG_OFF 0' statement to data nodes (logs SQL statements to the remote server)
  - `1` Send 'SET SQL_LOG_OFF 1' statement to data nodes (doesn't log SQL statements to the remote server)
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric` (previously `boolean`)
- **Range:** `-1` to `1`
- **Default Session Value:** `-1` (previously `ON`)

spider_internal_unlock

- **Description:** Whether to transmit unlock tables to the connection of the table used with `SELECT` statements.
  - `0` Not transmitted.
  - `1` Transmitted.
- **Data Type:** `boolean`
- **Default Session Value:** `0`

spider_internal_xa

- **Description:** Whether to implement XA at the server- or storage engine-level. When using the server-level, set different values for the server_id system variable on all server instances to generate different `xid` values.
  - `OFF` Uses the storage engine protocol.
  - `ON` Uses the server protocol.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Session Value:** `OFF`

spider_internal_xa_id_type

- **Description:** The type of internal_xa id.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `0`
- **Range:** `-1` to `1`

spider_internal_xa_snapshot

- **Description:** Limitation for reading consistent data on all backend servers when using MariaDB's internal XA implementation and `START TRANSACTION WITH CONSISTENT SNAPSHOT`.

    - `0` Raise error when using a Spider table.
    - `1` Raise error when issued a `START TRANSACTION` statement.
    - `2` Takes a consistent snapshot on each backend, but loses global consistency.
    - `3` Starts transactions with XA, but removes `CONSISTENT SNAPSHOT`.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Range:** `0` to `3`
- **Default Session Value:** `0`

`spider_load_crd_at_startup`

- **Description:** Whether to load CRD from the system table at startup.
    - `-1` Use table parameter
    - `0` Do not load
    - `1` Load
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `1`
- **Range:** `-1` to `1`
- **Introduced:** MariaDB 10.3.3
- **Deprecated:** MariaDB 10.7.4, MariaDB 10.8.3

`spider_load_sts_at_startup`

- **Description:** Whether to load STS from the system table at startup.
    - `-1` Use table parameter
    - `0` Do not load
    - `1` Load
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Session Value:** `1`
- **Range:** `-1` to `1`
- **Introduced:** MariaDB 10.3.3
- **Deprecated:** MariaDB 10.7.4, MariaDB 10.8.3

`spider_local_lock_table`

- **Description:** Whether to push LOCK TABLES statements down to the remote server.
    - `0` Transmitted.
    - `1` Not transmitted.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `0`

`spider_lock_exchange`

- **Description:** Whether to convert SELECT... LOCK IN SHARE MODE and SELECT... FOR UPDATE statements into a LOCK TABLE statement.
    - `0` Not converted.
    - `1` Converted.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `0`

---

spider_log_result_error_with_sql

- **Description:** How to log SQL statements with result errors.
  - `0` No log
  - `1` Log error
  - `2` Log warning summary
  - `3` Log warning
  - `4` Log info (Added in MariaDB 10.5.4)
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `4`

---

spider_log_result_errors

- **Description:** Log results from data nodes to the Spider node in the Error Log. Performs no logging by default.
  - `0` : Logs no errors from data nodes.
  - `1` : Logs errors from data nodes.
  - `2` : Logs errors from data nodes, as well as warning summaries.
  - `3` : Logs errors from data nodes, as well as warning summaries and details.
  - `4` : Logs errors from data nodes, as well as warning summaries and details, and result summaries.
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `4`

---

spider_low_mem_read

- **Description:** Whether to use low memory mode when executing queries issued internally to remote servers that return result-sets.
  - `-1` Falls back to the default value, if the table parameter is not set.
  - `0` Doesn't use low memory mode.
  - `1` Uses low memory mode.
- **Data Type:** `numeric`
- **Default Session Value:** `1`
- **Default Table Value:** `1`
- **Range:** `-1` to `1`

---

spider_max_connections

- **Description:** Maximum number of connections from Spider to a remote MariaDB servers. Defaults to `0`, which is no limit.
- **Command-line:** `--spider-max-connections`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `0`
- **Range:** `0` to `99999`
- **Introduced:** MariaDB 10.3.3

---

spider_max_order

- **Description:** Maximum number of columns for `ORDER BY` operations.
  - `-1` The table parameter is adopted.
  - `0` and greater: Maximum number of columns.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `32767`

- **Default Table Value:** `32767`
- **Range:** `-1` to `32767`
- **DSN Parameter Name:** `mod`

`spider_multi_split_read`

- **Description:** Whether to divide a statement into multiple SQL statements sent to the remote backend server when the optimizer resolves range retrievals to multiple conditions.
  - `-1` Falls back to the default value, if the table parameter is not set.
  - `0` Doesn't divide statements.
  - `1` Divides statements.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `100`
- **Default Table Value:** `100`
- **Range:** `-1` to `2147483647`
- **DSN Parameter Name:** `msr`

`spider_net_read_timeout`

- **Description:** TCP timeout in seconds to declare remote backend servers unresponsive when reading from a connection. Change for high latency networks.
  - `-1` Falls back to the default value, if the table parameter is not set.
  - `0` Less than 1 second timeout.
  - `1` and greater: Timeout in seconds.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `600`
- **Default Table Value:** `600`
- **Range:** `-1` to `2147483647`
- **DSN Parameter Name:** `nrt`

`spider_net_write_timeout`

- **Description:** TCP timeout in seconds to declare remote backend servers unresponsive when writing to a connection. Change for high latency networks.
  - `-1` The table parameter is adopted.
  - `0` Less than 1 second timeout.
  - `1` and more: Timeout in seconds.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `600`
- **Default Table Value:** `600`
- **Range:** `-1` to `2147483647`
- **DSN Parameter Name:** `nwt`

`spider_ping_interval_at_trx_start`

- **Description:** Resets the connection with keepalive timeout in seconds by sending a ping.
  - `0` At every transaction.
  - `1` and greater: Number of seconds.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `3600`
- **Range:** `0` to `2147483647`

---

spider_quick_mode

- **Description:** Sets the backend query buffering to cache on the remote backend server or in the local buffer.
  - `-1` Falls back to the default value, if the table parameter is not set.
  - `0` Local buffering, it acquires records collectively with `store_result`.
  - `1` Remote buffering, it acquires records one by one. Interrupts don't wait and recovery on context switch back.
  - `2` Remote buffering, it acquires records one by one. Interrupts wait to the end of the acquisition.
  - `3` Local buffering, uses a temporary table on disk when the result-set is greater than the value of the spider_quick_page_size system variable.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `3`
- **Default Table Value:** `3`
- **Range:** `-1` to `3`
- **DSN Parameter Name:** `qmd`

---

spider_quick_page_byte

- **Description:** Memory limit by size in bytes in a page when acquired record by record.
  - `-1` The table parameter is used. When quick_mode is 1 or 2, Spider stores at least 1 record even if quick_page_byte is smaller than 1 record. When quick_mode is 3, quick_page_byte is used for judging using temporary tables. That is given priority when spider_quick_page_byte is set.
  - `0` or greater: Memory limit.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `10485760`
- **Range:** `-1` to `9223372036854775807`
- **Introduced:** MariaDB 10.4.3, MariaDB 10.3.13

---

spider_quick_page_size

- **Description:** Number of records in a page when acquired record by record.
  - `-1` The table parameter is adopted.
  - `0` or greater: Number of records.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `1024` (>=MariaDB 10.7), `-1` (<= MariaDB 10.6)
- **Default Table Value:** `100`
- **Range:** `-1` to `9223372036854775807`
- **DSN Parameter Name:** `qps`

---

spider_read_only_mode

- **Description:** Whether to allow writes on Spider tables.
  - `-1` Falls back to the default value, if the table parameter is not set.
  - `0` Allows writes to Spider tables.
  - `1` Makes tables read- only.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `0` (>=MariaDB 10.7), `-1` (<= MariaDB 10.6)
- **Default Table Value:** `0`
- **Range:** `-1` to `1`
- **DSN Parameter Name:** `rom`

---

spider_remote_access_charset

- **Description:** Forces the specified session character set when connecting to the backend server. This can improve

connection time performance.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Session Value:** `null`

`spider_remote_autocommit`

- **Description:** Sets the auto-commit mode when connecting to backend servers. This can improve connection time performance.
  - `-1` Doesn't change the auto-commit mode.
  - `0` Sets the auto-commit mode to `0`.
  - `1` Sets the auto-commit mode to `1`.
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `-1`
- **Range:** `-1` to `1`

`spider_remote_default_database`

- **Description:** Sets the local default database when connecting to backend servers. This can improve connection time performance.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Session Value:** Empty string

`spider_remote_sql_log_off`

- **Description:** Sets the sql_log_off system variable to use when connecting to backend servers.
  - `-1` Doesn't set the value.
  - `0` Doesn't log Spider SQL statements to remote backend servers.
  - `1` Logs SQL statements on remote backend
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `-1`
- **Range:** `-1` to `1`

`spider_remote_time_zone`

- **Description:** Forces the specified time zone setting when connecting to backend servers. This can improve connection performance when you know the time zone.
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Session Value:** `null`

`spider_remote_trx_isolation`

- **Description:** Sets the Transaction Isolation Level when connecting to the backend server.
  - `-1` Doesn't set the Isolation Level.
  - `0` Sets to the `READ UNCOMMITTED` level.
  - `1` Sets to the `READ COMMITTED` level.
  - `2` Sets to the `REPEATABLE READ` level.
  - `3` Sets to the `SERIALIZABLE` level.
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`

- **Default Session Value:** `-1`
- **Range:** `-1` to `3`

---

spider_remote_wait_timeout

- **Description:** Wait timeout in seconds on remote server. `-1` means not set.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `-1`
- **Range:** `-1` to `2147483647`
- **Introduced:** MariaDB 10.4.5

---

spider_reset_sql_alloc

- **Description:** Resets the per connection SQL buffer after an SQL statement executes.
  - `-1` Falls back to the default value, if the table parameter is not set.
  - `0` Doesn't reset.
  - `1` Resets.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `1`
- **Default Table Value:** `1`
- **Range:** `-1` to `1`
- **DSN Parameter Name:** `rsa`

---

spider_same_server_link

- **Description:** Enables the linking of a table to the same local instance.
  - `0` Disables linking.
  - `1` Enables linking.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Session Value:** `OFF`

---

spider_second_read

- **Description:** Number of second read records on the backend server when the first records are found from the first read. Spider continues scanning a range, adding a `LIMIT` using the spider_first_read and spider_second_read variables.
  - `-1` Falls back to the default value, if the table parameter is not set.
  - `0` Usually retrieves records.
  - `1` and greater: Number of records.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Default Session Value:** `0`
- **Default Table Value:** `0`
- **Range:** `-1` to `9223372036854775807`
- **DSN Parameter Name:** `srd`

---

spider_select_column_mode

- **Description:** Mode for column retrieval from remote backend server.
  - `-1` Falls back to the default value, if the table parameter is not set.
  - `0` Uses index columns when the `SELECT` statement can resolve with an index, otherwise uses all columns.
  - `1` Uses all columns judged necessary to resolve the query.
- **Scope:** Global, Session
- **Dynamic:** Yes

- **Data Type:** `numeric`
- **Default Session Value:** `1`
- **Default Table Value:** `1`
- **Range:** `-1` to `1`
- **DSN Parameter Name:** `scm`

`spider_selupd_lock_mode`

- **Description:** Local lock mode on `INSERT SELECT`.
  - `-1` Falls back to the default value, if the table parameter is not set.
  - `0` Takes no locks.
  - `1` Takes shared locks.
  - `2` Takes exclusive locks.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `1`
- **Default Table Value:** `1`
- **Range:** `-1` to `2`
- **DSN Parameter Name:** `slm#`

`spider_semi_split_read`

- **Description:** Whether to use chunk retrieval with offset and limit parameters on SQL statements sent to the remote backend server when using the spider_split_read system variable.
  - `-1` Falls back to the default value, if the table parameter is not set.
  - `0` Doesn't use chunk retrieval.
  - `1 or more` Uses chunk retrieval.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `2`
- **Default Table Value:** `2`
- **Range:** `-1` to `2147483647`
- **DSN Parameter Name:** `ssr#`

`spider_semi_split_read_limit`

- **Description:** Sets the limit value for the spider_semi_split_read system variable.
  - `-1` Falls back to the default value, if the table parameter is not set.
  - `0` or more: The limit value.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `9223372036854775807`
- **Default Table Value:** `9223372036854775807`
- **Range:** `-1` to `9223372036854775807`
- **DSN Parameter Name:** `ssl#`

`spider_semi_table_lock`

- **Description:** Enables semi-table locking. This adds a LOCK TABLES statement to SQL executions sent to the remote backend server when using non-transactional storage engines to preserve consistency between roundtrips.
  - `0` Disables semi-table locking.
  - `1` Enables semi-table locking.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `0` (>=MariaDB 10.8), `1` (<= MariaDB 10.7)
- **Range:** `0` to `1`

- **DSN Parameter Name:** `stl#`

`spider_semi_table_lock_connection`

- **Description:** Whether to use multiple connections with semi-table locking. To enable semi-table locking, use the spider_semi_table_lock system variable.
  - `-1` Falls back to the default value, if the table parameter is not set.
  - `0` Uses the same connection.
  - `1` Uses different connections.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `1`
- **Default Table Value:** `1`
- **Range:** `-1` to `1`
- **DSN Parameter Name:** `stc#`

`spider_semi_trx`

- **Description:** Enables semi-transactions. This controls transaction consistency when an SQL statement is split into multiple statements issued to the backend servers. You can preserve or relax consistency as need. Spider encapsulates auto-committed SQL statements within a transaction on the remote backend server. When using `READ COMMITTED` or `READ UNCOMMITTED` transaction isolation levels to force consistency, set the spider_semi_trx_isolation system variable to `2`.
  - `0` Disables semi-transaction consistency.
  - `1` Enables semi-transaction consistency.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Session Value:** `ON`

`spider_semi_trx_isolation`

- **Description:** Set consistency during range SQL execution when spider_sync_trx_isolation is 1
  - `-1` OFF
  - `0` READ UNCOMMITTED
  - `1` READ COMMITTED
  - `2` REPEATABLE READ
  - `3` SERIALIZABLE
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `-1`
- **Range:** `-1` to `3`

`spider_skip_default_condition`

- **Description:** Whether to compute condition push downs.
  - `-1` Falls back to the default value, if the table parameter is not set.
  - `0` Computes condition push downs.
  - `1` Doesn't compute condition push downs.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `0`
- **Default Table Value:** `0`
- **Range:** `-1` to `1`
- **DSN Parameter Name:** `sdc`

---

spider_skip_parallel_search

- **Description:** Whether to skip parallel search by specific conditions.
  - `-1` :use table parameter
  - `0` :not skip
  - `1` :skip parallel search if query is not SELECT statement
  - `2` :skip parallel search if query has SQL_NO_CACHE
  - `3` :1+2
- **Commandline:** `--spider-skip-parallel-search=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `0`
- **Range:** `-1` to `3`
- **Introduced:** MariaDB 10.3.3

---

spider_slave_trx_isolation

- **Description:** Transaction isolation level when Spider table is used by slave SQL thread.
  - `-1` off
  - `0` read uncommitted
  - `1` read committed
  - `2` repeatable read
  - `3` serializable
- **Commandline:** `--spider-slave-trx-isolation=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `-1`
- **Range:** `-1` to `3`
- **Introduced:** MariaDB 10.4.3, MariaDB 10.3.13

---

spider_split_read

- **Description:** Number of records in chunk to retry the result when a range query is sent to remote backend servers.
  - `-1` Falls back to the default value, if the table parameter is not set.
  - `0` or more: Number of records.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `9223372036854775807`
- **Default Table Value:** `9223372036854775807`
- **Range:** `-1` to `9223372036854775807`
- **DSN Parameter Name:** `srd`

---

spider_store_last_crd

- **Description:** Whether to store last CRD result in the system table.
  - `-1` Use table parameter.
  - `0` Do not store last CRD result in the system table.
  - `1` Store last CRD result in the system table.
- **Commandline:** `--spider-store-last-crd=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `1`
- **Range:** `-1` to `1`
- **Introduced:** MariaDB 10.3.3
- **Deprecated:** MariaDB 10.7.4, MariaDB 10.8.3

---

spider_store_last_sts

- **Description:** Whether to store last STS result in the system table.
  - `-1` Use table parameter.
  - `0` Do not store last STS result in the system table.
  - `1` Store last STS result in the system table.
- **Commandline:** `--spider-store-last-sts=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `1`
- **Range:** `-1` to `1`
- **Introduced:** MariaDB 10.3.3
- **Deprecated:** MariaDB 10.7.4, MariaDB 10.8.3

---

spider_strict_group_by

- **Description:** Whether to use columns in select clause strictly for group by clause
  - `-1` Use the table parameter.
  - `0` Do not strictly use columns in select clause for group by clause
  - `1` Use columns in select clause strictly for group by clause
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `1`
- **Range:** `-1` to `1`
- **Introduced:** MariaDB 10.5.4

---

spider_sts_bg_mode

- **Description:** Enables background confirmation for table statistics. When background confirmation is enabled, Spider uses one thread per partition to maintain table status. Disable when the spider_sts_interval system variable is set to `0` , which causes Spider to always retrieve the latest information as need. It is effective, when the spider_sts_interval system variable is set to `10` .
  - `-1` Falls back to the default value, if the table parameter is not set.
  - `0` Disables background confirmation.
  - `1` Enables background confirmation (create thread per table/partition).
  - `2` Enables background confirmation (use static threads). (from MariaDB 10.)
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `2`
- **Range:** `-1` to `2`
- **DSN Parameter Name:** `sbm`

---

spider_sts_interval

- **Description:** Time interval of table statistics from the remote backend servers.
  - `-1` Falls back to the default value, if the table parameter is not set.
  - `0` Retrieves the latest table statistics on request.
  - `1` or more: Interval in seconds for table state confirmation.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `10`
- **Default Table Value:** `10`
- **Range:** `-1` to `2147483647`
- **DSN Parameter Name:** `siv`

---

spider_sts_mode

- **Description:** Table statistics mode. Mode for table statistics. The SHOW command is used at the table level default.
  - `-1,0` Uses the table parameter.
  - `1` Uses the `SHOW` command.
  - `2` Uses the Information Schema.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `1`
- **Default Table Value:** `1`
- **Range:** `-1` to `2`
- **DSN Parameter Name:** `smd`
- **Deprecated:** MariaDB 10.7.4, MariaDB 10.8.3

---

spider_sts_sync

- **Description:** Synchronizes table statistics in partitioned tables.
  - `-1` Falls back to the default value, if the table parameter is not set.
  - `0` Doesn't synchronize table statistics in partitioned tables.
  - `1` Synchronizes table state when opening a table, doesn't synchronize after opening.
  - `2` Synchronizes table statistics.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Session Value:** `0`
- **Default Table Value:** `0`
- **Range:** `-1` to `2`
- **DSN Parameter Name:** `ssy`

---

spider_support_xa

- **Description:** XA Protocol for mirroring and for multi-shard transactions.
  - `1` Enables XA Protocol for these Spider operations.
  - `0` Disables XA Protocol for these Spider operations.
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Table Value:** `1`

---

spider_suppress_comment_ignored_warning

- **Description:** Whether to suppress warnings that table COMMENT or CONNECTION strings are ignored due to specified table options.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB 11.3.0

---

spider_sync_autocommit

- **Description:** Whether to push down local auto-commits to remote backend servers.
  - `OFF` Pushes down local auto-commits.
  - `ON` Doesn't push down local auto-commits.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Session Value:** `ON`

---

spider_sync_sql_mode

- **Description:** Whether to sync sql_mode.
  - `OFF` No sync
  - `ON` Sync
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `ON`
- **Introduced:** MariaDB 10.4.7

---

spider_sync_time_zone

- **Description:** Whether to push the local time zone down to remote backend servers.
  - `OFF` Doesn't synchronize time zones.
  - `ON` Synchronize time zones.
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Session Value:** `OFF`
- **Removed:** MariaDB 10.3.9

---

spider_sync_trx_isolation

- **Description:** Pushes local transaction isolation levels down to remote backend servers.
  - `OFF` Doesn't push down local isolation levels.
  - `ON` Pushes down local isolation levels.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Session Value:** `ON`

---

spider_table_crd_thread_count

- **Description:** Static thread count of table crd.
- **Commandline:** `--spider-table-crd-thread-count=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `10`
- **Range:** `1` to `4294967295`
- **Introduced:** MariaDB 10.3.3

---

spider_table_init_error_interval

- **Description:** Interval in seconds where the same error code is returned if table initialization fails. Use to protect against infinite loops in table links.
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1`
- **Range:** `0` to `4294967295`

---

spider_table_sts_thread_count

- **Description:** Static thread count of table sts.
- **Commandline:** `--spider-table-sts-thread-count=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `10`
- **Range:** `1` to `4294967295`

- **Introduced:** MariaDB 10.3.3

spider_udf_ct_bulk_insert_interval

- **Description:** Interval in milliseconds between bulk inserts at copying. For use with the UDF spider_copy_tables, which copies table data linked to a Spider table from the source server to destination server using bulk insert. If this interval is 0, it may cause higher write traffic.
  - `-1` Uses the UDF parameter.
  - `0` and more: Time in milliseconds.
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `10`
- **Default Table Value:** `10`
- **Range:** `-1` to `2147483647`
- **Deprecated:** MariaDB 10.7.4, MariaDB 10.8.3
- **Removed:** MariaDB 10.10

spider_udf_ct_bulk_insert_rows

- **Description:** Number of rows to insert at a time when copying during bulk inserts.
  - `-1, 0` : Uses the table parameter.
  - `1` and more: Number of rows
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `100`
- **Default Table Value:** `100`
- **Range:** `-1` to `9223372036854775807`
- **Deprecated:** MariaDB 10.7.4, MariaDB 10.8.3
- **Removed:** MariaDB 10.10

spider_udf_ds_bulk_insert_rows

- **Description:** Number of rows inserted at a time during bulk inserts when the result-set is stored in a temporary table on executing a UDF.
  - `-1, 0` Uses the UDF parameter.
  - `1` or more: Number of rows
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `3000`
- **Default Table Value:** `3000`
- **Range:** `-1` to `9223372036854775807`
- **Deprecated:** MariaDB 10.7.4, MariaDB 10.8.3
- **Removed:** MariaDB 10.10

spider_udf_ds_table_loop_mode

- **Description:** Whether to store the result-set in the same temporary table when the temporary table list count for UDF is less than the result-set count on UDF execution.
  - `-1` Falls back to the default value, if the table parameter is not set.
  - `0` Drops records.
  - `1` Inserts the last table.
  - `2` Inserts the first table and loops again.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `-1` to `2`
- **Deprecated:** MariaDB 10.7.4, MariaDB 10.8.3

- **Removed:** MariaDB 10.10

---

spider_udf_ds_use_real_table

- **Description:** Whether to use real table for temporary table list.
  - `-1` Use UDF parameter.
  - `0` Do not use real table.
  - `1` Use real table.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `-1` to `1`
- **Deprecated:** MariaDB 10.7.4, MariaDB 10.8.3
- **Removed:** MariaDB 10.10

---

spider_udf_table_lock_mutex_count

- **Description:** Mutex count of table lock for Spider UDFs.
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `20`
- **Range:** `1` to `4294967295`
- **Removed:** MariaDB 10.10

spider_udf_table_mon_mutex_count

- **Description:** Mutex count of table mon for Spider UDFs.
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `20`
- **Range:** `1` to `4294967295`
- **Removed:** MariaDB 10.10

---

spider_use_all_conns_snapshot

- **Description:** Whether to pass `START TRANSACTION WITH SNAPSHOT` statements to all connections.
  - `OFF` Doesn't pass statement to all connections.
  - `ON` Passes statement to all connections.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Session Value:** `OFF`

---

spider_use_cond_other_than_pk_for_update

- **Description:** Whether to use all conditions even if condition has a primary key.
  - `0` Don't use all conditions
  - `1` Use all conditions
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1`
- **Range:** `0` to `1`
- **Introduced:** MariaDB 10.3.13, MariaDB 10.4.3

---

spider_use_consistent_snapshot

- **Description:** Whether to push a local `START TRANSACTION WITH CONSISTENT` statement down to remote backend servers.
  - `OFF` Doesn't push the local statement down.
  - `ON` Pushes the local statement down.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

spider_use_default_database

- **Description:** Whether to use the default database.
  - `OFF` Doesn't use the default database.
  - `ON` Uses the default database.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `ON`

---

spider_use_flash_logs

- **Description:** Whether to push FLUSH LOGS statements down to remote backend servers.
  - `OFF` Doesn't push the statement down.
  - `ON` Pushes the statement down.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

spider_use_handler

- **Description:** Converts HANDLER SQL statements. When the spider_sync_trx_isolation system variable is set to `0`, Spider disables HANDLER conversions to prevent use of the statement on the SERIALIZABLE isolation level.
  - `-1` Falls back to the default value, if the table parameter is not set.
  - `0` Converts HANDLER statements into SELECT statements.
  - `1` Passes HANDLER to the remote backend server.
  - `2` Converts SQL statements to HANDLER statements.
  - `3` Converts SQL statements to HANDLER statements and HANDLER statements to SQL statements.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Default Table Value:** `0`
- **Range:** `-1` to `3`
- **DSN Parameter Name:** `uhd`
- **Deprecated:** MariaDB 10.7.4, MariaDB 10.8.3
- **Removed:** MariaDB 10.10

---

spider_use_pushdown_udf

- **Description:** When using a UDF function in a condition and the engine_condition_pushdown system variable is set to `1`, whether to execute the UDF function locally or push it down.
  - `-1` Falls back to the default value, if the table parameter is not set.
  - `0` Doesn't transmit the UDF
  - `1` Transmits the UDF.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `-1`
- **Default Table Value:** `1`

- **Range:** `-1` to `1`
- **DSN Parameter Name:** `upu`

`spider_use_snapshot_with_flush_tables`

- **Description:** Whether to encapsulate FLUSH LOGS and UNLOCK TABLES statements when `START TRANSACTION WITH CONSISTENT` and `FLUSH TABLE WITH READ LOCK` statements are sent to the remote backend servers.
  - `0` : No encapsulation.
  - `1` : Encapsulates, only when the spider_use_all_conns_snapshot system variable i set to `1` .
  - `2` : Synchronizes the snapshot using a LOCK TABLES statement and [flush|FLUSH TABLES]] at the XA transaction level. This is only effective when the spider_use_all_cons_snapshot system variable is set to `1` .
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `2`

`spider_use_table_charset`

- **Description:** Whether to use the local table character set for the remote backend server connections.
  - `-1` Falls back to the default value, if the table parameter is not set.
  - `0` Use `utf8` .
  - `1` Uses the table character set.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1`
- **Default Table Value:** `1`
- **Range:** `-1` to `1`
- **DSN Parameter Name:** `utc`

`spider_version`

- **Description:** The current Spider version. Removed in MariaDB 10.9.2 when the Spider version number was matched with the server version.
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Removed:** MariaDB 10.9.2

`spider_wait_timeout`

- **Description:** Wait timeout in seconds of setting to remote server. `-1` means not set.
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `604800`
- **Range:** `-1` to `2147483647`
- **Introduced:** MariaDB 10.4.5

`spider_xa_register_mode`

- **Description:** Mode of XA transaction register into system table.
  - `0` Register all XA transactions
  - `1` Register only write XA transactions
- **Command-line:** `--spider-xa-register-mode=#`
- **Scope:** Global, Session
- **Dynamic:** Yes
- **Data Type:** `numeric`

- **Default Value:** `1`
- **Range:** `0` to `1`
- **Introduced:** MariaDB 10.3.3
- **Deprecated:** MariaDB 10.7.4, MariaDB 10.8.3

5.3.19.8 Spider Table Parameters

When a table uses the Spider storage engine, the following Spider table parameters can be set in the `COMMENT` clause of the CREATE TABLE statement. Many Spider table parameters have corresponding system variables, so they can be set for all Spider tables on the node. For additional information, see the Spider System Variables page.

Contents

`access_balances`

- **Description:** Connection load balancing integer weight.
- **Default Table Value:** `0`
- **DSN Parameter Name:** `abl`

`active_link_count`

- **Description:** Number of active remote servers, for use in load balancing read connections
- **Default Table Value:** `all backends`
- **DSN Parameter Name:** `alc`

`casual_read`

- **Description:**
- **Default Table Value:**
- **DSN Parameter Name:**
- **Introduced:** Spider 3.2

`database`

- **Description:** Database name for reference table that exists on remote backend server.

- **Default Table Value:** `local table database`
- **DSN Parameter Name:** `database`

default_file

- **Description:** Configuration file used when connecting to remote servers. When the `default_group` table variable is set, this variable defaults to the values of the `--defaults-extra-file` or `--defaults-file` options. When the `default_group` table variable is not set, it defaults to `none`.
- **Default Table Value:** `none`
- **DSN Parameter Name:** `dff`

default_group

- **Description:** Group name in configuration file used when connecting to remote servers.
- **Default Table Value:** `none`
- **DSN Parameter Name:** `dfg`

delete_all_rows_type

- **Description:**
- **Default Table Value:**
- **DSN Parameter Name:**
- **Introduced:** Spider 3.2

host

- **Description:** Host name of remote server.
- **Default Table Value:** `localhost`
- **DSN Parameter Name:** `host`

idx000

- **Description:** When using an index on Spider tables for searching, Spider uses this hint to search the remote table. The remote table index is related to the Spider table index by this hint. The number represented by `000` is the index ID, which is the number of the index shown by the `SHOW CREATE TABLE` statement. `000` is the Primary Key. For instance, `idx000 "force index(PRIMARY)"` (in abbreviated format `idx000 "f PRIMARY"`).
  - `f` force index
  - `u` use index
  - `ig` ignore index
- **Default Table Value:** `none`

internal_delayed

- **Description:** Whether to transmit existence of delay to remote servers when executing an `INSERT DELAYED` statement on local server.
  - `0` Doesn't transmit.
  - `1` Transmits.
- **Default Table Value:** `0`
- **DSN Parameter Name:** `idl`

link_status

- **Description:** Change status of the remote backend server link.
  - `0` Doesn't change status.
  - `1` Changes status to `OK`.
  - `2` Changes status to `RECOVERY`.
  - `3` Changes status to no more in group communication.
- **Default Table Value:** `0`
- **DSN Parameter Name:** `lst`

monitoring_bg_interval

- **Description:** Interval of background monitoring in microseconds.
- **Default Table Value:** `10000000`
- **DSN Parameter Name:** `mbi`

monitoring_bg_kind

- **Description:** Kind of background monitoring to use.
  - `0` Disables background monitoring.
  - `1` Monitors connection state.
  - `2` Monitors state of table without `WHERE` clause.
  - `3` Monitors state of table with `WHERE` clause (currently unsupported).
- **Default Table Value:** `0`
- **DSN Parameter Name:** `mbk`

monitoring_kind

- **Description:** Kind of monitoring.
  - `0` Disables monitoring
  - `1` Monitors connection state.
  - `2` Monitors state of table without `WHERE` clause.
  - `3` Monitors state of table with `WHERE` clause (currently unsupported).
- **Default Table Value:** `0`
- **DSN Parameter Name:** `mkd`

monitoring_limit

- **Description:** Limits the number of records in the monitoring table. This is only effective when Spider monitors the state of a table, which occurs when the `monitoring_kind` table variable is set to a value greater than `1`.
- **Default Table Value:** `1`
- **Range:** `0` upwards
- **DSN Parameter Name:** `mlt`

monitoring_server_id

- **Description:** Preferred monitoring `@@server_id` for each backend failure. You can use this to geo-localize backend servers and set the first Spider monitoring node to contact for failover. In the event that this monitor fails, other monitoring nodes are contacted. For multiple copy backends, you can set a lazy configuration with a single MSI instead of one per backend.
- **Default Table Value:** `server_id`
- **DSN Parameter Name:** `msi`

password

- **Description:** Remote server password.
- **Default Table Value:** `none`
- **DSN Parameter Name:** `password`

port

- **Description:** Remote server port.
- **Default Table Value:** `3306`
- **DSN Parameter Name:** `port`

priority

- **Description:** Priority. Used to define the order of execution. For instance, Spider uses priority when deciding the order in which to lock tables on a remote server.
- **Default Table Value:** `1000000`
- **DSN Parameter Name:** `prt`

query_cache

- **Description:** Passes the option for the Query Cache when issuing `SELECT` statements to the remote server.
  - `0` No option passed.
  - `1` Passes the `SQL_CACHE` option.
  - `2` Passes the `SQL_NO_CACHE` option.
- **Default Table Value:** `0`
- **DSN Parameter Name:** `qch`

read_rate

- **Description:** Rate used to calculate the amount of time Spider requires when executing index scans.
- **Default Table Value:** `0.0002`
- **DSN Parameter Name:** `rrt`

scan_rate

- **Description:** Rate used to calculate the amount of time Spider requires when scanning tables.
- **Default Table Value:** `0.0001`
- **DSN Parameter Name:** `srt`

server

- **Description:** Server name. Used when generating connection information with `CREATE SERVER` statements.
- **Default Table Value:** `none`
- **DSN Parameter Name:** `srv`

socket

- **Description:** Remote server socket.
- **Default Table Value:** `none`
- **DSN Parameter Name:** `socket`

ssl_ca

- **Description:** Path to the Certificate Authority file.
- **Default Table Value:** `none`
- **DSN Parameter Name:** `sca`

ssl_capath

- **Description:** Path to directory containing trusted TLS CA certificates in PEM format.
- **Default Table Value:** `none`
- **DSN Parameter Name:** `scp`

ssl_cert

- **Description:** Path to the certificate file.
- **Default Table Value:** `none`
- **DSN Parameter Name:** `scr`

ssl_cipher

- **Description:** List of allowed ciphers to use with TLS encryption.
- **Default Table Value:** `none`
- **DSN Parameter Name:** `sch`

ssl_key

- **Description:** Path to the key file.
- **Default Table Value:** `none`
- **DSN Parameter Name:** `sky`

ssl_verify_server_cert

- **Description:** Enables verification of the server's Common Name value in the certificate against the host name used when connecting to the server.
  - `0` Disables verification.
  - `1` Enables verification.
- **Default Table Value:** `0`
- **DSN Parameter Name:** `svc`

table

- **Description:** Destination table name.
- **Default Table Value:** `Same table name`
- **DSN Parameter Name:** `tbl`

5.3.19.9 Spider Status Variables

Contents

The following status variables are associated with the Spider storage engine. See Server Status Variables for a complete list of status variables that can be viewed with SHOW STATUS.

See also the Full list of MariaDB options, system and status variables.

`Spider_direct_aggregate`

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Spider_direct_delete`

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.3.3

`Spider_direct_order_limit`

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Spider_direct_update`

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.3.3

`Spider_mon_table_cache_version`

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Spider_mon_table_cache_version_req`

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`

`Spider_parallel_search`

- **Description:**
- **Scope:** Global, Session
- **Data Type:** `numeric`
- **Introduced:** MariaDB 10.3.3

1.2.9.6 Spider Functions

1.2.9.6.1 SPIDER_BG_DIRECT_SQL

1.2.9.6.2 SPIDER_COPY_TABLES

1.2.9.6.3 SPIDER_DIRECT_SQL

1.2.9.6.4 SPIDER_FLUSH_TABLE_MON_CACHE

1.1.1.2.9.3.32 Spider mysql Database Tables

1.1.1.2.9.3.32.1 mysqlspider_link_failed_log Table

1.1.1.2.9.3.32.2 mysqlspider_link_mon_servers Table

1.1.1.2.9.3.32.3 mysqlspider_tables Table

1.1.1.2.9.3.32.4 mysqlspider_table_crd Table

1.1.1.2.9.3.32.5 mysqlspider_table_position_for_recovery Table

1.1.1.2.9.3.32.6 mysqlspider_table_sts Table

1.1.1.2.9.3.32.7 mysqlspider_xa Table

1.1.1.2.9.3.32.8 mysqlspider_xa_failed_log Table

1.1.1.2.9.3.32.9 mysqlspider_xa_member Table

1.1.1.2.9.1.1.44 Information Schema SPIDER_ALLOC_MEM Table

1.1.1.2.9.1.1.45 Information Schema SPIDER_WRAPPER_PROTOCOLS Table

5.3.19.14 Spider Differences Between SpiderForMySQL and MariaDB

Contents

SQL Syntax

- With `SpiderForMySQL`, the CREATE TABLE statement uses `CONNECTION` to define spider table variables whereas MariaDB uses `COMMENT`.

Features

- `HANDLER` can not be translated to SQL in MariaDB
- Concurrent background search is not yet implemented in MariaDB
- Vertical partitioning storage engine VP is not implemented in MariaDB
- `CREATE TABLE` can use table discovery in MariaDB
- `JOIN` performance improvement using `join_cache_level>1` and `join_buffer_size` in MariaDB

5.3.19.15 Spider Case Studies

A list of users or clients that are using Spider and agree to be referenced:

- Tencent Games. They handle 100TB data on 396 Spider nodes and 2800 data nodes. They use this cluster for their online games.
- Kadokawa Corporation
- MicroAd, Inc.
- Sansan, Inc.
- teamLab Inc.
- CCM Benchmark http://www.slideshare.net/skysql/ccm-escape-case-study-skysql-paris-meetup-17122013
- Softlink http://fr.slideshare.net/skysql/galaxy-big-data-with-mariadb
- Gavo http://wiki.ivoa.net/internal/IVOA/InterOpMay2014NewTechnologies/Spider-MariaDB.pdf
- Blablacar Using for storing various logs
- Believe Digital Using for back office analytics queries to aggregate multi billions tables in real time

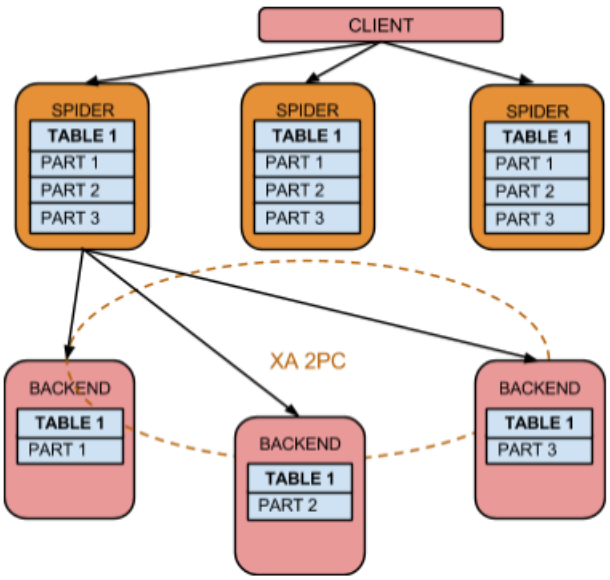
5.3.19.16 Spider Benchmarks

This is best run on a cluster of 3 nodes intel NUC servers 12 virtual cores model name : Intel® Core(TM) i3-3217U CPU @ 1.80GHz

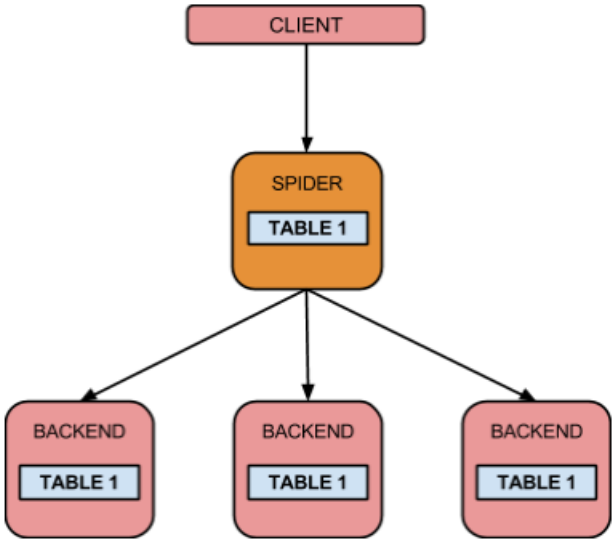
All nodes have been running a mysqlslap client attached to the local spider node in the best run.

```
/usr/local/skysql/mysql-client/bin/mysqlslap --user=skysql --password=skyvodka --
host=192.168.0.201 --port=5012 -i1000000 -c32 -q "insert into test(c) values('0-31091-
138522330')" --create-schema=test
```

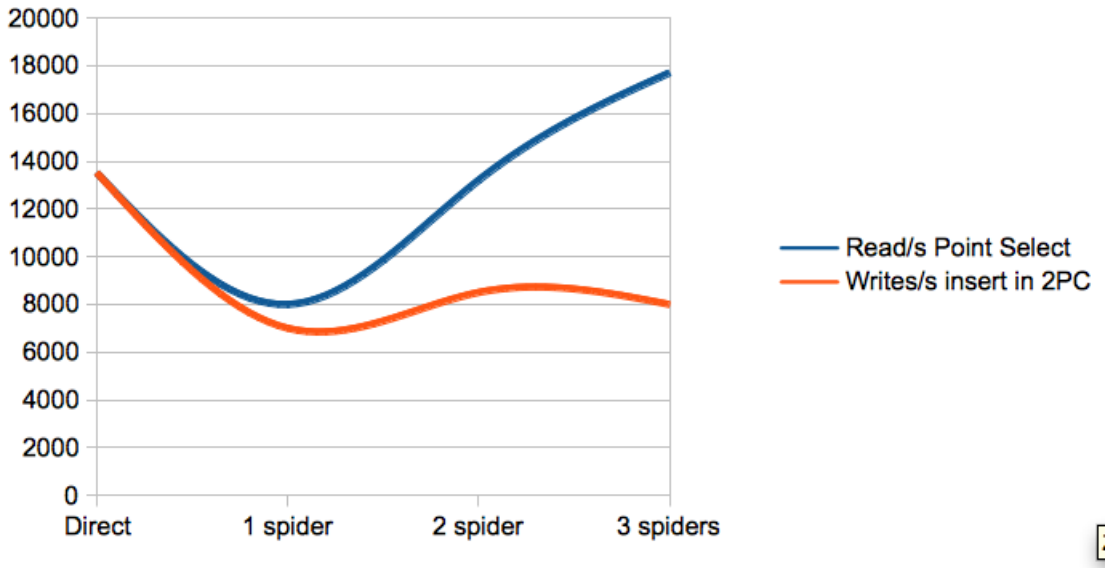
```
spider_conn_recycle_mode=1;
```



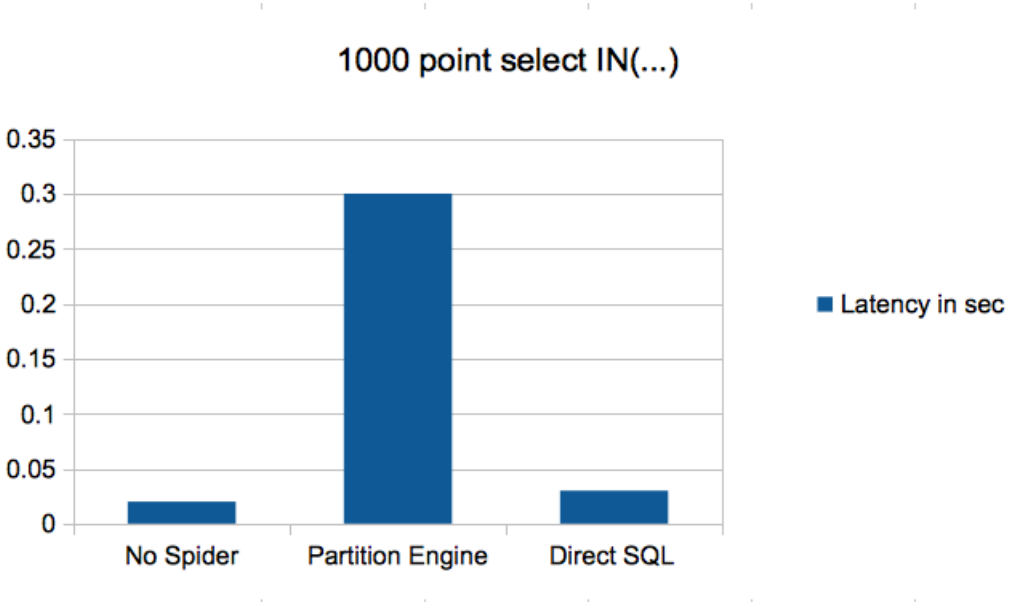
The read point select is produce with a 10M rows sysbench table



The write insert a single string into a memory table



Before Engine Condition Push Down patch .



Spider can benefit by 10% additional performance with Independent Storage Engine Statistics.

```
set global use_stat_tables='preferably';
USE backend;
ANALYZE TABLE sbtest;
```

5.3.19.17 Spider FAQ

Contents

What does "[ERROR] mysqld: Can't find record in 'spider_tables'" mean?

This happens when you have a Spider table defined that does not point to an existing table on a data node.

Are there minimum Spider settings?

```
myisam-recover=FORCE,BACKUP
```

MariaDB until 10.1.1

```
optimizer_switch='engine_condition_pushdown=on'
```

MariaDB until 10.3.7

When using spider_autoincrement_mode = 0, partitioned Spider tables work as spider_autoincrement_mode = 1 see : MDEV-21404

What does "select spider_ping_table()" in the general log mean?

This is used by Spider monitoring to ask other monitoring nodes the status of a table.

Do I need a primary key on physical tables?

Not having a primary key will generate errors for resynchronizing tables via spider_copy_table().

Can I use Spider on top of Galera shards?

Yes, XA transactions can be disabled from Spider. Until Galera 4.0 fully supports xa transactions, spider can point to a maxscale proxy that can manage transparent node election in case of failure inside a shard group. Note that disabling XA will break cross shard WRITES in case of transaction ROLLBACK. This architecture need to be used with care if you have a highly transactional workload that can generate cross shard deadlocks.

What are the most used architectures for Spider HA?

- Delegation of shard node replication using asynchronous replication and slave election with GTID.
- Delegation of shard node replication via active passive HA solutions.
- Shard builds via replication into Spider tables is interesting when you can route READS to a pool of Spider nodes reattaching the shards.

What are the most used architectures for Spider Map Reduce?

- Map reduce in Spider is limited to a single table. Building spider on top of some views can eliminate the need to use joins.
- Replication to universal tables to every shard is commonly used to enable the views on each shard.

What about Grants on shards?

- When using MRR and BKA (and you do so with network storage), when Spider needs to create temporary tables on

the backends, use the CREATE TEMPORARY TABLES privilege. Spider can still switch to a lower performance solution using spider_bka_mode=2, or Query push down or range predicate using spider_bka_mode=0

1.1.1.2.9.1.1.15 Information Schema ENGINES Table

1.1.1.2.9.2.6 PERFORMANCE_SCHEMA Storage Engine

5.3.22 Storage Engine Development

Storage Engine FAQ

Are storage engines designed for MySQL compatible with MariaDB? In most cas...

Engine-defined New Table/Field/Index Attributes

A storage engine can allow the user to specify additional attributes per index, field, or table.

Table Discovery

Mechanism for an engine to tell the server that the table exists

Table Discovery (before 10.0.2)

This page describes the old discovery API, created in MySQL for NDB Cluste...

There are 1 related questions.

5.3.22.1 Storage Engine FAQ

Are storage engines designed for MySQL compatible with MariaDB?

In most cases, yes. MariaDB tries to keep API compatibility with MySQL, even across major versions.

Will storage engines created for MariaDB work in MySQL?

It will mostly work. It would need #ifdef's to adjust to MySQL-5.6 API, for example, for multi-read-range API, for table discovery API, etc. But most of the code will work as is, without any changes.

Do storage engine binaries need to be recompiled for MariaDB?

Yes. You will need to recompile the storage engine against the exact version of MySQL or MariaDB you intend to run it on. This is due to the version of the server being stored in the storage engine binary, and the server will refuse to load it if it was compiled for a different version.

5.3.22.2 Engine-defined New Table/Field/Index Attributes

Contents

In MariaDB, a storage engine can allow the user to specify additional attributes per index, field, or table. The engine needs to declare what attributes it introduces.

API

There are three new members in the `handlerton` structure, they can be set in the engine's initialization function as follows:

```
example_hton->table_options= example_table_option_array;
example_hton->field_options= example_field_option_array;
example_hton->index_options= example_index_option_array;
```

The arrays are declared statically, as in the following example:

```
static MYSQL_THDVAR_ULONG(varopt_default, PLUGIN_VAR_RQCMDARG,
  "default value of the VAROPT table option", NULL, NULL, 5, 0, 100, 0);

struct ha_table_option_struct
{
  char *strparam;
  ulonglong ullparam;
  uint enumparam;
  bool boolparam;
  ulonglong varparam;
};

ha_create_table_option example_table_option_list[]=
{
  HA_TOPTION_NUMBER("NUMBER", ullparam, UINT_MAX32, 0, UINT_MAX32, 10),
  HA_TOPTION_STRING("STR", strparam),
  HA_TOPTION_ENUM("ONE_OR_TWO", enumparam, "one,two", 0),
  HA_TOPTION_BOOL("YESNO", boolparam, 1),
  HA_TOPTION_SYSVAR("VAROPT", varopt, varparam),
  HA_TOPTION_END
};
```

The engine declares a structure `ha_table_option_struct` that will hold values of these new attributes.

And it describes these attributes to MySQL by creating an array of `HA_TOPTION_*` macros. Note a detail: these macros expect a structure called `ha_table_option_struct`, if the structure is called differently, a `#define` will be needed.

There are five supported kinds of attributes:

macro name	attribure value type	corresponding C type	additional parameters of a macro
`HA_TOPTION_NUMBER`	an integer number	`unsigned long long`	a default value, minimal allowed value, maximal allowed value, a factor, that any allowed should be a multiple of.
`HA_TOPTION_STRING`	a string	`char *`	none. The default value is a null pointer.
`HA_TOPTION_ENUM`	one value from a list of allowed values	`unsigned int`	a string with a comma-separated list of allowed values, and a default value as a number, starting from 0.
`HA_TOPTION_BOOL`	a boolean	`bool`	a default value
`HA_TOPTION_SYSVAR`	defined by the system variable	defined by the system variable	system variable name

*Do **not** use `enum` for your `HA_TOPTION_ENUM` C structure members, the size of the `enum` depends on the compiler, and even on the compilation options, and the plugin API uses only types with known storage sizes.*

In all macros the first two parameters are name of the attribute as should be used in SQL in the `CREATE TABLE` statement, and the name of the corresponding member of the `ha_table_option_struct` structure.

The `HA_TOPTION_SYSVAR` stands aside a bit. It does not specify the attribute type or the default value, instead it binds the attribute to a system variable. The attribute type and the range of allowed values will be the same as of the corresponding system variable. The attribute **default value** will be the **current value** of its system variable. And unlike other attribute types that are only stored in the `.frm` file if explicitly set in the `CREATE TABLE` statement, the `HA_TOPTION_SYSVAR` attributes are always stored. If the system variable value is changed, it will not affect existing tables. Note that for this very reason, if a table was created in the old version of a storage engine, and a new version has introduced a `HA_TOPTION_SYSVAR` attribute, the attribute value in the old tables will be the **default** value of the system variable, not its **current** value.

The array ends with a `HA_TOPTION_END` macro.

Field and index (key) attributes are declared similarly using `HA_FOPTION_*` and `HA_IOPTION_*` macros.

When in a `CREATE TABLE` statement, the `::create()` handler method is called, the table attributes are available in the `table_arg->s->option_struct`, field attributes - in the `option_struct` member of the individual fields (objects of the

`Field` class), index attributes - in the `option_struct` member of the individual keys (objects of the `KEY` class).

Additionally, they are available in most other handler methods: the attributes are stored in the `.frm` file and on every open MySQL makes them available to the engine by filling the corresponding `option_struct` members of the table, fields, and keys.

The `ALTER TABLE` needs a special support from the engine. MySQL compares old and new table definitions to decide whether it needs to rebuild the table or not. As the semantics of the engine declared attributes is unknown, MySQL cannot make this decision by analyzing attribute values - this is delegated to the engine. The `HA_CREATE_INFO` structure has three new members:

```
ha_table_option_struct *option_struct;          ///< structure with parsed table options
ha_field_option_struct **fields_option_struct;  ///< array of field option structures
ha_index_option_struct **indexes_option_struct; ///< array of index option structures
```

The engine (in the `::check_if_incompatible_data()` method) is responsible for comparing new values of the attributes from the `HA_CREATE_INFO` structure with the old values from the table and returning `COMPATIBLE_DATA_NO` if they were changed in such a way that requires the table to be rebuild.

The example of declaring the attributes and comparing the values for the `ALTER TABLE` can be found in the EXAMPLE engine.

SQL

The engine declared attributes can be specified per field, index, or table in the `CREATE TABLE` or `ALTER TABLE` . The syntax is the conventional:

```
CREATE TABLE ... (
  field ... [attribute=value [attribute=value ...]],
  ...
  index ... [attribute=value [attribute=value ...]],
  ...
) ...  [attribute=value [attribute=value ...]]
```

All values must be specified as literals, not expressions. The value of a boolean option may be specified as one of YES, NO, ON, OFF, 1, or 0. A string value may be specified either quoted or not, as an identifier (if it is a valid identifier, of course). Compare with the old behavior:

```
CREATE TABLE ... ENGINE=FEDERATED CONNECTION='mysql://root@127.0.0.1';
```

where the value of the ENGINE attribute is specified not quoted, while the value of the CONNECTION is quoted.

When an attribute is set, it will be stored with the table definition and shown in the `SHOW CREATE TABLE;`
. To remove an attribute from a table definition use `ALTER TABLE`
to set its value to a `DEFAULT`
.

The values of unknown attributes or attributes with the illegal values cause an error by default. But with ALTER TABLE one can change the storage engine and some previously valid attributes may become unknown — to the new engine. They are not removed automatically, though, because the table might be altered back to the first engine, and these attributes will be valid again. Still SHOW CREATE TABLE will comment these unknown attributes out in the output, otherwise they would make a generated CREATE TABLE statement invalid.

With the `IGNORE_BAD_TABLE_OPTIONS`
sql mode this behavior changes. Unknown attributes do not cause an error, they only result in a warning. And SHOW CREATE TABLE will not comment them out. This mode is implicitly enabled in the replication slave thread.

5.3.22.3 Table Discovery

In MariaDB it is not always necessary to run an explicit `CREATE TABLE` statement for a table to appear. Sometimes a table may already exist in the storage engine, but the server does not know about it, because there is no `.frm` file for this table. This can happen for various reasons; for example, for a cluster engine the table might have been created in the cluster by another MariaDB server node. Or for the engine that supports table shipping a table file might have been simply copied into the MariaDB data directory. But no matter what the reason is, there is a mechanism for an engine to tell the server that the table exists. This mechanism is called **table discovery** and if an engine wants the server to discover its tables, the engine should support the table discovery API.

Contents

There are two different kinds of table discovery — a fully automatic discovery and a user-assisted one. In the former, the engine can automatically discover the table whenever an SQL statement needs it. In MariaDB, the Archive and Sequence engines support this kind of discovery. For example, one can copy a `t1.ARZ` file into the database directory and immediately start using it — the corresponding `.frm` file will be created automatically. Or one can select from say, the `seq_1_to_10` table without any explicit `CREATE TABLE` statement.

In the latter, user-assisted, discovery the engine does not have enough information to discover the table all on its own. But it can discover the table structure if the user provides certain additional information. In this case, an explicit `CREATE TABLE` statement is still necessary, but it should contain no table structure — only the table name and the table attributes. In MariaDB, the FederatedX storage engine supports this. When creating a table, one only needs to specify the `CONNECTION` attribute and the table structure — fields and indexes — will be provided automatically by the engine.

Automatic Discovery

As far as automatic table discovery is concerned, the tables, from the server point of view, may appear, disappear, or change structure anytime. Thus the server needs to be able to ask whether a given table exists and what its structure is. It needs to be notified when a table structure changes outside of the server. And it needs to be able to get a list of all (unknown to the server) tables, for statements like `SHOW TABLES`. The server does all that by invoking specific methods of the `handlerton`:

```
const char **tablefile_extensions;
int (*discover_table_names)(handlerton *hton, LEX_STRING *db, MY_DIR *dir,
                            discovered_list *result);
int (*discover_table_existence)(handlerton *hton, const char *db,
                                const char *table_name);
int (*discover_table)(handlerton *hton, THD* thd, TABLE_SHARE *share);
```

handlerton::tablefile_extensions

Engines that store tables in separate files (one table might occupy many files with different extensions, but having the same base file name) should store the list of possible extensions in the **tablefile_extensions** member of the `handlerton` (earlier this list was returned by the `handler::bas_ext()` method). This will significantly simplify the discovery implementation for these engines, as you will see below.

handlerton::discover_table_names()

When a user asks for a list of tables in a specific database — for example, by using `SHOW TABLES` or by selecting from `INFORMATION_SCHEMA.TABLES` — the server invokes **discover_table_names()** method of the `handlerton`. For convenience this method, besides the database name in question, gets the list of all files in this database directory, so that the engine can look for table files without doing any filesystem i/o. All discovered tables should be added to the `result` collector object. It is defined as

```
class discovered_list
{
  public:
  bool add_table(const char *tname, size_t tlen);
  bool add_file(const char *fname);
};
```

and the engine should call `result->add_table()` or `result->add_file()` for every discovered table (use

`add_file()` if the name to add is in the MariaDB file name encoding, and `add_table()` if it's a true table name, as shown in `SHOW TABLES`).

If the engine is file-based, that is, it has non-empty list in the `tablefile_extensions`, this method is optional. For any file-based engine that does not implement `discover_table_names()`, MariaDB will automatically discover the list of all tables of this engine, by looking for files with the extension `tablefile_extensions[0]`.

handlerton::discover_table_existence()

In some rare cases MariaDB needs to know whether a given table exists, but does not particularly care about this table structure (for example, when executing a `DROP TABLE` statement). In these cases, the server uses the **discover_table_existence()** method to find out whether a table with the given name exists in the engine.

This method is optional. For the engine that does not implement it, MariaDB will look for files with the `tablefile_extensions[0]`, if possible. But if the engine is not file-based, MariaDB will use the `discover_table()` method to perform a full table discovery. While this will allow determining correctly whether a table exists, a full discovery is usually slower than the simple existence check. In other words, engines that are not file-based might want to support `discover_table_existence()` method as a useful optimization.

handlerton::discover_table()

This is the main method of table discovery, the heart of it. The server invokes it when it wants to use the table. The **discover_table()** method gets the `TABLE_SHARE` structure, which is not completely initialized — only the table and the database name (and a path to the table file) are filled in. It should initialize this `TABLE_SHARE` with the desired table structure.

MariaDB provides convenient and easy to use helpers that allow the engine to initialize the `TABLE_SHARE` with minimal efforts. They are the `TABLE_SHARE` methods `init_from_binary_frm_image()` and `init_from_sql_statement_string()`.

TABLE_SHARE::init_from_binary_frm_image()

This method is used by engines that use "frm shipping" — such as Archive or NDB Cluster in MySQL. An frm shipping engine reads the frm file for a given table, exactly as it was generated by the server, and stores it internally. Later it can discover the table structure by using this very frm image. In this sense, a separate frm file in the database directory becomes redundant, because a copy of it is stored in the engine.

TABLE_SHARE::init_from_sql_statement_string()

This method allows initializing the `TABLE_SHARE` using a conventional SQL `CREATE TABLE` syntax.

TABLE_SHARE::read_frm_image()

Engines that use frm shipping need to get the frm image corresponding to a particular table (typically in the `handler::create()` method). They do it via the **read_frm_image()** method. It returns an allocated buffer with the binary frm image, that the engine can use the way it needs.

TABLE_SHARE::free_frm_image()

The frm image that was returned by `read_frm_image()` must be freed with the **free_frm_image()**.

HA_ERR_TABLE_DEF_CHANGED

One of the consequences of automatic discovery is that the table definition might change when the server doesn't expect it to. Between two `SELECT` queries, for example. If this happens, if the engine detects that the server is using an outdated version of the table definition, it should return a **HA_ERR_TABLE_DEF_CHANGED** handler error. Depending on when in the query processing this error has happened, MariaDB will either re-discover the table and execute the query with the correct table structure, or abort the query and return an error message to the user.

TABLE_SHARE::tabledef_version

The previous paragraph doesn't cover one important question — how can the engine know that the server uses an outdated table definition? The answer is — by checking the **tabledef_version**, the table definition version. Every table gets a unique `tabledef_version` value. Normally it is generated automatically when a table is created. When a table is discovered the engine can force it to have a specific `tabledef_version` value (simply by setting it in the `TABLE_SHARE` before calling the `init_from_binary_frm_image()` or `init_from_sql_statement_string()` methods).

Now the engine can compare the table definition version that the server is using (from any handler method it can be accessed as `this->table->s->tabledef_version`) with the version of the actual table definition. If they differ — it is `HA_ERR_TABLE_DEF_CHANGED`.

Assisted discovery

Assisted discovery is a lot simpler from the server point of view, a lot more controlled. The table cannot appear or disappear at will, one still needs explicit DDL statements to manipulate it. There is only one new handlerton method that the server uses to discover the table structure when a user has issued an explicit `CREATE TABLE` statement without declaring any columns or indexes.

```
int (*discover_table_structure)(handlerton *hton, THD* thd,
                                TABLE_SHARE *share, HA_CREATE_INFO *info);
```

The assisted discovery API is pretty much independent from the automatic discovery API. An engine can implement either of them or both (or none); there is no requirement to support automatic discovery if only assisted discovery is needed.

handlerton::discover_table_structure()

Much like the `discover_table()` method, the **discover_table_structure()** handlerton method gets a partially initialized `TABLE_SHARE` with the table name, database name, and a path to table files filled in, but without a table structure. Unlike `discover_table()`, here the `TABLE_SHARE` has all the engine-defined table attributes in the the `TABLE_SHARE::option_struct` structure. Based on the values of these attributes the `discover_table_structure()` method should initialize the `TABLE_SHARE` with the desired set of fields and keys. It can use `TABLE_SHARE` helper methods `init_from_binary_frm_image()` and `init_from_sql_statement_string()` for that.

The role of `.frm` files

Before table discovery was introduced, MariaDB used `.frm` files to store the table definition. But now the engine can store the table definition (if the engine supports automatic discovery, of course), and `.frm` files become redundant. Still, the server *can* use `.frm` files for such an engine — but they are no longer the only source of the table definition. Now `.frm` files are merely a *cache* of the table definition, while the original authoritative table definition is stored in the engine. Like any cache, its purpose is to reduce discovery attempts for a table. The engine decides whether it makes sense to cache table definition in the `.frm` file or not (see the second argument for the `TABLE_SHARE::init_from_binary_frm_image()`). For example, the Archive engine uses `.frm` cache, while the Sequence engine does not. In other words, MariaDB creates `.frm` files for Archive tables, but not for Sequence tables.

The cache is completely transparent for a user; MariaDB makes sure that it always stores the actual table definition and invalidates the `.frm` file automatically when it becomes out of date. This can happen, for example, if a user copies a new Archive table into the datadir and forgets to delete the `.frm` file of the old table with the same name.

5.3.23 Converting Tables from MyISAM to InnoDB

Contents

The task

You have decided to change one or more tables from MyISAM to InnoDB. That should be as simple as `ALTER TABLE foo ENGINE=InnoDB`. But you have heard that there might be some subtle issues.

This describes possible issues that may arise and what to do about them.

Recommendation. One way to assist in searching for issues in is to do (at least in *nix)

```
mysqldump --no-data --all-databases >schemas
egrep 'CREATE|PRIMARY' schemas    # Focusing on PRIMARY KEYs
egrep 'CREATE|FULLTEXT' schemas   # Looking for FULLTEXT indexes
egrep 'CREATE|KEY' schemas        # Looking for various combinations of indexes
```

Understanding how the indexes work will help you better understand what might run faster or slower in InnoDB.

INDEX Issues

(Most of these Recommendations and some of these Facts have exceptions.)

Fact. Every InnoDB table has a PRIMARY KEY. If you do not provide one, then the first non-NULL UNIQUE key is used. If that can't be done, then a 6-byte, hidden, integer is provided.

Recommendation. Look for tables without a PRIMARY KEY. Explicitly specify a PRIMARY KEY, even if it's an artificial AUTO_INCREMENT. This is not an absolute requirement, but it is a stronger admonishment for InnoDB than for MyISAM. Some day you may need to walk through the table; without an explicit PK, you can't do it.

Fact. The fields of the PRIMARY KEY are included in each Secondary key.

- Check for redundant indexes with this in mind.

  ```
  PRIMARY KEY(id),
  INDEX(b), -- effectively the same as INDEX(b, id)
  INDEX(b, id) -- effectively the same as INDEX(b)
  ```

  (Keep one of the INDEXes, not both)

- Note subtle things like

  ```
  PRIMARY KEY(id),
  UNIQUE(b), -- keep for uniqueness constraint
  INDEX(b, id) -- DROP this one
  ```

- Also, since the PK and the data coexist:

  ```
  PRIMARY KEY(id),
  INDEX(id, b) -- DROP this one; it adds almost nothing
  ```

Contrast. This feature of MyISAM is not available in InnoDB; the value of 'id' will start over at 1 for each different value of 'abc':

```
id INT UNSIGNED NOT NULL AUTO_INCREMENT,
PRIMARY KEY (abc, id)
```

A way to simulate the MyISAM 'feature' might be something like: What you want is this, but it won't work because it is referencing the table twice:

```
INSERT INTO foo
    (other, id, ...)
    VALUES
    (123, (SELECT MAX(id)+1 FROM foo WHERE other = 123), ...);
```

Instead, you need some variant on this. (You may already have a BEGIN...COMMIT.)

```
BEGIN;
SELECT @id := MAX(id)+1 FROM foo WHERE other = 123 FOR UPDATE;
INSERT INTO foo
    (other, id, ...)
    VALUES
    (123, @id, ...);
COMMIT;
```

Having a transaction is mandatory to prevent another thread from grabbing the same id.

Recommendation. Look for such PRIMARY KEYs. If you find such, ponder how to change the design. There is no straightforward workaround. However, the following may be ok. (Be sure that the datatype for id is big enough since it won't start over.):

```
id INT UNSIGNED NOT NULL AUTO_INCREMENT,
PRIMARY KEY (abc, id),
UNIQUE(id)
```

Recommendation. Keep the PRIMARY KEY short. If you have Secondary keys, remember that they include the fields of the PK. A long PK would make the Secondary keys bulky. Well, maybe not — if the is a lot of overlap in fields. Example: `PRIMARY KEY(a,b,c), INDEX(c,b,a)` — no extra bulk.

Recommendation. Check AUTO_INCREMENT sizes.

- BIGINT is almost never needed. It wastes at least 4 bytes per row (versus INT).
- Always use UNSIGNED and NOT NULL.
- MEDIUMINT UNSIGNED (16M max) might suffice instead of INT
- Be sure to be pessimistic — it is painful to ALTER.

Contrast. "Vertical Partitioning". This is where you artificially split a table to move bulky columns (eg, a BLOB) into another, parallel, table. It is beneficial in MyISAM to avoid stepping over the blob when you don't need to read it. InnoDB stores BLOB and TEXT differently — 767 bytes are in the record, the rest is in some other block. So, it may (or may not) be worth putting the tables back together. Caution: An InnoDB row is limited to 8KB, and the 767 counts against that.

Fact. FULLTEXT (prior to MariaDB 10.0.5) and SPATIAL indexes are not available in InnoDB. Note that MyISAM and InnoDB FULLTEXT indexes use different stopword lists and different system variables.

Recommendation. Search for such indexes. Keep such tables in MyISAM. Better yet, do Vertical Partitioning (see above) to split out the minimum number of columns from InnoDB.

Fact. The maximum length of an INDEX is different between the Engines. (This change is not likely to hit you, but watch out.) MyISAM allows 1000 bytes; InnoDB allows 767 bytes, just big enough for a

```
VARCHAR(255) CHARACTER SET utf8.

ERROR 1071 (42000): Specified key was too long; max key length is 767 bytes
```

Fact. The PRIMARY KEY is included in the data. Hence, SHOW TABLE STATUS will show and `Index_length` of 0 bytes (or 16KB) for a table with no secondary indexes. Otherwise, `Index_length` is the total size for the secondary keys.

Fact. The PRIMARY KEY is included in the data. Hence, exact match by PK may be a little faster with InnoDB. And, "range" scans by PK are likely to be faster.

Fact. A lookup by Secondary Key traverses the secondary key's BTree, grabs the PRIMARY KEY, then traverses the PK's BTree. Hence, secondary key lookups are a little more cumbersome in InnoDB.

Contrast. The fields of the PRIMARY KEY are included in each Secondary key. This may lead to "Using index" (in the EXPLAIN plan) for InnoDB for cases where it did not happen in MyISAM. (This is a slight performance boost, and counteracts the double-lookup otherwise needed.) However, when "Using index" would be useful on the PRIMARY KEY, MyISAM would do an "index scan", yet InnoDB effectively has to do a "table scan".

Same as MyISAM. Almost always

```
INDEX(a)   -- DROP this one because the other one handles it.
INDEX(a,b)
```

Contrast. The data is stored in PK order. This means that "recent" records are 'clustered' together at the end. This may give you better 'locality of reference' than in MyISAM.

Same as MyISAM. The optimizer almost never uses two indexes in a single SELECT. (5.1 will occasionally do "index merge".) SELECT in subqueries and UNIONs can independently pick indexes.

Subtle issue. When you DELETE a row, the AUTO_INCREMENT id will be burned. Ditto for REPLACE, which is a DELETE plus an INSERT.

Very subtle issue. Replication occurs on COMMIT. If you have multiple threads using transactions, the AUTO_INCREMENTs can arrive at a slave out of order. One transaction BEGINs, grabs an id. Then another transaction grabs an id but COMMITs before the first finishes.

Same as MyISAM. "Prefix" indexing is usually bad in both InnoDB and MyISAM. Example: `INDEX(foo(30))`

Non-INDEX Issues

Disk space for InnoDB is likely to be 2-3 times as much as for MyISAM.

MyISAM and InnoDB use RAM radically differently. If you change all your tables, you should make significant adjustments:

- key_buffer_size — small but non-zero; say, 10M;
- innodb_buffer_pool_size — 70% of available RAM

InnoDB has essentially no need for CHECK, OPTIMIZE, or ANALYZE. Remove them from your maintenance scripts. (No real harm if you keep them.)

Backup scripts may need checking. A MyISAM table can be backed up by copying three files. With InnoDB this is only possible if innodb_file_per_table is set to 1. Before MariaDB 10.0, capturing a table or database for copying from production to a development environment was not possible. Change to mysqldump. Since MariaDB 10.0 a hot copy can be created - see Backup and restore overview.

Before MariaDB 5.5, the DATA DIRECTORY table option was not supported for InnoDB. Since MariaDB 5.5 it is supported, but only in CREATE TABLE. INDEX DIRECTORY has no effect, since InnoDB does not use separate files for indexes. To better balance the workload through several disks, the paths of some InnoDB log files can also be changed.

Understand autocommit and BEGIN/COMMIT.

- (default) autocommit = 1: In the absence of any BEGIN or COMMIT statements, every statement is a transaction by itself. This is close to the MyISAM behavior, but is not really the best.
- autocommit = 0: COMMIT will close a transaction and start another one. To me, this is kludgy.
- (recommended) BEGIN...COMMIT gives you control over what sequence of operation(s) are to be considered a transaction and "atomic". Include the ROLLBACK statement if you need to undo stuff back to the BEGIN.

Perl's DBIx::DWIW and Java's JDBC have API calls to do BEGIN and COMMIT. These are probably better than 'executing' BEGIN and COMMIT.

Test for errors everywhere! Because InnoDB uses row-level locking, it can stumble into deadlocks that you are not expecting. The engine will automatically ROLLBACK to the BEGIN. The normal recovery is to redo, beginning at the BEGIN. Note that this is a strong reason to have BEGINs.

LOCK/UNLOCK TABLES — remove them. Replace them (sort of) with BEGIN ... COMMIT. (LOCK will work if innodb_table_locks is set to 1, but it is less efficient, and may have subtle issues.)

In 5.1, ALTER ONLINE TABLE can speed up some operations significantly. (Normally ALTER TABLE copies the table over and rebuilds the indexes.)

The "limits" on virtually everything are different between MyISAM and InnoDB. Unless you have huge tables, wide rows, lots of indexes, etc, you are unlikely to stumble into a different limit.

Mixture of MyISAM and InnoDB? This is OK. But there are caveats.

- RAM settings should be adjusted to accordingly.
- JOINing tables of different Engines works.
- A transaction that affects tables of both types can ROLLBACK InnoDB changes, but will leave MyISAM changes intact.
- Replication: MyISAM statements are replicated when finished; InnoDB statements are held until the COMMIT.

FIXED (vs DYNAMIC) is meaningless in InnoDB.

PARTITION — You can partition MyISAM and InnoDB tables. Remember the screwball rule: You must either

- have no UNIQUE (or PRIMARY) keys, or
- have the value you are "partitioning on" in every UNIQUE key.

The former is not advised for InnoDB. The latter is messy if you want an AUTO_INCREMENT.

PRIMARY KEY in PARTITION — Since every key must include the field on which you are PARTITIONing, how can AUTO_INCREMENT work? Well, there seems to be a convenient special case:

- This works: PRIMARY KEY(autoinc, partition_key)
- This does not work for InnoDB: PRIMARY KEY(partition_key, autoinc)

That is, an AUTO_INCREMENT will correctly increment, and be unique across all PARTITINOs, when it is the first field of the PRIMARY KEY, but not otherwise.

5.3.24 Machine Learning with MindsDB

Contents

Overview

MindsDB is a third-party application that interfaces with MariaDB Server to provide Machine Learning capabilities through SQL. The interface is done via the Connect Storage Engine.

Installation

To get a functional MariaDB - MindsDB installation, one needs to install the following components:

- **MindsDB**: follow the instructions in the project's official documentation.
- **Connect Storage Engine** must be enabled for the integration to work. See installing the connect storage engine.

MindsDB connects to MariaDB Server via a regular user to setup a dedicated database called `mindsdb`. Which user will be used is specified within MindsDB's configuration file.

For example, if MindsDB is installed locally, one can create a user called `mindsdb@localhost`. MindsDB only authenticates via the mysql_native_password plugin, hence one must set a password for the user:

```
CREATE USER mindsdb@localhost;
SET PASSWORD for mindsdb@localhost=PASSWORD("password");
```

The user must be granted the global FILE privilege and all privileges on the `mindsdb` database.

```
GRANT FILE on *.* to mindsdb@localhost;
GRANT ALL on mindsdb.* to mindsdb@localhost;
```

Assuming MindsDB is in the python path one can start up MindsDB with the following parameters:

```
python -m mindsdb --config=$CONFIG_PATH --api=http,mysql
```

Make sure `$CONFIG_PATH` points to the appropriate MindsDB configuration file.

Usage

Always consult the project's official documentation for up-to-date usage scenarios as MindsDB is an actively developed project.

For a step-by-step example, you can consult the following blog post.

If the connection between MindsDB and MariaDB is successful, you should see the `mindsdb` database present and two tables within it: `commands` and `predictors`.

MindsDB, as an AutoML framework does all the work when it comes to training the AI model. What is necessary is to pass it the initial data, which MindsDB retrieves via a SELECT statement. This can be done by inserting into the `predictors` table.

```
INSERT INTO `predictors`
       (`name`, `predict`, `select_data_query`)
VALUES ('bikes_model', 'count', 'SELECT * FROM test.bike_data');
```

The values inserted into predictors act as a command instructing MindsDB to:

1. Train a model called 'bikes_model'
2. From the input data, learn to predict the 'count' column.
3. The input data is generated via the select statement 'SELECT * FROM test.bike_data'. The `select_data_query` should be a valid select that MindsDB can run against MariaDB.

5.4 Plugins

MariaDB supports the use of plugins, software components that may be added to the core software without having to rebuild the MariaDB server from source code. Therefore, plugins can be loaded at start-up, or loaded and unloaded while the server is running without interruption. Plugins are commonly used for adding desired storage engines, additional security requirements, and logging special information about the server.

Plugin Overview

Basics of listing, installing and uninstalling plugins.

Information on Plugins

Information on installed and disabled plugins on a MariaDB Server.

Plugin SQL Statements

List of SQL statements related to plugins.

Creating and Building Plugins
Documentation on how to create new plugins and build existing ones.

MariaDB Audit Plugin
Logging user activity with the MariaDB Audit Plugin.

Authentication Plugins
Authentication plugins allow various authentication methods to be used, and new ones developed.

Password Validation Plugins
Ensuring that user passwords meet certain minimal security requirements.

Key Management and Encryption Plugins
MariaDB uses plugins to handle key management and encryption of data.

MariaDB Replication & Cluster Plugins
Plugins related to MariaDB replication and other replication cluster systems.

Storage Engines
Various storage engines available for MariaDB.

Other Plugins
Information on installing and using other plugins.

There are 5 related questions.

5.4.1 Plugin Overview

Contents

Plugins are server components that enhance MariaDB in some way. These can be anything from new storage engines, plugins for enhancing full-text parsing, or even small enhancements, such as a plugin to get a timestamp as an integer.

Querying Plugin Information

There are a number of ways to see which plugins are currently active.

A server almost always has a large number of active plugins, because the server contains a large number of built-in plugins, which are active by default and cannot be uninstalled.

Querying Plugin Information with `SHOW PLUGINS`

The SHOW PLUGINS statement can be used to query information about all active plugins.

For example:

```
SHOW PLUGINS\G;
*********************** 1. row ***********************
   Name: binlog
 Status: ACTIVE
   Type: STORAGE ENGINE
Library: NULL
License: GPL
*********************** 2. row ***********************
   Name: mysql_native_password
 Status: ACTIVE
   Type: AUTHENTICATION
Library: NULL
License: GPL
*********************** 3. row ***********************
   Name: mysql_old_password
 Status: ACTIVE
   Type: AUTHENTICATION
Library: NULL
License: GPL
...
```

If a plugin's `Library` column has a `NULL` value, then the plugin is built-in, and it cannot be uninstalled.

Querying Plugin Information with information_schema.PLUGINS

The information_schema.PLUGINS table can be queried to get more detailed information about plugins.

For example:

```
SELECT * FROM information_schema.PLUGINS\G
...
*************************** 6. row ***************************
           PLUGIN_NAME: CSV
        PLUGIN_VERSION: 1.0
         PLUGIN_STATUS: ACTIVE
           PLUGIN_TYPE: STORAGE ENGINE
   PLUGIN_TYPE_VERSION: 100003.0
        PLUGIN_LIBRARY: NULL
PLUGIN_LIBRARY_VERSION: NULL
         PLUGIN_AUTHOR: Brian Aker, MySQL AB
    PLUGIN_DESCRIPTION: CSV storage engine
        PLUGIN_LICENSE: GPL
           LOAD_OPTION: FORCE
       PLUGIN_MATURITY: Stable
   PLUGIN_AUTH_VERSION: 1.0
*************************** 7. row ***************************
           PLUGIN_NAME: MEMORY
        PLUGIN_VERSION: 1.0
         PLUGIN_STATUS: ACTIVE
           PLUGIN_TYPE: STORAGE ENGINE
   PLUGIN_TYPE_VERSION: 100003.0
        PLUGIN_LIBRARY: NULL
PLUGIN_LIBRARY_VERSION: NULL
         PLUGIN_AUTHOR: MySQL AB
    PLUGIN_DESCRIPTION: Hash based, stored in memory, useful for temporary tables
        PLUGIN_LICENSE: GPL
           LOAD_OPTION: FORCE
       PLUGIN_MATURITY: Stable
   PLUGIN_AUTH_VERSION: 1.0
...
```

If a plugin's `PLUGIN_LIBRARY` column has the `NULL` value, then the plugin is built-in, and it cannot be uninstalled.

Querying Plugin Information with `mysql.plugin`

The mysql.plugin table can be queried to get information about installed plugins.

This table only contains information about plugins that have been installed via the following methods:

- The INSTALL SONAME statement.
- The INSTALL PLUGIN statement.

- The mariadb-plugin utility.

This table does not contain information about:

- Built-in plugins.
- Plugins loaded with the --plugin-load-add option.
- Plugins loaded with the --plugin-load option.

This table only contains enough information to reload the plugin when the server is restarted, which means it only contains the plugin name and the plugin library.

For example:

```
SELECT * FROM mysql.plugin;

+------+------------+
| name | dl         |
+------+------------+
| PBXT | libpbxt.so |
+------+------------+
```

Installing a Plugin

There are three primary ways to install a plugin:

- A plugin can be installed dynamically with an SQL statement.
- A plugin can be installed with a mariadbd option, but it requires a server restart.
- A plugin can be installed with the mariadb-plugin utility, while the server is completely offline.

When you are installing a plugin, you also have to ensure that:

- The server's plugin directory is properly configured, and the plugin's library is in the plugin directory.
- The server's minimum plugin maturity is properly configured, and the plugin is mature enough to be installed.

Installing a Plugin Dynamically

A plugin can be installed dynamically by executing either the INSTALL SONAME or the INSTALL PLUGIN statement.

If a plugin is installed with one of these statements, then a record will be added to the mysql.plugins table for the plugin. This means that the plugin will automatically be loaded every time the server restarts, unless specifically uninstalled or deactivated.

Installing a Plugin with `INSTALL SONAME`

You can install a plugin dynamically by executing the INSTALL SONAME statement. INSTALL SONAME installs all plugins from the given plugin library. This could be required for some plugin libraries.

For example, to install all plugins in the `server_audit` plugin library (which is currently only the server_audit audit plugin), you could execute the following:

```
INSTALL SONAME 'server_audit';
```

Installing a Plugin with `INSTALL PLUGIN`

You can install a plugin dynamically by executing the INSTALL PLUGIN statement. INSTALL PLUGIN installs a single plugin from the given plugin library.

For example, to install the server_audit audit plugin from the `server_audit` plugin library, you could execute the following:

```
INSTALL PLUGIN server_audit SONAME 'server_audit';
```

Installing a Plugin with Plugin Load Options

A plugin can be installed with a mariadbd option by providing either the --plugin-load-add or the --plugin-load option.

If a plugin is installed with one of these options, then a record will **not** be added to the mysql.plugins table for the plugin. This means that if the server is restarted without the same option set, then the plugin will **not** automatically be loaded.

Installing a Plugin with `--plugin-load-add`

You can install a plugin with the --plugin-load-add option by specifying the option as a command-line argument to mariadbd or by specifying the option in a relevant server option group in an option file.

The --plugin-load-add option uses the following format:

- Plugins can be specified in the format `name=library`, where `name` is the plugin name and `library` is the plugin library. This format installs a single plugin from the given plugin library.
- Plugins can also be specified in the format `library`, where `library` is the plugin library. This format installs all plugins from the given plugin library.
- Multiple plugins can be specified by separating them with semicolons.

For example, to install all plugins in the `server_audit` plugin library (which is currently only the `server_audit` audit plugin) and also the `ed25519` authentication plugin from the `auth_ed25519` plugin library, you could set the option to the following values on the command-line:

```
$ mariadbd --user=mysql --plugin-load-add='server_audit' --plugin-load-add='ed25519=auth_ed25519'
```

You could also set the option to the same values in an option file:

```
[mariadb]
...
plugin_load_add = server_audit
plugin_load_add = ed25519=auth_ed25519
```

Special care must be taken when specifying both the --plugin-load option and the --plugin-load-add option together. The --plugin-load option resets the plugin load list, and this can cause unexpected problems if you are not aware. The --plugin-load-add option does **not** reset the plugin load list, so it is much safer to use. See Specifying Multiple Plugin Load Options for more information.

Installing a Plugin with `--plugin-load`

You can install a plugin with the --plugin-load option by specifying the option as a command-line argument to mariadbd or by specifying the option in a relevant server option group in an option file.

The --plugin-load option uses the following format:

- Plugins can be specified in the format `name=library`, where `name` is the plugin name and `library` is the plugin library. This format installs a single plugin from the given plugin library.
- Plugins can also be specified in the format `library`, where `library` is the plugin library. This format installs all plugins from the given plugin library.
- Multiple plugins can be specified by separating them with semicolons.

For example, to install all plugins in the `server_audit` plugin library (which is currently only the `server_audit` audit plugin) and also the `ed25519` authentication plugin from the `auth_ed25519` plugin library, you could set the option to the following values on the command-line:

```
$ mariadbd --user=mysql --plugin-load='server_audit;ed25519=auth_ed25519'
```

You could also set the option to the same values in an option file:

```
[mariadb]
...
plugin_load = server_audit;ed25519=auth_ed25519
```

Special care must be taken when specifying the `--plugin-load` option multiple times, or when specifying both the `--plugin-load` option and the `--plugin-load-add` option together. The `--plugin-load` option resets the plugin load list, and this can cause unexpected problems if you are not aware. The `--plugin-load-add` option does **not** reset the plugin load list, so it is much safer to use. See Specifying Multiple Plugin Load Options for more information.

Specifying Multiple Plugin Load Options

Special care must be taken when specifying the `--plugin-load` option multiple times, or when specifying both the `--plugin-load` option and the `--plugin-load-add` option. The `--plugin-load` option resets the plugin load list, and

this can cause unexpected problems if you are not aware. The `--plugin-load-add` option does **not** reset the plugin load list, so it is much safer to use.

This can have the following consequences:

- If the `--plugin-load` option is specified **multiple times**, then only the last instance will have any effect. For example, in the following case, the first instance of the option is reset:

```
[mariadb]
...
plugin_load = server_audit
plugin_load = ed25519=auth_ed25519
```

- If the `--plugin-load` option is specified **after** the `--plugin-load-add` option, then it will also reset the changes made by that option. For example, in the following case, the `--plugin-load-add` option does not do anything, because the subsequent `--plugin-load` option resets the plugin load list:

```
[mariadb]
...
plugin_load_add = server_audit
plugin_load = ed25519=auth_ed25519
```

- In contrast, if the `--plugin-load` option is specified **before** the `--plugin-load-add` option, then it will work fine, because the `--plugin-load-add` option does not reset the plugin load list. For example, in the following case, both plugins are properly loaded:

```
[mariadb]
...
plugin_load = server_audit
plugin_load_add = ed25519=auth_ed25519
```

Installing a Plugin with mariadb-plugin

A plugin can be installed with the mariadb-plugin utility if the server is completely offline.

The syntax is:

```
mariadb-plugin [options] <plugin> ENABLE|DISABLE
```

For example, to install the server_audit audit plugin, you could execute the following:

```
mariadb-plugin server_audit ENABLE
```

If a plugin is installed with this utility, then a record will be added to the mysql.plugins table for the plugin. This means that the plugin will automatically be loaded every time the server restarts, unless specifically uninstalled or deactivated.

Configuring the Plugin Directory

When a plugin is being installed, the server looks for the plugin's library in the server's plugin directory. This directory is configured by the `plugin_dir` system variable. This can be specified as a command-line argument to `mariadbd` or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
plugin_dir = /usr/lib64/mysql/plugin
```

Configuring the Minimum Plugin Maturity

When a plugin is being installed, the server compares the plugin's maturity level against the server's minimum allowed plugin maturity. This can help prevent users from using unstable plugins on production servers. This minimum plugin maturity is configured by the `plugin_maturity` system variable. This can be specified as a command-line argument to `mariadbd` or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
plugin_maturity = stable
```

Configuring Plugin Activation at Server Startup

A plugin will be loaded by default when the server starts if:

- The plugin was installed with the INSTALL SONAME statement.
- The plugin was installed with the INSTALL PLUGIN statement.
- The plugin was installed with the mariadb-plugin utility.
- The server is configured to load the plugin with the --plugin-load-add option.
- The server is configured to load the plugin with the --plugin-load option.

This behavior can be changed with special options that take the form `--plugin-name`. For example, for the `server_audit` audit plugin, the special option is called --server-audit.

The possible values for these special options are:

Option Value	Description
`OFF`	Disables the plugin without removing it from the mysql.plugins table.
`ON`	Enables the plugin. If the plugin cannot be initialized, then the server will still continue starting up, but the plugin will be disabled.
`FORCE`	Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error.
`FORCE_PLUS_PERMANENT`	Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error. In addition, the plugin cannot be uninstalled with UNINSTALL SONAME `or` UNINSTALL PLUGIN while the server is running.

A plugin's status can be found by looking at the `PLUGIN_STATUS` column of the information_schema.PLUGINS table.

Uninstalling Plugins

Plugins that are found in the mysql.plugin table, that is those that were installed with INSTALL SONAME, INSTALL PLUGIN or mariadb-plugin can be uninstalled in one of two ways:

- The UNINSTALL SONAME or the UNINSTALL PLUGIN statement while the server is running
- With mariadb-plugin while the server is offline.

Plugins that were enabled as a `--plugin-load` option do not need to be uninstalled. If `--plugin-load` is omitted the next time the server starts, or the plugin is not listed as one of the `--plugin-load` entries, the plugin will not be loaded.

UNINSTALL PLUGIN uninstalls a single installed plugin, while UNINSTALL SONAME uninstalls all plugins belonging to a given library.

5.4.2 Information on Plugins

List of Plugins

List of plugins included in MariaDB, ordered by their maturity.

Information Schema PLUGINS Table

Information Schema table containing information on plugins installed on a server.

Information Schema ALL_PLUGINS Table

Information about server plugins, whether installed or not.

5.4.2.1 List of Plugins

Contents

1. MariaDB Plugin Maturity

MariaDB Plugin Maturity

The following table lists the various plugins included in MariaDB ordered by their maturity. Note that maturity will differ across MariaDB versions - see below for an easy way to get a complete list of plugins and their maturity in your version of MariaDB:

Plugin	Version	Maturity	Version
Archive	3.0	Stable	
Aria	1.5	Stable	
Audit Plugin	1.4	Stable	
aws_key_management	1.0	Stable	
binlog	1.0	Stable	
Blackhole	1.0	Stable	
Connect	1.7	Stable	MariaDB 10.4.12, MariaDB 10.3.22
CLIENT_STATISTICS	2.0	Stable	
cracklib_password_check	1.0	Stable	
CSV	1.0	Stable	
DISKS	1.1	Stable	MariaDB 10.4.7, MariaDB 10.3.17
ed25519	1.1	Stable	MariaDB 10.4.0
FederatedX[1]	2.1	Stable	
Feedback	1.1	Stable	
file_key_management	1.0	Stable	
gssapi	1.0	Stable	
INDEX_STATISTICS	2.0	Stable	
INET6	1.0	Stable	MariaDB 10.5.12
InnoDB	10.*	Stable	
LOCALES	1.0	Stable	
Memory	1.0	Stable	
METADATA_LOCK_INFO	0.1	Stable	
MRG_MyISAM	1.0	Stable	
Mroonga	7.7	Stable	
MyISAM	1.0	Stable	
MyRocks	1.0	Stable	MariaDB 10.3.7
mysql_json	0.1	Stable	MariaDB 10.5.17
mysql_native_password	1.0	Stable	
mysql_old_password	1.0	Stable	
named_pipe	1.0	Stable	
pam	1.0	Stable	
password_reuse_check	2.0	Stable	MariaDB 10.8.7, MariaDB 10.9.5, MariaDB 10.10.2
partition	1.0	Stable	
Performance_Schema	0.1	Stable	
QUERY_CACHE_INFO	1.1	Stable	
query_response_time	1.0	Stable	
S3	1.0	Stable	MariaDB 10.5.12
semisync	1.0	Stable	Built-in, no longer a plugin from MariaDB 10.3.3
Sequence	1.0	Stable	
SERVER_AUDIT	1.4	Stable	

simple_password_check	1.0	Stable	
Spider	3.3	Stable	<= MariaDB 10.4, MariaDB 10.5.7
SQL_ERROR_LOG	1.0	Stable	
TABLE_STATISTICS	2.0	Stable	
USER_STATISTICS	2.0	Stable	
user_variables	1.0	Stable	MariaDB 10.3.13
TokuDB	4.0	Stable	Disabled in MariaDB 10.5 and removed in MariaDB 10.6
unix_socket	1.0	Stable	
UUID	1.0	Stable	MariaDB 10.9.1
wsrep	1.0	Stable	
WSREP_INFO	1.0	Stable	
Plugin	**Version**	**Maturity**	**From**
Federated [2]	1.0	Gamma	
OQGraph	3.0	Gamma	
Sphinx	2.0	Gamma	
Plugin	**Version**	**Maturity**	**From**
Columnstore	1.0	Beta	MariaDB 10.5.4
handlersocket	1.0	Beta	
Plugin	**Version**	**Maturity**	**From**
Cassandra	0.1	Experimental	Removed in MariaDB 10.6
debug_key_management	1.0	Experimental	
example_key_management	1.0	Experimental	
Plugin	**Version**	**Maturity**	**Version**

Execute the following on your MariaDB server to get a complete list of plugins and their maturity for your version of MariaDB:

```
SELECT plugin_name, plugin_version, plugin_maturity
FROM information_schema.plugins
ORDER BY plugin_name;
```

1.1.1.2.9.1.1.34 Information Schema PLUGINS Table

1.1.1.2.9.1.1.4 Information Schema ALL_PLUGINS Table

1.1.1.2.6 Plugin SQL Statements

5.4.4 Creating and Building Plugins

Specifying Which Plugins to Build
Specifying which plugins to build.

Writing Plugins for MariaDB
Writing plugins for MariaDB.

2.1.2.8.5.2 Specifying Which Plugins to Build

5.4.4.2 Writing Plugins for MariaDB

Contents

About

Generally speaking, writing plugins for MariaDB is very similar to writing plugins for MySQL.

Authentication Plugins

See Pluggable Authentication.

Storage Engine Plugins

Storage engines can extend `CREATE TABLE` syntax with optional index, field, and table attribute clauses. See Extending CREATE TABLE for more information.

See Storage Engine Development.

Information Schema Plugins

Information Schema plugins can have their own FLUSH and SHOW statements. See FLUSH and SHOW for Information Schema plugins.

Encryption Plugins

Encryption plugins in MariaDB are used for the data at rest encryption feature. They are responsible for both key management and for the actual encryption and decryption of data.

Plugin Declaration Structure

The MariaDB plugin declaration differs from the MySQL plugin declaration in the following ways:

1. it has no useless 'reserved' field (the very last field in the MySQL plugin declaration)
2. it has a 'maturity' declaration
3. it has a field for a text representation of the version field

MariaDB can load plugins that only have the MySQL plugin declaration but both `PLUGIN_MATURITY` and `PLUGIN_AUTH_VERSION` will show up as 'Unknown' in the INFORMATION_SCHEMA.PLUGINS table.

For compiled-in (not dynamically loaded) plugins, the presence of the MariaDB plugin declaration is mandatory.

Example Plugin Declaration

The MariaDB plugin declaration looks like this:

```
/* MariaDB plugin declaration */
maria_declare_plugin(example)
{
   MYSQL_STORAGE_ENGINE_PLUGIN, /* the plugin type (see include/mysql/plugin.h) */
   &example_storage_engine_info, /* pointer to type-specific plugin descriptor   */
   "EXAMPLEDB", /* plugin name */
   "John Smith",  /* plugin author */
   "Example of plugin interface", /* the plugin description */
   PLUGIN_LICENSE_GPL, /* the plugin license (see include/mysql/plugin.h) */
   example_init_func,   /* Pointer to plugin initialization function */
   example_deinit_func,  /* Pointer to plugin deinitialization function */
   0x0001 /* Numeric version 0xAABB means AA.BB version */,
   example_status_variables,  /* Status variables */
   example_system_variables,  /* System variables */
   "0.1 example",  /* String version representation */
   MariaDB_PLUGIN_MATURITY_EXPERIMENTAL /* Maturity (see include/mysql/plugin.h)*/
}
maria_declare_plugin_end;
```

5.4.5 MariaDB Audit Plugin

Contents

MariaDB and MySQL are used in a broad range of environments, but if you needed to record user access to be in compliance with auditing regulations for your organization, you would previously have had to use other database solutions. To meet this need, though, MariaDB has developed the MariaDB Audit Plugin. Although the MariaDB Audit Plugin has some unique features available only for MariaDB, it can be used also with MySQL.

Basically, the purpose of the MariaDB Audit Plugin is to log the server's activity. For each client session, it records who connected to the server (i.e., user name and host), what queries were executed, and which tables were accessed and server variables that were changed. This information is stored in a rotating log file or it may be sent to the local `syslogd`.

The MariaDB Audit Plugin works with MariaDB, MySQL (as of, version 5.5.34 and 10.0.7) and Percona Server. MariaDB started including by default the Audit Plugin from versions 10.0.10 and 5.5.37, and it can be installed in any version from MariaDB 5.5.20.

Additional documentation

Below are links to additional documentation on the MariaDB Audit Plugin. They explain in detail how to install, configure and use the Audit Plugin.

- Installation
- Configuration
- Log Settings
- Log Location & Rotation
- Log Format
- Status Variables
- System Variables
- Release Notes

Tutorials

Below are links to some tutorials on MariaDB's site and other sites. They may help you to get more out of the MariaDB Audit Plugin.

- Introducing the MariaDB Audit Plugin
  by Anatoliy Dimitrov, September 2, 2014
- Activating MariaDB Audit Log by Jaykishan Mutkawoa, May 30, 2016
- Installing MariaDB Audit Plugin on Amazon RDS
  Amazon RDS supports using the MariaDB Audit Plugin on MySQL and MariaDB database instances.

Web Log Articles

Below are links to web log articles on the MariaDB Audit Plugin. You may find them useful in understanding better how to

use the Audit Plugin. Since some of these articles are older, they won't include changes and improvements in newer versions. You can rely on the documentation pages listed above for the most current information.

- Activating Auditing for MariaDB in 5 Minutes
  by Ralf Gebhardt, September 29, 2013
- Query and Password Filtering with the MariaDB Audit Plugin
  by Ralf Gebhardt, May 4, 2015
- Set Up a Remote Log File using rsyslog
  by Ralf Gebhardt, December 16, 2013
- MySQL Auditing with MariaDB Auditing Plugin by Peter Zeitsev, February 15, 2016

Sub-Documents

MariaDB Audit Plugin - Installation
Installing the MariaDB Audit Plugin.

MariaDB Audit Plugin - Configuration
Audit Plugin global variables within MariaDB

MariaDB Audit Plugin - Log Settings
Log audit events to a file or syslog.

MariaDB Audit Plugin - Location and Rotation of Logs
Logs can be written to a separate file or to the system logs

MariaDB Audit Plugin - Log Format
The audit log is a set of records written as a list of fields to a file in plain-text format.

MariaDB Audit Plugin - Versions
Releases of the MariaDB Audit Plugin, and in which versions of MariaDB each...

MariaDB Audit Plugin Options and System Variables
Description of Server_Audit plugin options and system variables.

MariaDB Audit Plugin - Status Variables
Server Audit plugin status variables

Release Notes - MariaDB Audit Plugin
MariaDB Audit Plugin release notes

There are 7 related questions.

5.4.5.1 MariaDB Audit Plugin - Installation

The `server_audit` plugin logs the server's activity. For each client session, it records who connected to the server (i.e., user name and host), what queries were executed, and which tables were accessed and server variables that were changed. This information is stored in a rotating log file or it may be sent to the local syslogd.

Contents

Locating the Plugin

The `server_audit` plugin's shared library is included in MariaDB packages as the `server_audit.so` or `server_audit.dll` shared library on systems where it can be built.

The plugin must be located in the plugin directory, the directory containing all plugin libraries for MariaDB. The path to this directory is configured by the `plugin_dir` system variable. To see the value of this variable and thereby determine the file path of the plugin library, execute the following SQL statement:

```
SHOW GLOBAL VARIABLES LIKE 'plugin_dir';

+---------------+--------------------------+
| Variable_name | Value                    |
+---------------+--------------------------+
| plugin_dir    | /usr/lib64/mysql/plugin/ |
+---------------+--------------------------+
```

Check the directory returned at the filesystem level to make sure that you have a copy of the plugin library, `server_audit.so` or `server_audit.dll`, depending on your system. It's included in recent installations of MariaDB. If you do not have it, you should upgrade MariaDB.

Installing the Plugin

Although the plugin's shared library is distributed with MariaDB by default, the plugin is not actually installed by MariaDB by default. There are two methods that can be used to install the plugin with MariaDB.

The first method can be used to install the plugin without restarting the server. You can install the plugin dynamically by executing `INSTALL SONAME` or `INSTALL PLUGIN`. For example:

```
INSTALL SONAME 'server_audit';
```

The second method can be used to tell the server to load the plugin when it starts up. The plugin can be installed this way by providing the `--plugin-load` or the `--plugin-load-add` options. This can be specified as a command-line argument to `mysqld` or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
plugin_load_add = server_audit
```

Uninstalling the Plugin

You can uninstall the plugin dynamically by executing `UNINSTALL SONAME` or `UNINSTALL PLUGIN`. For example:

```
UNINSTALL SONAME 'server_audit';
```

If you installed the plugin by providing the `--plugin-load` or the `--plugin-load-add` options in a relevant server option group in an option file, then those options should be removed to prevent the plugin from being loaded the next time the server is restarted.

Prohibiting Uninstallation

The `UNINSTALL SONAME` or `UNINSTALL PLUGIN` statements may be used to uninstall plugins. For the `server_audit` plugin, you might want to disable this capability. To prevent the plugin from being uninstalled, you could set the the `server_audit` option to `FORCE_PLUS_PERMANENT` in a relevant server option group in an option file after the plugin is loaded once:

```
[mariadb]
...
plugin_load_add = server_audit
server_audit=FORCE_PLUS_PERMANENT
```

Once you've added the option to the server's option file and restarted the server, the plugin can't be uninstalled. If someone tries to uninstall the audit plugin, then an error message will be returned. Below is an example of the error message:

```
UNINSTALL PLUGIN server_audit;

ERROR 1702 (HY000):
Plugin 'server_audit' is force_plus_permanent and can not be unloaded
```

For more information on `FORCE_PLUS_PERMANENT` and other option values for the `server_audit` option, see Plugin Overview: Configuring Plugin Activation at Server Startup for more information.

5.4.5.2 MariaDB Audit Plugin - Configuration

After the audit plugin has been installed and loaded, there will be some new global variables within MariaDB. These can be used to configure many components, limits, and methods related to auditing the server. You may set these variables related to the logs, such as their location, size limits, rotation parameters, and method of logging information. You may also set what information is logged, such connects, disconnects, and failed attempts to connect. You can also have the audit plugin log queries, read and write access to tables. So as not to overload your logs, the audit plugin can be configured based on lists of users. You can include or exclude the activities of specific users in the logs.

To see a list of audit plugin-related variables on the server and their values, execute the follow while connected to the server:

```
SHOW GLOBAL VARIABLES LIKE 'server_audit%';

+-------------------------------+-----------------------+
| Variable_name                 | Value                 |
+-------------------------------+-----------------------+
| server_audit_events           | CONNECT,QUERY,TABLE   |
| server_audit_excl_users       |                       |
| server_audit_file_path        | server_audit.log      |
| server_audit_file_rotate_now  | OFF                   |
| server_audit_file_rotate_size | 1000000               |
| server_audit_file_rotations   | 9                     |
| server_audit_incl_users       |                       |
| server_audit_logging          | ON                    |
| server_audit_mode             | 0                     |
| server_audit_output_type      | file                  |
| server_audit_query_log_limit  | 1024                  |
| server_audit_syslog_facility  | LOG_USER              |
| server_audit_syslog_ident     | mysql-server_auditing |
| server_audit_syslog_info      |                       |
| server_audit_syslog_priority  | LOG_INFO              |
+-------------------------------+-----------------------+
```

The values of these variables can be changed by an administrator with the `SUPER` privilege, using the `SET` statement. Below is an example of how to disable audit logging:

```
SET GLOBAL server_audit_logging=OFF;
```

Although it is possible to change all of the variables shown above, some of them may be reset when the server restarts. Therefore, you may want set them in the configuration file (e.g., `/etc/my.cnf.d/server.cnf`) to ensure the values are the same after a restart:

```
[server]
...
server_audit_logging=OFF
…
```

For the reason given in the paragraph above, you would not generally set variables related to the auditing plugin using the `SET` statement. However, you might do so to test settings before making them more permanent. Since one cannot always restart the server, you would use the `SET` statement to change immediately the variables and then include the same settings in the configuration file so that the variables are set again as you prefer when the server is restarted.

Configuring Logs and Setting Other Variables

Of all of the server variables you can set, you may want to set initially the server_audit_events variable to tell the Audit Plugin which events to log. The Log Settings documentation page describes in detail the choices you have and provides examples of log entries related to them.

You can see a detailed list of system variables related to the MariaDB Audit Plugin on the System Variables documentation page. Status variables related to the Audit Plugin are listed and explained on the Status Variables documentation page.

5.4.5.3 MariaDB Audit Plugin - Log Settings

Events that are logged by the MariaDB Audit Plugin are grouped generally into different types: connect, query, and table events. To log based on these types of events, set the variable, server_audit_events to `CONNECT`, `QUERY`, or `TABLE`. To

have the Audit Plugin log more than one type of event, put them in a comma-separated list like so:

```
SET GLOBAL server_audit_events = 'CONNECT,QUERY,TABLE';
```

Contents

You can put the equivalent of this in the configuration file like so:

```
[mysqld]
...
server_audit_events=connect,query
```

By default, logging is set to `OFF`. To enable it, set the server_audit_logging variable to `ON`. Note that if the query cache is enabled, and a query is returned from the query cache, no `TABLE` records will appear in the log since the server didn't open or access any tables and instead relied on the cached results. So you may want to disable query caching.

There are actually a few types of events that may be logged, not just the three common ones mentioned above. A full list of related system variables is detailed on the Server_Audit System Variables page, and status variables on the Server_Audit Status Variables page of this documentation. Some of the major ones are highlighted below:

Type	Description
CONNECT	Connects, disconnects and failed connects—including the error code
QUERY	Queries executed and their results in plain text, including failed queries due to syntax or permission errors
TABLE	Tables affected by query execution
QUERY_DDL	Similar to `QUERY`, but filters only DDL-type queries (`CREATE`, `ALTER`, `DROP`, `RENAME` and `TRUNCATE`). There are some exceptions however. `RENAME USER` is not logged, while `CREATE/DROP [PROCEDURE / FUNCTION / USER]` are only logged from MariaDB 10.2.38, MariaDB 10.3.29, MariaDB 10.4.22, MariaDB 10.5.13 and MariaDB 10.6.5. In earlier versions they are not logged. See MDEV-23457.
QUERY_DML	Similar to `QUERY`, but filters only DML-type queries (`DO`, `CALL`, `LOAD DATA/XML`, `DELETE`, `INSERT`, `SELECT`, `UPDATE`, `HANDLER` and `REPLACE` statements)
QUERY_DML_NO_SELECT	Similar to `QUERY_DML`, but doesn't log SELECT queries. (since version 1.4.4) (`DO`, `CALL`, `LOAD DATA/XML`, `DELETE`, `INSERT`, `UPDATE`, `HANDLER` and `REPLACE` statements)
QUERY_DCL	Similar to `QUERY`, but filters only DCL-type queries (`CREATE USER`, `DROP USER`, `RENAME USER`, `GRANT`, `REVOKE` and `SET PASSWORD` statements)

Since there are other types of queries besides DDL and DML, using the `QUERY_DDL` and `QUERY_DML` options together is not equivalent to using `QUERY`. Starting in version 1.3.0 of the Audit Plugin, there is the `QUERY_DCL` option for logging DCL types of queries (e.g., `GRANT` and `REVOKE` statements). In the same version, the server_audit_query_log_limit variable was added to be able to set the length of a log record. Previously, a log entry would be truncated due to long query strings.

Logging Connect Events

If the Audit Plugin has been configured to log connect events, it will log connects, disconnects, and failed connects. For a failed connection, the log includes the error code.

It's possible to define a list of users for which events can be excluded or included for tracing their database activities. This list will be ignored, though, for the loggings of connect events. This is because auditing standards distinguish between technical and physical users. Connects need to be logged for all types of users; access to objects need to be logged only for physical users.

Logging Query Events

If `QUERY`, `QUERY_DDL`, `QUERY_DML`, `QUERY_DML_NO_SELECT`, and/or `QUERY_DCL` event types are enabled, then the corresponding types of queries that are executed will be logged for defined users. The queries will be logged exactly as they are executed, in plain text. This is a security vulnerability: anyone who has access to the log files will be able to read the queries. So make sure that only trusted users have access to the log files and that the files are in a protected location. An alternative is to use `TABLE` event type instead of the query-related event types.

Queries are also logged if they cannot be executed, if they're unsuccessful. For example, a query will be logged because of a syntax error or because the user doesn't have the privileges necessary to access an object. These queries can be parsed by the error code that's provided in the log.

You may find failed queries to be more interesting: They can reveal problems with applications (e.g., an SQL statement in an application that doesn't match the current schema). They can also reveal if a malicious user is guessing at the names of tables and columns to try to get access to data.

Below is an example in which a user attempts to execute an `UPDATE` statement on a table for which he does not have permission:

```
UPDATE employees
SET salary = salary * 1.2
WHERE emp_id = 18236;

ERROR 1142 (42000):
UPDATE command denied to user 'bob'@'localhost' for table 'employees'
```

Looking in the Audit Plugin log (`server_audit.log`) for this entry, you can see the following entry:

```
20170817 11:07:18,ip-172-30-0-38,bob,localhost,15,46,QUERY,company,
'UPDATE employees SET salary = salary * 1.2 WHERE emp_id = 18236',1142
```

This log entry would be on one line, but it's reformatted here for better rendering. Looking at this log entry, you can see the date and time of the query, followed by the server host, the user and host for the account. Next is the connection and query identification numbers (i.e., `15` and `46`). After the log event type (i.e., `QUERY`), the database name (i.e., `company`), the query, and the error number is recorded.

Notice that the last value in the log entry is `1142` . That's the error number for the query. To find failed queries, you would look for two elements: the notation indicating that it's a `QUERY` entry, and the last value for the entry. If the query is successful, the value will be `0` .

Queries Not Included in Subordinate Query Event Types

Note that the `QUERY` event type will log queries that are not included in any of the subordinate `QUERY_*` event types, such as:

- CREATE FUNCTION
- DROP FUNCTION
- CREATE PROCEDURE
- DROP PROCEDURE
- SET
- CHANGE MASTER TO
- FLUSH
- KILL
- CHECK
- OPTIMIZE
- LOCK
- UNLOCK
- ANALYZE
- INSTALL PLUGIN
- UNINSTALL PLUGIN
- INSTALL SONAME
- UNINSTALL SONAME
- EXPLAIN

Logging Table Events

MariaDB has the ability to record table events in the logs—this is not a feature of MySQL. This feature is the only way to log which tables have been accessed through a view, a stored procedure, a stored function, or a trigger. Without this feature, a log entry for a query shows only the view, stored procedure or function used, not the underlying tables. Of course, you could

create a custom application to parse each query executed to find the SQL statements used and the tables accessed, but that would be a drain on system resources. Table event logging is much simpler: it adds a line to the log for each table accessed, without any parsing. It includes notes as to whether it was a read or a write.

If you want to monitor user access to specific databases or tables (e.g., `mysql.user`), you can search the log for them. Then if you want to see a query which accessed a certain table, the audit log entry will include the query identificaiton number. You can use it to search the same log for the query entry. This can be useful when searching a log containing tens of thousands of entries.

Because of the `TABLE` option, you may disable query logging and still know who accessed which tables. You might want to disable `QUERY` event logging to prevent sensitive data from being logged. Since *table* event logging will log who accessed which table, you can still watch for malicious activities with the log. This is often enough to fulfill auditing requirements.

Below is an example with both `TABLE` and `QUERY` events logging. For this scenario, suppose there is a VIEW in which columns are selected from a few tables in a `company` database. The underlying tables are related to sensitive employee information, in particular salaries. Although we may have taken precautions to ensure that only certain user accounts have access to those tables, we will monitor the Audit Plugin logs for anyone who queries them—directly or indirectly through a view.

```
20170817 16:04:33,ip-172-30-0-38,root,localhost,29,913,READ,company,employees,
20170817 16:04:33,ip-172-30-0-38,root,localhost,29,913,READ,company,employees_salaries,
20170817 16:04:33,ip-172-30-0-38,root,localhost,29,913,READ,company,ref_job_titles,
20170817 16:04:33,ip-172-30-0-38,root,localhost,29,913,READ,company,org_departments,
20170817 16:04:33,ip-172-30-0-38,root,localhost,29,913,QUERY,company,
'SELECT * FROM employee_pay WHERE title LIKE \'%Executive%\'  OR title LIKE \'%Manager%\'',0
```

Although the user executed only one SELECT statement, there are multiple entries to the log: one for each table accessed and one entry for the query on the view, (i.e., `employee_pay`). We know primarily this is all for one query because they all have the same connection and query identification numbers (i.e., `29` and `913`).

Logging User Activities

The Audit Plugin will log the database activities of all users, or only the users that you specify. A database activity is defined as a *query* event or a *table* event. *Connect* events are logged for all users.

You may specify users to include in the log with the `server_audit_incl_users` variable or exclude users with the `server_audit_excl_users` variable. This can be useful if you would like to log entries, but are not interested in entries from trusted applications and would like to exclude them from the logs.

You would typically use either the `server_audit_incl_users` variable or the `server_audit_excl_users` variable. You may, though, use both variables. If a username is inadvertently listed in both variables, database activities for that user will be logged because `server_audit_incl_users` takes priority.

Although MariaDB considers a user as the combination of the username and hostname, the Audit Plugin logs only based on the username. MariaDB uses both the username and hostname so as to grant privileges relevant to the location of the user. Privileges are not relevant though for tracing the access to database objects. The host name is still recorded in the log, but logging is not determined based on that information.

The following example shows how to add a new username to the `server_audit_incl_users` variable without removing previous usernames:

```
SET GLOBAL server_audit_incl_users = CONCAT(@@global.server_audit_incl_users, ',Maria');
```

Remember to add also any new users to be included in the logs to the same variable in MariaDB configuration file. Otherwise, when the server restarts it will discard the setting.

Excluding or Including Users

By default events from all users are logged, but certain users can be excluded from logging by using the server_audit_excl_users variable. For example, to exclude users *valerianus* and *rocky* from having their events logged:

```
server_audit_excl_users=valerianus,rocky
```

This option is primarily used to exclude the activities of trusted applications.

Alternatively, server_audit_incl_users can be used to specifically include users. Both variables can be used, but if a user appears on both lists, server_audit_incl_users has a higher priority, and their activities will be logged.

Note that `CONNECT` events are always logged for all users, regardless of these two settings. Logging is also based on username only, not the username and hostname combination that MariaDB uses to determine privileges.

5.4.5.4 MariaDB Audit Plugin - Location and Rotation of Logs

Contents

Logs can be written to a separate file or to the system logs. If you prefer to have the logging separated from other system information, the value of the variable, server_audit_output_type should be set to `file`. Incidentally, `file` is the only option on Windows systems.

You can force a rotation by enabling the server_audit_file_rotate_now variable like so:

```
SET GLOBAL server_audit_file_rotate_now = ON;
```

Separate log files

In addition to setting server_audit_output_type, you will have to provide the file path and name of the audit file. This is set in the variable, server_audit_file_path. You can set the file size limit of the log file with the variable, server_audit_file_rotate_size.

So, if rotation is on and the log file has reached the size limit you set, a copy is created with a consecutive number as extension, the original file will be truncated to be used for the auditing again. To limit the number of log files created, set the variable, server_audit_file_rotations. You can force log file rotation by setting the variable, server_audit_file_rotate_now to a value of `ON`. When the number of files permitted is reached, the oldest file will be overwritten. Below is an example of how the variables described above might be set in a server's configuration files:

```
[mysqld]
...
server_audit_file_rotate_now=ON
server_audit_file_rotate_size=1000000
server_audit_file_rotations=5
...
```

System logs

For security reasons, it's better sometimes to use the system logs instead of a local file owned by the `mysql` user. To do this, the value of server_audit_output_type needs to be set to `syslog`. Advanced configurations, such as using a remote `syslogd` service, are part of the `syslogd` configuration.

The variables, server_audit_syslog_ident and server_audit_syslog_info can be used to identify a system log entry made by the audit plugin. If a remote `syslogd` service is used for several MariaDB servers, these same variables are also used to identify the MariaDB server.

Below is an example of a system log entry taken from a server which had server_audit_syslog_ident set to the default value of `mysql -server_auditing`, and server_audit_syslog_info set to `<prod1>`.

```
Aug 717:19:58localhostmysql- server_auditing:
<prod1> localhost.localdomain,root,localhost,1,7,
QUERY, mysql, 'SELECT * FROM user',0
```

Although the default values for server_audit_syslog_facility and server_audit_syslog_priority should be sufficient in most cases, they can be changed based on the definition in `syslog.h` for the functions `openlog()` and `syslog()`.

5.4.5.5 MariaDB Audit Plugin - Log Format

The audit plugin logs user access to MariaDB and its objects. The audit trail (i.e., audit log) is a set of records, written as a list of fields to a file in a plain-text format. The fields in the log are separated by commas. The format used for the plugin's own log file is slightly different from the format used if it logs to the system log because it has its own standard format. The general format for the logging to the plugin's own file is defined like the following:

```
[timestamp],[serverhost],[username],[host],[connectionid],
[queryid],[operation],[database],[object],[retcode]
```

If the server_audit_output_type variable is set to `syslog` instead of the default, `file`, the audit log file format will be as follows:

```
[timestamp][syslog_host][syslog_ident]:[syslog_info][serverhost],[username],[host],
[connectionid],[queryid],[operation],[database],[object],[retcode]
```

Item logged	Description
timestamp	Time at which the event occurred. If syslog is used, the format is defined by `syslogd`.
syslog_host	Host from which the syslog entry was received.
syslog_ident	For identifying a system log entry, including the MariaDB server.
syslog_info	For providing information for identifying a system log entry.
serverhost	The MariaDB server host name.
username	Connected user.
host	Host from which the user connected.
connectionid	Connection ID number for the related operation.
queryid	Query ID number, which can be used for finding the relational table events and related queries. For TABLE events, multiple lines will be added.
operation	Recorded action type: CONNECT, QUERY, READ, WRITE, CREATE, ALTER, RENAME, DROP.
database	Active database (as set by USE).
object	Executed query for QUERY events, or the table name in the case of TABLE events.
retcode	Return code of the logged operation.

Various events will result in different audit records. Some events will not return a value for some fields (e.g., when the active database is not set when connecting to the server).

Below is a generic example of the output for connect events, with placeholders representing data. These are events in which a user connected, disconnected, or tried unsuccessfully to connect to the server.

```
[timestamp],[serverhost],[username],[host],[connectionid],0,CONNECT,[database],,0
[timestamp],[serverhost],[username],[host],[connectionid],0,DISCONNECT,,,0
[timestamp],[serverhost],[username],[host],[connectionid],0,FAILED_CONNECT,,,[retcode]
```

Here is the one audit record generated for each query event:

```
[timestamp],[serverhost],[username],[host],[connectionid],[queryid],QUERY,[database],[object],
[retcode]
```

Below are generic examples of records that are entered in the audit log for each type of table event:

```
[timestamp],[serverhost],[username],[host],[connectionid],[queryid],CREATE,[database],[object],
[timestamp],[serverhost],[username],[host],[connectionid],[queryid],READ,[database],[object],
[timestamp],[serverhost],[username],[host],[connectionid],[queryid],WRITE,[database],[object],
[timestamp],[serverhost],[username],[host],[connectionid],[queryid],ALTER,[database],[object],
[timestamp],[serverhost],[username],[host],[connectionid],[queryid],RENAME,[database],
[object_old]|[database_new].[object_new],
[timestamp],[serverhost],[username],[host],[connectionid],[queryid],DROP,[database],[object],
```

Starting in version 1.2.0, passwords are hidden in the log for certain types of queries. They are replaced with asterisks for `GRANT`, `CREATE USER`, `CREATE MASTER`, `CREATE SERVER`, and `ALTER SERVER` statements. Passwords, however, are not replaced for the `PASSWORD()` and `OLD_PASSWORD()` functions when they are used inside other SQL statements (i.e., SET PASSWORD`)`.

5.4.5.6 MariaDB Audit Plugin - Versions

Below is a list of the releases of the MariaDB Audit Plugin, the most recent version first, and in which versions of MariaDB each plugin version was included.

Version	Introduced
1.4.13	MariaDB 10.2.38, MariaDB 10.3.29, MariaDB 10.4.19, MariaDB 10.5.10
1.4.10	MariaDB 10.2.35, MariaDB 10.3.26, MariaDB 10.5.7
1.4.7	MariaDB 10.1.41, MariaDB 10.2.26, MariaDB 10.3.17, MariaDB 10.4.7
1.4.5	MariaDB 10.2.24, MariaDB 10.3.15, MariaDB 10.4.5
1.4.4	MariaDB 5.5.61, MariaDB 10.0.36, MariaDB 10.1.34, MariaDB 10.2.15, MariaDB 10.3.7, MariaDB 10.4.0
1.4.0	MariaDB 5.5.48, MariaDB 10.0.24, MariaDB 10.1.11
1.3.0	MariaDB 5.5.43, MariaDB 10.0.18, MariaDB 10.1.5
1.2.0	MariaDB 5.5.42, MariaDB 10.0.17, MariaDB 10.1.4
1.1.7	MariaDB 5.5.38, MariaDB 10.0.11, MariaDB 10.1.0
1.1.6	MariaDB 5.5.37, MariaDB 10.0.10
1.1.5	MariaDB 10.0.09
1.1.4	MariaDB 5.5.36
1.1.3	MariaDB 5.5.34, MariaDB 10.0.7

5.4.5.7 MariaDB Audit Plugin Options and System Variables

Contents

There are a several options and system variables related to the MariaDB Audit Plugin, once it has been installed. System variables can be displayed using the SHOW VARIABLES statement like so:

```
SHOW GLOBAL VARIABLES LIKE '%server_audit%';

+-------------------------------+----------------------+
| Variable_name                 | Value                |
+-------------------------------+----------------------+
| server_audit_events           | CONNECT,QUERY,TABLE  |
| server_audit_excl_users       |                      |
| server_audit_file_path        | server_audit.log     |
| server_audit_file_rotate_now  | OFF                  |
| server_audit_file_rotate_size | 1000000              |
| server_audit_file_rotations   | 9                    |
| server_audit_incl_users       |                      |
| server_audit_logging          | ON                   |
| server_audit_mode             | 0                    |
| server_audit_output_type      | file                 |
| server_audit_query_log_limit  | 1024                 |
| server_audit_syslog_facility  | LOG_USER             |
| server_audit_syslog_ident     | mysql-server_auditing |
| server_audit_syslog_info      |                      |
| server_audit_syslog_priority  | LOG_INFO             |
+-------------------------------+----------------------+
```

To change the value of one of these variables, you can use the `SET` statement, or set them at the command-line when starting MariaDB. It's recommended that you set them in the MariaDB configuration for the server like so:

```
[mariadb]
...
server_audit_excl_users='bob,ted'
...
```

System Variables

Below is a list of all system variables related to the Audit Plugin. See Server System Variables for a complete list of system variables and instructions on setting them. See also the full list of MariaDB options, system and status variables.

`server_audit_events`

- **Description:** If set, then this restricts audit logging to certain event types. If not set, then every event type is logged to the audit log. For example: *SET GLOBAL server_audit_events='connect, query'*
- **Commandline:** `--server-audit-events=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** Empty string
- **Valid Values:**
  - `CONNECT`, `QUERY`, `TABLE` (MariaDB Audit Plugin < 1.2.0)
  - `CONNECT`, `QUERY`, `TABLE`, `QUERY_DDL`, `QUERY_DML` (MariaDB Audit Plugin >= 1.2.0)
  - `CONNECT`, `QUERY`, `TABLE`, `QUERY_DDL`, `QUERY_DML`, `QUERY_DCL` (MariaDB Audit Plugin >=1.3.0)
  - `CONNECT`, `QUERY`, `TABLE`, `QUERY_DDL`, `QUERY_DML`, `QUERY_DCL`, `QUERY_DML_NO_SELECT` (MariaDB Audit Plugin >= 1.4.4)
  - See MariaDB Audit Plugin - Versions to determine which MariaDB releases contain each MariaDB Audit Plugin versions.

`server_audit_excl_users`

- **Description:** If not empty, it contains the list of users whose activity will NOT be logged. For example: `SET GLOBAL server_audit_excl_users='user_foo, user_bar'`. CONNECT records aren't affected by this variable - they are always logged. The user is still logged if it's specified in server_audit_incl_users.
- **Commandline:** `--server-audit-excl-users=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** Empty string
- **Size limit:** 1024 characters

---

server_audit_file_path

- **Description:** When server_audit_output_type=file, sets the path and the filename to the log file. If the specified path exists as a directory, then the log will be created inside that directory with the name 'server_audit.log'. Otherwise the value is treated as a filename. The default value is 'server_audit.log', which means this file will be created in the database directory.
- **Commandline:** `--server-audit-file-path=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `server_audit.log`

---

server_audit_file_rotate_now

- **Description:** When server_audit_output_type=file, the user can force the log file rotation by setting this variable to ON or 1.
- **Commandline:** `--server-audit-rotate-now[={0|1}]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

server_audit_file_rotate_size

- **Description:** When server_audit_output_type=file, it limits the size of the log file to the given amount of bytes. Reaching that limit turns on the rotation - the current log file is renamed as 'file_path.1'. The empty log file is created as 'file_path' to log into it. The default value is 1000000.
- **Commandline:** `--server-audit-rotate-size=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1000000`
- **Range:** `100` to `9223372036854775807`

---

server_audit_file_rotations

- **Description:** When server_audit_output_type=file', this specifies the number of rotations to save. If set to 0 then the log never rotates. The default value is 9.
- **Commandline:** `--server-audit-rotations=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `9`
- **Range:** `0` to `999`

---

server_audit_incl_users

- **Description:** If not empty, it contains a comma-delimited list of users whose activity will be logged. For example: `SET GLOBAL server_audit_incl_users='user_foo, user_bar'` . CONNECT records aren't affected by this variable - they are always logged. This setting has higher priority than server_audit_excl_users. So if the same user is specified both in incl_ and excl_ lists, they will still be logged.
- **Commandline:** `--server-audit-incl-users=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** Empty string
- **Size limit:** 1024 characters

---

server_audit_loc_info

- **Description:** Used by plugin internals. It has no useful meaning to users.
  - In earlier versions, users see it as a read-only variable.
  - In later versions, it is hidden from the user.
- **Commandline:** N/A
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** Empty string
- **Introduced:** MariaDB 10.1.12, MariaDB 10.0.24, MariaDB 5.5.48
- **Hidden:** MariaDB 10.1.18, MariaDB 10.0.28, MariaDB 5.5.53

---

server_audit_logging

- **Description:** Enables/disables the logging. Expected values are ON/OFF. For example: `SET GLOBAL server_audit_logging=on` If the server_audit_output_type is FILE, this will actually create/open the logfile so the server_audit_file_path should be properly specified beforehand. Same about the SYSLOG-related parameters. The logging is turned off by default.
- **Commandline:** `--server-audit-logging[={0|1}]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

server_audit_mode

- **Description:** This variable doesn't have any distinctive meaning for a user. Its value mostly reflects the server version with which the plugin was started and is intended to be used by developers for testing.
- **Commandline:** `--server-audit-mode[=#]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `1`

---

server_audit_output_type

- **Description:** Specifies the desired output type. Can be SYSLOG, FILE or null as no output. For example: `SET GLOBAL server_audit_output_type=file` file: log records will be saved into the rotating log file. The name of the file set by server_audit_file_path variable. syslog: log records will be sent to the local syslogd daemon with the standard <syslog.h> API. The default value is 'file'.
- **Commandline:** `--server-audit-output-type=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `enum`
- **Default Value:** `file`
- **Valid Values:** `SYSLOG`, `FILE`

---

server_audit_query_log_limit

- **Description:** Limit on the length of the query string in a record.
- **Commandline:** `--server-audit-query-log-limit=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1024`
- **Range:** `0` to `2147483647`

---

server_audit_syslog_facility

- **Description:** SYSLOG-mode variable. It defines the 'facility' of the records that will be sent to the syslog. Later the

log can be filtered by this parameter.
- **Commandline:** `--server-audit-syslog-facility=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `enum`
- **Default Value:** `LOG_USER`
- **Valid Values:** `LOG_USER`, `LOG_MAIL`, `LOG_DAEMON`, `LOG_AUTH`, `LOG_SYSLOG`, `LOG_LPR`, `LOG_NEWS`, `LOG_UUCP`, `LOG_CRON`, `LOG_AUTHPRIV`, `LOG_FTP`, and `LOG_LOCAL0` – `LOG_LOCAL7`.

`server_audit_syslog_ident`

- **Description:** SYSLOG-mode variable. String value for the 'ident' part of each syslog record. Default value is 'mysql-server_auditing'. New value becomes effective only after restarting the logging.
- **Commandline:** `--server-audit-syslog-ident=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** `mysql-server_auditing`

`server_audit_syslog_info`

- **Description:** SYSLOG-mode variable. The 'info' string to be added to the syslog records. Can be changed any time.
- **Commandline:** `--server-audit-syslog-info=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `string`
- **Default Value:** Empty string

`server_audit_syslog_priority`

- **Description:** SYSLOG-mode variable. Defines the priority of the log records for the syslogd.
- **Commandline:** `--server-audit-syslog-priority=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `enum`
- **Default Value:** `LOG_INFO`
- **Valid Values:** `LOG_EMERG`, `LOG_ALERT`, `LOG_CRIT`, `LOG_ERR`, `LOG_WARNING`, `LOG_NOTICE`, `LOG_INFO`, `LOG_DEBUG`

Options

`server_audit`

- **Description:** Controls how the server should treat the plugin when the server starts up.
  - Valid values are:
    - `OFF` - Disables the plugin without removing it from the `mysql.plugins` table.
    - `ON` - Enables the plugin. If the plugin cannot be initialized, then the server will still continue starting up, but the plugin will be disabled.
    - `FORCE` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error.
    - `FORCE_PLUS_PERMANENT` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error. In addition, the plugin cannot be uninstalled with `UNINSTALL SONAME` or `UNINSTALL PLUGIN` while the server is running.
  - See MariaDB Audit Plugin - Installation: Prohibiting Uninstallation for more information on one use case.
  - See Plugin Overview: Configuring Plugin Activation at Server Startup for more information.
- **Commandline:** `--server-audit=val`
- **Data Type:** `enumerated`
- **Default Value:** `ON`
- **Valid Values:** `OFF`, `ON`, `FORCE`, `FORCE_PLUS_PERMANENT`

5.4.5.8 MariaDB Audit Plugin - Status Variables

Contents

There are a few status variables related to the MariaDB Audit Plugin, once it has been installed. These variables can be displayed using the SHOW STATUS statement like so:

```
SHOW STATUS LIKE 'server_audit%';

+----------------------------+------------------+
| Variable_name              | Value            |
+----------------------------+------------------+
| Server_audit_active        | ON               |
| Server_audit_current_log   | server_audit.log |
| Server_audit_last_error    |                  |
| Server_audit_writes_failed | 0                |
+----------------------------+------------------+
```

Status Variables

Below is a list of all status variables related to the Audit Plugin. These cannot be set: These are not to be confused with system variables, which can be set. See Server Status Variables for a complete list of status variables that can be viewed with the SHOW STATUS statement. See also the Full list of MariaDB options, system and status variables.

`Server_audit_active`

- **Description:** If the auditing is actually working. It gets the ON value when the logging is successfully started. Then it can get the OFF value if the logging was stopped or log records can't be properly stored due to file or syslog errors.
- **Data Type:** `boolean`

`Server_audit_current_log`

- **Description:** The name of the logfile or the SYSLOG parameters that are in current use.
- **Data Type:** `string`

`Server_audit_last_error`

- **Description:** If something went wrong with the logging here you can see the message.
- **Data Type:** `string`

`Server_audit_writes_failed`

- **Description:** The number of log records since last logging-start that weren't properly stored because of errors of any kind. The global value can be flushed by `FLUSH STATUS`.
- **Data Type:** `numeric`
- **Default Value:** `0`

5.4.6 Authentication Plugins

When a user attempts to log in, the authentication plugin controls how MariaDB Server determines whether the connection is from a legitimate user.

When creating or altering a user account with the GRANT, CREATE USER or ALTER USER statements, you can specify the authentication plugin you want the user account to use by providing the `IDENTIFIED VIA` clause. By default, when you create a user account without specifying an authentication plugin, MariaDB uses the mysql_native_password plugin.

MariaDB starting with 10.4

In MariaDB 10.4 and later, there are some notable changes, such as:

- You can specify multiple authentication plugins for each user account.
- The `root@localhost` user created by mariadb-install-db is created with the ability to use two authentication plugins. First, it is configured to try to use the unix_socket authentication plugin. This allows the the `root@localhost` user to login without a password via the local Unix socket file defined by the socket system variable, as long as the login is attempted from a process owned by the operating system `root` user account. Second, if authentication fails with the unix_socket authentication plugin, then it is configured to try to use the mysql_native_password authentication plugin. However, an invalid password is initially set, so in order to authenticate this way, a password must be set with SET PASSWORD.

Pluggable Authentication Overview

The authentication of users is delegated to plugins.

Authentication Plugin - mysql_native_password

Uses the password hashing algorithm introduced in MySQL 4.1.

Authentication Plugin - mysql_old_password

The mysql_old_password authentication plugin uses the pre-MySQL 4.1 password hashing algorithm.

Authentication Plugin - ed25519

Uses the Elliptic Curve Digital Signature Algorithm to securely store users' passwords.

Authentication Plugin - GSSAPI

The gssapi authentication plugin uses the GSSAPI interface to authenticate with Kerberos or NTLM.

Authentication with Pluggable Authentication Modules (PAM)

Uses the Pluggable Authentication Module (PAM) framework to authenticate MariaDB users.

Authentication Plugin - Unix Socket

Uses the user name that owns the process connected to MariaDB's unix socket file.

Authentication Plugin - Named Pipe

Uses the user name that owns the process connected to MariaDB's named pipe on Windows.

Authentication Plugin - SHA-256

MySQL supports the sha256_password and caching_sha2_password authentication plugins.

There are 1 related questions.

5.4.6.1 Pluggable Authentication Overview

Contents

When a user attempts to log in, the authentication plugin controls how MariaDB Server determines whether the connection is from a legitimate user.

When creating or altering a user account with the GRANT, CREATE USER or ALTER USER statements, you can specify the authentication plugin you want the user account to use by providing the `IDENTIFIED VIA` clause. By default, when you create a user account without specifying an authentication plugin, MariaDB uses the mysql_native_password plugin.

MariaDB starting with 10.4

In MariaDB 10.4 and later, there are some notable changes, such as:

- You can specify multiple authentication plugins for each user account.
- The `root@localhost` user created by mariadb-install-db is created with the ability to use two authentication plugins. First, it is configured to try to use the unix_socket authentication plugin. This allows the the `root@localhost` user to login without a password via the local Unix socket file defined by the socket system variable, as long as the login is attempted from a process owned by the operating system `root` user account. Second, if authentication fails with the unix_socket authentication plugin, then it is configured to try to use the mysql_native_password authentication plugin. However, an invalid password is initially set, so in order to authenticate this way, a password must be set with SET PASSWORD.

Supported Authentication Plugins

The authentication process is a conversation between the server and a client. MariaDB implements both server-side and client-side authentication plugins.

Supported Server Authentication Plugins

MariaDB provides seven server-side authentication plugins:

- mysql_native_password
- mysql_old_password
- ed25519
- gssapi
- pam (Unix only)
- unix_socket (Unix only)
- named_pipe (Windows only)

Supported Client Authentication Plugins

MariaDB provides eight client-side authentication plugins:

- mysql_native_password
- mysql_old_password
- client_ed25519
- auth_gssapi_client
- dialog
- mysql_clear_password
- sha256_password
- caching_sha256_password

Options Related to Authentication Plugins

Server Options Related to Authentication Plugins

MariaDB supports the following server options related to authentication plugins:

Server Option	Description
`old_passwords={1 | 0}`	If set to `1` (`0` is default), MariaDB reverts to using the mysql_old_password authentication plugin by default for newly created users and passwords, instead of the mysql_native_password authentication plugin.
`plugin_dir=path`	Path to the plugin directory. For security reasons, either make sure this directory can only be read by the server, or set `secure_file_priv`.
`plugin_maturity=level`	The lowest acceptable plugin maturity. MariaDB will not load plugins less mature than the specified level.
`secure_auth`	Connections will be blocked if they use the the mysql_old_password authentication plugin. The server will also fail to start if the privilege tables are in the old, pre-MySQL 4.1 format.

Client Options Related to Authentication Plugins

Most clients and utilities support some command line arguments related to client authentication plugins:

Client Option	Description
`--connect-expired-password`	Notify the server that this client is prepared to handle expired password sandbox mode even if `--batch` was specified. From MariaDB 10.4.3.
`--default-auth=name`	Default authentication client-side plugin to use.
`--plugin-dir=path`	Directory for client-side plugins.
`--secure-auth`	Refuse to connect to the server if the server uses the mysql_old_password authentication plugin. This mode is off by default, which is a difference in behavior compared to MySQL 5.6 and later, where it is on by default.

Developers who are using MariaDB Connector/C can implement similar functionality in their application by setting the following options with the mysql_optionsv function:

- `MYSQL_OPT_CAN_HANDLE_EXPIRED_PASSWORDS`
- `MYSQL_PLUGIN_DIR`
- `MYSQL_DEFAULT_AUTH`
- `MYSQL_SECURE_AUTH`

For example:

```
mysql_optionsv(mysql, MYSQL_OPT_CAN_HANDLE_EXPIRED_PASSWORDS, 1);
mysql_optionsv(mysql, MYSQL_DEFAULT_AUTH, "name");
mysql_optionsv(mysql, MYSQL_PLUGIN_DIR, "path");
mysql_optionsv(mysql, MYSQL_SECURE_AUTH, 1);
```

Installation Options Related to Authentication Plugins

mariadb-install-db supports the following installation options related to authentication plugins:

Installation Option	Description
`--auth-root-authentication-method={normal | socket}`	If set to `normal`, it creates a `root@localhost` account that authenticates with the mysql_native_password authentication plugin and that has no initial password set, which can be insecure. If set to `socket`, it creates a `root@localhost` account that authenticates with the unix_socket authentication plugin. Set to `normal` by default. Available since MariaDB 10.1.
`--auth-root-socket-user=USER`	Used with `--auth-root-authentication-method=socket`. It specifies the name of the second account to create with SUPER privileges in addition to `root`, as well as of the system account allowed to access it. Defaults to the value of `--user`.

Extended SQL Syntax

MariaDB has extended the SQL standard GRANT, CREATE USER, and ALTER USER statements, so that they support specifying different authentication plugins for specific users. An authentication plugin can be specified with these statements by providing the `IDENTIFIED VIA` clause.

For example, the GRANT syntax is:

```
GRANT <privileges> ON <level> TO <user>
   IDENTIFIED VIA <plugin> [ USING <string> ]
```

And the CREATE USER syntax is:

```
CREATE USER <user>
   IDENTIFIED VIA <plugin> [ USING <string> ]
```

And the ALTER USER syntax is:

```
ALTER USER <user>
   IDENTIFIED VIA <plugin> [ USING <string> ]
```

The optional `USING` clause allows users to provide an authentication string to a plugin. The authentication string's format and meaning is completely defined by the plugin.

For example, for the mysql_native_password authentication plugin, the authentication string should be a password hash:

```
CREATE USER mysqltest_up1
   IDENTIFIED VIA mysql_native_password USING '*E8D46CE25265E545D225A8A6F1BAF642FEBEE5CB';
```

Since mysql_native_password is the default authentication plugin, the above is just another way of saying the following:

```
CREATE USER mysqltest_up1
   IDENTIFIED BY PASSWORD '*E8D46CE25265E545D225A8A6F1BAF642FEBEE5CB';
```

In contrast, for the pam authentication plugin, the authentication string should refer to a PAM service name:

```
CREATE USER mysqltest_up1
   IDENTIFIED VIA pam USING 'mariadb';
```

MariaDB starting with 10.4

In MariaDB 10.4 and later, a user account can be associated with multiple authentication plugins.

For example, to configure the `root@localhost` user account to try the unix_socket authentication plugin, followed by the mysql_native_password authentication plugin as a backup, you could execute the following:

```
CREATE USER root@localhost
   IDENTIFIED VIA unix_socket
   OR mysql_native_password USING PASSWORD("verysecret");
```

See Authentication from MariaDB 10.4 for more information.

Authentication Plugins Installed by Default

Server Authentication Plugins Installed by Default

Not all server-side authentication plugins are installed by default. If a specific server-side authentication plugin is not installed by default, then you can find the installation procedure on the documentation page for the specific authentication plugin.

MariaDB starting with 10.4

In MariaDB 10.4 and later, the following server-side authentication plugins are installed by default:

- The mysql_native_password and mysql_old_password authentication plugins authentication plugins are installed by default in all builds.
- The unix_socket authentication plugin is installed by default in all builds on Unix and Linux.
- The named_pipe authentication plugin is installed by default in all builds on Windows.

MariaDB until 10.3

In MariaDB 10.3 and below, the following server-side authentication plugins are installed by default:

- The mysql_native_password and mysql_old_password authentication plugins are installed by default in all builds.
- The unix_socket authentication plugin is installed by default in **new installations** that use the .deb packages provided by Debian's default repositories in Debian 9 and later and Ubuntu's default repositories in Ubuntu 15.10 and later. See Differences in MariaDB in Debian (and Ubuntu) for more information.
- The named_pipe authentication plugin is installed by default in all builds on Windows.

Client Authentication Plugins Installed by Default

Client-side authentication plugins do not need to be *installed* in the same way that server-side authentication plugins do. If the client uses either the `libmysqlclient` or MariaDB Connector/C library, then the library automatically loads client-side authentication plugins from the library's plugin directory whenever they are needed.

Most clients and utilities support the `--plugin-dir` command line argument that can be used to set the path to the library's plugin directory:

Client Option	Description
`--plugin-dir=path`	Directory for client-side plugins.

Developers who are using MariaDB Connector/C can implement similar functionality in their application by setting the `MYSQL_PLUGIN_DIR` option with the mysql_optionsv function.

For example:

```
mysql_optionsv(mysql, MYSQL_PLUGIN_DIR, "path");
```

If your client encounters errors similar to the following, then you may need to set the path to the library's plugin directory:

```
ERROR 2059 (HY000): Authentication plugin 'dialog' cannot be loaded:
/usr/lib/mysql/plugin/dialog.so: cannot open shared object file: No such file or directory
```

If the client does **not** use either the `libmysqlclient` or MariaDB Connector/C library, then you will have to determine which authentication plugins are supported by the specific client library used by the client.

If the client uses either the `libmysqlclient` or MariaDB Connector/C library, but the client is not bundled with either library's *optional* client authentication plugins, then you can only use the conventional authentication plugins (like mysql_native_password and mysql_old_password) and the non-conventional authentication plugins that don't require special client-side authentication plugins (like unix_socket and named_pipe).

Default Authentication Plugin

Default Server Authentication Plugin

The mysql_native_password authentication plugin is currently the default authentication plugin in all versions of MariaDB if the old_passwords system variable is set to `0`, which is the default.

On a system with the old_passwords system variable set to `0`, this means that if you create a user account with either the GRANT or CREATE USER `statements, and if you do not specify an authentication plugin with the` IDENTIFIED VIA `clause, then MariaDB will use the mysql_native_password authentication plugin for the user account.`

For example, this user account will use the mysql_native_password authentication plugin:

```
CREATE USER username@hostname;
```

And so will this user account:

```
CREATE USER username@hostname IDENTIFIED BY 'notagoodpassword';
```

The mysql_old_password authentication plugin becomes the default authentication plugin in all versions of MariaDB if the old_passwords system variable is explicitly set to `1`.

However, the mysql_old_password authentication plugin is not considered secure, so it is recommended to avoid using this authentication plugin. To help prevent undesired use of the mysql_old_password authentication plugin, the server supports the secure_auth system variable that can be used to configured the server to refuse connections that try to use the mysql_old_password authentication plugin:

Server Option	Description
`old_passwords={1 | 0}`	If set to `1` (`0` is default), MariaDB reverts to using the mysql_old_password authentication plugin by default for newly created users and passwords, instead of the mysql_native_password authentication plugin.
`secure_auth`	Connections will be blocked if they use the the mysql_old_password authentication plugin. The server will also fail to start if the privilege tables are in the old, pre-MySQL 4.1 format.

Most clients and utilities also support the `--secure-auth` command line argument that can also be used to configure the client to refuse to connect to servers that use the mysql_old_password authentication plugin:

Client Option	Description
`--secure-auth`	Refuse to connect to the server if the server uses the mysql_old_password authentication plugin. This mode is off by default, which is a difference in behavior compared to MySQL 5.6 and later, where it is on by default.

Developers who are using MariaDB Connector/C can implement similar functionality in their application by setting the `MYSQL_SECURE_AUTH` option with the mysql_optionsv function.

For example:

```
mysql_optionsv(mysql, MYSQL_SECURE_AUTH, 1);
```

Default Client Authentication Plugin

The default client-side authentication plugin depends on a few factors.

If a client doesn't explicitly set the default client-side authentication plugin, then the client will determine which authentication plugin to use by checking the length of the scramble in the server's handshake packet.

If the server's handshake packet contains a 9-byte scramble, then the client will default to the mysql_old_password authentication plugin.

If the server's handshake packet contains a 20-byte scramble, then the client will default to the mysql_native_password authentication plugin.

Setting the Default Client Authentication Plugin

Most clients and utilities support the `--default-auth` command line argument that can be used to set the default client-side authentication plugin:

Client Option	Description
`--default-auth=name`	Default authentication client-side plugin to use.

Developers who are using MariaDB Connector/C can implement similar functionality in their application by setting the `MYSQL_DEFAULT_AUTH` option with the mysql_optionsv function.

For example:

```
mysql_optionsv(mysql, MYSQL_DEFAULT_AUTH, "name");
```

If you know that your user account is configured to require a client-side authentication plugin that isn't mysql_old_password or mysql_native_password, then it can help speed up your connection process to explicitly set the default client-side authentication plugin.

According to the client-server protocol, the server first sends the handshake packet to the client, then the client replies with a packet containing the user name of the user account that is requesting access. The server handshake packet initially tells the client to use the default server authentication plugin, and the client reply initially tells the server that it will use the default client authentication plugin.

However, the server-side and client-side authentication plugins mentioned in these initial packets may not be the correct ones for this specific user account. The server only knows what authentication plugin to use for this specific user account after reading the user name from the client reply packet and finding the appropriate row for the user account in either the mysql.user table or the mysql.global_priv table, depending on the MariaDB version.

If the server finds that either the server-side or client-side default authentication plugin does not match the actual authentication plugin that should be used for the given user account, then the server restarts the authentication on either the server side or the client side.

This means that, if you know what client authentication plugin your user account requires, then you can avoid an unnecessary authentication restart and you can save two packets and two round-trips.between the client and server by configuring your client to use the correct authentication plugin by default.

Authentication Plugins

Server Authentication Plugins

`mysql_native_password`

The mysql_native_password authentication plugin uses the password hashing algorithm introduced in MySQL 4.1, which is also used by the PASSWORD() `function when old_passwords=0 is set. This hashing algorithm is based on SHA-1`.

`mysql_old_password`

The mysql_old_password authentication plugin uses the pre-MySQL 4.1 password hashing algorithm, which is also used by the OLD_PASSWORD() function and by the PASSWORD() function when old_passwords=1 is set.

`ed25519`

The ed25519 authentication plugin uses Elliptic Curve Digital Signature Algorithm to securely store users' passwords and to authenticate users. The ed25519 algorithm is the same one that is used by OpenSSH. It is based on the elliptic curve and code created by Daniel J. Bernstein.

From a user's perspective, the ed25519 authentication plugin still provides conventional password-based authentication.

`gssapi`

The gssapi authentication plugin allows the user to authenticate with services that use the Generic Security Services Application Program Interface (GSSAPI). Windows has a slightly different but very similar API called Security Support Provider Interface (SSPI).

On Windows, this authentication plugin supports Kerberos and NTLM authentication. Windows authentication is supported regardless of whether a domain is used in the environment.

On Unix systems, the most dominant GSSAPI service is Kerberos. However, it is less commonly used on Unix systems than it is on Windows. Regardless, this authentication plugin also supports Kerberos authentication on Unix.

The gssapi authentication plugin is most often used for authenticating with Microsoft Active Directory.

pam

The pam authentication plugin allows MariaDB to offload user authentication to the system's Pluggable Authentication Module (PAM) framework. PAM is an authentication framework used by Linux, FreeBSD, Solaris, and other Unix-like operating systems.

unix_socket

The unix_socket authentication plugin allows the user to use operating system credentials when connecting to MariaDB via the local Unix socket file. This Unix socket file is defined by the socket system variable.

The unix_socket authentication plugin works by calling the getsockopt system call with the `SO_PEERCRED` socket option, which allows it to retrieve the `uid` of the process that is connected to the socket. It is then able to get the user name associated with that `uid`. Once it has the user name, it will authenticate the connecting user as the MariaDB account that has the same user name.

For example:

```
$ mysql -uroot
MariaDB []> CREATE USER serg IDENTIFIED VIA unix_socket;
MariaDB []> CREATE USER monty IDENTIFIED VIA unix_socket;
MariaDB []> quit
Bye
$ whoami
serg
$ mysql --user=serg
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 2
Server version: 5.2.0-MariaDB-alpha-debug Source distribution
MariaDB []> quit
Bye
$ mysql --user=monty
ERROR 1045 (28000): Access denied for user 'monty'@'localhost' (using password: NO)
```

In this example, a user `serg` is already logged into the operating system and has full shell access. He has already authenticated with the operating system and his MariaDB account is configured to use the unix_socket authentication plugin, so he does not need to authenticate again for the database. MariaDB accepts his operating system credentials and allows him to connect. However, any attempt to connect to the database as another operating system user will be denied.

named_pipe

The named_pipe authentication plugin allows the user to use operating system credentials when connecting to MariaDB via named pipe on Windows. Named pipe connections are enabled by the named_pipe system variable.

The named_pipe authentication plugin works by using named pipe impersonation and calling `GetUserName()` to retrieve the user name of the process that is connected to the named pipe. Once it has the user name, it authenticates the connecting user as the MariaDB account that has the same user name.

For example:

```
CREATE USER wlad IDENTIFIED VIA named_pipe;
CREATE USER monty IDENTIFIED VIA named_pipe;
quit

C:\>echo %USERNAME%
wlad

C:\> mysql --user=wlad --protocol=PIPE
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 4
Server version: 10.1.12-MariaDB-debug Source distribution

Copyright (c) 2000, 2015, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> quit
Bye

C:\> mysql --user=monty  --protocol=PIPE
ERROR 1698 (28000): Access denied for user 'monty'@'localhost'
```

Authentication Plugin API

The authentication plugin API is extensively documented in the source code in the following files:

- `mysql/plugin_auth.h` (server part)
- `mysql/client_plugin.h` (client part)
- `mysql/plugin_auth_common.h` (common parts)

The MariaDB source code also contains some authentication plugins that are intended explicitly to be examples for developers. They are located in `plugin/auth_examples`.

The definitions of two example authentication plugins called `two_questions` and `three_attempts` can be seen in `plugin/auth_examples/dialog_examples.c`. These authentication plugins demonstrate how to communicate with the user using the dialog client authentication plugin.

The `two_questions` authentication plugin asks the user for a password and a confirmation ("Are you sure?").

The `three_attempts` authentication plugin gives the user three attempts to enter a correct password.

The password for both of these plugins should be specified in the plain text in the `USING` clause:

```
CREATE USER insecure IDENTIFIED VIA two_questions USING 'notverysecret';
```

Dialog Client Authentication Plugin - Client Library Extension

The dialog client authentication plugin, strictly speaking, is not part of the client-server or authentication plugin API. But it can be loaded into any client application that uses the `libmysqlclient` or MariaDB Connector/C libraries. This authentication plugin provides a way for the application to customize the UI of the dialog function.

In order to use the dialog client authentication plugin to communicate with the user in a customized way, the application will need to implement a function with the following signature:

```
extern "C" char *mysql_authentication_dialog_ask(
  MYSQL *mysql, int type, const char *prompt, char *buf, int buf_len)
```

The function takes the following arguments:

- The connection handle.
- A question "type", which has one of the following values:
  - `1` - Normal question
  - `2` - Password (no echo)
- A prompt.
- A buffer.
- The length of the buffer.

The function returns a pointer to a string of characters, as entered by the user. It may be stored in `buf` or allocated with `malloc()`.

Using this function a GUI application can pop up a dialog window, a network application can send the question over the network, as required. If no `mysql_authentication_dialog_ask` function is provided by the application, the dialog client authentication plugin falls back to fputs() and fgets().

Providing this callback is particularly important on Windows, because Windows GUI applications have no associated console and the default dialog function will not be able to reach the user. An example of Windows GUI client that does it correctly is HeidiSQL.

5.4.6.2 Authentication Plugin - mysql_native_password

The `mysql_native_password` authentication plugin is the default authentication plugin that will be used for an account created when no authentication plugin is explicitly mentioned and old_passwords=0 is set. It uses the password hashing algorithm introduced in MySQL 4.1, which is also used by the PASSWORD() function when old_passwords=0 is set. This hashing algorithm is based on SHA-1.

It is not recommended to use the `mysql_native_password` authentication plugin for new installations that require **high password security**. If someone is able to both listen to the connection protocol and get a copy of the mysql.user table, then the person would be able to use this information to connect to the MariaDB server. The ed25519

authentication plugin is a more modern authentication plugin that provides simple password authentication using a more secure algorithm.

Contents

Installing the Plugin

The `mysql_native_password` authentication plugin is statically linked into the server, so no installation is necessary.

Creating Users

The easiest way to create a user account with the `mysql_native_password` authentication plugin is to make sure that old_passwords=0 is set, and then create a user account via CREATE USER that does not specify an authentication plugin, but does specify a password via the IDENTIFIED BY clause. For example:

```
SET old_passwords=0;
CREATE USER username@hostname IDENTIFIED BY 'mariadb';
```

If SQL_MODE does not have `NO_AUTO_CREATE_USER` set, then you can also create the user account via GRANT. For example:

```
SET old_passwords=0;
GRANT SELECT ON db.* TO username@hostname IDENTIFIED BY 'mariadb';
```

You can also create the user account by providing a password hash via the IDENTIFIED BY PASSWORD clause, and MariaDB will validate whether the password hash is one that is compatible with `mysql_native_password`. For example:

```
SET old_passwords=0;

SELECT PASSWORD('mariadb');
+-------------------------------------------+
| PASSWORD('mariadb')                       |
+-------------------------------------------+
| *54958E764CE10E50764C2EECBB71D01F08549980 |
+-------------------------------------------+

CREATE USER username@hostname
  IDENTIFIED BY PASSWORD '*54958E764CE10E50764C2EECBB71D01F08549980';
```

Similar to all other authentication plugins, you could also specify the name of the plugin in the IDENTIFIED VIA clause while providing the password hash as the `USING` clause. For example:

```
CREATE USER username@hostname
  IDENTIFIED VIA mysql_native_password USING '*54958E764CE10E50764C2EECBB71D01F08549980';
```

Changing User Passwords

You can change a user account's password with the SET PASSWORD statement while providing the plain-text password as an argument to the PASSWORD() function. For example:

```
SET PASSWORD =  PASSWORD('new_secret')
```

You can also change the user account's password with the ALTER USER statement. You would have to make sure that old_passwords=0 is set, and then you would have to specify a password via the IDENTIFIED BY clause. For example:

```
SET old_passwords=0;
ALTER USER username@hostname IDENTIFIED BY 'new_secret';
```

Client Authentication Plugins

For clients that use the `libmysqlclient` or MariaDB Connector/C libraries, MariaDB provides one client authentication plugin that is compatible with the `mysql_native_password` authentication plugin:

- `mysql_native_password`

When connecting with a client or utility to a server as a user account that authenticates with the `mysql_native_password` authentication plugin, you may need to tell the client where to find the relevant client authentication plugin by specifying the `--plugin-dir` option. For example:

```
mysql --plugin-dir=/usr/local/mysql/lib64/mysql/plugin --user=alice
```

However, the `mysql_native_password` client authentication plugin is generally statically linked into client libraries like `libmysqlclient` or MariaDB Connector/C, so this is not usually necessary.

`mysql_native_password`

The `mysql_native_password` client authentication plugin hashes the password before sending it to the server.

Support in Client Libraries

The `mysql_native_password` authentication plugin is one of the conventional authentication plugins, so all client libraries should support it.

Known Old Issues (Only Relevant for Old Installations)

Mismatches Between Password and authentication_string Columns

For compatibility reasons,the `mysql_native_password` authentication plugin tries to read the password hash from both the `Password` and `authentication_string` columns in the mysql.user table. This has caused issues in the past if one of the columns had a different value than the other.

Starting with MariaDB 10.2.19 and MariaDB 10.3.11, CREATE USER, ALTER USER, GRANT, and SET PASSWORD will set both columns whenever an account's password is changed.

See MDEV-16774 for more information.

5.4.6.3 Authentication Plugin - mysql_old_password

The `mysql_old_password` authentication plugin is the default authentication plugin that will be used for an account created when no authentication plugin is explicitly mentioned and `old_passwords=1` is set. It uses the pre-MySQL 4.1 password hashing algorithm, which is also used by the `OLD_PASSWORD()` function and by the `PASSWORD()` function when `old_passwords=1` is set.

It is not recommended to use the `mysql_old_password` authentication plugin for new installations. The password hashing algorithm is no longer as secure as it used to be, and the plugin is primarily provided for backward-compatibility. The `ed25519` authentication plugin is a more modern authentication plugin that provides simple password authentication.

Contents

Installing the Plugin

The `mysql_old_password` authentication plugin is statically linked into the server, so no installation is necessary.

Creating Users

The easiest way to create a user account with the `mysql_old_password` authentication plugin is to make sure that `old_passwords=1` is set, and then create a user account via `CREATE USER` that does not specify an authentication plugin, but does specify a password via the `IDENTIFIED BY` clause. For example:

```
SET old_passwords=1;
CREATE USER username@hostname IDENTIFIED BY 'mariadb';
```

If `SQL_MODE` does not have `NO_AUTO_CREATE_USER` set, then you can also create the user via `GRANT`. For example:

```
SET old_passwords=1;
GRANT SELECT ON db.* TO username@hostname IDENTIFIED BY 'mariadb';
```

You can also create the user account by providing a password hash via the `IDENTIFIED BY PASSWORD` clause, and MariaDB will validate whether the password hash is one that is compatible with `mysql_old_password`. For example:

```
SET old_passwords=1;
Query OK, 0 rows affected (0.000 sec)

SELECT PASSWORD('mariadb');
+---------------------+
| PASSWORD('mariadb') |
+---------------------+
| 021bec665bf663f1    |
+---------------------+
1 row in set (0.000 sec)

CREATE USER username@hostname IDENTIFIED BY PASSWORD '021bec665bf663f1';
Query OK, 0 rows affected (0.000 sec)
```

Similar to all other authentication plugins, you could also specify the name of the plugin in the `IDENTIFIED VIA` clause while providing the password hash as the `USING` clause. For example:

```
CREATE USER username@hostname IDENTIFIED VIA mysql_old_password USING '021bec665bf663f1';
Query OK, 0 rows affected (0.000 sec)
```

Changing User Passwords

You can change a user account's password with the `SET PASSWORD` statement while providing the plain-text password as an argument to the `PASSWORD()` function. For example:

```
SET PASSWORD =  PASSWORD('new_secret')
```

You can also change the user account's password with the `ALTER USER` statement. You would have to make sure that `old_passwords=1` is set, and then you would have to specify a password via the `IDENTIFIED BY` clause. For example:

```
SET old_passwords=1;
ALTER USER username@hostname IDENTIFIED BY 'new_secret';
```

Client Authentication Plugins

For clients that use the `libmysqlclient` or MariaDB Connector/C libraries, MariaDB provides one client authentication plugin that is compatible with the `mysql_old_password` authentication plugin:

- `mysql_old_password`

When connecting with a client or utility to a server as a user account that authenticates with the `mysql_old_password` authentication plugin, you may need to tell the client where to find the relevant client authentication plugin by specifying the `--plugin-dir` option. For example:

```
mysql --plugin-dir=/usr/local/mysql/lib64/mysql/plugin --user=alice
```

However, the `mysql_old_password` client authentication plugin is generally statically linked into client libraries like `libmysqlclient` or MariaDB Connector/C, so this is not usually necessary.

`mysql_old_password`

The `mysql_old_password` client authentication plugin hashes the password before sending it to the server.

Support in Client Libraries

The `mysql_old_password` authentication plugin is one of the conventional authentication plugins, so all client libraries should support it.

5.4.6.4 Authentication Plugin - ed25519

MySQL has used SHA-1 based authentication since version 4.1. Since MariaDB 5.2 this authentication plugin has been called mysql_native_password. Over the years as computers became faster, new attacks on SHA-1 were being developed. Nowadays SHA-1 is no longer considered as secure as it was in 2001. That's why the `ed25519` authentication plugin was created.

The `ed25519` authentication plugin uses Elliptic Curve Digital Signature Algorithm (ECDSA) to securely store users' passwords and to authenticate users. The ed25519 algorithm is the same one that is used by OpenSSH. It is based on the elliptic curve and code created by Daniel J. Bernstein.

From a user's perspective, the `ed25519` authentication plugin still provides conventional password-based authentication.

Contents

Installing the Plugin

Although the plugin's shared library is distributed with MariaDB by default as `auth_ed25519.so` or `auth_ed25519.dll` depending on the operating system, the plugin is not actually installed by MariaDB by default. There are two methods that can be used to install the plugin with MariaDB.

The first method can be used to install the plugin without restarting the server. You can install the plugin dynamically by executing INSTALL SONAME or INSTALL PLUGIN. For example:

```
INSTALL SONAME 'auth_ed25519';
```

The second method can be used to tell the server to load the plugin when it starts up. The plugin can be installed this way by providing the --plugin-load or the --plugin-load-add options. This can be specified as a command-line argument to mariadbd or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
plugin_load_add = auth_ed25519
```

Uninstalling the Plugin

You can uninstall the plugin dynamically by executing UNINSTALL SONAME or UNINSTALL PLUGIN. For example:

```
UNINSTALL SONAME 'auth_ed25519';
```

If you installed the plugin by providing the --plugin-load or the --plugin-load-add options in a relevant server option group in an option file, then those options should be removed to prevent the plugin from being loaded the next time the server is restarted.

Creating Users

MariaDB starting with 10.4

In MariaDB 10.4 and later, you can create a user account by executing the CREATE USER statement and providing the IDENTIFIED VIA clause followied by the the name of the plugin, which is `ed25519` , and providing the the `USING` clause followed by the PASSWORD() function with the plain-text password as an argument. For example:

```
CREATE USER username@hostname IDENTIFIED VIA ed25519 USING PASSWORD('secret');
```

If SQL_MODE does not have `NO_AUTO_CREATE_USER` set, then you can also create the user account via GRANT. For example:

```
GRANT SELECT ON db.* TO username@hostname IDENTIFIED VIA ed25519 USING PASSWORD('secret');
```

MariaDB until 10.3

In MariaDB 10.3 and before, the PASSWORD() function and SET PASSWORD statement did not work with the `ed25519` authentication plugin. Instead, you would have to use the UDF that comes with the authentication plugin to calculate the password hash. For example:

```
CREATE FUNCTION ed25519_password RETURNS STRING SONAME "auth_ed25519.so";
```

Now you can calculate a password hash by executing:

```
SELECT ed25519_password("secret");
+-----------------------------------------------+
| SELECT ed25519_password("secret");            |
+-----------------------------------------------+
| ZIgUREUg5PVgQ6LskhXmO+eZLS0nC8be6HPjYWR4YJY |
+-----------------------------------------------+
```

Now you can use it to create the user account using the new password hash.

To create a user account via CREATE USER, specify the name of the plugin in the IDENTIFIED VIA clause while providing the password hash as the `USING` clause. For example:

```
CREATE USER username@hostname IDENTIFIED VIA ed25519
  USING 'ZIgUREUg5PVgQ6LskhXmO+eZLS0nC8be6HPjYWR4YJY';
```

If SQL_MODE does not have `NO_AUTO_CREATE_USER` set, then you can also create the user account via GRANT. For example:

```
GRANT SELECT ON db.* TO username@hostname IDENTIFIED VIA ed25519
  USING 'ZIgUREUg5PVgQ6LskhXmO+eZLS0nC8be6HPjYWR4YJY';
```

Note that users require a password in order to be able to connect. It is possible to create a user without specifying a password, but they will be unable to connect.

Changing User Passwords

MariaDB starting with 10.4

In MariaDB 10.4 and later, you can change a user account's password by executing the SET PASSWORD statement followed by the PASSWORD() function and providing the plain-text password as an argument. For example:

```
SET PASSWORD =  PASSWORD('new_secret')
```

You can also change the user account's password with the ALTER USER statement. You would have to specify the name of the plugin in the IDENTIFIED VIA clause while providing the plain-text password as an argument to the PASSWORD() function in the `USING` clause. For example:

```
ALTER USER username@hostname IDENTIFIED VIA ed25519 USING PASSWORD('new_secret');
```

MariaDB until 10.3

In MariaDB 10.3 and before, the PASSWORD() function and SET PASSWORD statement did not work with the `ed25519` authentication plugin. Instead, you would have to use the UDF that comes with the authentication plugin to calculate the password hash. For example:

```
CREATE FUNCTION ed25519_password RETURNS STRING SONAME "auth_ed25519.so";
```

Now you can calculate a password hash by executing:

```
SELECT ed25519_password("secret");
+-----------------------------------------------+
| SELECT ed25519_password("secret");            |
+-----------------------------------------------+
| ZIgUREUg5PVgQ6LskhXmO+eZLS0nC8be6HPjYWR4YJY |
+-----------------------------------------------+
```

Now you can change the user account's password using the new password hash.

You can change the user account's password with the ALTER USER statement. You would have to specify the name of the plugin in the IDENTIFIED VIA clause while providing the password hash as the `USING` clause. For example:

```
ALTER USER username@hostname IDENTIFIED VIA ed25519
  USING 'ZIgUREUg5PVgQ6LskhXmO+eZLS0nC8be6HPjYWR4YJY';
```

Client Authentication Plugins

For clients that use the `libmysqlclient` or MariaDB Connector/C libraries, MariaDB provides one client authentication plugin that is compatible with the `ed25519` authentication plugin:

- `client_ed25519`

When connecting with a client or utility to a server as a user account that authenticates with the `ed25519` authentication plugin, you may need to tell the client where to find the relevant client authentication plugin by specifying the `--plugin-dir` option. For example:

```
mysql --plugin-dir=/usr/local/mysql/lib64/mysql/plugin --user=alice
```

`client_ed25519`

The `client_ed25519` client authentication plugin hashes and signs the password using the Elliptic Curve Digital Signature Algorithm (ECDSA) before sending it to the server.

Support in Client Libraries

Using the Plugin with MariaDB Connector/C

MariaDB Connector/C supports `ed25519` authentication using the client authentication plugins mentioned in the previous section since MariaDB Connector/C 3.1.0.

Using the Plugin with MariaDB Connector/ODBC

MariaDB Connector/ODBC supports `ed25519` authentication using the client authentication plugins mentioned in the previous section since MariaDB Connector/ODBC 3.1.2.

Using the Plugin with MariaDB Connector/J

MariaDB Connector/J supports `ed25519` authentication since MariaDB Connector/J 2.2.1.

Using the Plugin with MariaDB Connector/Node.js

MariaDB Connector/Node.js supports `ed25519` authentication since MariaDB Connector/Node.js 2.1.0.

Using the Plugin with MySqlConnector for .NET

MySqlConnector for ADO.NET supports `ed25519` authentication since MySqlConnector 0.56.0.

The connector implemented support for this authentication plugin in a separate NuGet package called MySqlConnector.Authentication.Ed25519. After the package is installed, your application must call `Ed25519AuthenticationPlugin.Install` to enable it.

Versions

Version	Status	Introduced
1.1	Stable	MariaDB 10.4.0
1.0	Stable	MariaDB 10.3.8, MariaDB 10.2.17, MariaDB 10.1.35
1.0	Beta	MariaDB 10.2.5, MariaDB 10.1.22

Options

`ed25519`

- **Description:** Controls how the server should treat the plugin when the server starts up.
  - Valid values are:
    - `OFF` - Disables the plugin without removing it from the mysql.plugins table.
    - `ON` - Enables the plugin. If the plugin cannot be initialized, then the server will still continue starting up, but the plugin will be disabled.
    - `FORCE` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error.
    - `FORCE_PLUS_PERMANENT` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error. In addition, the plugin cannot be uninstalled with UNINSTALL SONAME or UNINSTALL PLUGIN while the server is running.
  - See Plugin Overview: Configuring Plugin Activation at Server Startup for more information.
- **Commandline:** `--ed25519=value`
- **Data Type:** `enumerated`
- **Default Value:** `ON`
- **Valid Values:** `OFF`, `ON`, `FORCE`, `FORCE_PLUS_PERMANENT`

5.4.6.5 Authentication Plugin - GSSAPI

The `gssapi` authentication plugin allows the user to authenticate with services that use the Generic Security Services Application Program Interface (GSSAPI). Windows has a slightly different but very similar API called Security Support Provider Interface (SSPI). The GSSAPI is a standardized API described in RFC2743 and RFC2744. The client and

server negotiate using a standardized protocol described in RFC7546.

On Windows, this authentication plugin supports Kerberos and NTLM authentication. Windows authentication is supported regardless of whether a domain is used in the environment.

On Unix systems, the most dominant GSSAPI service is Kerberos. However, it is less commonly used on Unix systems than it is on Windows. Regardless, this authentication plugin also supports Kerberos authentication on Unix.

The `gssapi` authentication plugin is most often used for authenticating with Microsoft Active Directory.

This article gives instructions on configuring the `gssapi` authentication plugin for MariaDB for passwordless login.

Contents

Installing the Plugin's Package

Since MariaDB 10.11, on Windows, the plugin is included in the server, and there is no need for separate installation.

The `gssapi` authentication plugin's shared library is included in MariaDB packages as the `auth_gssapi.so` or `auth_gssapi.dll` shared library on systems where it can be built.

Installing on Linux

The `gssapi` authentication plugin is included in binary tarballs on Linux.

Installing with a Package Manager

The `gssapi` authentication plugin can also be installed via a package manager on Linux. In order to do so, your system needs to be configured to install from one of the MariaDB repositories.

You can configure your package manager to install it from MariaDB Corporation's MariaDB Package Repository by using the MariaDB Package Repository setup script.

You can also configure your package manager to install it from MariaDB Foundation's MariaDB Repository by using the MariaDB Repository Configuration Tool.

Installing with yum/dnf

On RHEL, CentOS, Fedora, and other similar Linux distributions, it is highly recommended to install the relevant RPM package from MariaDB's repository using `yum` or `dnf`. Starting with RHEL 8 and Fedora 22, `yum` has been replaced by `dnf`, which is the next major version of `yum`. However, `yum` commands still work on many systems that use `dnf`. For example:

```
sudo yum install MariaDB-gssapi-server
```

Installing with apt-get

On Debian, Ubuntu, and other similar Linux distributions, it is highly recommended to install the relevant DEB package from MariaDB's repository using `apt-get`. For example:

```
sudo apt-get install mariadb-plugin-gssapi-server
```

Installing with zypper

On SLES, OpenSUSE, and other similar Linux distributions, it is highly recommended to install the relevant RPM package from MariaDB's repository using `zypper`. For example:

```
sudo zypper install MariaDB-gssapi-server
```

Installing on Windows

Since MariaDB 10.11, the plugin is included in the server, and there is no need for separate installation.

Before MariaDB 10.11, the `gssapi` authentication plugin is included in MSI and ZIP packages on Windows.

Installing the Plugin

Since MariaDB 10.11, on Windows, the plugin is included in the server, and there is no need for separate installation.

Before MariaDB 10.11 on Windows, and on other operating systems, although the plugin's shared library is distributed with MariaDB by default, the plugin is not actually installed by MariaDB by default. There are two methods that can be used to install the plugin with MariaDB.

The first method can be used to install the plugin without restarting the server. You can install the plugin dynamically by executing `INSTALL SONAME` or `INSTALL PLUGIN`. For example:

```
INSTALL SONAME 'auth_gssapi';
```

The second method can be used to tell the server to load the plugin when it starts up. The plugin can be installed this way by providing the --plugin-load or the --plugin-load-add options. This can be specified as a command-line argument to mariadbd or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
plugin_load_add = auth_gssapi
```

Uninstalling the Plugin

You can uninstall the plugin dynamically by executing UNINSTALL SONAME or UNINSTALL PLUGIN. For example:

```
UNINSTALL SONAME 'auth_gssapi';
```

If you installed the plugin by providing the `--plugin-load` or the `--plugin-load-add` options in a relevant server option group in an option file, then those options should be removed to prevent the plugin from being loaded the next time the server is restarted.

Configuring the Plugin

If the MariaDB server is running on Unix, then some additional configuration steps will need to be implemented in order to use the plugin.

If the MariaDB server is running on Windows, then no special configuration steps will need to be implemented in order to use the plugin, as long as the following is true:

- The Windows server is joined to a domain.
- The MariaDB server process is running as either a NetworkService Account or a Domain User Account.

Creating a Keytab File on Unix

If the MariaDB server is running on Unix, then the KDC server will need to create a keytab file for the MariaDB server. The keytab file contains the service principal name, which is the identity that the MariaDB server will use to communicate with the KDC server. The keytab will need to be transferred to the MariaDB server, and the `mysqld` server process will need read access to this keytab file.

How this keytab file is generated depends on whether the KDC server is **Microsoft Active Directory KDC** or **MIT Kerberos KDC**.

Creating a Keytab File with Microsoft Active Directory

If you are using **Microsoft Active Directory KDC**, then you may need to create a keytab using the `ktpass.exe` utility on a Windows host. The service principal will need to be mapped to an existing domain user. To do so, follow the steps listed below.

Be sure to replace the following items in the step below:

- Replace `${HOST}` with the fully qualified DNS name for the MariaDB server host.
- Replace `${DOMAIN}` with the Active Directory domain.
- Replace `${AD_USER}` with the existing domain user.
- Replace `${PASSWORD}` with the password for the service principal.

To create the service principal, execute the following:

```
ktpass.exe /princ mariadb/${HOST}@${DOMAIN} /mapuser ${AD_USER} /pass ${PASSWORD} /out
mariadb.keytab /crypto all /ptype KRB5_NT_PRINCIPAL /mapop set
```

Creating a Keytab File with MIT Kerberos

If you are using **MIT Kerberos KDC**, then you can create a keytab file using the `kadmin` utility. To do so, follow the steps listed below.

In the following steps, be sure to replace `${HOST}` with the fully qualified DNS name for the MariaDB server host.

First, create the service principal using the `kadmin` utility. For example:

```
kadmin -q "addprinc -randkey mariadb/${HOST}"
```

Then, export the newly created user to the keytab file using the `kadmin` utility. For example:

```
kadmin -q "ktadd -k /path/to/mariadb.keytab mariadb/${HOST}"
```

More details can be found at the following links:

- MIT Kerberos Documentation: Database administration
- MIT Kerberos Documentation: Application servers

Configuring the Path to the Keytab File on Unix

If the MariaDB server is running on Unix, then the path to the keytab file that was previously created can be set by configuring the `gssapi_keytab_path` system variable. This can be specified as a command-line argument to `mysqld` or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
gssapi_keytab_path=/path/to/mariadb.keytab
```

Configuring the Service Principal Name

The service principal name can be set by configuring the `gssapi_principal_name` system variable. This can be specified

as a command-line argument to `mysqld` or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
gssapi_principal_name=service_principal_name/host.domain.com@REALM
```

If a service principal name is not provided, then the plugin will try to use `mariadb/host.domain.com@REALM` by default.

If the MariaDB server is running on Unix, then the plugin needs a service principal name in order to function.

If the MariaDB server is running on Windows, then the plugin does not usually need a service principal in order to function. However, if you want to use one anyway, then one can be created with the `setspn` utility.

Different KDC implementations may use different canonical forms to identify principals. See RFC2744: Section 3.10 to learn what the standard says about principal names.

More details can be found at the following links:

- Active Directory Domain Services: Service Principal Names
- MIT Kerberos Documentation: Realm configuration decisions
- MIT Kerberos Documentation: Principal names and DNS

Creating Users

To create a user account via `CREATE USER`, specify the name of the plugin in the `IDENTIFIED VIA` clause. For example:

```
CREATE USER username@hostname IDENTIFIED VIA gssapi;
```

If `SQL_MODE` does not have `NO_AUTO_CREATE_USER` set, then you can also create the user account via `GRANT`. For example:

```
GRANT SELECT ON db.* TO username@hostname IDENTIFIED VIA gssapi;
```

You can also specify the user's realm for MariaDB with the `USING` clause. For example:

```
CREATE USER username@hostname IDENTIFIED VIA gssapi USING 'username@EXAMPLE.COM';
```

The format of the realm depends on the specific authentication mechanism that is used. For example, the format would need to be `machine\\username` for Windows users authenticating with NTLM.

If the realm is not provided in the user account's definition, then the realm is **not** used for comparison. Therefore, 'usr1@EXAMPLE.COM', 'usr1@EXAMPLE.CO.UK' and 'mymachine\usr1' would all identify as the following user account:

```
CREATE USER usr1@hostname IDENTIFIED VIA gssapi;
```

Creating Users Identified Via Group Membership or SID (Windows-specific)

Since 10.6.0, on Windows only, it is possible to login using a AD or local group-membership. This is achieved by using GROUP prefix in IDENTIFIED ... AS

```
CREATE USER root IDENTIFIED VIA gssapi as 'GROUP:Administrators'
CREATE USER root IDENTIFIED VIA gssapi as 'GROUP:BUILTIN\\Administrators'
```

Effect of the above definition is that every user that identifies as member of group Administrators can login using user name root, passwordless.

User can also login using own or group SID

```
CREATE USER root IDENTIFIED VIA gssapi as 'SID:S-1-5-32-544'
```

Using SIDs will perform slightly faster than using name (since it will spare translation between SID and name which is otherwise done), also SIDs immune against user or group renaming.

Passwordless login on Windows

MariaDB starting with 10.11

From MariaDB 10.11, on Windows, in addition to the usual authentication with a password, passwordless authentication is permitted, when creating the 'root' user during install.

This works in a similar manner to Unix socket authentication. However, since auth_gssapi, unlike unix_socket, requires client support, to avoid failures when MariaDB is used with 3rd party drivers, authentication on Windows first attempts password-based native_authentication, and only if it fails, falls back to passwordless auth_gssapi.

Client Authentication Plugins

For clients that use the `libmysqlclient` or MariaDB Connector/C libraries, MariaDB provides one client authentication plugin that is compatible with the `gssapi` authentication plugin:

- `auth_gssapi_client`

When connecting with a client or utility to a server as a user account that authenticates with the `gssapi` authentication plugin, you may need to tell the client where to find the relevant client authentication plugin by specifying the `--plugin-dir` option. For example:

```
mysql --plugin-dir=/usr/local/mysql/lib64/mysql/plugin --user=alice
```

`auth_gssapi_client`

The `auth_gssapi_client` client authentication plugin receives the principal name from the server, and then uses either the `gss_init_sec_context` function (on Unix) or the `InitializeSecurityContext` function (on Windows) to establish a security context on the client.

Support in Client Libraries

Using the Plugin with MariaDB Connector/C

MariaDB Connector/C supports `gssapi` authentication using the client authentication plugins mentioned in the previous section since MariaDB Connector/C 3.0.1.

Using the Plugin with MariaDB Connector/ODBC

MariaDB Connector/ODBC supports `gssapi` authentication using the client authentication plugins mentioned in the previous section since MariaDB Connector/ODBC 3.0.0.

Using the Plugin with MariaDB Connector/J

MariaDB Connector/J supports `gssapi` authentication since MariaDB Connector/J 1.4.0. Current documentation can be found here.

Using the Plugin with MariaDB Connector/Node.js

MariaDB Connector/Node.js does not yet support `gssapi` authentication. See CONJS-72 for more information.

Using the Plugin with MySqlConnector for .NET

MySqlConnector for ADO.NET supports `gssapi` authentication since MySqlConnector 0.47.0.

The support is transparent. Normally, the connector only needs to be provided the correct user name, and no other parameters are required.

However, this connector also supports the `ServerSPN` connection string parameter, which can be used for mutual authentication.

.NET specific problems/workarounds

When connecting from Unix client to Windows server with ADO.NET, in an Active Directory domain environment, be aware that .NET Core on Unix does not support principal names in UPN(User Principal Name) form, which is default on Windows (e.g machine$@domain.com) . Thus, upon encountering an authentication exception with "server not found in Kerberos database", use one of workarounds below

- Force host-based SPN on server side.
  - For example, this can be done by setting the `gssapi_principal_name` system variable to `HOST/machine` in a server option group in an option file.
- Pass host-based SPN on client side.
  - For example, this can be done by setting the connector's `ServerSPN` connection string parameter to `HOST/machine` .

Versions

Version	Status	Introduced
1.0	Stable	MariaDB 10.1.15
1.0	Beta	MariaDB 10.1.11

System Variables

gssapi_keytab_path

- **Description:** Defines the path to the server's keytab file.
  - This system variable is only meaningful on **Unix**.
  - See Creating a Keytab File on Unix and Configuring the Path to the Keytab File on Unix for more information.
- **Commandline:** `--gssapi-keytab-path`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** ''
- **Introduced:** MariaDB 10.1.11

---

gssapi_principal_name

- **Description:** Name of the service principal.
  - See Configuring the Service Principal Name for more information.
- **Commandline:** `--gssapi-principal-name`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** ''
- **Introduced:** MariaDB 10.1.11

---

gssapi_mech_name

- **Description:** Name of the SSPI package used by server. Can be either 'Kerberos' or 'Negotiate'. Set it to 'Kerberos', to prevent less secure NTLM in domain environments, but leave it as default (Negotiate) to allow non-domain environments (e.g if server does not run in a domain environment).
  - This system variable is only meaningful on **Windows**.
- **Commandline:** `--gssapi-mech-name`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `enumerated`
- **Default Value:** `Negotiate`
- **Valid Values:** `Kerberos` , `Negotiate`
- **Introduced:** MariaDB 10.1.11

Options

gssapi

- **Description:** Controls how the server should treat the plugin when the server starts up.
  - Valid values are:
    - `OFF` - Disables the plugin without removing it from the `mysql.plugins` table.
    - `ON` - Enables the plugin. If the plugin cannot be initialized, then the server will still continue starting up, but the plugin will be disabled.
    - `FORCE` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error.
    - `FORCE_PLUS_PERMANENT` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error. In addition, the plugin cannot be uninstalled with `UNINSTALL SONAME` or `UNINSTALL PLUGIN` while the server is running.
  - See Plugin Overview: Configuring Plugin Activation at Server Startup for more information.
- **Commandline:** `--gssapi=value`
- **Data Type:** `enumerated`
- **Default Value:** `ON`
- **Valid Values:** `OFF`, `ON`, `FORCE`, `FORCE_PLUS_PERMANENT`
- **Introduced:** MariaDB 10.1.11

5.4.6.6 Authentication with Pluggable Authentication Modules (PAM)

Authentication Plugin - PAM

Uses the Pluggable Authentication Module (PAM) framework to authenticate MariaDB users.

User and Group Mapping with PAM

Configure PAM to map a given PAM user or group to a different MariaDB user.

Configuring PAM Authentication and User Mapping with Unix Authentication

Walkthrough configuration of PAM authentication and user mapping with Unix authentication.

Configuring PAM Authentication and User Mapping with LDAP Authentication

Configuring PAM authentication and user mapping with LDAP authentication.

5.4.6.6.1 Authentication Plugin - PAM

The `pam` authentication plugin allows MariaDB to offload user authentication to the system's Pluggable Authentication Module (PAM) framework. PAM is an authentication framework used by Linux, FreeBSD, Solaris, and other Unix-like operating systems.

> **Note:** Windows does not support PAM, so the `pam` authentication plugin does not support Windows. However, one can use a MariaDB client on Windows to connect to MariaDB server that is installed on a Unix-like operating system and that is configured to use the `pam` authentication plugin. For an example of how to do this, see the blog post: MariaDB: Improve Security with Two-Step Verification.

Contents

Use Cases

PAM makes it possible to implement various authentication scenarios of different complexity. For example,

- Authentication using passwords from `/etc/shadow` (indeed, this is what a default PAM configuration usually does). See the pam_unix PAM module.
- Authentication using LDAP. See the pam_ldap PAM module.
- Authentication using Microsoft's Active Directory. See the pam_lsass, pam_winbind, and pam_centrifydc PAM modules.
- Authentication using one-time passwords (even with SMS confirmation!). See the pam_google_authenticator and pam_securid PAM modules.
- Authentication using SSH keys. See the pam_ssh PAM module.
- User and group mapping. See the pam_user_map PAM module.
- Combining different authentication modules in interesting ways in a PAM service.
- Password expiration.

- Limiting access by time, date, day of the week, etc. See the pam_time PAM module.
- Logging every login attempt.
- and so on, the list is in no way exhaustive.

Installing the Plugin

The `pam` authentication plugin's library is provided in binary packages in all releases on Linux.

Although the plugin's shared library is distributed with MariaDB by default, the plugin is not actually installed by MariaDB by default. There are two methods that can be used to install the plugin with MariaDB.

The first method can be used to install the plugin without restarting the server. You can install the plugin dynamically by executing INSTALL SONAME or INSTALL PLUGIN. For example:

```
INSTALL SONAME 'auth_pam';
```

The second method can be used to tell the server to load the plugin when it starts up. The plugin can be installed this way by providing the --plugin-load or the --plugin-load-add options. This can be specified as a command-line argument to mysqld or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
plugin_load_add = auth_pam
```

Installing the v1 Plugin

MariaDB starting with 10.4.0

Starting in MariaDB 10.4.0, the `auth_pam` shared library actually refers to version `2.0` of the `pam` authentication plugin. MariaDB 10.4.0 and later also provides version `1.0` of the plugin as the `auth_pam_v1` shared library.

In MariaDB 10.4.0 and later, if you need to install version `1.0` of the authentication plugin instead of version `2.0`, then you can do so. For example, with INSTALL SONAME or INSTALL PLUGIN:

```
INSTALL SONAME 'auth_pam_v1';
```

Or by specifying in a relevant server option group in an option file:

```
[mariadb]
...
plugin_load_add = auth_pam_v1
```

Uninstalling the Plugin

You can uninstall the plugin dynamically by executing UNINSTALL SONAME or UNINSTALL PLUGIN. For example:

```
UNINSTALL SONAME 'auth_pam';
```

If you installed the plugin by providing the --plugin-load or the --plugin-load-add options in a relevant server option group in an option file, then those options should be removed to prevent the plugin from being loaded the next time the server is restarted.

Uninstalling the v1 Plugin

If you installed version `1.0` of the authentication plugin, then you can uninstall that by executing a similar statement for `auth_pam_v1`. For example:

```
UNINSTALL SONAME 'auth_pam_v1';
```

Configuring PAM

The `pam` authentication plugin tells MariaDB to delegate the authentication to the PAM authentication framework. How exactly that authentication is performed depends on how PAM was configured.

Configuring the PAM Service

PAM is divided into `services`. PAM services are configured by PAM configuration files. Typically, the global PAM configuration file is located at `/etc/pam.conf` and PAM directory-based configuration files for individual services are located in `/etc/pam.d/`.

If you want to use a PAM service called `mariadb` for your MariaDB PAM authentication, then the PAM configuration file for that service would also be called `mariadb`, and it would typically be located at `/etc/pam.d/mariadb`.

For example, here is a minimal PAM service configuration file that performs simple password authentication with UNIX passwords:

```
auth required pam_unix.so audit
account required pam_unix.so audit
```

Let's breakdown this relatively simple PAM service configuration file.

Each line of a PAM service configuration file has the following general format:

type control module-path module-arguments

The above PAM service configuration file instructs the PAM authentication framework that for successful **authentication** (i.e. type=auth), it is **required** that the `pam_unix.so` PAM module returns a success.

The above PAM service configuration file also instructs the PAM authentication framework that for an **account** (i.e. type=account) to be valid, it is **required** that the `pam_unix.so` PAM module returns a success.

PAM also supports session and password types, but MariaDB's `pam` authentication plugin does not support those.

The above PAM service configuration file also provides the `audit` module argument to the `pam_unix` PAM module. The pam_unix manual says that this module argument enables extreme debug logging to the syslog.

On most systems, you can find many other examples of PAM service configuration files in your `/etc/pam.d/` directory.

Configuring the pam_unix PAM Module

If you configure PAM to use the pam_unix PAM module (as in the above example), then you might notice on some systems that this will fail by default with errors like the following:

```
Apr 14 12:56:23 localhost unix_chkpwd[3332]: check pass; user unknown
Apr 14 12:56:23 localhost unix_chkpwd[3332]: password check failed for user (alice)
Apr 14 12:56:23 localhost mysqld: pam_unix(mysql:auth): authentication failure; logname= uid=991
euid=991 tty= ruser= rhost=  user=alice
```

The problem is that on some systems, the `pam_unix` PAM module needs access to `/etc/shadow` in order to function, and most systems only allow `root` to access that file by default.

Newer versions of PAM do not have this limitation, so you may want to try upgrading your version of PAM to see if that fixes the issue.

If that does not work, then you can work around this problem by giving the user that runs mysqld access to `/etc/shadow`. For example, if the `mysql` user runs mysqld, then you could do the following:

```
sudo groupadd shadow
sudo usermod -a -G shadow mysql
sudo chown root:shadow /etc/shadow
sudo chmod g+r /etc/shadow
```

And then you would have to restart the server.

At that point, the server should be able to read `/etc/shadow`.

MariaDB starting with 10.4.0

Starting in MariaDB 10.4.0, the `pam` authentication plugin uses a setuid wrapper to perform its PAM checks, so it should not need any special workarounds to perform privileged operations, such as reading `/etc/shadow` when using the `pam_unix` PAM module. See MDEV-7032 for more information.

Creating Users

Similar to all other authentication plugins, to create a user in MariaDB which uses the `pam` authentication plugin, you would execute CREATE USER while specifying the name of the plugin in the IDENTIFIED VIA clause. For example:

```
CREATE USER username@hostname IDENTIFIED VIA pam;
```

If SQL_MODE does not have `NO_AUTO_CREATE_USER` set, then you can also create the user this way with GRANT. For example:

```
GRANT SELECT ON db.* TO username@hostname IDENTIFIED VIA pam;
```

You can also specify a PAM service name for MariaDB to use by providing it with the `USING` clause. For example:

```
CREATE USER username@hostname IDENTIFIED VIA pam USING 'mariadb';
```

This line creates a user that needs to be authenticated via the `pam` authentication plugin using the PAM service name `mariadb`. As mentioned in a previous section, this service's configuration file will typically be present in `/etc/pam.d/mariadb`.

If no service name is specified, then the plugin will use `mysql` as the default PAM service name.

Client Authentication Plugins

For clients that use the `libmysqlclient` or MariaDB Connector/C libraries, MariaDB provides two client authentication plugins that are compatible with the `pam` authentication plugin:

- `dialog`
- `mysql_clear_password`

When connecting with a client or utility to a server as a user account that authenticates with the `pam` authentication plugin, you may need to tell the client where to find the relevant client authentication plugin by specifying the `--plugin-dir` option. For example:

```
mariadb --plugin-dir=/usr/local/mysql/lib64/mysql/plugin --user=alice
```

Both the `dialog` and the `mysql_clear_password` client authentication plugins transmit the password to the server in clear text. Therefore, when you use the `pam` authentication plugin, it is incredibly important to encrypt client connections using TLS to prevent the clear-text passwords from being seen by unauthorized users.

`dialog`

Usually the `pam` authentication plugin uses the `dialog` client authentication plugin to communicate with the user. This client authentication plugin allows MariaDB to support arbitrarily complex PAM configurations with regular or one-time passwords, challenge-response, multiple questions, or just about anything else. When using a MariaDB client library, there is no need to install or enable anything — the `dialog` client authentication plugin is loaded by the client library completely automatically and transparently for the application.

The `dialog` client authentication plugin was developed by MariaDB, so MySQL's clients and client libraries as well as third party applications that bundle MySQL's client libraries do not support the `dialog` client authentication plugin out of the box. If the server tells an unsupported client to use the `dialog` client authentication plugin, then the client is likely to throw an error like the following:

```
ERROR 2059 (HY000): Authentication plugin 'dialog' cannot be loaded:
/usr/lib/mysql/plugin/dialog.so: cannot open shared object file: No such file or directory
```

For some libraries or applications, this problem can be fixed by copying `dialog.so` or `dialog.dll` from a MariaDB client installation that is compatible with the system into the system's MySQL client authentication plugin directory. However, not all clients are compatible with the `dialog` client authentication plugin, so this may not work for every client.

If your client does not support the `dialog` client authentication plugin, then you may need to use the mysql_clear_password client authentication plugin instead.

The `dialog` client authentication plugin transmits the password to the server in clear text. Therefore, when you use

the `pam` authentication plugin, it is incredibly important to encrypt client connections using TLS to prevent the clear-text passwords from being seen by unauthorized users.

`mysql_clear_password`

Users can instruct the `pam` authentication plugin to use the `mysql_clear_password` client authentication plugin instead of the dialog client authentication plugin by configuring the pam_use_cleartext_plugin system variable on the server. It can be set in a relevant server option group in an option file. For example:

```
[mariadb]
...
pam_use_cleartext_plugin
```

It is important to note that the `mysql_clear_password` plugin has very limited functionality.

- The `mysql_clear_password` client authentication plugin only supports PAM services that require password-based authentication.
- The `mysql_clear_password` client authentication plugin also only supports PAM services that ask the user a single question.
- If the PAM service requires challenge-responses, multiple questions, or other similar complicated authentication schemes, then the PAM service is not compatible with `mysql_clear_password` client authentication plugin. In that case, the dialog client authentication plugin will have to be used instead.

The `mysql_clear_password` client authentication plugin transmits the password to the server in clear text. Therefore, when you use the `pam` authentication plugin, it is incredibly important to encrypt client connections using TLS to prevent the clear-text passwords from being seen by unauthorized users.

Compatiblity with MySQL Clients and Client Libraries

The `mysql_clear_password` client authentication plugin is similar to MySQL's mysql_clear_password client authentication plugin.

The `mysql_clear_password` client authentication plugin is compatible with MySQL clients and most MySQL client libraries, while the dialog client authentication plugin is not always compatible with them. Therefore, the `mysql_clear_password` client authentication plugin is most useful if you need some kind of MySQL compatibility in your environment, but you still want to use the `pam` authentication plugin.

Even though the `mysql_clear_password` client authentication plugin is compatible with MySQL clients and most MySQL client libraries, the `mysql_clear_password` client authentication plugin may be disabled by default by these clients and client libraries. For example, MySQL's version of the mysql command-line client has the --enable-cleartext-plugin option that must be set in order to use the `mysql_clear_password` client authentication plugin. For example:

```
mysql --enable-cleartext-plugin --user=alice -p
```

Other clients may require other methods to enable the authentication plugin. For example, MySQL Workbench has a checkbox titled **Enable Cleartext Authentication Plugin** under the **Advanced** tab on the connection configuration screen.

For applications that use MySQL's `libmysqlclient`, the authentication plugin can be enabled by setting the `MYSQL_ENABLE_CLEARTEXT_PLUGIN` option with the `mysql_options()` function. For example:

```
mysql_options(mysql, MYSQL_ENABLE_CLEARTEXT_PLUGIN, 1);
```

For MySQL compatibility, MariaDB Connector/C also allows applications to set the `MYSQL_ENABLE_CLEARTEXT_PLUGIN` option with the mysql_optionsv function. However, this option does not actually do anything in MariaDB Connector/C, because the `mysql_clear_password` client authentication plugin is always enabled for MariaDB clients and client libraries.

Support in Client Libraries

Using the Plugin with MariaDB Connector/C

MariaDB Connector/C supports `pam` authentication using the client authentication plugins mentioned in the previous section since MariaDB Connector/C 2.1.0, regardless of the value of the pam_use_cleartext_plugin system variable.

Using the Plugin with MariaDB Connector/ODBC

MariaDB Connector/ODBC supports `pam` authentication using the client authentication plugins mentioned in the previous section since MariaDB Connector/ODBC 1.0.0, regardless of the value of the pam_use_cleartext_plugin system variable.

Using the Plugin with MariaDB Connector/J

MariaDB Connector/J supports `pam v1` authentication since MariaDB Connector/J 1.4.0, regardless of the value of the pam_use_cleartext_plugin system variable.

MariaDB Connector/J supports `pam v2` authentication since MariaDB Connector/J 2.4.4, regardless of the value of the pam_use_cleartext_plugin system variable.

Using the Plugin with MariaDB Connector/Node.js

MariaDB Connector/Node.js supports `pam` authentication since MariaDB Connector/Node.js 0.7.0, regardless of the value of the pam_use_cleartext_plugin system variable.

Using the Plugin with MySqlConnector for .NET

MySqlConnector for ADO.NET supports `pam` authentication since MySqlConnector 0.20.0, but only if the pam_use_cleartext_plugin system variable is enabled on the server.

Logging

PAM Module Logging

Errors and messages from PAM modules are usually logged using the syslog daemon with the `authpriv` facility. To determine the specific log file where the `authpriv` facility is logged, you can check rsyslog.conf.

On RHEL, CentOS, Fedora, and other similar Linux distributions, the default location for these messages is usually `/var/log/secure`.

On Debian, Ubuntu, and other similar Linux distributions, the default location for these messages is usually `/var/log/auth.log`.

For example, the syslog can contain messages like the following when MariaDB's `pam` authentication plugin is configured to use the pam_unix PAM module, and the user enters an incorrect password:

```
Jan  9 05:35:41 ip-172-30-0-198 unix_chkpwd[1205]: password check failed for user (foo)
Jan  9 05:35:41 ip-172-30-0-198 mysqld: pam_unix(mariadb:auth): authentication failure; logname=
uid=997 euid=997 tty= ruser= rhost=  user=foo
```

PAM Authentication Plugin's Debug Logging

MariaDB's `pam` authentication plugin can also log additional verbose debug logging to the error log. This is only done if the plugin is a debug build and if pam_debug is set.

The output looks like this:

```
PAM: pam_start(mariadb, alice)
PAM: pam_authenticate(0)
PAM: conv: send(Enter PASSCODE:)
PAM: conv: recv(123456789)
PAM: pam_acct_mgmt(0)
PAM: pam_get_item(PAM_USER)
PAM: status = 0 user = ��\>
```

Custom Logging with pam_exec

The pam_exec PAM module can be used to implement some custom logging. This can be very useful when debugging

certain kinds of issues.

For example, first, create a script that writes the log output:

```
tee /tmp/pam_log_script.sh <<EOF
#!/bin/bash
echo "\${PAM_SERVICE}:\${PAM_TYPE} - \${PAM_RUSER}@\${PAM_RHOST} is authenticating as \${PAM_USER
EOF
chmod 0775 /tmp/pam_log_script.sh
```

And then change the PAM service configuration to execute the script using the pam_exec PAM module. For example:

```
auth optional pam_exec.so log=/tmp/pam_output.txt /tmp/pam_log_script.sh
auth required pam_unix.so audit
account optional pam_exec.so log=/tmp/pam_output.txt /tmp/pam_log_script.sh
account required pam_unix.so audit
```

Whenever the above PAM service is used, the output of the script will be written to `/tmp/pam_output.txt`. It may look similar to this:

```
*** Tue May 14 14:53:23 2019
mariadb:auth - @ is authenticating as alice
*** Tue May 14 14:53:25 2019
mariadb:account - @ is authenticating as alice
*** Tue May 14 14:53:28 2019
mariadb:auth - @ is authenticating as alice
*** Tue May 14 14:53:31 2019
mariadb:account - @ is authenticating as alice
```

User and Group Mapping

Even when using the `pam` authentication plugin, the authenticating PAM user account still needs to exist in MariaDB, and the account needs to have privileges in the database. Creating these MariaDB accounts and making sure the privileges are correct can be a lot of work. To decrease the amount of work involved, some users would like to be able to map a PAM user to a different MariaDB user. For example, let's say that `alice` and `bob` are both DBAs. It would be nice if each of them could log into MariaDB with their own PAM username and password, while MariaDB sees both of them as the same `dba` user. That way, there is only one MariaDB account to keep track of. See User and Group Mapping with PAM for more information on how to do this.

PAM Modules

There are many PAM modules. The ones described below are the ones that have been seen most often by MariaDB.

pam_unix

The pam_unix PAM module provides support for Unix password authentication. It is the default PAM module on most systems.

For a tutorial on setting up PAM authentication and user or group mapping with Unix authentication, see Configuring PAM Authentication and User Mapping with Unix Authentication.

pam_user_map

The pam_user_map PAM module was developed by MariaDB to support user and group mapping.

pam_ldap

The pam_ldap PAM module provides support for LDAP authentication.

For a tutorial on setting up PAM authentication and user or group mapping with LDAP authentication, see Configuring PAM Authentication and User Mapping with LDAP Authentication.

This can also be configured for Active Directory authentication.

pam_sss

The pam_sss PAM module provides support for authentication with System Security Services Daemon (SSSD).

This can be configured for Active Directory authentication.

pam_lsass

The `pam_lsass` PAM module provides support for Active Directory authentication. It is provided by PowerBroker Identity Services – Open Edition.

pam_winbind

The pam_winbind PAM module provides support for Active Directory authentication. It is provided by winbindd from the samba suite.

This PAM module converts all provided user names to lowercase. There is no way to disable this functionality. If you do not want to be forced to use all lowercase user names, then you may need to configure the pam_winbind_workaround system variable. See MDEV-18686 for more information.

pam_centrifydc

The pam_centrifydc PAM module provides support for Active Directory authentication. It integrates with the commercial Active Directory Bridge from Centrify.

pam_krb5

The pam_krb5 PAM module provides support for Kerberos authentication.

This can be configured for Active Directory authentication.

pam_google_authenticator

The `pam_google_authenticator` PAM module provides two-factor identification with Google Authenticator. It is from Google's google-authenticator-libpam open-source project. The PAM module should work with the open-source mobile apps built by Google's google-authenticator and google-authenticator-android projects as well as the the closed source Google Authenticator mobile apps that are present in each mobile app store.

For an example of how to use this PAM module, see the blog post: MariaDB: Improve Security with Two-Step Verification.

pam_securid

The `pam_securid` PAM module provides support for multi-factor authentication. It is part of the commercial RSA SecurID Suite.

Note that current versions of this module are not safe for multi-threaded environments, and the vendor does not officially support the product on MariaDB. See MDEV-10361 about that. However, the module may work with MariaDB 10.4.0 and above.

pam_ssh

The pam_ssh PAM module provides authentication using SSH keys.

pam_time

The pam_time PAM module provides time-controlled access.

Known Issues

Multi-Threaded Issues

MariaDB is a multi-threaded program, which means that different connections concurrently run in different threads. Current versions of MariaDB's `pam` authentication plugin execute PAM module code in the server address space. This means that any PAM modules used with MariaDB must be safe for multi-threaded environments. Otherwise, if multiple clients try to authenticate with the same PAM module in parallel, undefined behavior can occur. For example, the `pam_fprintd` PAM module is not safe for multi-threaded environments, and if you use it with MariaDB, you may experience server crashes.

MariaDB starting with 10.4.0

Starting in MariaDB 10.4.0, the `pam` authentication plugin isolates PAM module code from the server address space, so even PAM modules that are known to be unsafe for multi-threaded environments should not cause issues with MariaDB. See MDEV-15473 for more information.

Conflicts with Password Validation

When a password validation plugin is enabled, MariaDB won't allow an account to be created if the password validation plugin says that the account's password is too weak. This creates a problem for accounts that authenticate with the `pam` authentication plugin, since MariaDB has no knowledge of the user's password. When a user tries to create an account that authenticates with the `pam` authentication plugin, the password validation plugin would throw an error, even with strict_password_validation=OFF set.

The workaround is to uninstall the password validation plugin with UNINSTALL PLUGIN, and then create the account, and then reinstall the password validation plugin with INSTALL PLUGIN.

For example:

```
INSTALL PLUGIN simple_password_check SONAME 'simple_password_check';
Query OK, 0 rows affected (0.002 sec)

SET GLOBAL strict_password_validation=OFF;
Query OK, 0 rows affected (0.000 sec)

CREATE USER ''@'%' IDENTIFIED VIA pam USING 'mariadb';
ERROR 1819 (HY000): Your password does not satisfy the current policy requirements
UNINSTALL PLUGIN simple_password_check;
Query OK, 0 rows affected (0.000 sec)

CREATE USER ''@'%' IDENTIFIED VIA pam USING 'mariadb';
Query OK, 0 rows affected (0.000 sec)

INSTALL PLUGIN simple_password_check SONAME 'simple_password_check';
Query OK, 0 rows affected (0.001 sec)
```

MariaDB starting with 10.4.0

Starting in MariaDB 10.4.0, accounts that authenticate with the `pam` authentication plugin should be exempt from password validation checks. See MDEV-12321 and MDEV-10457 for more information.

SELinux

SELinux may cause issues when using the `pam` authentication plugin. For example, using pam_unix with the `pam` authentication plugin while SELinux is enabled can sometimes lead to SELinux errors involving unix_chkpwd, such as the following::

```
Apr 14 12:37:59 localhost setroubleshoot: Plugin Exception restorecon_source
Apr 14 12:37:59 localhost setroubleshoot: SELinux is preventing /usr/sbin/unix_chkpwd from
execute access on the file . For complete SELinux messages. run sealert -l c56fe6e0-c78c-4bdb-
a80f-27ef86a1ea85
Apr 14 12:37:59 localhost python: SELinux is preventing /usr/sbin/unix_chkpwd from execute
access on the file .

*****  Plugin catchall (100. confidence) suggests   **************************

If you believe that unix_chkpwd should be allowed execute access on the  file by default.
Then you should report this as a bug.
You can generate a local policy module to allow this access.
Do
allow this access for now by executing:
# grep unix_chkpwd /var/log/audit/audit.log | audit2allow -M mypol
# semodule -i mypol.pp
```

Sometimes issues like this can be fixed by updating the system's SELinux policies. You may be able to update the policies using audit2allow. See SELinux: Generating SELinux Policies with audit2allow for more information.

If you can't get the `pam` authentication plugin to work with SELinux at all, then it can help to disable SELinux entirely. See SELinux: Changing SELinux's Mode for information on how to do this.

Memory Overcommit

On MariaDB 10.4 and later you may run into authentication failures with the following log message in the MariaDB error log:

```
pam: cannot exec /usr/lib64/mysql/plugin/auth_pam_tool_dir/auth_pam_tool (errno: 12 "Cannot alloc
```

This can happen on operating system setups that are configured to prevent memory overcommit. When the MariaDB server process spawns the `auth_pam_tool` helper process there's a brief period where the new process inherits the memory of the MariaDB process before releasing that memory and executing the new command. When having a MariaDB server configured to use more than 50% of the server machnines RAM -- which is common for dedicated database servers -- this duplication would lead to an over-commit situation.

Starting with MariaDB 10.4.25, MariaDB 10.5.16, MariaDB 10.6.8, and MariaDB 10.7.4, we changed to use `posix_spawn()` instead of the classic `fork();exec()` to prevent this, but systems with older glibc versions prior to 2.26 still use `fork();exec()` to implement `posix_spawn()` internally and so are still affected; this is for example still the case on RedHat Enterprise Linux 7.

To solve this you can either:

- change the `vm.overcommit_memory` kernel setting to allow memory overcommit
- install the older `auth_pam_v1` plugin version that does not spawn a helper process (but may run into problems with file permissions or multi threading with some PAM modules)

See also MDEV-26212 and MDEV-30734

Tutorials

You may find the following PAM-related tutorials helpful:

- Configuring PAM Authentication and User Mapping with Unix Authentication
- Configuring PAM Authentication and User Mapping with LDAP Authentication

Versions

Version	Status	Introduced
2.0	Beta	MariaDB 10.4.0
1.0	Stable	MariaDB 10.0.10
1.0	Beta	MariaDB 5.2.10

System Variables

`pam_debug`

- **Description:** Enables verbose debug logging to the error log for all authentication handled by the plugin.
  - This system variable is only available when the plugin is a debug build.
- **Commandline:** `--pam-debug`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB 10.2.2, MariaDB 10.1.17

`pam_use_cleartext_plugin`

- **Description:** Use the mysql_clear_password client authentication plugin instead of the dialog client authentication plugin. This may be needed for compatibility reasons, but it only supports simple PAM configurations that don't require any input besides a password.
- **Commandline:** `--pam-use-cleartext-plugin`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`

- **Default Value:** `OFF`
- **Introduced:** MariaDB 10.1.1, MariaDB 5.5.32

`pam_winbind_workaround`

- **Description:** Configures the authentication plugin to compare the user name provided by the client with the user name returned by the PAM module in a case insensitive manner. This may be needed if you use the pam_winbind PAM module, which is known to convert all user names to lowercase, and which does not allow this behavior to be disabled.
- **Commandline:** `--pam-winbind-workaround`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB 10.4.5, MariaDB 10.3.15, MariaDB 10.2.24, MariaDB 10.1.39

Options

`pam`

- **Description:** Controls how the server should treat the plugin when the server starts up.
  - Valid values are:
    - `OFF` - Disables the plugin without removing it from the mysql.plugins table.
    - `ON` - Enables the plugin. If the plugin cannot be initialized, then the server will still continue starting up, but the plugin will be disabled.
    - `FORCE` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error.
    - `FORCE_PLUS_PERMANENT` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error. In addition, the plugin cannot be uninstalled with UNINSTALL SONAME or UNINSTALL PLUGIN while the server is running.
  - See Plugin Overview: Configuring Plugin Activation at Server Startup for more information.
- **Commandline:** `--pam=value`
- **Data Type:** `enumerated`
- **Default Value:** `ON`
- **Valid Values:** `OFF`, `ON`, `FORCE`, `FORCE_PLUS_PERMANENT`

5.4.6.6.2 User and Group Mapping with PAM

Even when using the `pam` authentication plugin, the authenticating PAM user account still needs to exist in MariaDB, and the account needs to have privileges in the database. Creating these MariaDB accounts and making sure the privileges are correct can be a lot of work. To decrease the amount of work involved, some users would like to be able to map a PAM user to a different MariaDB user. For example, let's say that `alice` and `bob` are both DBAs. It would be nice if each of them could log into MariaDB with their own PAM username and password, while MariaDB sees both of them as the same `dba` user. That way, there is only one MariaDB account to keep track of.

Although most PAM modules usually do not do things like this, PAM supports the ability to change the user name in the process of authentication.The MariaDB `pam` authentication plugin fully supports this feature of PAM.

Contents

The pam_user_map PAM Module

Rather than building user and group mapping into the `pam` authentication plugin, MariaDB thought that it would cover the most use cases and offer the most flexibility to offload this functionality to an external PAM module. The `pam_user_map` PAM module was implemented by MariaDB to facilitate this. This PAM module can be configured in the PAM service used by the `pam` authentication plugin, just like other PAM modules.

Lack of Support for MySQL/Percona Group Mapping Syntax

Unlike MariaDB, MySQL and Percona implemented group mapping in their PAM authentication plugins. If you've read through MySQL's PAM authentication documentation on group mapping or Percona's PAM authentication documentation on group mapping, you've probably seen syntax where the group mappings are provided in the `CREATE USER` statement like this:

```
CREATE USER ''@''
  IDENTIFIED WITH authentication_pam
  AS 'mysql, root=developer, users=data_entry';
```

Since MariaDB's user and group mapping is performed by an external PAM module, MariaDB's `pam` authentication plugin does not support this syntax. Instead, the user and group mappings for the `pam_user_map` PAM module are configured in an external configuration file. This is discussed in a later section.

Installing the pam_user_map PAM Module

The `pam_user_map` PAM module gets installed as part of all our MariaDB server packages since MariaDB 10.5, and was added since 10.2.31, 10.3.22, and 10.4.12 in previous MariaDB major releases where it was not present from the beginning.

Some Linux distributions have not picked up this change in their own packages yet, so when e.g. installing MariaDB server from stock Ubuntu packages on Ubuntu 20.04LTS you still won't have the `pam_user_map` module installed even though the MariaDB server installed is more recent than MariaDB 10.3.22.

When using such an installation, and not being able to switch to our own MariaDB package repositories, it may be necessary to compile the PAM module from source as described in the next section, or to manually extract it from one of our server packages and copy it to the target system.

Installing the pam_user_map PAM Module from Source

Installing Compilation Dependencies

Before the module can be compiled from source, you may need to install some dependencies.

On RHEL, CentOS, and other similar Linux distributions that use RPM packages, you need to install `gcc`, `pam-devel` and `MariaDB-devel`. For example:

```
sudo yum install gcc pam-devel MariaDB-devel
```

On Debian, Ubuntu, and other similar Linux distributions that use DEB packages, you need to install `gcc`, `libpam0g-dev`. For example:

```
sudo apt-get install gcc libpam0g-dev libmariadb-dev
```

Compiling and Installing the pam_user_map PAM Module

The `pam_user_map` PAM module can be built by downloading `plugin/auth_pam/mapper/pam_user_map.c` file from the MariaDB source tree and compiling it after minor adjustments. Once it is built, it can be installed to the system's PAM module directory, which is typically `/lib64/security/` .

For example: (replace 10.4 in the URL with the actual server versions)

```
wget https://raw.githubusercontent.com/MariaDB/server/10.4/plugin/auth_pam/mapper/pam_user_map.c
sed -ie 's/config_auth_pam/plugin_auth_common/' pam_user_map.c
gcc -I/usr/include/mysql/ pam_user_map.c -shared -lpam -fPIC -o pam_user_map.so
sudo install --mode=0755 pam_user_map.so /lib64/security/
```

You will also need to adjust the major version number in the URL on the first line to match your installed MariaDB version, and the #-I# include path argument on the `gcc` line, as depending on operating system and MariaDB server version the plugin_auth_common.h file may be installed in different directories than `/usr/include/mysql/`

Configuring the pam_user_map PAM Module

The `pam_user_map` PAM module uses the configuration file at the path `/etc/security/user_map.conf` to determine its user and group mappings. The file's format is described below.

To map a specific PAM user to a specific MariaDB user:

```
orig_pam_user_name: mapped_mariadb_user_name
```

Or to map any PAM user in a specific PAM group to a specific MariaDB user, the group name is prefixed with `@` :

```
@orig_pam_group_name: mapped_mariadb_user_name
```

For example, here is an example `/etc/security/user_map.conf` :

```
============================================================
#comments and empty lines are ignored
john: jack
bob:  admin
top:  accounting
@group_ro: readonly
```

Configuring PAM

With user and group mapping, configuring PAM is done similar to how it is normally done with the `pam` authentication plugin. However, when configuring the PAM service, you will have to add an `auth` line for the `pam_user_map` PAM module to the service's PAM configuration file. For example:

```
auth required pam_unix.so audit
auth required pam_user_map.so
account required pam_unix.so audit
```

Creating Users

With user and group mapping, creating users is done similar to how it is normally done with the `pam` authentication plugin. However, one major difference is that you will need to `GRANT` the `PROXY` privilege on the mapped user to the original user.

For example, if you have the following configured in `/etc/security/user_map.conf` :

```
foo: bar
@dba:dba
```

Then you could execute the following to grant the relevant privileges:

```
CREATE USER 'bar'@'%' IDENTIFIED BY 'strongpassword';
GRANT ALL PRIVILEGES ON *.* TO 'bar'@'%' ;

CREATE USER 'dba'@'%' IDENTIFIED BY 'strongpassword';
GRANT ALL PRIVILEGES ON *.* TO 'dba'@'%' ;

CREATE USER ''@'%' IDENTIFIED VIA pam USING 'mariadb';
GRANT PROXY ON 'bar'@'%' TO ''@'%';
GRANT PROXY ON 'dba'@'%' TO ''@'%';
```

Note that the `''@'%'` account is a special catch-all anonymous account. Any login by a user that has no more specific account match in the system will be matched by this anonymous account.

Also note that you might not be able to create the `''@'%'` anonymous account by default on some systems without doing some extra steps first. See Fixing a Legacy Default Anonymous Account for more information.

Verifying that Mapping is Occurring

In case any user mapping is performed, the original user name is returned by the SQL function `USER()`, while the authenticated user name is returned by the SQL function `CURRENT_USER()`. The latter actually defines what privileges are available to a connected user.

For example, if we have the following configured:

```
foo: bar
```

Then the following output would verify that it is working properly:

```
$ mysql -u foo -h 172.30.0.198
[mariadb] Password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 22
Server version: 10.3.10-MariaDB MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> SELECT USER(), CURRENT_USER();
+-----------------------------------------------+----------------+
| USER()                                        | CURRENT_USER() |
+-----------------------------------------------+----------------+
| foo@ip-172-30-0-198.us-west-2.compute.internal | bar@%          |
+-----------------------------------------------+----------------+
1 row in set (0.000 sec)
```

We can verify that our `foo` PAM user was properly mapped to the `bar` MariaDB user by looking at the return value of `CURRENT_USER()`.

Logging

By default, the `pam_user_map` PAM module does not perform any logging. However, if you want to enable debug logging, then you can add the `debug` module argument to the service's PAM configuration file. For example:

```
auth required pam_unix.so audit
auth required pam_user_map.so debug
account required pam_unix.so audit
```

When debug logging is enabled, the `pam_user_map` PAM module will write log entries to the same syslog location as other PAM modules, which is typically `/var/log/secure` on many systems.

For example, this debug log output can look like the following:

```
Jan  9 05:42:13 ip-172-30-0-198 mysqld: pam_user_map(mariadb:auth): Opening file
'/etc/security/user_map.conf'.
Jan  9 05:42:13 ip-172-30-0-198 mysqld: pam_user_map(mariadb:auth): Incoming username 'alice'.
Jan  9 05:42:13 ip-172-30-0-198 mysqld: pam_user_map(mariadb:auth): User belongs to 2 groups
[alice,dba].
Jan  9 05:42:13 ip-172-30-0-198 mysqld: pam_user_map(mariadb:auth): Check if user is in group
'dba': YES
Jan  9 05:42:13 ip-172-30-0-198 mysqld: pam_user_map(mariadb:auth): User mapped as 'dba'
Jan  9 05:43:36 ip-172-30-0-198 mysqld: pam_user_map(mariadb:auth): Opening file
'/etc/security/user_map.conf'.
Jan  9 05:43:36 ip-172-30-0-198 mysqld: pam_user_map(mariadb:auth): Incoming username 'bob'.
Jan  9 05:43:36 ip-172-30-0-198 mysqld: pam_user_map(mariadb:auth): User belongs to 2 groups
[bob,dba].
Jan  9 05:43:36 ip-172-30-0-198 mysqld: pam_user_map(mariadb:auth): Check if user is in group
'dba': YES
Jan  9 05:43:36 ip-172-30-0-198 mysqld: pam_user_map(mariadb:auth): User mapped as 'dba'
Jan  9 06:08:36 ip-172-30-0-198 mysqld: pam_user_map(mariadb:auth): Opening file
'/etc/security/user_map.conf'.
Jan  9 06:08:36 ip-172-30-0-198 mysqld: pam_user_map(mariadb:auth): Incoming username 'foo'.
Jan  9 06:08:36 ip-172-30-0-198 mysqld: pam_user_map(mariadb:auth): User belongs to 1 group
[foo].
Jan  9 06:08:36 ip-172-30-0-198 mysqld: pam_user_map(mariadb:auth): Check if user is in group
'dba': NO
Jan  9 06:08:36 ip-172-30-0-198 mysqld: pam_user_map(mariadb:auth): Check if username 'foo': YES
Jan  9 06:08:36 ip-172-30-0-198 mysqld: pam_user_map(mariadb:auth): User mapped as 'bar'
```

Known Issues

PAM User with Same Name as Mapped MariaDB User Must Exist

With user and group mapping, any PAM user or any PAM user in a given PAM group can be mapped to a specific MariaDB user account. However, due to the way PAM works, a PAM user with the same name as the mapped MariaDB user account must exist.

For example, if the configuration file for the PAM service file contained the following:

```
auth required pam_sss.so
auth required pam_user_map.so debug
account sufficient pam_unix.so
account sufficient pam_sss.so
```

And if `/etc/security/user_map.conf` contained the following:

```
@dba: dba
```

Then any PAM user in the PAM group `dba` would be mapped to the MariaDB user account `dba`. But if a PAM user with the name `dba` did not also exist, then the `pam_user_map` PAM module's debug logging would write errors to the syslog like the following:

```
Sep 27 17:17:05 dbserver1 mysqld: pam_user_map(mysql:auth): Opening file
'/etc/security/user_map.conf'.
Sep 27 17:17:05 dbserver1 mysqld: pam_user_map(mysql:auth): Incoming username 'alice'.
Sep 27 17:17:05 dbserver1 mysqld: pam_user_map(mysql:auth): User belongs to 4 groups
[dba,mongod,mongodba,mysql].
Sep 27 17:17:05 dbserver1 mysqld: pam_user_map(mysql:auth): Check if user is in group 'mysql':
YES
Sep 27 17:17:05 dbserver1 mysqld: pam_user_map(mysql:auth): User mapped as 'dba'
Sep 27 17:17:05 dbserver1 mysqld: pam_unix(mysql:account): could not identify user (from
getpwnam(dba))
Sep 27 17:17:05 dbserver1 mysqld: pam_sss(mysql:account): Access denied for user dba: 10 (User
not known to the underlying authentication module)
Sep 27 17:17:05 dbserver1 mysqld: 2018-09-27 17:17:05 72 [Warning] Access denied for user
'alice'@'localhost' (using password: NO)
```

In the above log snippet, notice that both the `pam_unix` and the `pam_sss` PAM modules are complaining that the `dba` PAM user does not appear to exist, and that these complaints cause the PAM authentication process to fail, which causes the MariaDB authentication process to fail as well.

This can be fixed by creating a PAM user with the same name as the mapped MariaDB user account, which is `dba` in this case.

You may also be able to work around this problem by essentially disabling PAM's account verification for the service with the `pam_permit` PAM module. For example, in the above case, that would be:

```
auth required pam_sss.so
auth required pam_user_map.so debug
account required pam_permit.so
```

See MDEV-17315 for more information.

Tutorials

You may find the following PAM and user mapping-related tutorials helpful:

- Configuring PAM Authentication and User Mapping with Unix Authentication
- Configuring PAM Authentication and User Mapping with LDAP Authentication

5.4.6.6.3 Configuring PAM Authentication and User Mapping with Unix Authentication

In this article, we will walk through the configuration of PAM authentication using the `pam` authentication plugin and user and group mapping with the `pam_user_map` PAM module. The primary authentication will be handled by the `pam_unix` PAM module, which performs standard Unix password authentication.

Contents

Hypothetical Requirements

In this walkthrough, we are going to assume the following hypothetical requirements:

- The Unix user `foo` should be mapped to the MariaDB user `bar`. (`foo: bar`)
- Any Unix user in the Unix group `dba` should be mapped to the MariaDB user `dba`. (`@dba: dba`)

Creating Our Unix Users and Groups

Let's go ahead and create the Unix users and groups that we are using for this hypothetical scenario.

First, let's create the the `foo` user and a couple users to go into the `dba` group. Note that each of these users needs a password.

```
sudo useradd foo
sudo passwd foo
sudo useradd alice
sudo passwd alice
sudo useradd bob
sudo passwd bob
```

And then let's create our `dba` group and add our two users to it:

```
sudo groupadd dba
sudo usermod -a -G dba alice
sudo usermod -a -G dba bob
```

We also need to create Unix users with the same name as the `bar` and `dba` MariaDB users. See here to read more about why. No one will be logging in as these users, so they do not need passwords.

```
sudo useradd bar
sudo useradd dba -g dba
```

Installing the pam_user_map PAM Module

Next, let's install the pam_user_map PAM module.

Before the module can be compiled from source, we may need to install some dependencies.

On RHEL, CentOS, and other similar Linux distributions that use RPM packages, we need to install `gcc` and `pam-devel`:

```
sudo yum install gcc pam-devel
```

On Debian, Ubuntu, and other similar Linux distributions that use DEB packages, we need to install `gcc` and `libpam0g-dev`:

```
sudo apt-get install gcc libpam0g-dev
```

And then we can build and install the library with the following:

```
wget https://raw.githubusercontent.com/MariaDB/server/10.4/plugin/auth_pam/mapper/pam_user_map.c
gcc pam_user_map.c -shared -lpam -fPIC -o pam_user_map.so
sudo install --mode=0755 pam_user_map.so /lib64/security/
```

Configuring the pam_user_map PAM Module

Next, let's configure the pam_user_map PAM module based on our hypothetical requirements.

The configuration file for the `pam_user_map` PAM module is `/etc/security/user_map.conf`. Based on our hypothetical requirements, ours would look like:

```
foo: bar
@dba:dba
```

Installing the PAM Authentication Plugin

Next, let's install the `pam` authentication plugin.

Log into the MariaDB Server and execute the following:

```
INSTALL SONAME 'auth_pam';
```

Configuring the PAM Service

Next, let's configure the PAM service. We will call our service `mariadb`, so our PAM service configuration file will be located at `/etc/pam.d/mariadb` on most systems.

Since we are only doing Unix authentication with the `pam_unix` PAM module and group mapping with the `pam_user_map` PAM module, our configuration file would look like this:

```
auth required pam_unix.so audit
auth required pam_user_map.so
account required pam_unix.so audit
```

Configuring the pam_unix PAM Module

The `pam_unix` PAM module adds some additional configuration steps on a lot of systems. We basically have to give the user that runs `mysqld` access to `/etc/shadow`.

If the `mysql` user is running `mysqld`, then we can do that by executing the following:

```
sudo groupadd shadow
sudo usermod -a -G shadow mysql
sudo chown root:shadow /etc/shadow
sudo chmod g+r /etc/shadow
```

The server needs to be restarted for this change to take affect.

Creating MariaDB Users

Next, let's create the MariaDB users. Remember that our PAM service is called `mariadb`.

First, let's create the MariaDB user for the user mapping: `foo: bar`

That means that we need to create a `bar` user:

```
CREATE USER 'bar'@'%' IDENTIFIED BY 'strongpassword';
GRANT ALL PRIVILEGES ON *.* TO 'bar'@'%' ;
```

And then let's create the MariaDB user for the group mapping: `@dba: dba`

That means that we need to create a `dba` user:

```
CREATE USER 'dba'@'%' IDENTIFIED BY 'strongpassword';
GRANT ALL PRIVILEGES ON *.* TO 'dba'@'%' ;
```

And then to allow for the user and group mapping, we need to create an anonymous user that authenticates with the `pam` authentication plugin that is also able to `PROXY` as the `bar` and `dba` users. Before we can create the proxy user, we might need to clean up some defaults:

```
DELETE FROM mysql.db WHERE User='' AND Host='%';
FLUSH PRIVILEGES;
```

And then let's create the anonymous proxy user:

```
CREATE USER ''@'%' IDENTIFIED VIA pam USING 'mariadb';
GRANT PROXY ON 'bar'@'%' TO ''@'%';
GRANT PROXY ON 'dba'@'%' TO ''@'%';
```

Testing our Configuration

Next, let's test out our configuration by verifying that mapping is occurring. We can verify this by logging in as each of our users and comparing the return value of `USER()`, which is the original user name and the return value of `CURRENT_USER()`, which is the authenticated user name.

First, let's test out our `foo` user:

```
$ mysql -u foo -h 172.30.0.198
[mariadb] Password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 22
Server version: 10.3.10-MariaDB MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> SELECT USER(), CURRENT_USER();
+-----------------------------------------------+----------------+
| USER()                                        | CURRENT_USER() |
+-----------------------------------------------+----------------+
| foo@ip-172-30-0-198.us-west-2.compute.internal | bar@%          |
+-----------------------------------------------+----------------+
1 row in set (0.000 sec)
```

We can verify that our `foo` Unix user was properly mapped to the `bar` MariaDB user by looking at the return value of `CURRENT_USER()`.

Then let's test out our `alice` user in the `dba` group:

```
$ mysql -u alice -h 172.30.0.198
[mariadb] Password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 19
Server version: 10.3.10-MariaDB MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> SELECT USER(), CURRENT_USER();
+---------------------------------------------------+----------------+
| USER()                                            | CURRENT_USER() |
+---------------------------------------------------+----------------+
| alice@ip-172-30-0-198.us-west-2.compute.internal | dba@%          |
+---------------------------------------------------+----------------+
1 row in set (0.000 sec)
```

And then let's test out our `bob` user in the `dba` group:

```
$ mysql -u bob -h 172.30.0.198
[mariadb] Password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 20
Server version: 10.3.10-MariaDB MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> SELECT USER(), CURRENT_USER();
+-------------------------------------------------+----------------+
| USER()                                          | CURRENT_USER() |
+-------------------------------------------------+----------------+
| bob@ip-172-30-0-198.us-west-2.compute.internal | dba@%          |
+-------------------------------------------------+----------------+
1 row in set (0.000 sec)
```

We can verify that our `alice` and `bob` Unix users in the `dba` Unix group were properly mapped to the `dba` MariaDB user by looking at the return values of `CURRENT_USER()`.

5.4.6.6.4 Configuring PAM Authentication and User Mapping with LDAP Authentication

In this article, we will walk through the configuration of PAM authentication using the `pam` authentication plugin and user and group mapping with the `pam_user_map` PAM module. The primary authentication will be handled by the `pam_ldap` PAM module, which performs LDAP authentication. We will also set up an OpenLDAP server.

Contents

Hypothetical Requirements

In this walkthrough, we are going to assume the following hypothetical requirements:

- The LDAP user `foo` should be mapped to the MariaDB user `bar`. (`foo: bar`)
- Any LDAP user in the LDAP group `dba` should be mapped to the MariaDB user `dba`. (`@dba: dba`)

Setting up the OpenLDAP Server

Before we can use LDAP authentication, we first need to set up our OpenLDAP Server. This is usually done on a server that is completely separate from the database server.

Installing the OpenLDAP Server and Client Components

On the server acting as the OpenLDAP Server, first, we need to install the OpenLDAP components.

On RHEL, CentOS, and other similar Linux distributions that use RPM packages, that would go like this:

```
sudo yum install openldap openldap-servers openldap-clients nss-pam-ldapd
```

Configuring the OpenLDAP Server

Next, let's to configure the OpenLDAP Server. The easiest way to do that is to copy the template configuration file that is included with the installation. In many installations, that will be at `/usr/share/openldap-servers/DB_CONFIG.example`. For example:

```
sudo cp /usr/share/openldap-servers/DB_CONFIG.example /var/lib/ldap/DB_CONFIG
sudo chown ldap. /var/lib/ldap/DB_CONFIG
```

Configuring the OpenLDAP Port

Sometimes it is useful to change the port used by OpenLDAP. For example, some cloud environments block well-known authentication services, so they block the default LDAP port.

On some systems, the port can be changed by setting `SLAPD_URLS` in `/etc/sysconfig/slapd`:

```
SLAPD_URLS="ldapi:/// ldap://0.0.0.0:3306/"
```

I used `3306` because that is the port that is usually used by `mysqld`, so I know that it is not blocked.

Starting the OpenLDAP Server

Next, let's start the OpenLDAP Server and configure it to start on reboot. On `systemd` systems, that would go like this:

```
sudo systemctl start slapd
sudo systemctl enable slapd
```

Installing the Standard LDAP objectClasses

In order to use LDAP for authentication, we also need to install some standard `objectClasses`, such as `posixAccount` and `posixGroup`. In LDAP, things like `objectClasses` are defined in `LDIF` files. In many installations, these specific `objectClasses` are defined in `/etc/openldap/schema/nis.ldif`. `nis.ldif` also depends on `core.ldif` and `cosine.ldif`. However, `core.ldif` is usually installed by default.

We can install them with `ldapmodify`:

```
sudo ldapmodify -a -Y EXTERNAL -H ldapi:/// -f /etc/openldap/schema/cosine.ldif
sudo ldapmodify -a -Y EXTERNAL -H ldapi:/// -f /etc/openldap/schema/nis.ldif
```

Creating the LDAP Directory Manager User

Next, let's create a directory manager user. We can do this by using OpenLDAP's olc configuration system to change the `olcRootDN` directive to the DN of the directory manager user, which means that the user will be a privileged LDAP user that is not subject to access controls. We will also set the root password for the user by changing the `olcRootPW` directive.

We will also set the DN suffix for our backend LDAP database by changing the `olcSuffix` directive.

Let's use the `slappasswd` utility to generate a password hash from a clear-text password. Simply execute:

```
slappasswd
```

This utility should provide a password hash that looks kind of like this: `{SSHA}AwT4jrvmokeCkbDrFAnGvzzjCMb7bvEl`

OpenLDAP's olc configuration system also uses `LDIF` files. Now that we have the password hash, let's create an `LDIF` file to create the directory manager user:

```
tee ~/setupDirectoryManager.ldif <<EOF
dn: olcDatabase={1}monitor,cn=config
changetype: modify
replace: olcAccess
olcAccess: {0}to *
    by dn.base="gidNumber=0+uidNumber=0,cn=peercred,cn=external,cn=auth" read
    by dn.base="cn=Manager,dc=support,dc=mariadb,dc=com" read
    by * none

dn: olcDatabase={2}hdb,cn=config
changetype: modify
replace: olcSuffix
olcSuffix: dc=support,dc=mariadb,dc=com

dn: olcDatabase={2}hdb,cn=config
changetype: modify
replace: olcRootDN
olcRootDN: cn=Manager,dc=support,dc=mariadb,dc=com

dn: olcDatabase={2}hdb,cn=config
changetype: modify
add: olcRootPW
olcRootPW: {SSHA}AwT4jrvmokeCkbDrFAnGvzzjCMb7bvEl

dn: olcDatabase={2}hdb,cn=config
changetype: modify
add: olcAccess
olcAccess: {0}to attrs=userPassword,shadowLastChange
    by   dn="cn=Manager,dc=support,dc=mariadb,dc=com" write
    by anonymous auth
    by self write
    by * none
olcAccess: {1}to dn.base=""
    by * read
olcAccess: {2}to *
    by dn="cn=Manager,dc=support,dc=mariadb,dc=com" write
    by * read
EOF
```

Note that I am using the `dc=support,dc=mariadb,dc=com` domain for my directory. You can change this to whatever is relevant to you.

Now let's run the ldif file with `ldapmodify` :

```
sudo ldapmodify -Y EXTERNAL -H ldapi:/// -f ~/setupDirectoryManager.ldif
```

We will use the new directory manager user to make changes to the LDAP directory after this step.

Creating the Structure of the Directory

Next, let's create the structure of the directory by creating parts of our tree.

```
tee ~/setupDirectoryStructure.ldif <<EOF
dn: dc=support,dc=mariadb,dc=com
objectClass: top
objectClass: dcObject
objectclass: organization
o: MariaDB Support Team
dc: support

dn: cn=Manager,dc=support,dc=mariadb,dc=com
objectClass: top
objectClass: organizationalRole
cn: Manager
description: Directory Manager

dn: ou=People,dc=support,dc=mariadb,dc=com
objectClass: top
objectClass: organizationalUnit
ou: People

dn: ou=Groups,dc=support,dc=mariadb,dc=com
objectClass: top
objectClass: organizationalUnit
ou: Groups

dn: ou=System Users,dc=support,dc=mariadb,dc=com
objectClass: top
objectClass: organizationalUnit
ou: System Users
EOF
```

Now let's use our new directory manager user and run the `LDIF` file with `ldapmodify`:

```
ldapmodify -a -x -D cn=Manager,dc=support,dc=mariadb,dc=com -W -f ~/setupDirectoryStructure.ldif
```

Creating the LDAP Users and Groups

Let's go ahead and create the LDAP users and groups that we are using for this hypothetical scenario.

First, let's create the the `foo` user:

```
tee ~/createFooUser.ldif <<EOF
dn: uid=foo,ou=People,dc=support,dc=mariadb,dc=com
objectClass: top
objectClass: account
objectClass: posixAccount
objectClass: shadowAccount
cn: foo
uid: foo
uidNumber: 16859
gidNumber: 100
homeDirectory: /home/foo
loginShell: /bin/bash
gecos: foo
userPassword: {crypt}x
shadowLastChange: -1
shadowMax: -1
shadowWarning: 0
EOF
ldapmodify -a -x -D cn=Manager,dc=support,dc=mariadb,dc=com -W -f ~/createFooUser.ldif
```

Then let's create a couple users to go into the `dba` group:

```
tee ~/createDbaUsers.ldif <<EOF
dn: uid=gmontee,ou=People,dc=support,dc=mariadb,dc=com
objectClass: top
objectClass: account
objectClass: posixAccount
objectClass: shadowAccount
cn: gmontee
uid: gmontee
uidNumber: 16860
gidNumber: 100
homeDirectory: /home/gmontee
loginShell: /bin/bash
gecos: gmontee
userPassword: {crypt}x
shadowLastChange: -1
shadowMax: -1
shadowWarning: 0

dn: uid=bstillman,ou=People,dc=support,dc=mariadb,dc=com
objectClass: top
objectClass: account
objectClass: posixAccount
objectClass: shadowAccount
cn: bstillman
uid: bstillman
uidNumber: 16861
gidNumber: 100
homeDirectory: /home/bstillman
loginShell: /bin/bash
gecos: bstillman
userPassword: {crypt}x
shadowLastChange: -1
shadowMax: -1
shadowWarning: 0
EOF
ldapmodify -a -x -D cn=Manager,dc=support,dc=mariadb,dc=com -W -f ~/createDbaUsers.ldif
```

Note that each of these users needs a password, so we can set it for each user with `ldappasswd`:

```
ldappasswd -x -D cn=Manager,dc=support,dc=mariadb,dc=com -W -S
uid=foo,ou=People,dc=support,dc=mariadb,dc=com
ldappasswd -x -D cn=Manager,dc=support,dc=mariadb,dc=com -W -S
uid=gmontee,ou=People,dc=support,dc=mariadb,dc=com
ldappasswd -x -D cn=Manager,dc=support,dc=mariadb,dc=com -W -S
uid=bstillman,ou=People,dc=support,dc=mariadb,dc=com
```

And then let's create our `dba` group

```
tee ~/createDbaGroup.ldif <<EOF
dn: cn=dba,ou=Groups,dc=support,dc=mariadb,dc=com
objectClass: top
objectClass: posixGroup
gidNumber: 678
EOF
ldapmodify -a -x -D cn=Manager,dc=support,dc=mariadb,dc=com -W -f ~/createDbaGroup.ldif
```

And then let's add our two users to it:

```
tee ~/addUsersToDbaGroup.ldif <<EOF
dn: cn=dba,ou=Groups,dc=support,dc=mariadb,dc=com
changetype: modify
add: memberuid
memberuid: gmontee

dn: cn=dba,ou=Groups,dc=support,dc=mariadb,dc=com
changetype: modify
add: memberuid
memberuid: bstillman
EOF
ldapmodify -a -x -D cn=Manager,dc=support,dc=mariadb,dc=com -W -f ~/addUsersToDbaGroup.ldif
```

We also need to create LDAP users with the same name as the `bar` and `dba` MariaDB users. See here to read more about why. No one will be logging in as these users, so they do not need passwords. Instead of the `People organizationalUnit`, we will create them in the `System Users organizationalUnit`.

```
tee ~/createSystemUsers.ldif <<EOF
dn: uid=bar,ou=System Users,dc=support,dc=mariadb,dc=com
objectClass: top
objectClass: account
objectClass: posixAccount
objectClass: shadowAccount
cn: bar
uid: bar
uidNumber: 16862
gidNumber: 100
homeDirectory: /home/bar
loginShell: /bin/bash
gecos: bar
userPassword: {crypt}x
shadowLastChange: -1
shadowMax: -1
shadowWarning: 0

dn: uid=dba,ou=System Users,dc=support,dc=mariadb,dc=com
objectClass: top
objectClass: account
objectClass: posixAccount
objectClass: shadowAccount
cn: dba
uid: dba
uidNumber: 16863
gidNumber: 100
homeDirectory: /home/dba
loginShell: /bin/bash
gecos: dba
userPassword: {crypt}x
shadowLastChange: -1
shadowMax: -1
shadowWarning: 0
EOF
ldapmodify -a -x -D cn=Manager,dc=support,dc=mariadb,dc=com -W -f ~/createSystemUsers.ldif
```

Setting up the MariaDB Server

At this point, we can move onto setting up the MariaDB Server.

Installing LDAP and PAM Libraries

First, we need to make sure that the LDAP and PAM libraries are installed.

On RHEL, CentOS, and other similar Linux distributions that use RPM packages, we need to install the following packages:

```
sudo yum install openldap-clients nss-pam-ldapd pam pam-devel
```

Configuring LDAP

Next, let's configure LDAP on the system. We can use `authconfig` for this:

```
sudo authconfig --enableldap \
   --enableldapauth \
   --ldapserver="ldap://172.30.0.238:3306" \
   --ldapbasedn="dc=support,dc=mariadb,dc=com" \
   --enablemkhomedir \
   --update
```

Be sure to replace `--ldapserver` and `--ldapbasedn` with values that are relevant for your environment.

Installing the pam_user_map PAM Module

The following steps apply to MariaDB Server in versions 10.2.32.7, 10.3.23., 10.4.13.6, 10.5.2 and earlier. In later releases, the pam_user_map PAM module is now included in the base install.

Next, let's install the pam_user_map PAM module.

Before the module can be compiled from source, we may need to install some dependencies.

On RHEL, CentOS, and other similar Linux distributions that use RPM packages, we need to install `gcc` and `pam-devel`:

```
sudo yum install gcc pam-devel
```

On Debian, Ubuntu, and other similar Linux distributions that use DEB packages, we need to install `gcc` and `libpam0g-dev`:

```
sudo apt-get install gcc libpam0g-dev
```

And then we can build and install the library with the following:

```
wget https://raw.githubusercontent.com/MariaDB/server/10.4/plugin/auth_pam/mapper/pam_user_map.c
gcc pam_user_map.c -shared -lpam -fPIC -o pam_user_map.so
sudo install --mode=0755 pam_user_map.so /lib64/security/
```

Configuring the pam_user_map PAM Module

Next, let's configure the pam_user_map PAM module based on our hypothetical requirements.

The configuration file for the `pam_user_map` PAM module is `/etc/security/user_map.conf`. Based on our hypothetical requirements, ours would look like:

```
foo: bar
@dba:dba
```

Installing the PAM Authentication Plugin

Next, let's install the `pam` authentication plugin.

Log into the MariaDB Server and execute the following:

```
INSTALL SONAME 'auth_pam';
```

Configuring the PAM Service

Next, let's configure the PAM service. We will call our service `mariadb`, so our PAM service configuration file will be located at `/etc/pam.d/mariadb` on most systems.

Configuring PAM to Allow Only LDAP Authentication

Since we are only doing LDAP authentication with the `pam_ldap` PAM module and group mapping with the `pam_user_map` PAM module, our configuration file would look like this:

```
auth required pam_ldap.so
auth required pam_user_map.so
account required pam_ldap.so
```

Configuring PAM to Allow LDAP and Local Unix Authentication

If we want to allow authentication from LDAP users **and** from local Unix users through `pam_unix`, while giving priority to the local users, then we could do this instead:

```
auth [success=1 new_authtok_reqd=1 default=ignore] pam_unix.so audit
auth required pam_ldap.so try_first_pass
auth required pam_user_map.so
account sufficient pam_unix.so audit
account required pam_ldap.so
```

Configuring the pam_unix PAM Module

If you also want to allow authentication from local Unix users, the `pam_unix` PAM module adds some additional configuration steps on a lot of systems. We basically have to give the user that runs `mysqld` access to `/etc/shadow`.

If the `mysql` user is running `mysqld`, then we can do that by executing the following:

```
sudo groupadd shadow
sudo usermod -a -G shadow mysql
sudo chown root:shadow /etc/shadow
sudo chmod g+r /etc/shadow
```

The server needs to be restarted for this change to take affect.

Creating MariaDB Users

Next, let's create the MariaDB users. Remember that our PAM service is called `mariadb`.

First, let's create the MariaDB user for the user mapping: `foo: bar`

That means that we need to create a `bar` user:

```
CREATE USER 'bar'@'%' IDENTIFIED BY 'strongpassword';
GRANT ALL PRIVILEGES ON *.* TO 'bar'@'%' ;
```

And then let's create the MariaDB user for the group mapping: `@dba: dba`

That means that we need to create a `dba` user:

```
CREATE USER 'dba'@'%' IDENTIFIED BY 'strongpassword';
GRANT ALL PRIVILEGES ON *.* TO 'dba'@'%' ;
```

And then to allow for the user and group mapping, we need to create an anonymous user that authenticates with the `pam` authentication plugin that is also able to `PROXY` as the `bar` and `dba` users. Before we can create the proxy user, we might need to clean up some defaults:

```
DELETE FROM mysql.db WHERE User='' AND Host='%';
FLUSH PRIVILEGES;
```

And then let's create the anonymous proxy user:

```
CREATE USER ''@'%' IDENTIFIED VIA pam USING 'mariadb';
GRANT PROXY ON 'bar'@'%' TO ''@'%';
GRANT PROXY ON 'dba'@'%' TO ''@'%';
```

Testing our Configuration

Next, let's test out our configuration by verifying that mapping is occurring. We can verify this by logging in as each of our users and comparing the return value of `USER()`, which is the original user name and the return value of `CURRENT_USER()`, which is the authenticated user name.

Testing LDAP Authentication

First, let's test out our `foo` user:

```
$ mysql -u foo -h 172.30.0.198
[mariadb] Password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 134
Server version: 10.3.10-MariaDB MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> SELECT USER(), CURRENT_USER();
+-----------------------------------------------+----------------+
| USER()                                        | CURRENT_USER() |
+-----------------------------------------------+----------------+
| foo@ip-172-30-0-198.us-west-2.compute.internal | bar@%          |
+-----------------------------------------------+----------------+
1 row in set (0.000 sec)
```

We can verify that our `foo` LDAP user was properly mapped to the `bar` MariaDB user by looking at the return value of `CURRENT_USER()`.

Then let's test out our `gmontee` user in the `dba` group:

```
$ mysql -u gmontee -h 172.30.0.198
[mariadb] Password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 135
Server version: 10.3.10-MariaDB MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> SELECT USER(), CURRENT_USER();
+-------------------------------------------------+----------------+
| USER()                                          | CURRENT_USER() |
+-------------------------------------------------+----------------+
| gmontee@ip-172-30-0-198.us-west-2.compute.internal | dba@%          |
+-------------------------------------------------+----------------+
1 row in set (0.000 sec)
```

And then let's test out our `bstillman` user in the `dba` group:

```
$ mysql -u bstillman -h 172.30.0.198
[mariadb] Password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 136
Server version: 10.3.10-MariaDB MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> SELECT USER(), CURRENT_USER();
+---------------------------------------------------+----------------+
| USER()                                            | CURRENT_USER() |
+---------------------------------------------------+----------------+
| bstillman@ip-172-30-0-198.us-west-2.compute.internal | dba@%          |
+---------------------------------------------------+----------------+
1 row in set (0.000 sec)
```

We can verify that our `gmontee` and `bstillman` LDAP users in the `dba` LDAP group were properly mapped to the `dba` MariaDB user by looking at the return values of `CURRENT_USER()`.

Testing Local Unix Authentication

If you chose the option that also allowed local Unix authentication, then let's test that out. Let's create a Unix user and give the user a password real quick:

```
sudo useradd alice
sudo passwd alice
```

And let's also map this user to `dba`:

```
@dba:dba
foo: bar
alice: dba
```

And we know that the existing anonymous user already has the `PROXY` privilege granted to the `dba` user, so this should just work without any other configuration. Let's test it out:

```
$ mysql -u alice -h 172.30.0.198
[mariadb] Password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 141
Server version: 10.3.10-MariaDB MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> SELECT USER(), CURRENT_USER();
+---------------------------------------------------+----------------+
| USER()                                            | CURRENT_USER() |
+---------------------------------------------------+----------------+
| alice@ip-172-30-0-198.us-west-2.compute.internal | dba@%          |
+---------------------------------------------------+----------------+
1 row in set (0.000 sec)
```

We can verify that our `alice` Unix user was properly mapped to the `dba` MariaDB user by looking at the return values of `CURRENT_USER()`.

5.4.6.7 Authentication Plugin - Unix Socket

MariaDB starting with 10.4.3

In MariaDB 10.4.3 and later, the `unix_socket` authentication plugin is installed by default, and it is used by the `'root'@'localhost'` user account by default. See Authentication from MariaDB 10.4 for more information.

The `unix_socket` authentication plugin allows the user to use operating system credentials when connecting to MariaDB via the local Unix socket file. This Unix socket file is defined by the socket system variable.

The `unix_socket` authentication plugin works by calling the getsockopt system call with the `SO_PEERCRED` socket option, which allows it to retrieve the `uid` of the process that is connected to the socket. It is then able to get the user name associated with that `uid`. Once it has the user name, it will authenticate the connecting user as the MariaDB account that has the same user name.

The `unix_socket` authentication plugin is not suited to multiple Unix users accessing a single MariaDB user account.

Contents

Security

A `unix_socket` authentication plugin is a passwordless security mechanism. Its security is in the strength of the access to the Unix user rather than the complexity and the secrecy of the password. As the security is different from passwords, the strengths and weaknesses need to be considered, and these aren't the same in every installation.

Strengths

- Access is limited to the Unix user so, for example, a `www-data` user cannot access `root` with the `unix_socket` authentication plugin.
- There is no password to brute force.
- There is no password that can be accidentally exposed by user accident, poor security on backups, or poor security on passwords in configuration files.
- Default Unix user security is usually strong on preventing remote access and password brute force attempts.

Weaknesses

The strength of a `unix_socket` authentication plugin is effectively the strength of the security of the Unix users on the system. The Unix user default installation in most cases is sufficiently secure, however, business requirements or unskilled management may expose risks. The following is a non-exhaustive list of potential Unix user security issues that may arise.

- Common access areas without screen locks, where an unauthorized user accesses the logged in Unix user of an authorized user.
- Extensive sudo access grants that provide users with access to execute commands of a different Unix user.
- Scripts writable by Unix users other than the Unix user that are executed (cron or directly) by the unix user.
- Web pages that are susceptible to command injection, where the Unix user running the web page has elevated privileges in the database that weren't intended to be used.
- Poor Unix user password practices including weak user passwords, password exposure and password reuse accompanied by an access vulnerability/mechanism of an unauthorized user to exploit this weakness.
- Weak remote access mechanisms and network file system privileges.
- Poor user security behavior including running untrusted scripts and software.

In some of these scenarios a database password may prevent these security exploits, however it will remove all the strengths of the `unix_socket` authentication plugin previously mentioned.

Disabling the Plugin

MariaDB starting with 10.4.3

In MariaDB 10.4.3 and later, the `unix_socket` authentication plugin is installed by default, so **if you do not want it to be available by default on those versions, then you will need to disable it**.

The `unix_socket` authentication plugin is also installed by default in **new installations** that use the `.deb` packages provided by Debian's default repositories in Debian 9 and later and Ubuntu's default repositories in Ubuntu 15.10 and later, so **if you do not want it to be available by default on those systems when those packages are used, then you will need to disable it**. See Differences in MariaDB in Debian (and Ubuntu) for more information.

The `unix_socket` authentication plugin can be disabled by starting the server with the `unix_socket` option set to `OFF`. This can be specified as a command-line argument to `mysqld` or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
unix_socket=OFF
```

As an alternative, the `unix_socket` option can also be set to `OFF` by pairing the option with the `disable` option prefix. For example:

```
[mariadb]
...
disable_unix_socket
```

Installing the Plugin

MariaDB starting with 10.4.3

In MariaDB 10.4.3 and later, the `unix_socket` authentication plugin is installed by default, so **this step can be skipped on those versions**.

The `unix_socket` authentication plugin is also installed by default in **new installations** that use the `.deb` packages provided by Debian's default repositories in Debian 9 and later and Ubuntu's default repositories in Ubuntu 15.10 and later, so **this step can be skipped on those systems when those packages are used**. See Differences in MariaDB in Debian (and Ubuntu) for more information.

In other systems, although the plugin's shared library is distributed with MariaDB by default as `auth_socket.so`, the plugin is not actually installed by MariaDB by default. There are two methods that can be used to install the plugin with MariaDB.

The first method can be used to install the plugin without restarting the server. You can install the plugin dynamically by executing `INSTALL SONAME` or `INSTALL PLUGIN`. For example:

```
INSTALL SONAME 'auth_socket';
```

The second method can be used to tell the server to load the plugin when it starts up. The plugin can be installed this way by providing the `--plugin-load` or the `--plugin-load-add` options. This can be specified as a command-line argument to `mysqld` or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
plugin_load_add = auth_socket
```

Uninstalling the Plugin

You can uninstall the plugin dynamically by executing `UNINSTALL SONAME` or `UNINSTALL PLUGIN`. For example:

```
UNINSTALL SONAME 'auth_socket';
```

If you installed the plugin by providing the `--plugin-load` or the `--plugin-load-add` options in a relevant server option group in an option file, then those options should be removed to prevent the plugin from being loaded the next time the server is restarted.

Creating Users

To create a user account via `CREATE USER`, specify the name of the plugin in the `IDENTIFIED VIA` clause. For example:

```
CREATE USER username@hostname IDENTIFIED VIA unix_socket;
```

If `SQL_MODE` does not have `NO_AUTO_CREATE_USER` set, then you can also create the user account via `GRANT`. For example:

```
GRANT SELECT ON db.* TO username@hostname IDENTIFIED VIA unix_socket;
```

Switching to Password-based Authentication

Sometimes Unix socket authentication does not meet your needs, so it can be desirable to switch a user account back to password-based authentication. This can easily be done by telling MariaDB to use another authentication plugin for the account by executing the `ALTER USER` statement. The specific authentication plugin is specified with the `IDENTIFIED VIA` clause. For example, if you wanted to switch to the `mysql_native_password` authentication plugin, then you could execute:

```
ALTER USER root@localhost IDENTIFIED VIA mysql_native_password;
SET PASSWORD = PASSWORD('foo');
```

Note that if your operating system has scripts that require password-less access to MariaDB, then this may break those scripts. You may be able to fix that by setting a password in the `[client]` option group in your /root/.my.cnf option file. For example:

```
[client]
password=foo
```

Client Authentication Plugins

The `unix_socket` authentication plugin does not require any specific client authentication plugins. It should work with all clients.

Support in Client Libraries

The `unix_socket` authentication plugin does not require any special support in client libraries. It should work with all client libraries.

Example

```
$ mysql -uroot
MariaDB []> CREATE USER serg IDENTIFIED VIA unix_socket;
MariaDB []> CREATE USER monty IDENTIFIED VIA unix_socket;
MariaDB []> quit
Bye
$ whoami
serg
$ mysql --user=serg
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 2
Server version: 5.2.0-MariaDB-alpha-debug Source distribution
MariaDB []> quit
Bye
$ mysql --user=monty
ERROR 1045 (28000): Access denied for user 'monty'@'localhost' (using password: NO)
```

In this example, a user `serg` is already logged into the operating system and has full shell access. He has already authenticated with the operating system and his MariaDB account is configured to use the `unix_socket` authentication plugin, so he does not need to authenticate again for the database. MariaDB accepts his operating system credentials and allows him to connect. However, any attempt to connect to the database as another operating system user will be denied.

Versions

Version	Status	Introduced
1.0	Stable	MariaDB 10.0.11
1.0	Beta	MariaDB 5.2.0

Options

`unix_socket`

- **Description:** Controls how the server should treat the plugin when the server starts up.
  - Valid values are:
    - `OFF` - Disables the plugin without removing it from the `mysql.plugin` table.
    - `ON` - Enables the plugin. If the plugin cannot be initialized, then the server will still continue starting up, but the plugin will be disabled.
    - `FORCE` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error.
    - `FORCE_PLUS_PERMANENT` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error. In addition, the plugin cannot be uninstalled with `UNINSTALL SONAME` or `UNINSTALL PLUGIN` while the server is running.

    - See Plugin Overview: Configuring Plugin Activation at Server Startup for more information.
- **Commandline:** `--unix-socket=value`
- **Data Type:** `enumerated`
- **Default Value:** `ON`
- **Valid Values:** `OFF`, `ON`, `FORCE`, `FORCE_PLUS_PERMANENT`

5.4.6.8 Authentication Plugin - Named Pipe

The `named_pipe` authentication plugin allows the user to use operating system credentials when connecting to MariaDB via named pipe on Windows. Named pipe connections are enabled by the `named_pipe` system variable.

The `named_pipe` authentication plugin works by using named pipe impersonation and calling `GetUserName()` to retrieve the user name of the process that is connected to the named pipe. Once it has the user name, it authenticates the connecting user as the MariaDB account that has the same user name.

Contents

Installing the Plugin

Although the plugin's shared library is distributed with MariaDB by default, the plugin is not actually installed by MariaDB by default. There are two methods that can be used to install the plugin with MariaDB.

The first method can be used to install the plugin without restarting the server. You can install the plugin dynamically by executing `INSTALL SONAME` or `INSTALL PLUGIN`. For example:

```
INSTALL SONAME 'auth_named_pipe';
```

The second method can be used to tell the server to load the plugin when it starts up. The plugin can be installed this way by providing the `--plugin-load` or the `--plugin-load-add` options. This can be specified as a command-line argument to `mysqld` or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
plugin_load_add = auth_named_pipe
```

Uninstalling the Plugin

You can uninstall the plugin dynamically by executing `UNINSTALL SONAME` or `UNINSTALL PLUGIN`. For example:

```
UNINSTALL SONAME 'auth_named_pipe';
```

If you installed the plugin by providing the `--plugin-load` or the `--plugin-load-add` options in a relevant server option group in an option file, then those options should be removed to prevent the plugin from being loaded the next time the server is restarted.

Creating Users

To create a user account via `CREATE USER`, specify the name of the plugin in the `IDENTIFIED VIA` clause. For example:

```
CREATE USER username@hostname IDENTIFIED VIA named_pipe;
```

If `SQL_MODE` does not have `NO_AUTO_CREATE_USER` set, then you can also create the user account via `GRANT`. For

example:

```
GRANT SELECT ON db.* TO username@hostname IDENTIFIED VIA named_pipe;
```

Client Authentication Plugins

The `named_pipe` authentication plugin does not require any specific client authentication plugins. It should work with all clients.

Support in Client Libraries

The `named_pipe` authentication plugin does not require any special support in client libraries. It should work with all client libraries.

Example

```
CREATE USER wlad IDENTIFIED VIA named_pipe;
CREATE USER monty IDENTIFIED VIA named_pipe;
quit

C:\>echo %USERNAME%
wlad

C:\> mysql --user=wlad --protocol=PIPE
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 4
Server version: 10.1.12-MariaDB-debug Source distribution

Copyright (c) 2000, 2015, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> quit
Bye

C:\> mysql --user=monty  --protocol=PIPE
ERROR 1698 (28000): Access denied for user 'monty'@'localhost'
```

In this example, a user `wlad` is already logged into the system. Because he has identified himself to the operating system, he does not need to do it again for the database — MariaDB trusts the operating system credentials. However, he cannot connect to the database as another user.

Versions

Version	Status	Introduced
1.0	Stable	MariaDB 10.1.11

Options

`named_pipe`

- **Description:** Controls how the server should treat the plugin when the server starts up.
  - Valid values are:
    - `OFF` - Disables the plugin without removing it from the `mysql.plugins` table.
    - `ON` - Enables the plugin. If the plugin cannot be initialized, then the server will still continue starting up, but the plugin will be disabled.
    - `FORCE` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error.
    - `FORCE_PLUS_PERMANENT` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error. In addition, the plugin cannot be uninstalled with `UNINSTALL SONAME` or `UNINSTALL PLUGIN` while the server is running.
  - See Plugin Overview: Configuring Plugin Activation at Server Startup for more information.

    - There may be ambiguity between this option and the `named_pipe` system variable. See MDEV-19625 about that.
- **Commandline:** `--named-pipe=value`
- **Data Type:** `enumerated`
- **Default Value:** `ON`
- **Valid Values:** `OFF`, `ON`, `FORCE`, `FORCE_PLUS_PERMANENT`
- **Introduced:** MariaDB 10.1.11

5.4.6.9 Authentication Plugin - SHA-256

Contents

MySQL 5.6 added support for the `sha256_password` authentication plugin, and MySQL 8.0 also added support for the `caching_sha2_password` authentication plugin.

The `caching_sha2_password` plugin is now the default authentication plugin in MySQL 8.0.4 and above, based on the value of the `default_authentication_plugin` system variable.

Support in MariaDB Server

MariaDB Server does not currently support either the `sha256_password` or the `caching_sha2_password` authentication plugins. See MDEV-9804 for more information.

MariaDB Server does not support either of these authentication plugins. This is mainly because:

- To use the protocol, one has to distribute the server's public key to all MariaDB users, which can be cumbersome and impractical.
- The server gets the password in clear text which can cause problems if the user is convinced to connect to a malicious server.

Client Authentication Plugins

For clients that use the MariaDB Connector/C library, MariaDB provides two client authentication plugins that are compatible with MySQL's SHA-256 authentication plugins:

- `sha256_password`
- `caching_sha256_password`

When connecting with a client or utility to a server as a user account that authenticates with the `sha256_password` or `caching_sha256_password` authentication plugin, you may need to tell the client where to find the relevant client authentication plugin by specifying the `--plugin-dir` option. For example:

```
mysql --plugin-dir=/usr/local/mysql/lib64/mysql/plugin --user=alice
```

For clients that use MariaDB's `libmysqlclient` library instead of MariaDB Connector/C, these client authentication plugins are not supported.

`sha256_password`

The `sha256_password` client authentication plugin is compatible with MySQL's `sha256_password` authentication plugin, which was added in MySQL 5.6.

`caching_sha256_password`

The `caching_sha256_password` client authentication plugin is compatible with MySQL's `caching_sha2_password` authentication plugin, which was added in MySQL 8.0.

The `caching_sha2_password` plugin is now the default authentication plugin in MySQL 8.0.4 and above, based on the value of the `default_authentication_plugin` system variable.

Support in Client Libraries

Using the Plugin with MariaDB Connector/C

MariaDB Connector/C supports `sha256_password` and `caching_sha2_password` authentication using the client authentication plugins mentioned in the previous section.

It has supported the `sha256_password` client authentication plugin since MariaDB Connector/C 3.0.2. See CONC-229 for more information.

It has supported the `caching_sha256_password` client authentication plugin since MariaDB Connector/C 3.0.8 and MariaDB Connector/C 3.1.0. See CONC-312 for more information.

Using the Plugin with MariaDB Connector/ODBC

MariaDB Connector/ODBC supports `sha256_password` and `caching_sha2_password` authentication using the client authentication plugins mentioned in the previous section.

It has supported `sha256_password` and `caching_sha2_password` authentication since MariaDB Connector/ODBC 3.1.4. See ODBC-241 for more information.

Using the Plugin with MariaDB Connector/J

MariaDB Connector/J supports `sha256_password` and `caching_sha2_password` authentication since MariaDB Connector/J 2.5.0. See CONJ-327 and CONJ-663 for more information.

Using the Plugin with MariaDB Connector/Node.js

MariaDB Connector/Node.js supports `sha256_password` and `caching_sha2_password` authentication since MariaDB Connector/Node.js 2.5.0. See CONJS-76 and CONJS-77 for more information.

5.4.7 Password Validation Plugins

Simple Password Check Plugin

This plugin checks that passwords meet certain simple criteria.

Cracklib Password Check Plugin

This plugin checks password strength using the CrackLib library.

Password Reuse Check Plugin

Plugin for preventing password reuse.

Password Validation Plugin API

Allows the creation of password validation plugins to check user passwords as they are set.

password_reuse_check_interval

Retention period for password history.

5.4.7.1 Simple Password Check Plugin

`simple_password_check` is a password validation plugin. It can check whether a password contains at least a certain number of characters of a specific type. When first installed, a password is required to be at least eight characters, and requires at least one digit, one uppercase character, one lowercase character, and one character that is neither a digit nor a letter.

Note that passwords can be directly set as a hash, bypassing the password validation, if the strict_password_validation variable is `OFF` (it is `ON` by default).

Contents

Installing the Plugin

Although the plugin's shared library is distributed with MariaDB by default, the plugin is not actually installed by MariaDB by default. There are two methods that can be used to install the plugin with MariaDB.

The first method can be used to install the plugin without restarting the server. You can install the plugin dynamically by executing `INSTALL SONAME` or `INSTALL PLUGIN`. For example:

```
INSTALL SONAME 'simple_password_check';
```

The second method can be used to tell the server to load the plugin when it starts up. The plugin can be installed this way by providing the `--plugin-load` or the `--plugin-load-add` options. This can be specified as a command-line argument to `mysqld` or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
plugin_load_add = simple_password_check
```

Uninstalling the Plugin

You can uninstall the plugin dynamically by executing `UNINSTALL SONAME` or `UNINSTALL PLUGIN`. For example:

```
UNINSTALL SONAME 'simple_password_check';
```

If you installed the plugin by providing the `--plugin-load` or the `--plugin-load-add` options in a relevant server option group in an option file, then those options should be removed to prevent the plugin from being loaded the next time the server is restarted.

Example

When creating a new password, if the criteria are not met, the following error is returned:

```
SET PASSWORD FOR 'bob'@'%.loc.gov' = PASSWORD('abc');
ERROR 1819 (HY000): Your password does not satisfy the current policy requirements
```

Known Issues

Issues with PAM Authentication Plugin

Prior to MariaDB 10.4.0, all password validation plugins are incompatible with the `pam` authentication plugin. See Authentication Plugin - PAM: Conflicts with Password Validation for more information.

Versions

Version	Status	Introduced
1.0	Stable	MariaDB 10.1.18
1.0	Gamma	MariaDB 10.1.13
1.0	Beta	MariaDB 10.1.11
1.0	Alpha	MariaDB 10.1.2

System Variables

`simple_password_check_digits`

- **Description:** A password must contain at least this many digits.
- **Commandline:** `--simple-password-check-digits=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1`
- **Range:** `0` to `1000`

`simple_password_check_letters_same_case`

- **Description:** A password must contain at least this many upper-case and this many lower-case letters.
- **Commandline:** `--simple-password-check-letters-same-case=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1`
- **Range:** `0` to `1000`

`simple_password_check_minimal_length`

- **Description:** A password must contain at least this many characters.
- **Commandline:** `--simple-password-check-minimal-length=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `8`
- **Range:** `0` to `1000`

`simple_password_check_other_characters`

- **Description:** A password must contain at least this many characters that are neither digits nor letters.
- **Commandline:** `--simple-password-check-other-characters=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1`
- **Range:** `0` to `1000`

Options

`simple_password_check`

- **Description:** Controls how the server should treat the plugin when the server starts up.
  - Valid values are:
    - `OFF` - Disables the plugin without removing it from the `mysql.plugins` table.
    - `ON` - Enables the plugin. If the plugin cannot be initialized, then the server will still continue starting up,

but the plugin will be disabled.
    - `FORCE` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error.
    - `FORCE_PLUS_PERMANENT` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error. In addition, the plugin cannot be uninstalled with `UNINSTALL SONAME` or `UNINSTALL PLUGIN` while the server is running.
  - See Plugin Overview: Configuring Plugin Activation at Server Startup for more information.
- **Commandline:** `--simple-password-check=value`
- **Data Type:** `enumerated`
- **Default Value:** `ON`
- **Valid Values:** `OFF`, `ON`, `FORCE`, `FORCE_PLUS_PERMANENT`

5.4.7.2 Cracklib Password Check Plugin

Contents

`cracklib_password_check` is a password validation plugin. It uses the CrackLib library to check the strength of new passwords. CrackLib is installed by default in many Linux distributions, since the system's Pluggable Authentication Module (PAM) authentication framework is usually configured to check the strength of new passwords with the `pam_cracklib` PAM module.

Note that passwords can be directly set as a hash, bypassing the password validation, if the strict_password_validation variable is `OFF` (it is `ON` by default).

The plugin requires at least cracklib 2.9.0, so it is not available on Debian/Ubuntu builds before Debian 8 Jessie/Ubuntu 14.04 Trusty, RedHat Enterprise Linux / CentOS 6.

Installing the Plugin's Package

The `cracklib_password_check` plugin's shared library is included in MariaDB packages as the `cracklib_password_check.so` or `cracklib_password_check.dll` shared library on systems where it can be built.

Installing on Linux

The `cracklib_password_check` plugin is included in `systemd` binary tarballs on Linux, but not in the older generic and `glibc_214` tarballs.

Installing with a Package Manager

The `cracklib_password_check` plugin can also be installed via a package manager on Linux. In order to do so, your system needs to be configured to install from one of the MariaDB repositories.

You can configure your package manager to install it from MariaDB Corporation's MariaDB Package Repository by using the MariaDB Package Repository setup script.

You can also configure your package manager to install it from MariaDB Foundation's MariaDB Repository by using the

MariaDB Repository Configuration Tool.

Installing with yum/dnf

On RHEL, CentOS, Fedora, and other similar Linux distributions, it is highly recommended to install the relevant RPM package from MariaDB's repository using `yum` or `dnf`. Starting with RHEL 8 and Fedora 22, `yum` has been replaced by `dnf`, which is the next major version of `yum`. However, `yum` commands still work on many systems that use `dnf`. For example:

```
sudo yum install MariaDB-cracklib-password-check
```

Installing with apt-get

On Debian, Ubuntu, and other similar Linux distributions, it is highly recommended to install the relevant DEB package from MariaDB's repository using `apt-get`. For example:

```
sudo apt-get install mariadb-plugin-cracklib-password-check
```

Installing with zypper

On SLES, OpenSUSE, and other similar Linux distributions, it is highly recommended to install the relevant RPM package from MariaDB's repository using `zypper`. For example:

```
sudo zypper install MariaDB-cracklib-password-check
```

Installing the Plugin

Once the shared library is in place, the plugin is not actually installed by MariaDB by default. There are two methods that can be used to install the plugin with MariaDB.

The first method can be used to install the plugin without restarting the server. You can install the plugin dynamically by executing `INSTALL SONAME` or `INSTALL PLUGIN`. For example:

```
INSTALL SONAME 'cracklib_password_check';
```

The second method can be used to tell the server to load the plugin when it starts up. The plugin can be installed this way by providing the `--plugin-load` or the `--plugin-load-add` options. This can be specified as a command-line argument to `mysqld` or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
plugin_load_add = cracklib_password_check
```

Uninstalling the Plugin

You can uninstall the plugin dynamically by executing `UNINSTALL SONAME` or `UNINSTALL PLUGIN`. For example:

```
UNINSTALL SONAME 'cracklib_password_check';
```

If you installed the plugin by providing the `--plugin-load` or the `--plugin-load-add` options in a relevant server option group in an option file, then those options should be removed to prevent the plugin from being loaded the next time the server is restarted.

Viewing CrackLib Errors

If password validation fails, then the original CrackLib error message can be viewed by executing `SHOW WARNINGS`.

Example

When creating a new password, if the criteria are not met, the following error is returned:

```
SET PASSWORD FOR 'bob'@'%.loc.gov' = PASSWORD('abc');
ERROR 1819 (HY000): Your password does not satisfy the current policy requirements
```

Known Issues

Issues with PAM Authentication Plugin

Prior to MariaDB 10.4.0, all password validation plugins are incompatible with the `pam` authentication plugin. See Authentication Plugin - PAM: Conflicts with Password Validation for more information.

SELinux

When using the standard SELinux policy with the mode set to `enforcing`, `mysqld` does not have access to `/usr/share/cracklib`, and you may see the following error when attempting to use the `cracklib_password_check` plugin:

```
CREATE USER `user`@`hostname` IDENTIFIED BY 's0mePwd123.';
ERROR 1819 (HY000): Your password does not satisfy the current policy requirements

SHOW WARNINGS;
+---------+------+----------------------------------------------------------------+
| Level   | Code | Message                                                        |
+---------+------+----------------------------------------------------------------+
| Warning | 1819 | cracklib: error loading dictionary                             |
| Error   | 1819 | Your password does not satisfy the current policy requirements |
| Error   | 1396 | Operation CREATE USER failed for 'user'@'hostname'             |
+---------+------+----------------------------------------------------------------+
```

And the SELinux `audit.log` will contain errors like the following:

```
type=AVC msg=audit(1548371977.821:66): avc:  denied  { read } for  pid=3537 comm="mysqld"
name="pw_dict.pwd" dev="xvda2" ino=564747 scontext=system_u:system_r:mysqld_t:s0
tcontext=system_u:object_r:crack_db_t:s0 tclass=file
type=SYSCALL msg=audit(1548371977.821:66): arch=c000003e syscall=2 success=no exit=-13
a0=7fdd2a674580 a1=0 a2=1b6 a3=1b items=0 ppid=1 pid=3537 auid=4294967295 uid=995 gid=992
euid=995 suid=995 fsuid=995 egid=992 sgid=992 fsgid=992 tty=(none) ses=4294967295 comm="mysqld"
exe="/usr/sbin/mysqld" subj=system_u:system_r:mysqld_t:s0 key=(null)
```

This can be fixed by creating an SELinux policy that allows `mysqld` to load the CrackLib dictionary. For example:

```
cd /usr/share/mysql/policy/selinux/
tee ./mariadb-plugin-cracklib-password-check.te <<EOF

module mariadb-plugin-cracklib-password-check 1.0;

require {
        type mysqld_t;
        type crack_db_t;
        class file { execute setattr read create getattr execute_no_trans write ioctl open
append unlink };
        class dir { write search getattr add_name read remove_name open };
}

allow mysqld_t crack_db_t:dir { search read open };
allow mysqld_t crack_db_t:file { getattr read open };
EOF
sudo yum install selinux-policy-devel
make -f /usr/share/selinux/devel/Makefile mariadb-plugin-cracklib-password-check.pp
sudo semodule -i mariadb-plugin-cracklib-password-check.pp
```

See MDEV-18374 for more information.

Versions

Version	Status	Introduced

1.0	Stable	MariaDB 10.1.18
1.0	Gamma	MariaDB 10.1.13
1.0	Alpha	MariaDB 10.1.2

System Variables

`cracklib_password_check_dictionary`

- **Description:** Sets the path to the CrackLib dictionary. If not set, the default CrackLib dictionary path is used. The parameter expects the base name of a cracklib dictionary (a set of three files with endings `.hwm`, `.pwd`, `.pwi`), not a directory path.
- **Commandline:** `--cracklib-password-check-dictionary=value`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** Depends on the system. Often `/usr/share/cracklib/pw_dict`

Options

`cracklib_password_check`

- **Description:** Controls how the server should treat the plugin when the server starts up.
  - Valid values are:
    - `OFF` - Disables the plugin without removing it from the `mysql.plugins` table.
    - `ON` - Enables the plugin. If the plugin cannot be initialized, then the server will still continue starting up, but the plugin will be disabled.
    - `FORCE` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error.
    - `FORCE_PLUS_PERMANENT` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error. In addition, the plugin cannot be uninstalled with `UNINSTALL SONAME` or `UNINSTALL PLUGIN` while the server is running.
  - See Plugin Overview: Configuring Plugin Activation at Server Startup for more information.
- **Commandline:** `--cracklib-password-check=value`
- **Data Type:** `enumerated`
- **Default Value:** `ON`
- **Valid Values:** `OFF`, `ON`, `FORCE`, `FORCE_PLUS_PERMANENT`

5.4.7.3 Password Reuse Check Plugin

MariaDB starting with 10.7

`password_reuse_check` is a password validation plugin introduced in MariaDB 10.7.0.

Contents

Description

The plugin is used to prevent a user from reusing a password, which can be a requirement in some security policies. The password_reuse_check_interval system variable determines the retention period, in days, for a password. By default this is zero, meaning unlimited retention. Old passwords are stored in the mysql.password_reuse_check_history table.

Note that passwords can be directly set as a hash, bypassing the password validation, if the strict_password_validation variable is `OFF` (it is `ON` by default).

Installing the Plugin

Although the plugin's shared library is distributed with MariaDB by default, the plugin is not actually installed by MariaDB by default.

You can install the plugin dynamically, without restarting the server, by executing INSTALL SONAME or INSTALL PLUGIN. For example:

```
INSTALL SONAME 'password_reuse_check';
```

The second method can be used to tell the server to load the plugin when it starts up. The plugin can be installed this way by providing the --plugin-load or the --plugin-load-add options. This can be specified as a command-line argument to mysqld or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
plugin_load_add = password_reuse_check
```

Uninstalling the Plugin

You can uninstall the plugin dynamically by executing UNINSTALL SONAME or UNINSTALL PLUGIN. For example:

```
UNINSTALL SONAME 'password_reuse_check';
```

If you installed the plugin by providing the --plugin-load or the --plugin-load-add options in a relevant server option group in an option file, then those options should be removed to prevent the plugin from being loaded the next time the server is restarted.

Example

```
INSTALL SONAME 'password_reuse_check';

GRANT SELECT ON *.* TO user1@localhost identified by 'pwd1';
Query OK, 0 rows affected (0.038 sec)

GRANT SELECT ON *.* TO user1@localhost identified by 'pwd1';
ERROR 1819 (HY000): Your password does not satisfy the current policy requirements

GRANT SELECT ON *.* TO user1@localhost identified by 'pwd2';
Query OK, 0 rows affected (0.003 sec)

GRANT SELECT ON *.* TO user1@localhost identified by 'pwd1';
ERROR 1819 (HY000): Your password does not satisfy the current policy requirements
```

Versions

Version	Status	Introduced
1.0	Alpha	MariaDB 10.7.0
1.0	Beta	MariaDB 10.7.2
1.0	Gamma	MariaDB 10.7.4
2.0	Stable	MariaDB 10.7.7, MariaDB 10.8.7, MariaDB 10.9.5, MariaDB 10.10.2

The bump to version 2.0 required the change of the stored format to mitigate an implementation weakness (MDEV-28838) and as such the bump from 1.0 to 2.0 will invalidate previously saved password reuse protections.

5.4.7.4 Password Validation Plugin API

Contents

“Password validation” means ensuring that user passwords meet certain minimal security requirements. A dedicated plugin API allows the creation of password validation plugins that will check user passwords as they are set (in SET PASSWORD and GRANT statements) and either allow or reject them.

SQL-Level Extensions

MariaDB comes with three password validation plugins — the simple_password_check plugin, the cracklib_password_check plugin and the password_reuse_check plugin. They are not enabled by default; use INSTALL SONAME (or INSTALL PLUGIN) statement to install them.

When at least one password plugin is loaded, all new passwords will be validated and password-changing statements will fail if the password will not pass validation checks. Several password validation plugin can be loaded at the same time — in this case a password must pass **all** validation checks by **all** plugins.

Password-Changing Statements

One can use various SQL statements to change a user password:

With Plain Text Password

```
SET PASSWORD = PASSWORD('plain-text password');
SET PASSWORD FOR `user`@`host` = PASSWORD('plain-text password');
SET PASSWORD = OLD_PASSWORD('plain-text password');
SET PASSWORD FOR `user`@`host` = OLD_PASSWORD('plain-text password');
CREATE USER `user`@`host` IDENTIFIED BY 'plain-text password';
GRANT privileges TO `user`@`host` IDENTIFIED BY 'plain-text password';
```

These statements are subject to password validation. If at least one password validation plugin is loaded, plain-text passwords specified in these statements will be validated.

With Password Hash

```
SET PASSWORD = 'password hash';
SET PASSWORD FOR `user`@`host` = 'password hash';
CREATE USER `user`@`host` IDENTIFIED BY PASSWORD 'password hash';
CREATE USER `user`@`host` IDENTIFIED VIA mysql_native_password USING 'password hash';
CREATE USER `user`@`host` IDENTIFIED VIA mysql_old_password USING 'password hash';
GRANT privileges TO `user`@`host` IDENTIFIED BY PASSWORD 'password hash';
GRANT privileges TO `user`@`host` IDENTIFIED VIA mysql_native_password USING 'password hash';
GRANT privileges TO `user`@`host` IDENTIFIED VIA mysql_old_password USING 'password hash';
```

These statements can not possibly use password validation — there is nothing to validate, the original plain-text password is not available. MariaDB introduces a **strict password validation** mode — controlled by a strict_password_validation global server variable. If the strict password validation is enabled and at least one password validation plugin is loaded then these “unvalidatable” passwords will be rejected. Otherwise they will be accepted. By default a strict password validation is enabled (but note that it has no effect if no password validation plugin is loaded).

Examples

Failed password validation:

```
GRANT SELECT ON *.* to foobar IDENTIFIED BY 'raboof';
ERROR HY000: Your password does not satisfy the current policy requirements

SHOW WARNINGS;
+---------+------+-----------------------------------------------------------------+
| Level   | Code | Message                                                         |
+---------+------+-----------------------------------------------------------------+
| Warning | 1819 | cracklib: it is based on your username                          |
| Error   | 1819 | Your password does not satisfy the current policy requirements |
+---------+------+-----------------------------------------------------------------+
```

Strict password validation:

```
GRANT SELECT ON *.* TO foo IDENTIFIED BY PASSWORD '2222222222222222';
ERROR HY000: The MariaDB server is running with the --strict-password-validation option so it
cannot execute this statement
```

Plugin API

Password validation plugin API is very simple. A plugin must implement only one method — `validate_password()` . This method takes two arguments — user name and the plain-text password. And it returns 0 when the password has passed the validation and 1 otherwise,

See also `mysql/plugin_password_validation.h` and password validation plugins in `plugin/simple_password_check/` and `plugins/cracklib_password_check/` .

5.4.7.5 password_reuse_check_interval

The password_reuse_check_interval system variable is available when the password_reuse_check plugin is installed. It determines the retention period for the password history in days. Zero, the default, means passwords are never discarded.

- **Commandline:** `--password_reuse_check_interval=#`
- **Scope:** Global
- **Read-only:** No
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `36500`

5.4.8 Key Management and Encryption Plugins

Encryption Key Management
Managing encryption keys for data-at-rest encryption.

File Key Management Encryption Plugin
A key management and encryption plugin for data-at-rest encryption that uses a plain-text file.

Hashicorp Key Management Plugin
Implement encryption using keys stored in the Hashicorp Vault KMS.

AWS Key Management Encryption Plugin
A key management and encryption plugin for data-at-rest encryption that use...

Amazon Web Services (AWS) Key Management Service (KMS) Encryption Plugin Setup Guide
Plugin that uses the AWS Key Management Service.

Amazon Web Services (AWS) Key Management Service (KMS) Encryption Plugin Advanced Usage
This document assumes you've already set up an Amazon Web Services (AWS) a...

Eperi Key Management Encryption Plugin
A key management and encryption plugin for data-at-rest encryption that use...

There are 1 related questions.

5.4.8.1 Encryption Key Management

Contents

MariaDB's data-at-rest encryption requires the use of a key management and encryption plugin. These plugins are responsible both for the management of encryption keys and for the actual encryption and decryption of data.

MariaDB supports the use of multiple encryption keys. Each encryption key uses a 32-bit integer as a key identifier. If the specific plugin supports key rotation, then encryption keys can also be rotated, which creates a new version of the encryption key.

Choosing an Encryption Key Management Solution

How MariaDB manages encryption keys depends on which encryption key management solution you choose. Currently, MariaDB has three options:

File Key Management Plugin

The File Key Management plugin that ships with MariaDB is a basic key management and encryption plugin that reads keys from a plain-text file. It can also serve as example and as a starting point when developing a key management plugin.

For more information, see File Key Management Plugin.

AWS Key Management Plugin

The AWS Key Management plugin is a key management and encryption plugin that uses the Amazon Web Services (AWS) Key Management Service (KMS). The AWS Key Management plugin depends on the AWS SDK for C++, which uses the Apache License, Version 2.0. This license is not compatible with MariaDB Server's GPL 2.0 license, so we are not able to distribute packages that contain the AWS Key Management plugin. Therefore, the only way to currently obtain the plugin is to install it from source.

For more information, see AWS Key Management Plugin.

Eperi Key Management Plugin

The Eperi Key Management plugin is a key management and encryption plugin that uses the eperi Gateway for Databases. The eperi Gateway for Databases stores encryption keys on the key server outside of the database server itself, which provides an extra level of security. The eperi Gateway for Databases also supports performing all data encryption operations on the key server as well, but this is optional.

For more information, see Eperi Key Management Plugin.

Using Multiple Encryption Keys

Key management and encryption plugins support using multiple encryption keys. Each encryption key can be defined with a different 32-bit integer as a key identifier.

The support for multiple keys opens up some potential use cases. For example, let's say that a hypothetical key management and encryption plugin is configured to provide two encryption keys. One encryption key might be intended for

"low security" tables. It could use short keys, which might not be rotated, and data could be encrypted with a fast encryption algorithm. Another encryption key might be intended for "high security" tables. It could use long keys, which are rotated often, and data could be encrypted with a slower, but more secure encryption algorithm. The user would specify the identifier of the key that they want to use for different tables, only using high level security where it's needed.

There are two encryption key identifiers that have special meanings in MariaDB. Encryption key `1` is intended for encrypting system data, such as InnoDB redo logs, binary logs, and so on. It must always exist when data-at-rest encryption is enabled. Encryption key `2` is intended for encrypting temporary data, such as temporary files and temporary tables. It is optional. If it doesn't exist, then MariaDB uses encryption key `1` for these purposes instead.

When encrypting InnoDB tables, the key that is used to encrypt tables can be changed.

When encrypting Aria tables, the key that is used to encrypt tables cannot currently be changed.

Key Rotation

Encryption key rotation is optional in MariaDB Server. Key rotation is only supported if the backend key management service (KMS) supports key rotation, and if the corresponding key management and encryption plugin for MariaDB also supports key rotation. When a key management and encryption plugin supports key rotation, users can opt to rotate one or more encryption keys, which creates a new version of each rotated encryption key.

Key rotation allows users to improve data security in the following ways:

- If the server is configured to automatically re-encrypt table data with the newer version of the encryption key after the key is rotated, then that prevents an encryption key from being used for long periods of time.
- If the server is configured to simultaneously encrypt table data with multiple versions of the encryption key after the key is rotated, then that prevents all data from being leaked if a single encryption key version is compromised.

The InnoDB storage engine has background encryption threads that can automatically re-encrypt pages when key rotations occur.

The Aria storage engine does not currently have a similar mechanism to re-encrypt pages in the background when key rotations occur.

Support for Key Rotation in Encryption Plugins

Encryption Plugins with Key Rotation Support

- The AWS Key Management Service (KMS) supports encryption key rotation, and the corresponding AWS Key Management Plugin also supports encryption key rotation.
- The eperi Gateway for Databases supports encryption key rotation, and the corresponding Eperi Key Management Plugin also supports encryption key rotation.

Encryption Plugins without Key Rotation Support

- The File Key Management Plugin does not support encryption key rotation, because it does not use a backend key management service (KMS).

Encryption Plugin API

New key management and encryption plugins can be developed using the encryption plugin API.

5.4.8.2 File Key Management Encryption Plugin

Contents

MariaDB's data-at-rest encryption requires the use of a key management and encryption plugin. These plugins are responsible both for the management of encryption keys and for the actual encryption and decryption of data.

MariaDB supports the use of multiple encryption keys. Each encryption key uses a 32-bit integer as a key identifier. If the specific plugin supports key rotation, then encryption keys can also be rotated, which creates a new version of the encryption key.

The File Key Management plugin that ships with MariaDB is a key management and encryption plugin that reads encryption keys from a plain-text file.

Overview

The File Key Management plugin is the easiest key management and encryption plugin to set up for users who want to use data-at-rest encryption. Some of the plugin's primary features are:

- It reads encryption keys from a plain-text key file.
- As an extra protection mechanism, the plain-text key file can be encrypted.
- It supports multiple encryption keys.
- It does **not** support key rotation.
- It supports two different algorithms for encrypting data.

It can also serve as an example and as a starting point when developing a key management and encryption plugin with the encryption plugin API.

Installing the File Key Management Plugin's Package

The File Key Management plugin is included in MariaDB packages as the `file_key_management.so` or `file_key_management.dll` shared library. The shared library is in the main server package, so no additional package installations are necessary. The plug-in must be installed into MariaDB however as follows.

Installing the Plugin

Although the plugin's shared library is distributed with MariaDB by default, the plugin is not actually installed by MariaDB by default. The plugin can be installed by providing the `--plugin-load` or the `--plugin-load-add` options. This can be specified as a command-line argument to `mysqld` or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
plugin_load_add = file_key_management
```

Uninstalling the Plugin

Before you uninstall the plugin, you should ensure that data-at-rest encryption is completely disabled, and that MariaDB no longer needs the plugin to decrypt tables or other files.

You can uninstall the plugin dynamically by executing `UNINSTALL SONAME` or `UNINSTALL PLUGIN`. For example:

```
UNINSTALL SONAME 'file_key_management';
```

If you installed the plugin by providing the `--plugin-load` or the `--plugin-load-add` options in a relevant server option group in an option file, then those options should be removed to prevent the plugin from being loaded the next time the server is restarted.

Creating the Key File

In order to encrypt your tables with encryption keys using the File Key Management plugin, you first need to create the file that contains the encryption keys. The file needs to contain two pieces of information for each encryption key. First, each encryption key needs to be identified with a 32-bit integer as the key identifier. Second, the encryption key itself needs to be provided in hex-encoded form. These two pieces of information need to be separated by a semicolon. For example, the file is formatted in the following way:

```
<encryption_key_id1>;<hex-encoded_encryption_key1>
<encryption_key_id2>;<hex-encoded_encryption_key2>
```

You can also optionally encrypt the key file to make it less accessible from the file system. That is explained further in the section below.

The File Key Management plugin uses Advanced Encryption Standard (AES) to encrypt data, which supports 128-bit, 192-bit, and 256-bit encryption keys. Therefore, the plugin also supports 128-bit, 192-bit, and 256-bit encryption keys.

You can generate random encryption keys using the `openssl rand` command. For example, to create a random 256-bit (32-byte) encryption key, you would run the following command:

```
$ openssl rand -hex 32
a7addd9adea9978fda19f21e6be987880e68ac92632ca052e5bb42b1a506939a
```

The key file still needs to have a key identifier for each encryption key added to the beginning of each line. Key identifiers do not need to be contiguous.

For example, to append three new encryption keys to a new key file, you could execute the following:

```
$ (echo -n "1;" ; openssl rand -hex 32 ) | sudo tee -a  /etc/mysql/encryption/keyfile
$ (echo -n "2;" ; openssl rand -hex 32 ) | sudo tee -a  /etc/mysql/encryption/keyfile
$ (echo -n "100;" ; openssl rand -hex 32 ) | sudo tee -a  /etc/mysql/encryption/keyfile
```

The new key file would look something like the following after this step:

```
1;a7addd9adea9978fda19f21e6be987880e68ac92632ca052e5bb42b1a506939a
2;49c16acc2dffe616710c9ba9a10b94944a737de1beccb52dc1560abfdd67388b
100;8db1ee74580e7e93ab8cf157f02656d356c2f437d548d5bf16bf2a56932954a3
```

The key identifiers give you a way to reference the encryption keys from MariaDB. In the example above, you could reference these encryption keys using the key identifiers `1`, `2` or `100` with the `ENCRYPTION_KEY_ID` table option or with system variables such as `innodb_default_encryption_key_id`. You do not necessarily need multiple encryption keys--the encryption key with the key identifier `1` is the only mandatory encryption key.

Configuring the Path to an Unencrypted Key File

If the key file is unencrypted, then the File Key Management plugin only requires the `file_key_management_filename` system variable to be configured.

This system variable can be specified as command-line arguments to `mysqld` or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
loose_file_key_management_filename = /etc/mysql/encryption/keyfile
```

Note that the `loose` option prefix is specified. This option prefix is used in case the plugin hasn't been installed yet.

Encrypting the Key File

By enabling the File Key Management plugin and setting the appropriate path on the `file_key_management_filename` system variable, you can begin using the plugin to manage your encryption keys. But, there is a security risk in doing so, given that the keys are stored in plain text on your system. You can reduce this exposure using file permissions, but it's better to encrypt the whole key file to further restrict access.

There are some important details to keep in mind about encrypting the key file, such as:

- The only algorithm that MariaDB currently supports to encrypt the key file is Cipher Block Chaining (CBC) mode of Advanced Encryption Standard (AES).
- The encryption key size can be 128-bits, 192-bits, or 256-bits.
- The encryption key is created from the SHA-1 hash of the encryption password.
- The encryption password has a max length of 256 characters.

You can generate a random encryption password using the `openssl rand` command. For example, to create a random 256 character encryption password, you could execute the following:

```
$ sudo openssl rand -hex 128 > /etc/mysql/encryption/keyfile.key
```

You can encrypt the key file using the `openssl enc` command. For example, to encrypt the key file with the encryption password created in the previous step, you could execute the following:

```
$ sudo openssl enc -aes-256-cbc -md sha1 \
   -pass file:/etc/mysql/encryption/keyfile.key \
   -in /etc/mysql/encryption/keyfile \
   -out /etc/mysql/encryption/keyfile.enc
```

Running this command reads the unencrypted `keyfile` file created above and creates a new encrypted `keyfile.enc` file, using the encryption password stored in `keyfile.key`. Once you've finished preparing your system, you can delete the unencrypted `keyfile` file, as it's no longer necessary.

Configuring the Path to an Encrypted Key File

If the key file is encrypted, then the File Key Management plugin requires both the `file_key_management_filename` and the `file_key_management_filekey` system variables to be configured.

The `file_key_management_filekey` system variable can be provided in two forms:

- It can be the actual plain-text encryption password. This is not recommended, since the plain-text encryption password would be visible in the output of the `SHOW VARIABLES` statement.
- If it is prefixed with `FILE:`, then it can be a path to a file that contains the plain-text encryption password.

These system variables can be specified as command-line arguments to `mysqld` or they can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
loose_file_key_management_filename = /etc/mysql/encryption/keyfile.enc
loose_file_key_management_filekey = FILE:/etc/mysql/encryption/keyfile.key
```

Note that the `loose` option prefix is specified. This option prefix is used in case the plugin hasn't been installed yet.

Choosing an Encryption Algorithm

The File Key Management plugin currently supports two encryption algorithms for encrypting data: `AES_CBC` and `AES_CTR`. Both of these algorithms use Advanced Encryption Standard (AES) in different modes. AES uses 128-bit blocks, and supports 128-bit, 192-bit, and 256-bit keys. The modes are:

- The `AES_CBC` mode uses AES in the Cipher Block Chaining (CBC) mode.
- The `AES_CTR` mode uses AES in two slightly different modes in different contexts. When encrypting tablespace pages (such as pages in InnoDB, XtraDB, and Aria tables), it uses AES in the Counter (CTR) mode. When encrypting temporary files (where the cipher text is allowed to be larger than the plain text), it uses AES in the authenticated Galois/Counter Mode (GCM).

The recommended algorithm is `AES_CTR`, but this algorithm is only available when MariaDB is built with recent versions of OpenSSL. If the server is built with wolfSSL or yaSSL, then this algorithm is not available. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.

Configuring the Encryption Algorithm

The encryption algorithm can be configured by setting the `file_key_management_encryption_algorithm` system variable.

This system variable can be set to one of the following values:

System Variable Value	Description
`AES_CBC`	Data is encrypted using AES in the Cipher Block Chaining (CBC) mode. This is the default value.
`AES_CTR`	Data is encrypted using AES either in the Counter (CTR) mode or in the authenticated Galois/Counter Mode (GCM) mode, depending on context. This is only supported in some builds. See the previous section for more information.

This system variable can be specified as command-line arguments to `mysqld` or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
loose_file_key_management_encryption_algorithm = AES_CTR
```

Note that the `loose` option prefix is specified. This option prefix is used in case the plugin hasn't been installed yet.

> Note that this variable does not affect the algorithm that MariaDB uses to decrypt the key file. This variable only affects the encryption algorithm that MariaDB uses to encrypt and decrypt data. The only algorithm that MariaDB currently supports to encrypt the key file is Cipher Block Chaining (CBC) mode of Advanced Encryption Standard (AES).

Using the File Key Management Plugin

Once the File Key Management Plugin is enabled, you can use it by creating an encrypted table:

```
CREATE TABLE t (i int) ENGINE=InnoDB ENCRYPTED=YES
```

Now, table `t` will be encrypted using the encryption key from the key file.

For more information on how to use encryption, see Data at Rest Encryption.

Using Multiple Encryption Keys

The File Key Management Plugin supports using multiple encryption keys. Each encryption key can be defined with a different 32-bit integer as a key identifier.

When encrypting InnoDB tables, the key that is used to encrypt tables can be changed.

When encrypting Aria tables, the key that is used to encrypt tables cannot currently be changed.

Key Rotation

The File Key Management plugin does not currently support key rotation. See MDEV-20713 for more information.

Versions

Version	Status	Introduced
1.0	Stable	MariaDB 10.1.18
1.0	Gamma	MariaDB 10.1.13

1.0	Alpha	MariaDB 10.1.3

System Variables

file_key_management_encryption_algorithm

- **Description:** This system variable is used to determine which algorithm the plugin will use to encrypt data.
  - The recommended algorithm is `AES_CTR`, but this algorithm is only available when MariaDB is built with recent versions of OpenSSL. If the server is built with wolfSSL or yaSSL, then this algorithm is not available. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.
- **Commandline:** `--file-key-management-encryption-algorithm=value`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `enumerated`
- **Default Value:** `AES_CBC`
- **Valid Values:** `AES_CBC`, `AES_CTR`

---

file_key_management_filekey

- **Description:** This system variable is used to determine the encryption password that is used to decrypt the key file defined by `file_key_management_filename`.
  - If this system variable's value is prefixed with `FILE:`, then it is interpreted as a path to a file that contains the plain-text encryption password.
  - If this system variable's value is **not** prefixed with `FILE:`, then it is interpreted as the plain-text encryption password. However, this is not recommended.
  - The encryption password has a max length of 256 characters.
  - The only algorithm that MariaDB currently supports when decrypting the key file is Cipher Block Chaining (CBC) mode of Advanced Encryption Standard (AES). The encryption key size can be 128-bits, 192-bits, or 256-bits. The encryption key is calculated by taking a SHA-1 hash of the encryption password.
- **Commandline:** `--file-key-management-filekey=value`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** (empty)

---

file_key_management_filename

- **Description:** This system variable is used to determine the path to the file that contains the encryption keys. If `file_key_management_filekey` is set, then this file can be encrypted with Cipher Block Chaining (CBC) mode of Advanced Encryption Standard (AES).
- **Commandline:** `--file-key-management-filename=value`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** (empty)

Options

file_key_management

- **Description:** Controls how the server should treat the plugin when the server starts up.
  - Valid values are:
    - `OFF` - Disables the plugin without removing it from the `mysql.plugins` table.
    - `ON` - Enables the plugin. If the plugin cannot be initialized, then the server will still continue starting up, but the plugin will be disabled.
    - `FORCE` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error.

        - `FORCE_PLUS_PERMANENT` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error. In addition, the plugin cannot be uninstalled with `UNINSTALL SONAME` or `UNINSTALL PLUGIN` while the server is running.
    - See Plugin Overview: Configuring Plugin Activation at Server Startup for more information.
- **Commandline:** `--file-key-management=value`
- **Data Type:** `enumerated`
- **Default Value:** `ON`
- **Valid Values:** `OFF`, `ON`, `FORCE`, `FORCE_PLUS_PERMANENT`

5.4.8.3 Hashicorp Key Management Plugin

MariaDB starting with 10.9

The Hashicorp Key Management Plugin is used to implement encryption using keys stored in the Hashicorp Vault KMS. For more information, see Hashicorp Vault and MariaDB, and for how to install Vault, see Install Vault, as well as MySQL/MariaDB Database Secrets Engine.

Contents

The current version of this plugin implements the following features:

- Authentication is done using the Hashicorp Vault's token authentication method;
- If additional client authentication is required, then the path to the CA authentication bundle file may be passed as a plugin parameter;
- The creation of the keys and their management is carried out using the Hashicorp Vault KMS and their tools;
- The plugin uses libcurl (https) as an interface to the HashiCorp Vault server;
- JSON parsing is performed through the JSON service (through the include/mysql/service_json.h);
- HashiCorp Vault 1.2.4 was used for development and testing.

Since we require support for key versioning, the key-value storage must be configured in Hashicorp Vault as a key-value storage that uses the interface of the second version. For example, you can create it as follows:

```
~$ vault secrets enable -path /test -version=2 kv
```

Key names must correspond to their numerical identifiers. Key identifiers itself, their possible values and rules of use are described in more detail in the MariaDB main documentation.

From the point of view of the key-value storage (in terms of Hashicorp Vault), the key is a secret containing one key-value pair with the name "data" and a value representing a binary string containing the key value, for example:

```
~$ vault kv get /test/1

====== Metadata ======
Key              Value
---              -----
created_time     2019-12-14T14:19:19.42432951Z
deletion_time    n/a
destroyed        false
version          1

==== Data ====
Key     Value
---     -----
data    0123456789ABCDEF0123456789ABCDEF
```

Keys values are strings containing binary data. MariaDB currently uses the AES algorithm with 256-bit keys as the default encryption method. In this case, the keys that will be stored in the Hashicorp Vault should be 32-byte strings. Most likely you will use some utilities for creating and administering keys designed to work with Hashicorp Vault. But in the simplest case, keys can be created from the command line through the vault utility, for example, as follows:

```
~$ vault kv put /test/1 data="0123456789ABCDEF0123456789ABCDEF"
```

If you use default encryption (AES), you should ensure that the key length is 32 bytes, otherwise it may fail to use InnoDB as a data storage.

The plugin currently does not unseal Hashicorp Vault on its own, you must do this in advance and on your own.

To use Hashicorp Vault KMS, the plugin must be preloaded and activated on the server. Most of its parameters should not be changed during plugin operation and therefore must be preconfigured as part of the server configuration through configuration file or command line options:

```
--plugin-load-add=hashicorp_key_management.so
--loose-hashicorp-key-management
--loose-hashicorp-key-management-vault-url="$VAULT_ADDR/v1/test"
--loose-hashicorp-key-management-token="$VAULT_TOKEN"
```

Options

The plugin supports the following parameters, which must be set in advance and cannot be changed during server operation:

`hashicorp-key-management-vault-url`

- **Description:** HTTP[s] URL that is used to connect to the Hashicorp Vault server. It must include the name of the scheme (`https://` for a secure connection) and, according to the API rules for storages of the key-value type in Hashicorp Vault, after the server address, the path must begin with the "/v1/" string (as prefix), for example: `https://127.0.0.1:8200/v1/my_secrets` . By default, the path is not set, therefore you must replace with the correct path to your secrets.
- **Commandline:** `--[loose-]hashicorp-key-management-vault-url="<url>"`

`hashicorp-key-management-token`

- **Description:** Authentication token that passed to the Hashicorp Vault in the request header. By default, this parameter contains an empty string, so you must specify the correct value for it, otherwise the Hashicorp Vault server will refuse authorization.
- **Commandline:** `--[loose-]hashicorp-key-management-token="<token>"`

`hashicorp-key-management-vault-ca`

- **Description:** Path to the Certificate Authority (CA) bundle (is a file that contains root and intermediate certificates). By default, this parameter contains an empty string, which means no CA bundle.
- **Commandline:** `--[loose-]hashicorp-key-management-vault-ca="<path>"`

`hashicorp-key-management-timeout`

- **Description:** Set the duration (in seconds) for the Hashicorp Vault server connection timeout. The default value is 15 seconds. The allowed range is from 1 to 86400 seconds. The user can also specify a zero value, which means the default timeout value set by the libcurl library (currently 300 seconds).
- **Commandline:** `--[loose-]hashicorp-key-management-timeout=<timeout>`

`hashicorp-key-management-retries`

- **Description:** Number of server request retries in case of timeout. Default is three retries.
- **Commandline:** `----[loose-]hashicorp-key-management-retries=<retries>`

`hashicorp-key-management-caching-enabled`

- **Description:** Enable key caching (storing key values received from the Hashicorp Vault server in the local memory). By default caching is enabled.
- **Commandline:** `--[loose-]hashicorp-key-management-caching-enabled="on"|"off"`

`hashicorp-key-management-use-cache-on-timeout`

- **Description:** This parameter instructs the plugin to use the key values or version numbers taken from the cache in the event of a timeout when accessing the vault server. By default this option is disabled. Please note that key values or version numbers will be read from the cache when the timeout expires only after the number of attempts to read them from the storage server that specified by the --[loose-]hashicorp-key-management-retries parameter has been exhausted.
- **Commandline:** `--[loose-]hashicorp-key-management-use-cache-on-timeout="on"|"off"`

`hashicorp-key-management-cache-timeout`

- **Description:** The time (in milliseconds) after which the value of the key stored in the cache becomes invalid and an attempt to read this data causes a new request send to the vault server. By default, cache entries become invalid after 60,000 milliseconds (after one minute). If the value of this parameter is zero, then the keys will always be considered invalid, but they still can be used if the vault server is unavailable and the corresponding cache operating mode (--[loose-]hashicorp-key-management-use-cache-on-timeout="on") is enabled.
- **Commandline:** `--[loose-]hashicorp-key-management-cache-timeout=<timeout>`

`hashicorp-key-management-cache-version-timeout`

- **Description:** The time (in milliseconds) after which the information about latest version number of the key (which stored in the cache) becomes invalid and an attempt to read this information causes a new request send to the vault server. If the value of this parameter is zero, then information about latest key version numbers always considered invalid, unless there is no communication with the vault server and use of the cache is allowed when the server is unavailable. By default, this parameter is zero, that is, the latest version numbers for the keys stored in the cache are considered always invalid, except when the vault server is unavailable and use of the cache is allowed on server failures.
- **Commandline:** `--[loose-]hashicorp-key-management-cache-version-timeout=<timeout>`

`hashicorp-key-management-check-kv-version`

- **Description:** This parameter enables ("on", this is the default value) or disables ("off") checking the kv storage version during plugin initialization. The plugin requires storage to be version 2 or older in order for it to work properly.
- **Commandline:** `--[loose-]hashicorp-key-management-check-kv-version="on"|"off"`

5.4.8.4 AWS Key Management Encryption Plugin

Contents

Due to license incompatibilities between the MariaDB server source code and the Amazon AWS C++ SDK we can only

distribute the plugin in source code form, and not as ready-to-use binaries. See Installing the Plugin's Package for details.

MariaDB's data-at-rest encryption requires the use of a key management and encryption plugin. These plugins are responsible both for the management of encryption keys and for the actual encryption and decryption of data.

MariaDB supports the use of multiple encryption keys. Each encryption key uses a 32-bit integer as a key identifier. If the specific plugin supports key rotation, then encryption keys can also be rotated, which creates a new version of the encryption key.

The AWS Key Management plugin is a key management and encryption plugin that uses the Amazon Web Services (AWS) Key Management Service (KMS).

Overview

The AWS Key Management plugin uses the Amazon Web Services (AWS) Key Management Service (KMS) to generate and store AES keys on disk, in encrypted form, using the Customer Master Key (CMK) kept in AWS KMS. When MariaDB Server starts, the plugin will decrypt the encrypted keys, using the AWS KMS "Decrypt" API function. MariaDB data will then be encrypted and decrypted using the AES key. It supports multiple encryption keys. It supports key rotation.

Tutorials

Tutorials related to the AWS Key Management plugin can be found at the following pages:

- Amazon Web Services (AWS) Key Management Service (KMS) Encryption Plugin Setup Guide
- Amazon Web Services (AWS) Key Management Service (KMS) Encryption Plugin Advanced Usage

Preparation

- Before you use the plugin, you need to create a Customer Master Key (CMK). Create a key using the AWS Console as described in the AMS KMS developer guide.
- The easiest way to give the AWS key management plugin access to the key is to create an IAM Role with access to the key, and to apply that IAM Role to an EC2 instance where MariaDB Server runs.
- Make sure that MariaDB Server runs under the correct AWS identity that has access to the above key. For example, you can store the AWS credentials in a AWS credentials file for the user who runs `mysqld`. More information about the credentials file can be found in the AWS CLI Getting Started Guide.

Installing the Plugin's Package

The AWS Key Management plugin depends on the AWS SDK for C++, which uses the Apache License, Version 2.0. This license is not compatible with MariaDB Server's GPL 2.0 license, so we are not able to distribute packages that contain the AWS Key Management plugin. Therefore, the only way to currently obtain the plugin is to install it from source.

Installing from Source

When compiling MariaDB from source, the AWS Key Management plugin is not built by default in MariaDB 10.1, but it is built by default in MariaDB 10.2 and later, on systems that support it.

Compilation is controlled by the following `cmake` arguments:

- `-DPLUGIN_AWS_KEY_MANAGEMENT=DYNAMIC` to build a loadable plugin library
- `-DAWS_SDK_EXTERNAL_PROJECT=ON` to download the AWS C++ SDK code
- `-DNOT_FOR_DISTRIBUTION=ON` to confirm that you know to not distribute the resulting binaries

The plugin uses AWS C++ SDK, which introduces the following restrictions:

- The plugin can only be built on Windows, Linux and macOS.
- The plugin requires that one of the following compilers is used: `gcc` 4.8 or later, `clang` 3.3 or later, Visual Studio 2013 or later.
- On Unix, the `libcurl` development package (e.g. `libcurl3-dev` on Debian Jessie), `uuid` development package and `openssl` need to be installed.
- You may need to use a newer version of `cmake` than is provided by default in your OS.

You do not need to download / install the AWS C++ SDK yourself, the correct version of the SDK github repository will be cloned into the build directory at build time, and only the libraries for AWS components actually needed by the key management plugin will be built, which takes much less time than building the full AWS C++ SDK.

Building on Linux

With all prerequisites installed the actual build process pretty much comes down to:

```
# clone the MariaDB Server source code repository
git clone https://github.com/MariaDB/server.git
cd server

# prepare the build
mkdir _build
cd _build
cmake .. -DNOT_FOR_DISTRIBUTION=ON \
  -DPLUGIN_AWS_KEY_MANAGEMENT=DYNAMIC \
  -DAWS_SDK_EXTERNAL_PROJECT=1

# build the plugin only
cd plugin/aws_key_management
make
```

Cmake will print the following warning as part of its output, please take it serious and do not distribute the `aws_key_management.so` file to any third parties:

You have linked MariaDB with Apache 2.0 libraries! You may not distribute the resulting binary. If you do, you will put yourself into a legal problem with the Free Software Foundation.

After `make` succeeded you can copy the created `aws_key_management.so` plugin library file to the plugin directory of your actual MariaDB Server machines installation, e.g. `/usr/lib64/mysql/plugin` on RedHat/Fedora based systems or `/usr/lib/mysql/plugin` on Debian based systems.

Installing the Plugin

Even after the package that contains the plugin's shared library is installed on the operating system, the plugin is not actually installed by MariaDB by default. There are two methods that can be used to install the plugin with MariaDB.

The first method can be used to install the plugin without restarting the server. You can install the plugin dynamically by executing `INSTALL SONAME` or `INSTALL PLUGIN`. For example:

```
INSTALL SONAME 'aws_key_management';
```

The second method can be used to tell the server to load the plugin when it starts up. The plugin can be installed this way by providing the `--plugin-load` or the `--plugin-load-add` options. This can be specified as a command-line argument to `mysqld` or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
plugin_load_add = aws_key_management
```

Uninstalling the Plugin

Before you uninstall the plugin, you should ensure that data-at-rest encryption is completely disabled, and that MariaDB no longer needs the plugin to decrypt tables or other files.

You can uninstall the plugin dynamically by executing `UNINSTALL SONAME` or `UNINSTALL PLUGIN`. For example:

```
UNINSTALL SONAME 'aws_key_management';
```

If you installed the plugin by providing the `--plugin-load` or the `--plugin-load-add` options in a relevant server option group in an option file, then those options should be removed to prevent the plugin from being loaded the next time the server is restarted.

Configuring the AWS Key Management Plugin

To enable the AWS Key Management plugin, you also need to set the plugin's system variables. The `aws_key_management_master_key_id` system variable is the primary one to set. These system variables can be

specified as command-line arguments to `mysqld` or they can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
aws_key_management_master_key_id=alias/<your key's alias>
```

Once you've updated the configuration file, restart the MariaDB server to apply the changes and make the key management and encryption plugin available for use.

Using the AWS Key Management Plugin

Once the AWS Key Management Plugin is enabled, you can use it by creating an encrypted table:

```
CREATE TABLE t (i int) ENGINE=InnoDB ENCRYPTED=YES
```

Now, table `t` will be encrypted using the encryption key generated by AWS.

For more information on how to use encryption, see Data at Rest Encryption.

Using Multiple Encryption Keys

The AWS Key Management Plugin supports using multiple encryption keys. Each encryption key can be defined with a different 32-bit integer as a key identifier. If a previously unused identifier is used, then the plugin will automatically generate a new key.

When encrypting InnoDB tables, the key that is used to encrypt tables can be changed.

When encrypting Aria tables, the key that is used to encrypt tables cannot currently be changed.

Key Rotation

The AWS Key Management plugin does support key rotation. To rotate a key, set the `aws_key_management_rotate_key` system variable. For example, to rotate key with ID 2:

```
SET GLOBAL aws_key_management_rotate_key=2;
```

Or to rotate all keys, set the value to -1:

```
SET GLOBAL aws_key_management_rotate_key=-1;
```

Versions

Version	Status	Introduced
1.0	Stable	MariaDB 10.2.6, MariaDB 10.1.24
1.0	Beta	MariaDB 10.1.18
1.0	Experimental	MariaDB 10.1.13

System Variables

`aws_key_management_key_spec`

- **Description:** Encryption algorithm used to create new keys
- **Commandline:** `--aws-key-management-key-spec=value`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `enumerated`
- **Default Value:** `AES_128`
- **Valid Values:** `AES_128` , `AES_256`

aws_key_management_log_level

- **Description:** Dump log of the AWS SDK to MariaDB error log. Permitted values, in increasing verbosity, are **Off** (default), **Fatal**, **Error**, **Warn**, **Info**, **Debug**, and **Trace**.
- **Commandline:** `--aws-key-management-log-level=value`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `enumerated`
- **Default Value:** `Off`
- **Valid Values:** `Off`, `Fatal`, `Warn`, `Info`, `Debug` and `Trace`

aws_key_management_master_key_id

- **Description:** AWS KMS Customer Master Key ID (ARN or alias prefixed by alias/) for the master encryption key. Used to create new data keys. If not set, no new data keys will be created.
- **Commandline:** `--aws-key-management-master-key-id=value`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:**

aws_key_management_mock

- **Description:** Mock AWS KMS calls (for testing). Must be enabled at compile-time.
- **Commandline:** `--aws-key-management-mock`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Valid Values:** `OFF`, `ON`

aws_key_management_region

- **Description:** AWS region name, e.g us-east-1 . Default is SDK default, which is us-east-1.
- **Commandline:** `--aws-key-management-region=value`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** `'us-east-1'`

aws_key_management_request_timeout

- **Description:** Timeout in milliseconds for create HTTPS connection or execute AWS request. Specify 0 to use SDK default.
- **Commandline:** `--aws-key-management-request-timeout=value`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `integer`
- **Default Value:** 0

aws_key_management_rotate_key

- **Description:** Set this variable to a data key ID to perform rotation of the key to the master key given in `aws_key_management_master_key_id` . Specify -1 to rotate all keys.
- **Commandline:** `--aws-key-management-rotate-key=value`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `integer`

- **Default Value:**

Options

`aws_key_management`

- **Description:** Controls how the server should treat the plugin when the server starts up.
  - Valid values are:
    - `OFF` - Disables the plugin without removing it from the `mysql.plugins` table.
    - `ON` - Enables the plugin. If the plugin cannot be initialized, then the server will still continue starting up, but the plugin will be disabled.
    - `FORCE` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error.
    - `FORCE_PLUS_PERMANENT` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error. In addition, the plugin cannot be uninstalled with `UNINSTALL SONAME` or `UNINSTALL PLUGIN` while the server is running.
  - See Plugin Overview: Configuring Plugin Activation at Server Startup for more information.
- **Commandline:** `--aws-key-management=value`
- **Data Type:** `enumerated`
- **Default Value:** `ON`
- **Valid Values:** `OFF` , `ON` , `FORCE` , `FORCE_PLUS_PERMANENT`

5.4.8.5 Amazon Web Services (AWS) Key Management Service (KMS) Encryption Plugin Setup Guide

Contents

Overview

MariaDB contains a robust, full instance, at-rest encryption. This feature uses a flexible plugin interface to allow actual encryption to be done using a key management approach that meets the customer's needs. MariaDB Server, starting with MariaDB 10.2, includes a plugin that uses the Amazon Web Services (AWS) Key Management Service (KMS) to facilitate separation of responsibilities and remote logging & auditing of key access requests.

Rather than storing the encryption key in a local file, this plugin keeps the master key in AWS KMS. When you first start MariaDB, the AWS KMS plugin will connect to the AWS Key Management Service and ask it to generate a new key. MariaDB will store that key on-disk in an encrypted form. The key stored on-disk cannot be used to decrypt the data; rather, on each startup, MariaDB connects to AWS KMS and has the service decrypt the locally-stored key(s). The decrypted key is stored in-memory as long as the MariaDB server process is running, and that in-memory decrypted key is used to encrypt the local data.

This guide is based on CentOS 7, using systemd with SELinux enabled. Some steps will differ if you use other operating systems or configurations.

Installing the Plugin's Package

The AWS Key Management plugin depends on the AWS SDK for C++, which uses the Apache License, Version 2.0. This license is not compatible with MariaDB Server's GPL 2.0 license, so we are not able to distribute packages that contain the AWS Key Management plugin. Therefore, the only way to currently obtain the plugin is to install it from source.

Installing from Source

When compiling MariaDB from source, the AWS Key Management plugin is not built by default in MariaDB 10.1, but it is built by default in MariaDB 10.2 and later, on systems that support it.

Compilation is controlled by the `-DPLUGIN_AWS_KEY_MANAGEMENT=DYNAMIC -DAWS_SDK_EXTERNAL_PROJECT=1` cmake arguments.

The plugin uses AWS C++ SDK, which introduces the following restrictions:

- The plugin can only be built on Windows, Linux and macOS.
- The plugin requires that one of the following compilers is used: `gcc` 4.8 or later, `clang` 3.3 or later, Visual Studio 2013 or later.
- On Unix, the `libcurl` development package (e.g. `libcurl3-dev` on Debian Jessie), `uuid` development package and `openssl` need to be installed.
- You may need to use a newer version of cmake than is provided by default in your OS.

Installing the Plugin

Even after the package that contains the plugin's shared library is installed on the operating system, the plugin is not actually installed by MariaDB by default. There are two methods that can be used to install the plugin with MariaDB.

The first method can be used to install the plugin without restarting the server. You can install the plugin dynamically by executing INSTALL SONAME or INSTALL PLUGIN. For example:

```
INSTALL SONAME 'aws_key_management';
```

The second method can be used to tell the server to load the plugin when it starts up. The plugin can be installed this way by providing the --plugin-load or the --plugin-load-add options. This can be specified as a command-line argument to mysqld or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
plugin_load_add = aws_key_management
```

Sign up for Amazon Web Services

If you already have an AWS account, you can skip this section.

1. Load http://aws.amazon.com/.
2. Click "Create a Free Account" and complete the steps.
3. You'll need to enter credit card information. Charges related only to your use of the AWS KMS service should be limited to about $1/month for the single master key we will create. If you use other services, additional charges may apply. Consult AWS Cloud Pricing Principles https://aws.amazon.com/pricing/ for more information about pricing of AWS services.
4. You'll need to complete the AWS identify verification process.

Create an IAM User and/or Role

After creating an account or logging in to an existing account, follow these steps to create an IAM User or Role with restricted privileges that will use (but not administer) your master encryption key.

If you intend to run MariaDB Server on an EC2 instance, you should create a Role (or modify an existing Role already attached to your instance). If you intent to run MariaDB Server outside of AWS, you may want to create a User.

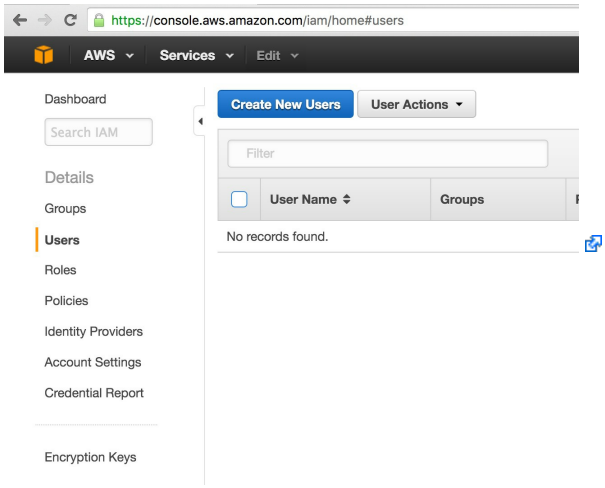
Creating an IAM Role

1. Load the Identity and Access Management Console at https://console.aws.amazon.com/iam/.
2. Click "Roles" in the left-hand sidebar
3. Click "Create new role"

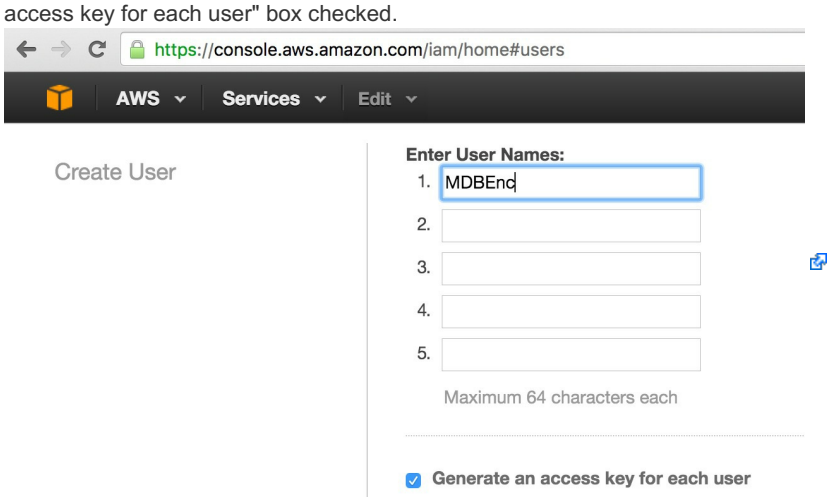
4. Select "AWS Service Role"
5. Click the "Select" button next to "Amazon EC2 / Allows EC2 instances to call AWS services on your behalf."
6. Do not select any policies on the "Attach Policy" screen. Click "Next Step"
7. Click "Next Step"
8. Give your Role a "Role name"
9. Click "Create role"

Creating an IAM User

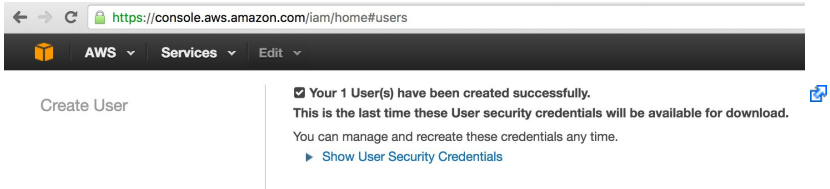
1. Load the Identity and Access Management Console at https://console.aws.amazon.com/iam/.
2. Click "Users" in the left-hand sidebar.



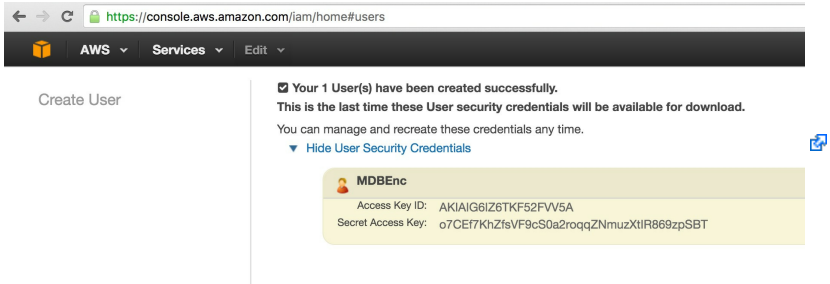
3. Click the "Create New Users" button
4. Enter a single User Name of your choosing. We'll use "MDBEnc" for this demonstration. Keep the "Generate an access key for each user" box checked.



5. Click "Create".
6. Click "Show User Security Credentials".



7. Copy the Access Key ID and Secret Access Key. Optionally, you can click "Download Credentials". We will need these in order for local programs to interact with AWS using its API.



8. Create a file on your computer to hold the credentials for this user. We'll use this file later. It should have this structure:

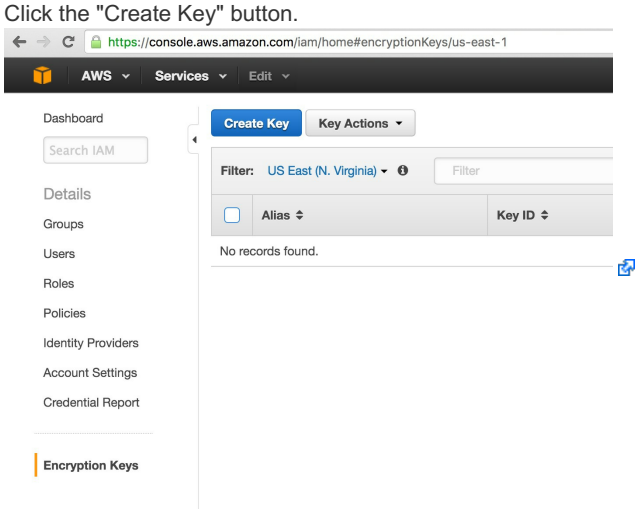
```
[default]
aws_access_key_id = AKIAIG6IZ6TKF52FVV5A
aws_secret_access_key = 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
```

9. Click "Close". If prompted because you did not Download Credentials, ensure that you've saved them somewhere, and click "Close".

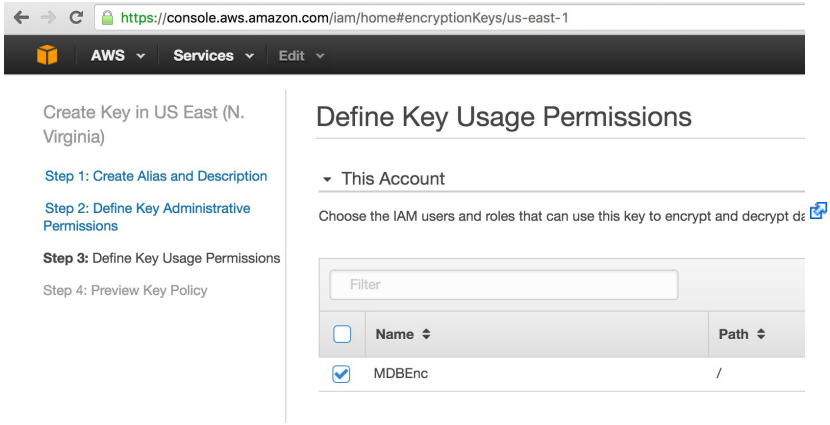
Create a Master Encryption Key

Now, we'll create a master encryption key. This key can *never* be retrieved by *any* application or user. This key is used remotely to encrypt (and decrypt) the actual encryption keys that will be used by MariaDB. If this key is deleted or you lose access to it, you will be unable to use the contents of your MariaDB data directory.

1. Click "Encryption Keys" in the left-hand sidebar.
2. Click the "Get Started Now" button.
3. Use the "Filter" dropdown to choose the region where you'd like to create your master key.
4. Click the "Create Key" button.

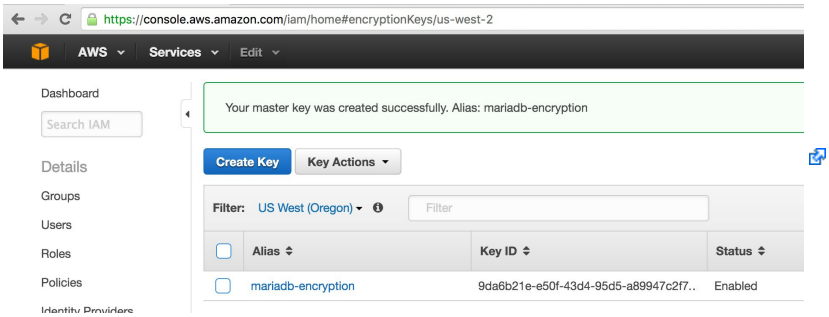


5. Enter an Alias and Description of your choosing.
6. Click "Next Step".
7. Do **not** check the box to make your IAM Role or IAM User user a Key Administrator.
8. Click "Next Step" again.
9. Check the boxes to give your IAM Role and/or IAM User permissions to use this key.



10. Click "Next Step".
11. Click "Finish".

You should now see your key listed in the console:



You'll use the "Alias" you provided when you configure MariaDB later.

We now have a Customer Master Key and an IAM user that has privileges to access it using access credentials. This is enough to begin using the AWS KMS plugin.

Configure AWS Credentials

There are a number of ways to give the IAM credentials to the AWS KMS plugin. The plugin supports reading credentials from all standard locations used across the various AWS API clients.

The easiest approach is to run MariaDB Server in an EC2 instance that has an IAM Role with User access to the CMK you wish to use. You can give key access privileges to a Role already attached to your EC2 instance, or you can create a new IAM Role and attach it to an already-running EC2 instance. If you've done that, no further credentials management is required and you do not need to create a `credentials` file.

If you're not running MariaDB Server on an EC2 instance, you can also place the credentials in the MariaDB data directory. The AWS API client looks for a `credentials` file in the `.aws` subdirectory of the home directory of the user running the client process. In the case of MariaDB, its home directory is its `datadir`.

1. Create a `credentials` file that MariaDB can read. Use the region you selected when creating the key. Master keys cannot be used across regions. For example:

```
$ cat /var/lib/mysql/.aws/credentials
[default]
aws_access_key_id = AKIAIG6IZ6TKF52FVV5A
aws_secret_access_key = o7CEf7KhZfsVF9cS0a2roqqZNmuzXtIR869zpSBT
region = us-east-1
```

2. Change the permissions of the file so that it it is owned by, and can only be read by, the `mysql` user:

```
chown mysql /var/lib/mysql/.aws/credentials
chmod 600 /var/lib/mysql/.aws/credentials
```

Configure MariaDB

1. Create a new option file to tell MariaDB to enable encryption functionality and to use the AWS KMS plugin. Create a new file under `/etc/my.cnf.d/` (or wherever your OS may have you create such files) with contents like this:

```
[mariadb]
plugin_load_add = aws_key_management
aws-key-management = FORCE_PLUS_PERMANENT
aws-key-management-master-key-id = alias/mariadb-encryption
aws-key-management-region = us-east-1
!include /etc/my.cnf.d/enable_encryption.preset
```

1. Append the "Alias" value you copied above to `alias/` to use as the value for the `aws-key-management-master-key-id` option.

Note that you **must** include `aws-key-management-region` in your .cnf file if you are not using the us-east-1 region.

Now, you have told MariaDB to use the AWS KMS plugin and you've put credentials for the plugin in a location where the plugin will find them. The /etc/my.cnf.d/enable_encryption.preset file contains a set of options that enable all available encryption functionality.

When you start MariaDB, the AWS KMS plugin will connect to the AWS Key Management Service and ask it to generate a new key. MariaDB will store that key on-disk in an encrypted form. The key stored on-disk cannot be used to decrypt the data; rather, on each startup, MariaDB must connect to AWS KMS and have the service decrypt the locally-stored key. The decrypted version is stored in-memory as long as the MariaDB server process is running, and that in-memory decrypted key is used to encrypt the local data.

SELinux and Outbound Connections from MariaDB

Because MariaDB needs to connect to the AWS KMS service, you must ensure that the host has outbound network connectivity over port 443 to AWS and you must ensure that local policies allow the MariaDB server process to make those outbound connections. By default, SELinux restricts MariaDB from making such connections.

The most simple way to cause SELinux to allow outbound HTTPS connections from MariaDB is to enable to mysql_connect_any boolean, like this:

```
setsebool -P mysql_connect_any 1
```

There are more complex alternatives that have a more granular effect, but those are beyond the scope of this document.

Start MariaDB

Start MariaDB using the `systemctl` tool:

```
systemctl start mariadb
```

If you do not use systemd, you may have to start MariaDB using some other mechanism.

You should see journal output similar to this:

```
# journalctl --no-pager -o cat -u mariadb.service

[Note] /usr/sbin/mysqld (mysqld 10.1.9-MariaDB-enterprise-log) starting as process 19831 ...
[Note] AWS KMS plugin: generated encrypted datakey for key id=1, version=1
[Note] AWS KMS plugin: loaded key 1, version 1, key length 128 bit
[Note] InnoDB: Using mutexes to ref count buffer pool pages
[Note] InnoDB: The InnoDB memory heap is disabled
[Note] InnoDB: Mutexes and rw_locks use GCC atomic builtins
[Note] InnoDB: Memory barrier is not used
[Note] InnoDB: Compressed tables use zlib 1.2.7
[Note] InnoDB: Using CPU crc32 instructions
[Note] InnoDB: Initializing buffer pool, size = 2.0G
[Note] InnoDB: Completed initialization of buffer pool
[Note] InnoDB: Highest supported file format is Barracuda.
[Note] InnoDB: 128 rollback segment(s) are active.
[Note] InnoDB: Waiting for purge to start
[Note] InnoDB:  Percona XtraDB (http://www.percona.com) 5.6.26-74.0 started; log sequence number
[Note] InnoDB: Dumping buffer pool(s) not yet started
[Note] Plugin 'FEEDBACK' is disabled.
[Note] AWS KMS plugin: generated encrypted datakey for key id=2, version=1
[Note] AWS KMS plugin: loaded key 2, version 1, key length 128 bit
[Note] Using encryption key id 2 for temporary files
[Note] Server socket created on IP: '::'.
[Note] Reading of all Master_info entries succeded
[Note] Added new Master_info '' to hash table
[Note] /usr/sbin/mysqld: ready for connections.
```

Note the several lines of output that refer explicitly to the "AWS KMS plugin". You can see that the plugin generates a "datakey", loads that data key, and then later generates and loads a second data key. The 2nd data key is used to encrypt temporary files and temporary tables.

You can see the encrypted keys stored on-disk in the datadir:

```
# ls -l /var/lib/mysql/aws*
-rw-rw----. 1 mysql mysql 188 Feb 25 18:55 /var/lib/mysql/aws-kms-key.1.1
-rw-rw----. 1 mysql mysql 188 Feb 25 18:55 /var/lib/mysql/aws-kms-key.2.1
```

Note that those keys are not useful alone. They are encrypted. When MariaDB starts up, the AWS KMS plugin decrypts those keys by interacting with AWS KMS.

For maximum security, you should start from an empty datadir and run mariadb-install-db after configuring encryption. Then you should re-import your data so that it is fully encrypted. Use `sudo` to run `mariadb-install-db` so that it finds your credentials file:

```
# sudo -u mysql mariadb-install-db
Installing MariaDB/MySQL system tables in '/var/lib/mysql' ...
2016-02-25 23:16:06 139731553998976 [Note] /usr/sbin/mysqld (mysqld 10.1.11-MariaDB-enterprise-
log) starting as process 39551 ...
2016-02-25 23:16:07 139731553998976 [Note] AWS KMS plugin: generated encrypted datakey for key
id=1, version=1
2016-02-25 23:16:07 139731553998976 [Note] AWS KMS plugin: loaded key 1, version 1, key length
128 bit
...
```

Create Encrypted Tables

With `innodb-encrypt-tables=ON`, new InnoDB tables will be encrypted by default, using the key ID set in `innodb_default_encryption_key_id` (default 1). With `innodb-encrypt-tables=FORCE` enabled, it is not possible to manually bypass encryption when creating a table.

You can cause the AWS KMS plugin to create new encryption keys at-will by specifying a new ENCRYPTION_KEY_ID when creating a table:

```
MariaDB [test]> create table t1 (id serial, v varchar(32)) ENCRYPTION_KEY_ID=3;
Query OK, 0 rows affected (0.91 sec)
```

```
[Note] AWS KMS plugin: generated encrypted datakey for key id=3, version=1
[Note] AWS KMS plugin: loaded key 3, version 1, key length 128 bit
```

```
# ls -l /var/lib/mysql/aws*
-rw-rw----. 1 mysql mysql 188 Feb 25 18:55 /var/lib/mysql/aws-kms-key.1.1
-rw-rw----. 1 mysql mysql 188 Feb 25 18:55 /var/lib/mysql/aws-kms-key.2.1
-rw-rw----. 1 mysql mysql 188 Feb 25 19:10 /var/lib/mysql/aws-kms-key.3.1
```

Read more about encrypting data in the Data at Rest Encryption section of the MariaDB Documentation.

AWS KMS Plugin Option Reference

- `aws_key_management_master_key_id`: AWS KMS Customer Master Key ID (ARN or alias prefixed by `alias/`) for master encryption key. Used to create new data keys. If not set, no new data keys will be created.
- `aws_key_management_rotate_key`: Set this variable to a data key ID to perform rotation of the key to the master key given in `aws_key_management_master_key_id`. Specify -1 to rotate all keys.
- `aws_key_management_key_spec`: Encryption algorithm used to create new keys. Allowed values are AES_128 (default) or AES_256.
- `aws_key_management_log_level`: Logging for AWS API. Allowed values, in increasing verbosity, are "Off" (default), "Fatal", "Error", "Warn", "Info", "Debug", and "Trace".

Next Steps

For more information about advanced usage, including strategies to manage credentials, enforce separation of responsibilities, and even require 2-factor authentication to start your MariaDB server, please review Amazon Web Services (AWS) Key Management Service (KMS) Encryption Plugin Advanced Usage.

5.4.8.6 Amazon Web Services (AWS) Key Management Service (KMS) Encryption Plugin Advanced Usage

Contents

This document assumes you've already set up an Amazon Web Services (AWS) account, created a master key in the Key Management Service (KMS), and have done the basic work to set up the MariaDB AWS KMS plugin. These steps are all described in Amazon Web Services (AWS) Key Management Service (KMS) Encryption Plugin Setup Guide.

Ultimately, keeping all the credentials required to read the key on a single host means that a user who has gained access to the host has enough information to read the encrypted files in the datadir, read the encrypted keys from the datadir, interact with AWS KMS to decrypt the encrypted keys, and then used the decrypted keys to decrypt the encrypted data.

Theoretically, a superuser can read the memory of the MariaDB server process to read the decrypted keys or restart MariaDB with password authentication disabled in order to dump data, or add new users to MariaDB in order to allow a user to connect and dump the data. Resolving these issues is beyond the scope of this document. A user who gains root access to your operating system or root access to your MariaDB server will have the ability to decrypt your data. Plan accordingly.

Managing AWS credentials

Putting the AWS credentials in a file inside the MariaDB home directory is not ideal. By default, any user with the FILE privilege can read any files that the MariaDB server has permission to read, which would include the credentials file. To protect against this, you should set `secure_file_priv` to restrict the location the server will allow a user to read from when executing `LOAD DATA INFILE` or the `LOAD_FILE()` function.

But putting them in other locations requires passing additional data to the server, which in the case of CentOS 7 requires customizing the systemd startup procedure. This is most easily done by creating a "drop-in" file in /etc/systemd/system/mariadb.service.d/. The file should end in ".conf" but can otherwise be named whatever you like. After making any changes to systemd files, execute `systemctl daemon-reload` and then start (or restart) the service as usual.

You can place the credentials file in a location of your choosing and then refer to that file by setting the `AWS_CREDENTIAL_PROFILES_FILE` environment variable in the drop-in file:

```
[Service]
Environment=AWS_CREDENTIAL_PROFILES_FILE=/etc/aws-kms-credentials
```

The credentials file will need to be readable by the "mysql" user, but it does not need to be readable by any other user.

AWS credentials can also be put directly into a "drop-in" systemd file that will be read when starting the MariaDB service:

```
# cat /etc/systemd/system/mariadb.service.d/aws-kms.conf
[Service]
Environment=AWS_ACCESS_KEY_ID=AKIAIRSG2XYZATCJLZ4A
Environment=AWS_SECRET_ACCESS_KEY=ux91LZIxCp4ZXabcdefgIViQNtTan42QAmJqJVqV
```

However, any OS user can read this information from systemd, which could be considered a security risk. Another solution is to put the credentials in a separate file that is only readable by root and then refer to that file using an `EnvironmentFile` directive in a drop-in systemd file.

```
# cat /etc/systemd/system/mariadb.service.d/aws-kms.env
AWS_ACCESS_KEY_ID=AKIAIRSG2XYZATCJLZ4A
AWS_SECRET_ACCESS_KEY=ux91LZIxCp4ZXabcdefgIViQNtTan42QAmJqJVqV

# chown root /etc/systemd/system/mariadb.service.d/aws-kms.env
# chmod 600 /etc/systemd/system/mariadb.service.d/aws-kms.env

# cat /etc/systemd/system/mariadb.service.d/aws-kms.conf
[Service]
EnvironmentFile=/etc/systemd/system/mariadb.service.d/aws-kms.env
```

That has the advantage the the credentials can only be read directly by root. systemd adds those variables to the environment of the MariaDB server when starting it, and MariaDB can use the credentials to interact with AWS. Note, though, that any process running as the "mysql" user can still read the credentials from the proc filesystem on Linux.

```
$ whoami
mysql
$ cat /proc/$(pgrep mysqld)/environ | tr '\0' '\n' | grep AWS
AWS_ACCESS_KEY_ID=AKIAIRSG2XYZATCJLZ4A
AWS_SECRET_ACCESS_KEY=ux91LZIxCp4ZXabcdefgIViQNtTan42QAmJqJVqV
```

AWS KMS Key Policy

AWS KMS allows flexible, user-editable key policy. This offers fine-grained control over which users can operate on keys. The possibilities range from simply restricting which IP addresses are allowed to perform operations on the key, to requiring a MFA (Multi-Factor Authentication) device to use the key, to enforcing separation of responsibilities by creating an additional user with limited privileges to enable and disable the key. All 3 of these options will be outlined in this section.

For more details about customizing the Key Policy for your master keys, please consult the AWS Key Management Service Key Policy documentation.

Source IP restrictions

A simple, common-sense restriction to put in place is to restrict the range of IP addresses that are allowed to use your master key. This way, even if someone obtains API credentials, they'll be unable to use them to decrypt your encryptions keys from a different host.

To restrict API access from only a specific IP address or range of IP addresses, you'll need to manually edit the key policy.

1. Load the IAM console at https://console.aws.amazon.com/iam/.
2. Click "Encryption Keys" in the left-hand sidebar.
3. Click the name of your encryption key to view its details.
4. Click the link labeled "Switch to policy view", to the right of the heading of the "Key Policy" section.
5. Locate the section that contains `"Sid": "Allow use of the key"`.
6. Add this text below the `"Sid"` line:

```
"Condition": {
  "IpAddress": {
    "aws:SourceIp": [
      "10.1.2.3/32"
    ]
  }
},
```

   ... replacing `10.1.2.3/32` above with an IP address or range of IP addresses in CIDR format. For example, a single address would be `192.168.12.34/32`, while a range of addresses might be `192.168.0.0/24`.
7. Click "Save Changes".
8. Click "Proceed" if prompted with a warning about using the default view in the future.

Access to the API will now be restricted to requests coming from the IP address or range of IP addresses specified in the policy.

Using a Multi-Factor Authentication (MFA) device

One approach is to modify the key policy for the master key so that MFA (Multi-Factor Authentication) is required in order to use the key. This is achieved with a wrapper that handles prompting the user for an MFA token, acquires temporary, limited-privilege credentials from the AWS Security Token Service (STS), and puts those credentials into the environment of the MariaDB server process. The credentials can expire after as little as 15 minutes.

To require an MFA token for users of the key, you'll need to manually edit the key policy.

1. Load the IAM console at https://console.aws.amazon.com/iam/.
2. Click "Encryption Keys" in the left-hand sidebar.
3. Click the name of your encryption key to view its details.
4. Click the link labeled "Switch to policy view", to the right of the heading of the "Key Policy" section.
5. Locate the section that contains `"Sid": "Allow use of the key"`.
6. Add this text below the `"Sid"` line:

```
"Condition": {
  "Bool": {
    "aws:MultiFactorAuthPresent": "True"
  }
},
```

7. Click "Save Changes".
8. Click "Proceed" if prompted with a warning about using the default view in the future.

Now, add an MFA device for your user. You'll need to have a hardware MFA device or an application such as Google Authenticator installed on your smartphone.

1. Click "Users" in the left-hand sidebar.
2. Click the name of your user.
3. Click the "Security Credentials" tab.
4. In the "Sign-In Credentials" section, click the "Manage MFA Device" button.
5. Complete the steps to activate your MFA device.
6. Copy the ARN for your MFA device. You will need to use this when configuring the wrapper program.

Now, set up the wrapper program.

1. Copy the iam-kms-wrapper file to /usr/local/bin/, and ensure that it is executable.
2. Create a drop-in systemd config file in `/etc/systemd/system/mariadb.service.d/`:

```
[Service]
EnvironmentFile=/etc/systemd/system/mariadb.service.d/aws-kms.env
ExecStart=
ExecStart=/usr/local/bin/iam-kms-wrapper --config=/etc/my.cnf.d/iam-kms-wrapper.config /usr/
```

3. Execute `systemctl daemon-reload`.
4. Create a file at `/etc/my.cnf.d/iam-kms-wrapper.config` with these contents, using the ARN for your MFA device as the value for `kms_mfa_id`:

```
[kms]
kms_mfa_id = arn:aws:iam::551888187628:mfa/MDBEnc
kms_mfa_socket = /tmp/kms_mfa_socket
```

When you start the MariaDB service now, the wrapper will temporarily create a socket file at the location given by the `kms_mfa_socket` option. The wrapper will read the MFA code from the socket and will use it to authenticate to KMS. To give the MFA code, simply write the digits to the socket file using `echo`: `echo 111676 > /tmp/kms_mfa_socket`.

The `systemctl` command will block until MariaDB starts, so you will need to write the code to the socket file via a separate terminal.

Note that the temporary credentials put into the environment of the MariaDB process will expire after a period of time defined by the request to the AWS Security Token Service (STS). In the example below, they will expire after 900 seconds. After that time, MariaDB may be unable to generate new encrypted data keys, which means that, for example, an attempt to create a table with a previously-unused key ID would fail.

Wrapper program example

Here's an example wrapper program written in go. Build this into an executable named `iam-kms-wrapper` and use it as instructed above. This could of course be written in any language for which an appropriate AWS SDK exists, but go has the benefit of compiling to a static binary, which means you do not have to worry about interpreter versions or installing complex dependencies on the host that runs your MariaDB server.

```
package main

import (
    "syscall"
    "os"
    "log"
    "flag"
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/aws/awserr"
    "github.com/aws/aws-sdk-go/service/sts"
    "github.com/robfig/config"
)

func main() {

    config_file_p := flag.String("config", "", "location of the config file")
    flag.Parse()

    if flag.NArg() < 1 {
        log.Fatal("Command to wrap must be given as first command-line argument")
    }

    cmd := flag.Arg(0)
    args := flag.Args()[0:]

    conf, err := config.ReadDefault(*config_file_p)
    if err != nil {
        log.Fatal(err)
    }

    kms_mfa_id, err := conf.String("kms","kms_mfa_id")
    mfa_socket_file, err := conf.String("kms","kms_mfa_socket")

    sess := session.New()
    svc := sts.New(sess)

    syscall.Umask(0044)
```

```
    log.Printf("Reading MFA token from %s\n",mfa_socket_file)

    if err := syscall.Mknod(mfa_socket_file, syscall.S_IFIFO|uint32(os.FileMode(0622)), 0); err
!= nil {
        log.Fatal(err)
    }

    file, err := os.Open(mfa_socket_file)
    if err != nil {
        log.Fatal(err)
    }

    token := make([]byte, 6)

    if _, err := file.Read(token); err != nil {
        log.Fatal(err)
    }

    file.Close()

    if err := syscall.Unlink(mfa_socket_file); err != nil {
        log.Fatal(err)
    }

    mfa_token := string(token)

    token_params := &sts.GetSessionTokenInput{
        DurationSeconds: aws.Int64(900),
        SerialNumber:    aws.String(kms_mfa_id),
        TokenCode:       aws.String(mfa_token),
    }

    resp, err := svc.GetSessionToken(token_params)
    if err != nil {
        if awsErr, ok := err.(awserr.Error); ok {
            // Prints out full error message, including original error if there was one.
            log.Fatal("Error:", awsErr.Error())
        } else {
            log.Fatal("Error:", err.Error())
        }
    }

    creds := resp.Credentials

    os.Setenv("AWS_ACCESS_KEY_ID",*creds.AccessKeyId)
    os.Setenv("AWS_SECRET_ACCESS_KEY",*creds.SecretAccessKey)
    os.Setenv("AWS_SESSION_TOKEN",*creds.SessionToken)

    execErr := syscall.Exec(cmd, args, os.Environ())
    if execErr != nil {
        panic(execErr)
    }
}
```

Disabling keys when not needed

Another possibility is to use the API to disable access to the master key and enable it only when a trusted administrator knows that the MariaDB service needs to be started. A specialized tool on a separate host could be used to enable the key for a very short period of time while the service starts and then quickly disable the key.

To do this, you can create an extra IAM User that can only use the kms:EnableKey and kms:DisableKey API endpoints for your key. This user will not be able to encrypt or decrypt any data using the key.

First, create a new user.

1. Load the IAM console at https://console.aws.amazon.com/iam/.
2. Click "Users" in the left-hand sidebar.
3. Click "Create New Users".
4. Enter a new user name. (The examples will use "MDBEncAdmin".)
5. Click "Show User Security Credentials".
6. Copy the credentials and put them in a `credentials` file with this structure:

```
[MDBEncAdmin]
aws_access_key_id=AKIAJMPPNO7EBKABCDEF
aws_secret_access_key=pVdGwbuK5/jG64aBK1oEJOXRlkdM0aAylgabCDef
```

7. Click "Close".
8. Click "Close" again if prompted.
9. Click the name of your new user to open the details view.
10. Copy the "User ARN" value for your user (for example "arn:aws:iam::551888181234:user/MDBEncAdmin"). You will need this for the next step.

Now, give the new user permission to perform API operations on your key.

1. Click "Encryption Keys" in the left-hand sidebar.
2. Click the name of your key to open the details view.
3. Click "Switch to policy view" if it is not already open. (The "policy view" is a large text field that contains JSON describing the key policy.)
4. Create a new item in the `Statement` array with this structure:

```
    {
      "Sid": "Allow Enable and Disable of the key",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::551888181234:user/MDBEncAdmin"
      },
      "Action": [
        "kms:EnableKey",
        "kms:DisableKey"
      ]
    },
```

   ...so that your Key Policy looks like this:

```
{
  "Version": "2012-10-17",
  "Id": "key-consolepolicy-2",
  "Statement": [
    {
      "Sid": "Allow Enable and Disable of the key",
      "Effect": "Allow",
      "Principal": {
      ...
```

5. Click "Save Changes".

You've now added a new IAM user and you've given that user privileges to enable and disable your key. This user will be able to perform those operations using the AWS CLI or via a script of your own design using the AWS API. For example, using the AWS CLI:

```
$ cat ~/.aws/credentials
[MDBEncAdmin]
aws_access_key_id=AKIAJMPPNO7EBKABCDEF
aws_secret_access_key=pVdGwbuK5/jG64aBK1oEJOXRlkdM0aAylgabCDef

$ AWS_PROFILE=MDBEncAdmin aws --region us-east-1 kms disable-key --key-id arn:aws:kms:us-east-
1:551888181234:key/abcdf8f6-084b-4cff-99ca-abcdef6c7907c
```

In order for MariaDB to start, this new user will have to enable the master key, then the DBA can start MariaDB, and this user can once again disable the master key after the service has started. Note that in this workflow, MariaDB will be unable to create new encryption keys, such as would be done when a user creates a table that refers to a non-existent key ID. The AWS KMS plugin will encounter an error if it tries to generate a new encryption key while the master key is disabled. In that scenario, the key administrator would have to enable the key before the operation could succeed. Here's what you should expect to see in the journal if MariaDB tries to interact with a disabled master key:

```
[ERROR] AWS KMS plugin : GenerateDataKeyWithoutPlaintext failed : DisabledException - Unable to
parse ExceptionName: DisabledException Message: arn:aws:kms:us-east-1:551888181234:key/abcdf8f6-
084b-4cff-99ca-abcdef6c7907c is disabled.
```

Adding MFA

It's also possible to add MFA to this technique so that the user that enables & disables the master key has to authenticate using an MFA device. Adapt the instructions in the MFA section above to add MFA to the policy section for the user with EnableKey and DisableKeys privileges, add an MFA device for that user, use Security Token Service (STS) to get temporary security credentials, and then use those credentials to make the API calls. Here's an example Python script that follows that workflow:

```
#!/usr/bin/env python

import boto3
import sys

# Command-line argument processing should be more robust than this...
action= sys.argv[1]
mfa_token= sys.argv[2]

# These should perhaps go into a config file instead of here
mfa_serial= 'arn:aws:iam::551888181234:mfa/MDBEncAdmin'
key_id= 'arn:aws:kms:us-east-1:551888181234:key/abcdf8f6-084b-4cff-99ca-abcdef6c7907c'

# Make the connection to the Security Token Service to get temporary credentials
token_client= boto3.client('sts')
token_response= token_client.get_session_token(
    DurationSeconds= 900,
    SerialNumber= mfa_serial,
    TokenCode= mfa_token
)

cred= token_response['Credentials']

# Start new session using temporary, MFA-authenticated credentials
kms_session= boto3.session.Session(
    aws_access_key_id= cred['AccessKeyId'],
    aws_secret_access_key= cred['SecretAccessKey'],
    aws_session_token= cred['SessionToken'],
    region_name= key_id.split(':')[3]
)

# Start KMS client and execute operation against key
kms_client= kms_session.client('kms')
if action == 'enable' or action == 'e':
    action_f= kms_client.enable_key
elif action == 'disable' or action == 'd':
    action_f= kms_client.disable_key
else:
    raise Exception('Action must be either "disable" or "enable"')

action_f(KeyId=key_id)
```

```
$ AWS_PROFILE=MDBEncAdmin python kms-manage-key disable 575290
$ AWS_PROFILE=MDBEncAdmin python kms-manage-key enable 799870
```

Logging and auditing

CloudTrail

Amazon's CloudTrail service creates JSON-formatted text log files for every API interaction. Enabling CloudTrail requires S3, which incurs additional fees.

First, enable CloudTrail and add a trail.

1. Load the CloudTrail console at https://console.aws.amazon.com/cloudtrail/.
2. If you've never used CloudTrail before, click "Get Started Now".
3. Enter a value for "trail name". This example uses "mariadb-encryption-key".
4. Create a new S3 bucket, using a globally unique name, or use an existing S3 bucket, according to your needs.
5. Click "Turn On".

If you navigate to the S3 bucket you created, you should find log files that contain JSON-formatted descriptions of your API interactions.

CloudWatch

Amazon's CloudWatch service allows you to create alarms and event rules that monitor log information.

First, send your CloudTrail logs to CloudWatch.

1. Load the CloudTrail console at https://console.aws.amazon.com/cloudtrail/.
2. Click "Trails" in the left-hand navigation sidebar.
3. Click the name of your trail to open the Configuration view.
4. In the "CloudWatch Logs" section, click "Configure".
5. Click "Continue".
6. Click "Allow".

Now, set up an SNS topic to facilitate email notifications.

1. Open https://console.aws.amazon.com/sns/.
2. *Make sure the region in the console (look in the upper-right corner) is the same as the region where you created your key!*
3. Click "Get Started" is prompted.
4. Click "Events" in the left-hand sidebar.
5. Click "Create new topic".
6. Enter a *Topic name* of your choosing.
7. Enter a *Display name* of your choosing.
8. Click "Create topic".
9. Click the ARN of your new SNS topic.
10. Click "Create Subscription".
11. Select "Email" from the Protocol dropdown.
12. Enter the desired notification email address in the Endpoint field.
13. Wait for the confirmation email to show up and follow the instructions.

Now, configure CloudWatch and create an Event Rule.

1. Open https://console.aws.amazon.com/cloudwatch/.
2. *Make sure the region in the console (look in the upper-right corner) is the same as the region where you created your key and your SNS topic!*
3. Click "Events" in the left-hand sidebar.
4. Click "Create rule".
5. Choose "AWS API call" from the "Select event source" dropdown.
6. Choose "KMS" from the "Service name" dropdown.
7. Decide which operations should trigger the event. (You can eep "Any operation" selected for simplicity.)
8. Click "Add target".
9. Select "SNS target" from the dropdown.
10. Select the SNS topic you created in the previous steps.
11. Click "Configure details".
12. Enter a *Name* and *Description* of your choosing.
13. Click "Create rule".

You should now get emails any time someone executes API calls on the KMS service in the region where you've created the CloudWatch Event rule. That means you should get email any time the key is enabled or disabled, and any time the AWS KMS plugin generates new keys or decrypts the keys stored on disk on the MariaDB server.

You may also wish to create an event rule (or an additional event) that matches only when an unauthorized user tries to access the key. You might accomplish that by manually editing the Event selector of the rule to look something like this:

```
{
  "detail-type": [
    "AWS API Call via CloudTrail"
  ],
  "detail": {
    "eventSource": [
      "kms.amazonaws.com"
    ],
    "errorCode": [
      "AccessDenied",
      "UnauthorizedOperation"
    ]
  }
}
```

The emails are formatted as JSON. Further customization of the CloudWatch email workflow is beyond the scope of this document.

There are many other workflows available using CloudWatch, including workflows with alarms and dashboards. Those are beyond the scope of this document.

5.4.8.7 Eperi Key Management Encryption Plugin

Eperi's Key Management Plugin Package no longer appears to be available.

Contents

MariaDB's data-at-rest encryption requires the use of a key management and encryption plugin. These plugins are responsible both for the management of encryption keys and for the actual encryption and decryption of data.

MariaDB supports the use of multiple encryption keys. Each encryption key uses a 32-bit integer as a key identifier. If the specific plugin supports key rotation, then encryption keys can also be rotated, which creates a new version of the encryption key.

The Eperi Key Management plugin is a key management and encryption plugin that integrates with eperi Gateway for Databases.

Overview

The Eperi Key Management plugin is one of the key management and encryption plugins that can be set up for users who want to use data-at-rest encryption. Some of the plugin's primary features are:

- It reads encryption keys from eperi Gateway for Databases.
- It supports multiple encryption keys.
- It supports key rotation.
- It supports two different algorithms for encrypting data.

The eperi Gateway for Databases stores encryption keys on the key server outside of the database server itself, which provides an extra level of security. The eperi Gateway for Databases also supports performing all data encryption operations on the key server as well, but this is optional.

It also provides the following benefits:

- Key management outside the database
- No keys on database server hard disk
- Graphical user interface for configuration
- Encryption and decryption outside the database, supporting HSM's for maximum security.

Support for MariaDB is provided in eperi Gateway for Databases 3.4.

Installing the Eperi Key Management Plugin's Package

For information on how to install the package, see Eperi's documentation at the Eperi Customer Portal.

Installing the Plugin

Even after the package that contains the plugin's shared library is installed on the operating system, the plugin is not actually installed by MariaDB by default. The plugin can be installed by providing the `--plugin-load` or the `--plugin-load-add` options. This can be specified as a command-line argument to `mysqld` or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
plugin_load_add = eperi_key_management_plugin
```

Uninstalling the Plugin

Before you uninstall the plugin, you should ensure that data-at-rest encryption is completely disabled, and that MariaDB no longer needs the plugin to decrypt tables or other files.

You can uninstall the plugin dynamically by executing `UNINSTALL SONAME` or `UNINSTALL PLUGIN`. For example:

```
UNINSTALL SONAME 'eperi_key_management_plugin';
```

If you installed the plugin by providing the `--plugin-load` or the `--plugin-load-add` options in a relevant server option group in an option file, then those options should be removed to prevent the plugin from being loaded the next time the server is restarted.

Configuring the Eperi Key Management Plugin

For information on how to configure the plugin, see Eperi's documentation at the Eperi Customer Portal.

Using the Eperi Key Management Plugin

Once the Eperi Key Management Plugin is enabled, you can use it by creating an encrypted table:

```
CREATE TABLE t (i int) ENGINE=InnoDB ENCRYPTED=YES
```

Now, table `t` will be encrypted using the encryption key from the key server.

For more information on how to use encryption, see Data at Rest Encryption.

Using Multiple Encryption Keys

The Eperi Key Management Plugin supports using multiple encryption keys. Each encryption key can be defined with a different 32-bit integer as a key identifier.

When encrypting InnoDB tables, the key that is used to encrypt tables can be changed.

When encrypting Aria tables, the key that is used to encrypt tables cannot currently be changed.

Key Rotation

The Eperi Key Management plugin supports key rotation.

Versions

Version	Status	Introduced
1.0	Unknown	eperi Gateway for Databases 3.4.0

System Variables

`eperi_key_management_plugin_databaseId`

- **Description:** Determines the database ID which is processed in the eperi Gateway.
- **Commandline:** `--eperi-key-management-plugin-databaseid=value`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `integer`
- **Default Value:** `1`

---

eperi_key_management_plugin_encryption_algorithm

- **Description:** This system variable is used to determine which algorithm the plugin will use to encrypt data.
  - The recommended algorithm is `AES_CTR`, but this algorithm is only available when MariaDB is built with recent versions of OpenSSL. If the server is built with wolfSSL or yaSSL, then this algorithm is not available. See TLS and Cryptography Libraries Used by MariaDB for more information about which libraries are used on which platforms.
- **Commandline:** `--eperi-key-management-plugin-encryption-algorithm=value`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `enumerated`
- **Default Value:** `AES_CBC`
- **Valid Values:** `AES_CBC`, `AES_CTR`

---

eperi_key_management_plugin_encryption_mode

- **Description:** Encryption mode.
- **Commandline:** `--eperi-key-management-plugin-encryption-mode=value`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `enumerated`
- **Default Value:** `database`
- **Valid Values:** `database`, `gateway`

---

eperi_key_management_plugin_osslmt

- **Description:** Determines, whether the plugin should register callback functions for OpenSSL thread support.
- **Commandline:** `--eperi-key-management-plugin-osslmt=value`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `0` (Linux), `1` (Windows)

---

eperi_key_management_plugin_port

- **Description:** Listener port for plugin.
- **Commandline:** `--eperi-key-management-plugin-port=value`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `integer`
- **Default Value:** `14332`

---

eperi_key_management_plugin_url

- **Description:** URL to key server. The expected format of the URL is <host>:<port>. The port number is optional, and the port number defaults to 14333 if it is not specified.
- **Commandline:** `--eperi-key-management-plugin-url=value`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** `NULL`

eperi_key_management_plugin_url_check_disabled

- **Description:** Determines, whether the connection between plugin and eperi Gateway is tested at server startup.
- **Commandline:** `--eperi-key-management-plugin-url-check-disabled=value`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `boolean`
- **Default Value:** `1`

Options

eperi_key_management_plugin

- **Description:** Controls how the server should treat the plugin when the server starts up.
  - Valid values are:
    - `OFF` - Disables the plugin without removing it from the `mysql.plugins` table.
    - `ON` - Enables the plugin. If the plugin cannot be initialized, then the server will still continue starting up, but the plugin will be disabled.
    - `FORCE` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error.
    - `FORCE_PLUS_PERMANENT` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error. In addition, the plugin cannot be uninstalled with `UNINSTALL SONAME` or `UNINSTALL PLUGIN` while the server is running.
  - See Plugin Overview: Configuring Plugin Activation at Server Startup for more information.
- **Commandline:** `--eperi-key-management-plugin=value`
- **Data Type:** `enumerated`
- **Default Value:** `ON`
- **Valid Values:** `OFF`, `ON`, `FORCE`, `FORCE_PLUS_PERMANENT`

5.4.8.8 Encryption Plugin API

Contents

MariaDB's data-at-rest encryption requires the use of a key management and encryption plugin. These plugins are responsible both for the management of encryption keys and for the actual encryption and decryption of data.

MariaDB supports the use of multiple encryption keys. Each encryption key uses a 32-bit integer as a key identifier. If the specific plugin supports key rotation, then encryption keys can also be rotated, which creates a new version of the encryption key.

See Data at Rest Encryption and Encryption Key Management for more information.

Encryption Plugin API

The Encryption plugin API was created to allow a plugin to:

- implement key management, provide encryption keys to the server on request and change them according to internal policies.
- implement actual data encryption and decryption with the algorithm defined by the plugin.

This is how the API reflects that:

```
/* Returned from get_latest_key_version() */
#define ENCRYPTION_KEY_VERSION_INVALID (~(unsigned int)0)
#define ENCRYPTION_KEY_NOT_ENCRYPTED (0)

#define ENCRYPTION_KEY_SYSTEM_DATA 1
#define ENCRYPTION_KEY_TEMPORARY_DATA 2

/* Returned from get_key()  */
#define ENCRYPTION_KEY_BUFFER_TOO_SMALL (100)

#define ENCRYPTION_FLAG_DECRYPT 0
#define ENCRYPTION_FLAG_ENCRYPT 1
#define ENCRYPTION_FLAG_NOPAD 2

struct st_mariadb_encryption {
  int interface_version; /**< version plugin uses */

  /******************** KEY MANAGEMENT ***********************************/

  /**
    Function returning latest key version for a given key id.

    @return A version or ENCRYPTION_KEY_VERSION_INVALID to indicate an error.
  */
  unsigned int (*get_latest_key_version)(unsigned int key_id);

  /**
    Function returning a key for a key version

    @param key_id       The requested key id
    @param version      The requested key version
    @param key          The key will be stored there. Can be NULL -
                        in which case no key will be returned
    @param key_length   in: key buffer size
                        out: the actual length of the key

    This method can be used to query the key length - the required
    buffer size - by passing key==NULL.

    If the buffer size is less than the key length the content of the
    key buffer is undefined (the plugin is free to partially fill it with
    the key data or leave it untouched).

    @return 0 on success, or
            ENCRYPTION_KEY_VERSION_INVALID, ENCRYPTION_KEY_BUFFER_TOO_SMALL
            or any other non-zero number for errors
  */
  unsigned int (*get_key)(unsigned int key_id, unsigned int version,
                          unsigned char *key, unsigned int *key_length);

  /******************** ENCRYPTION ***********************************/
  /*
    The caller uses encryption as follows:
      1. Create the encryption context object of the crypt_ctx_size() bytes.
      2. Initialize it with crypt_ctx_init().
      3. Repeat crypt_ctx_update() until there are no more data to encrypt.
      4. Write the remaining output bytes and destroy the context object
         with crypt_ctx_finish().
  */

  /**
    Returns the size of the encryption context object in bytes
  */
  unsigned int (*crypt_ctx_size)(unsigned int key_id, unsigned int key_version);
  /**
    Initializes the encryption context object.
  */
  int (*crypt_ctx_init)(void *ctx, const unsigned char *key, unsigned int klen,
                        const unsigned char *iv, unsigned int ivlen, int flags,
                        unsigned int key_id, unsigned int key_version);
  /**
    Processes (encrypts or decrypts) a chunk of data
```

```
    Writes the output to th dst buffer. note that it might write
    more bytes that were in the input. or less. or none at all.
  */
  int (*crypt_ctx_update)(void *ctx, const unsigned char *src,
                          unsigned int slen, unsigned char *dst,
                          unsigned int *dlen);
  /**
    Writes the remaining output bytes and destroys the encryption context

    crypt_ctx_update might've cached part of the output in the context,
    this method will flush these data out.
  */
  int (*crypt_ctx_finish)(void *ctx, unsigned char *dst, unsigned int *dlen);
  /**
    Returns the length of the encrypted data

    It returns the exact length, given only the source length.
    Which means, this API only supports encryption algorithms where
    the length of the encrypted data only depends on the length of the
    input (a.k.a. compression is not supported).
  */
  unsigned int (*encrypted_length)(unsigned int slen, unsigned int key_id,
                                   unsigned int key_version);
};
```

The first method is used for key rotation. A plugin that doesn't support key rotation — for example, **file_key_management** — can return a fixed version for any valid key id. Note that it still has to return an error for an invalid key id. The version `ENCRYPTION_KEY_NOT_ENCRYPTED` means that the data should not be encrypted.

The second method is used for key management, the server uses it to retrieve the key corresponding to a specific key identifier and a specific key version.

The last five methods deal with encryption. Note that they take the key to use and key identifier and version. This is needed because the server can derive a session-specific, user-specific, or a tablespace-specific key from the original encryption key as returned by `get_key()`, so the `key` argument doesn't have to match the encryption key as the plugin knows it. On the other hand, the encryption algorithm may depend on the key identifier and version (and in the **example_key_management** plugin it does) so the plugin needs to know them to be able to encrypt the data.

Encryption methods are optional — if unset (as in the **debug_key_management** plugin), the server will fall back to AES_CBC.

Current Encryption Plugins

The MariaDB source tree has four encryption plugins. All these plugins are fairly simple and can serve as good examples of the Encryption plugin API.

file_key_management

It reads encryption keys from a plain-text file. It supports two different encryption algorithms. It supports multiple encryption keys. It does not support key rotation. See the File Key Management Plugin article for more details.

Versions

Version	Status	Introduced
1.0	Stable	MariaDB 10.1.18
1.0	Gamma	MariaDB 10.1.13
1.0	Alpha	MariaDB 10.1.3

aws_key_management

The AWS Key Management plugin uses the Amazon Web Services (AWS) Key Management Service (KMS) to generate and store AES keys on disk, in encrypted form, using the Customer Master Key (CMK) kept in AWS KMS. When MariaDB Server starts, the plugin will decrypt the encrypted keys, using the AWS KMS "Decrypt" API function. MariaDB data will then be encrypted and decrypted using the AES key. It supports multiple encryption keys. It supports key rotation.

See the AWS Key Management Plugin article for more details.

Versions

Version	Status	Introduced
1.0	Stable	MariaDB 10.2.6, MariaDB 10.1.24
1.0	Beta	MariaDB 10.1.18
1.0	Experimental	MariaDB 10.1.13

example_key_management

Uses random time-based generated keys, ignores key identifiers, supports key versions and key rotation. Uses AES_ECB and AES_CBC as encryption algorithms and changes them automatically together with key versions.

Versions

Version	Status	Introduced
1.0	Experimental	MariaDB 10.1.3

debug_key_management

Key is generated from the version, user manually controls key rotation. Only supports key identifier 1, uses only AES_CBC.

Versions

Version	Status	Introduced
1.0	Experimental	MariaDB 10.1.3

Encryption Service

Encryption is generally needed on the very low level **inside the storage engine**. That is, the storage engine needs to support encryption and have access to the encryption and key management functionality. The usual way for a plugin to access some functionality in the server is via a *service*. In this case the server provides the Encryption Service for storage engines (and other interested plugins) to use. These service functions are directly hooked into encryption plugin methods (described above).

Service functions are declared as follows:

```
unsigned int encryption_key_get_latest_version(unsigned int key_id);
unsigned int encryption_key_get(unsigned int key_id, unsigned int key_version,
                                unsigned char *buffer, unsigned int *length);
unsigned int encryption_ctx_size(unsigned int key_id, unsigned int key_version);
int encryption_ctx_init(void *ctx, const unsigned char *key, unsigned int klen,
                        const unsigned char *iv, unsigned int ivlen, int flags,
                        unsigned int key_id, unsigned int key_version);
int encryption_ctx_update(void *ctx, const unsigned char *src,
                          unsigned int slen, unsigned char *dst,
                          unsigned int *dlen);
int encryption_ctx_finish(void *ctx, unsigned char *dst, unsigned int *dlen);
unsigned int encryption_encrypted_length(unsigned int slen, unsigned int key_id,
                                         unsigned int key_version);
```

There are also convenience helpers to check for a key or key version existence and to encrypt or decrypt a block of data with one function call.

```
unsigned int encryption_key_id_exists(unsigned int id);
unsigned int encryption_key_version_exists(unsigned int id,
                                           unsigned int version);
int encryption_crypt(const unsigned char *src, unsigned int slen,
                     unsigned char *dst, unsigned int *dlen,
                     const unsigned char *key, unsigned int klen,
                     const unsigned char *iv, unsigned int ivlen, int flags,
                     unsigned int key_id, unsigned int key_version);
```

5.4.9 MariaDB Replication & Cluster Plugins

Semisynchronous Replication

Semisynchronous replication.

WSREP_INFO Plugin

Adds two new Information Schema tables, WSREP_MEMBERSHIP and WSREP_STATUS.

3.1.24 Semisynchronous Replication

5.4.9.2 WSREP_INFO Plugin

The `WSREP_INFO` plugin library contains the following plugins:

- `WSREP_MEMBERSHIP`
- `WSREP_STATUS`

The `WSREP_MEMBERSHIP` plugin creates the WSREP_MEMBERSHIP table in the INFORMATION_SCHEMA database. The plugin also adds the SHOW WSREP_MEMBERSHIP statement.

The `WSREP_STATUS` plugin creates the WSREP_STATUS table in the INFORMATION_SCHEMA database. The plugin also adds the SHOW WSREP_STATUS statement.

These tables and statements provide information about Galera. Only users with the SUPER privilege can access this information.

Contents

Installing the Plugin

Although the plugin's shared library is distributed with MariaDB by default, the plugin is not actually installed by MariaDB by default. There are two methods that can be used to install the plugin with MariaDB.

The first method can be used to install the plugin without restarting the server. You can install the plugin dynamically by executing INSTALL SONAME or INSTALL PLUGIN. For example:

```
INSTALL SONAME 'wsrep_info';
```

The second method can be used to tell the server to load the plugin when it starts up. The plugin can be installed this way by providing the --plugin-load or the --plugin-load-add options. This can be specified as a command-line argument to mysqld or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
plugin_load_add = wsrep_info
```

Uninstalling the Plugin

You can uninstall the plugin dynamically by executing UNINSTALL SONAME or UNINSTALL PLUGIN. For example:

```
UNINSTALL SONAME 'wsrep_info';
```

If you installed the plugin by providing the --plugin-load or the --plugin-load-add options in a relevant server option group in an option file, then those options should be removed to prevent the plugin from being loaded the next time the server is restarted.

Example

```
SHOW TABLES FROM information_schema LIKE 'WSREP%';
+---------------------------------------+
| Tables_in_information_schema (WSREP%) |
+---------------------------------------+
| WSREP_STATUS                          |
| WSREP_MEMBERSHIP                      |
+---------------------------------------+
```

Versions

Version	Status	Introduced
1.0	Stable	MariaDB 10.1.18
1.0	Gamma	MariaDB 10.1.13
1.0	Alpha	MariaDB 10.1.2

Options

wsrep_membership

- **Description:** Controls how the server should treat the plugin when the server starts up.
  - Valid values are:
    - `OFF` - Disables the plugin without removing it from the mysql.plugins table.
    - `ON` - Enables the plugin. If the plugin cannot be initialized, then the server will still continue starting up, but the plugin will be disabled.
    - `FORCE` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error.
    - `FORCE_PLUS_PERMANENT` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error. In addition, the plugin cannot be uninstalled with UNINSTALL SONAME or UNINSTALL PLUGIN while the server is running.
  - See Plugin Overview: Configuring Plugin Activation at Server Startup for more information.
- **Commandline:** `--wsrep-membership=value`
- **Data Type:** `enumerated`
- **Default Value:** `ON`
- **Valid Values:** `OFF`, `ON`, `FORCE`, `FORCE_PLUS_PERMANENT`

---

wsrep_status

- **Description:** Controls how the server should treat the plugin when the server starts up.
  - Valid values are:
    - `OFF` - Disables the plugin without removing it from the mysql.plugins table.
    - `ON` - Enables the plugin. If the plugin cannot be initialized, then the server will still continue starting up, but the plugin will be disabled.
    - `FORCE` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error.
    - `FORCE_PLUS_PERMANENT` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error. In addition, the plugin cannot be uninstalled with UNINSTALL SONAME or UNINSTALL PLUGIN while the server is running.
  - See Plugin Overview: Configuring Plugin Activation at Server Startup for more information.
- **Commandline:** `--wsrep-status=value`
- **Data Type:** `enumerated`
- **Default Value:** `ON`
- **Valid Values:** `OFF`, `ON`, `FORCE`, `FORCE_PLUS_PERMANENT`

5.3 Storage Engines

5.4.11 Other Plugins

Compression Plugins

Five MariaDB compression libraries are available as plugins.

Disks Plugin

Shows metadata about disks on the system.

Feedback Plugin

Collect and send user statistics.

Locales Plugin

List compiled-in locales.

METADATA_LOCK_INFO Plugin

Active metadata locks.

MYSQL_JSON

Converting MySQL's JSON data type to MariaDB's format.

Query Cache Information Plugin

Examines the contents of the query cache.

Query Response Time Plugin

Records statistics on the time to execute queries on MariaDB Server.

SQL Error Log Plugin

Records SQL-level errors to a log file.

User Statistics

User Statistics.

User Variables Plugin

User Variables plugin.

There are 1 related questions.

5.4.11.1 Feedback Plugin

> The Feedback plugin is not currently working.

The `feedback` plugin is designed to collect and, optionally, upload configuration and usage information to MariaDB.org or to any other configured URL.

The `feedback` plugin exists in all MariaDB versions.

MariaDB is distributed with this plugin included, but it is not enabled by default. On Windows, this plugin is part of the server and has a special checkbox in the installer window. Either way, you need to explicitly install and enable it in order for feedback data to be sent.

Contents

Installing the Plugin

Although the plugin's shared library is distributed with MariaDB by default, the plugin is not actually installed by MariaDB by default. There are two methods that can be used to install the plugin with MariaDB.

The first method can be used to install the plugin without restarting the server. You can install the plugin dynamically by executing `INSTALL SONAME` or `INSTALL PLUGIN`. For example:

```
INSTALL SONAME 'feedback';
```

The second method can be used to tell the server to load the plugin when it starts up. The plugin can be installed this way by providing the `--plugin-load` or the `--plugin-load-add` options. This can be specified as a command-line argument to `mysqld` or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
plugin_load_add = feedback
```

Uninstalling the Plugin

You can uninstall the plugin dynamically by executing `UNINSTALL SONAME` or `UNINSTALL PLUGIN`. For example:

```
UNINSTALL SONAME 'feedback';
```

If you installed the plugin by providing the `--plugin-load` or the `--plugin-load-add` options in a relevant server option group in an option file, then those options should be removed to prevent the plugin from being loaded the next time the server is restarted.

Enabling the Plugin

You can enable the plugin by setting the `feedback` option to `ON` in a relevant server option group in an option file. For example:

```
[mariadb]
...
feedback=ON
```

In Windows, the plugin can also be enabled during a new MSI installation. The MSI GUI installation provides the "Enable feedback plugin" checkbox to enable the plugin. The MSI command-line installation provides the FEEDBACK=1 command-line option to enable the plugin.

See the next section for how to verify the plugin is installed and active and (if needed) install the plugin.

Verifying the Plugin's Status

To verify whether the `feedback` plugin is installed and enabled, execute the `SHOW PLUGINS` statement or query the `information_schema.plugins` table. For example:

```
SELECT plugin_status FROM information_schema.plugins
  WHERE plugin_name = 'feedback';
```

If that `SELECT` returns no rows, then you still need to install the plugin.

When the plugin is installed and enabled, you will see:

```
SELECT plugin_status FROM information_schema.plugins
  WHERE plugin_name = 'feedback';
+---------------+
| plugin_status |
+---------------+
| ACTIVE        |
+---------------+
```

If you see `DISABLED` instead of `ACTIVE`, then you still need to enable the plugin.

Collecting Data

The `feedback` plugin will collect:

- Certain rows from SHOW STATUS and SHOW VARIABLES.
- All installed plugins and their versions.
- System information such as CPU count, memory, architecture, and OS/linux distribution.
- The feedback_server_uid, which is a SHA1 hash of the MAC address of the first network interface and the TCP port that the server listens on.

The `feedback` plugin creates the FEEDBACK table in the INFORMATION_SCHEMA database. To see the data that has been collected by the plugin, you can execute:

```
SELECT * FROM information_schema.feedback;
```

Only the contents of this table are sent to the feedback_url.

MariaDB stores collation usage statistics. Each collation that has been used by the server will have a record in "SELECT * FROM information_schema.feedback" output, for example:

```
+----------------------------------------+---------------------+
| VARIABLE_NAME                          | VARIABLE_VALUE      |
+----------------------------------------+---------------------+
| Collation used utf8_unicode_ci         | 10                  |
| Collation used latin1_general_ci       | 20                  |
+----------------------------------------+---------------------+
```

Collations that have not been used will not be included into the result.

Sending Data

The `feedback` plugin sends the data using a `POST` request to any URL or a list of URLs that you specify by setting the feedback_url system variable. By default, this is set to the following URL:

- https://mariadb.org/feedback_plugin/post

Both HTTP and HTTPS protocols are supported.

If HTTP traffic requires a proxy in your environment, then you can specify the proxy by setting the feedback_http_proxy system variable.

If the feedback_url system variable is not set to an empty string, then the plugin will automatically send a report to all URLs in the list a few minutes after the server starts up and then once a week after that.

If the feedback_url system variable is set to an empty string, then the plugin will **not** automatically send any data. This may be necessary if outbound HTTP communication from your database server is not permitted. In this case, you can still upload the data manually, if you'd like.

First, generate the report file with the MariaDB command-line mariadb client:

```
$ mariadb -e 'select * from information_schema.feedback' > report.txt
```

Then you can upload the generated `report.txt` here using your web browser.

Or you can do it from the command line with tools such as curl. For example:

```
$ curl -F data=@report.txt https://mariadb.org/feedback_plugin/post
```

Manual uploading allows you to be absolutely sure that we receive only the data shown in the INFORMATION_SCHEMA.FEEDBACK table and that no private or sensitive information is being sent.

Versions

Version	Status	Introduced
1.1	Stable	MariaDB 10.0.10
1.1	Beta	MariaDB 5.5.20, MariaDB 5.3.3

System Variables

feedback_http_proxy

- **Description:** Proxy server for use when http calls cannot be made, such as in a firewall environment. The format is `host:port`.
- **Commandline:** `--feedback-http=proxy=value`
- **Read-only:** Yes
- **Data Type:** string
- **Default Value:** `''` (empty)

---

feedback_send_retry_wait

- **Description:** Time in seconds before retrying if the plugin failed to send the data for any reason.
- **Commandline:** `--feedback-send-retry-wait=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** numeric
- **Default Value:** `60`
- **Valid Values:** `1` to `86400`

---

feedback_send_timeout

- **Description:** An attempt to send the data times out and fails after this many seconds.
- **Commandline:** `--feedback-send-timeout=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** numeric
- **Default Value:** `60`
- **Valid Values:** `1` to `86400`

---

feedback_server_uid

- **Description:** Automatically calculated server unique id hash.
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** string

---

feedback_url

- **Description:** URL to which the data is sent. More than one URL, separated by spaces, can be specified. Set it to an empty string to disable data sending.
- **Commandline:** `--feedback-url=url`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** string
- **Default Value:** `https://mariadb.org/feedback_plugin/post`

feedback_user_info

- **Description:** The value of this option is not used by the plugin, but it is included in the feedback data. It can be used to add any user-specified string to the report. This could be used to help to identify it. For example, a support contract number, or a computer name (if you collect reports internally by specifying your own `feedback-url`).
- **Commandline:** `--feedback-user-info=string`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** string
- **Default Value:** Empty string

Options

feedback

- **Description:** Controls how the server should treat the plugin when the server starts up.
  - Valid values are:
    - `OFF` - Disables the plugin without removing it from the mysql.plugins table.
    - `ON` - Enables the plugin. If the plugin cannot be initialized, then the server will still continue starting up, but the plugin will be disabled.
    - `FORCE` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error.
    - `FORCE_PLUS_PERMANENT` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error. In addition, the plugin cannot be uninstalled with UNINSTALL SONAME or UNINSTALL PLUGIN while the server is running.
  - See Plugin Overview: Configuring Plugin Activation at Server Startup for more information.
- **Commandline:** `--feedback=value`
- **Data Type:** `enumerated`
- **Default Value:** `ON`
- **Valid Values:** `OFF` , `ON` , `FORCE` , `FORCE_PLUS_PERMANENT`

5.4.11.2 Locales Plugin

The `LOCALES` plugin creates the LOCALES table in the INFORMATION_SCHEMA database. The plugin also adds the SHOW LOCALES statement.The table and statement can be queried to see all locales that are compiled into the server.

Contents

Installing the Plugin

Although the plugin's shared library is distributed with MariaDB by default, the plugin is not actually installed by MariaDB by default. There are two methods that can be used to install the plugin with MariaDB.

The first method can be used to install the plugin without restarting the server. You can install the plugin dynamically by

executing INSTALL SONAME or INSTALL PLUGIN. For example:

```
INSTALL SONAME 'locales';
```

The second method can be used to tell the server to load the plugin when it starts up. The plugin can be installed this way by providing the --plugin-load or the --plugin-load-add options. This can be specified as a command-line argument to mysqld or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
plugin_load_add = locales
```

Uninstalling the Plugin

You can uninstall the plugin dynamically by executing UNINSTALL SONAME or UNINSTALL PLUGIN. For example:

```
UNINSTALL SONAME 'locales';
```

If you installed the plugin by providing the --plugin-load or the --plugin-load-add options in a relevant server option group in an option file, then those options should be removed to prevent the plugin from being loaded the next time the server is restarted.

Example

```
SELECT * FROM INFORMATION_SCHEMA.LOCALES;
+-----+-------+---------------------------------------+-----------------------+----------------
-----+---------------+--------------+------------------------+
| ID  | NAME  | DESCRIPTION                           | MAX_MONTH_NAME_LENGTH |
MAX_DAY_NAME_LENGTH | DECIMAL_POINT | THOUSAND_SEP | ERROR_MESSAGE_LANGUAGE |
+-----+-------+---------------------------------------+-----------------------+----------------
-----+---------------+--------------+------------------------+
|   0 | en_US | English - United States               |                     9 |
9 | .             | ,            | english                |
|   1 | en_GB | English - United Kingdom              |                     9 |
9 | .             | ,            | english                |
|   2 | ja_JP | Japanese - Japan                      |                     3 |
3 | .             | ,            | japanese               |
|   3 | sv_SE | Swedish - Sweden                      |                     9 |
7 | ,             |              | swedish                |
|   4 | de_DE | German - Germany                      |                     9 |
10 | ,             | .            | german                 |
|   5 | fr_FR | French - France                       |                     9 |
8 | ,             |              | french                 |
|   6 | ar_AE | Arabic - United Arab Emirates         |                     6 |
8 | .             | ,            | english                |
|   7 | ar_BH | Arabic - Bahrain                      |                     6 |
8 | .             | ,            | english                |
|   8 | ar_JO | Arabic - Jordan                       |                    12 |
8 | .             | ,            | english                |
...
| 106 | no_NO | Norwegian - Norway                    |                     9 |
7 | ,             | .            | norwegian              |
| 107 | sv_FI | Swedish - Finland                     |                     9 |
7 | ,             |              | swedish                |
| 108 | zh_HK | Chinese - Hong Kong SAR               |                     3 |
3 | .             | ,            | english                |
| 109 | el_GR | Greek - Greece                        |                    11 |
9 | ,             | .            | greek                  |
| 110 | rm_CH | Romansh - Switzerland                 |                     9 |
9 | ,             | .            | english                |
+-----+-------+---------------------------------------+-----------------------+----------------
-----+---------------+--------------+------------------------+
```

Versions

Version	Status	Introduced

1.0	Stable	MariaDB 10.1.13
1.0	Gamma	MariaDB 10.0.10
1.0	Alpha	MariaDB 10.0.4

Options

`locales`

- **Description:** Controls how the server should treat the plugin when the server starts up.
  - Valid values are:
    - `OFF` - Disables the plugin without removing it from the mysql.plugins table.
    - `ON` - Enables the plugin. If the plugin cannot be initialized, then the server will still continue starting up, but the plugin will be disabled.
    - `FORCE` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error.
    - `FORCE_PLUS_PERMANENT` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error. In addition, the plugin cannot be uninstalled with UNINSTALL SONAME or UNINSTALL PLUGIN while the server is running.
  - See Plugin Overview: Configuring Plugin Activation at Server Startup for more information.
- **Commandline:** `--locales=value`
- **Data Type:** `enumerated`
- **Default Value:** `ON`
- **Valid Values:** `OFF`, `ON`, `FORCE`, `FORCE_PLUS_PERMANENT`

5.4.11.3 METADATA_LOCK_INFO Plugin

The `METADATA_LOCK_INFO` plugin creates the METADATA_LOCK_INFO table in the INFORMATION_SCHEMA database. This table shows active metadata locks. The table will be empty if there are no active metadata locks.

Contents

Installing the Plugin

Although the plugin's shared library is distributed with MariaDB by default, the plugin is not actually installed by MariaDB by default. There are two methods that can be used to install the plugin with MariaDB.

The first method can be used to install the plugin without restarting the server. You can install the plugin dynamically by executing INSTALL SONAME or INSTALL PLUGIN. For example:

```
INSTALL SONAME 'metadata_lock_info';
```

The second method can be used to tell the server to load the plugin when it starts up. The plugin can be installed this way by providing the --plugin-load or the --plugin-load-add options. This can be specified as a command-line argument to mysqld or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
plugin_load_add = metadata_lock_info
```

Uninstalling the Plugin

You can uninstall the plugin dynamically by executing UNINSTALL SONAME or UNINSTALL PLUGIN. For example:

```
UNINSTALL SONAME 'metadata_lock_info';
```

If you installed the plugin by providing the --plugin-load or the --plugin-load-add options in a relevant server option group in an option file, then those options should be removed to prevent the plugin from being loaded the next time the server is restarted.

Examples

Viewing all Metadata Locks

```
SELECT * FROM information_schema.metadata_lock_info;
+-----------+-------------------------+--------------+----------------------+--------------
---+-------------+
| THREAD_ID | LOCK_MODE               | LOCK_DURATION | LOCK_TYPE            | TABLE_SCHEMA
| TABLE_NAME  |
+-----------+-------------------------+--------------+----------------------+--------------
---+-------------+
|        31 | MDL_INTENTION_EXCLUSIVE  | MDL_EXPLICIT  | Global read lock     |
|             |
|        31 | MDL_INTENTION_EXCLUSIVE  | MDL_EXPLICIT  | Commit lock          |
|             |
|        31 | MDL_INTENTION_EXCLUSIVE  | MDL_EXPLICIT  | Schema metadata lock | dbname
|             |
|        31 | MDL_SHARED_NO_READ_WRITE | MDL_EXPLICIT  | Table metadata lock  | dbname
| exotics     |
+-----------+-------------------------+--------------+----------------------+--------------
---+-------------+
4 rows in set (0.00 sec)
```

Matching Metadata Locks with Threads and Queries

```
SELECT
CONCAT('Thread ',P.ID,' executing "',P.INFO,'" IS LOCKED BY Thread ',
M.THREAD_ID) WhoLocksWho
FROM INFORMATION_SCHEMA.PROCESSLIST P,
INFORMATION_SCHEMA.METADATA_LOCK_INFO M
WHERE LOCATE(lcase(LOCK_TYPE), lcase(STATE))>0;
+-------------------------------------------------------------------------------------+
| WhoLocksWho                                                                         |
+-------------------------------------------------------------------------------------+
| Thread 3 executing "INSERT INTO foo ( b ) VALUES ( 'FOO' )" IS LOCKED BY Thread 2 |
+-------------------------------------------------------------------------------------+
1 row in set (0.00 sec)

SHOW PROCESSLIST;
+----+------+-----------+------+---------+------+-------------------------------+-------------
---------------------------+----------+
| Id | User | Host      | db   | Command | Time | State                         | Info
| Progress |
+----+------+-----------+------+---------+------+-------------------------------+-------------
---------------------------+----------+
|  2 | root | localhost | test | Sleep   |  123 |                               | NULL
|    0.000 |
|  3 | root | localhost | test | Query   |  103 | Waiting for global read lock | INSERT INTO
foo ( b ) VALUES ( 'FOO' ) |    0.000 |
|  4 | root | localhost | test | Query   |    0 | init                          | SHOW
PROCESSLIST                        |    0.000 |
+----+------+-----------+------+---------+------+-------------------------------+-------------
---------------------------+----------+
3 rows in set (0.00 sec)
```

Versions

Version	Status	Introduced
0.1	Stable	MariaDB 10.1.13

0.1	Beta	MariaDB 10.0.10
0.1	Alpha	MariaDB 10.0.7

Options

`metadata_lock_info`

- **Description:** Controls how the server should treat the plugin when the server starts up.
  - Valid values are:
    - `OFF` - Disables the plugin without removing it from the mysql.plugins table.
    - `ON` - Enables the plugin. If the plugin cannot be initialized, then the server will still continue starting up, but the plugin will be disabled.
    - `FORCE` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error.
    - `FORCE_PLUS_PERMANENT` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error. In addition, the plugin cannot be uninstalled with UNINSTALL SONAME or UNINSTALL PLUGIN while the server is running.
  - See Plugin Overview: Configuring Plugin Activation at Server Startup for more information.
- **Commandline:** `--metadata-lock-info=value`
- **Data Type:** `enumerated`
- **Default Value:** `ON`
- **Valid Values:** `OFF`, `ON`, `FORCE`, `FORCE_PLUS_PERMANENT`

5.4.11.4 MYSQL_JSON

MariaDB starting with 10.5.7

The `TYPE_MYSQL_JSON` plugin was first released in MariaDB 10.5.7.

The JSON type in MySQL stores the JSON object in its own native form, while in MariaDB the JSON type is a LONGTEXT. Opening a table with a JSON type created in MySQL would result in an error:

```
select * from mysql_json_table;
ERROR 4161 (HY000): Unknown data type: 'MYSQL_JSON'
```

The mysql_json plugin is used to make it easier to upgrade to MariaDB.

Installing

Installing can be done in a number of ways, for example:

```
install soname 'type_mysql_json';
```

See Making MariaDB understand MySQL JSON for a full description.

5.4.11.5 Query Cache Information Plugin

The `QUERY_CACHE_INFO` plugin creates the QUERY_CACHE_INFO table in the INFORMATION_SCHEMA database. This table shows all queries in the query cache. Querying this table acquires the query cache lock and will result in lock waits for queries that are using or expiring from the query cache. You must have the PROCESS privilege to query this table.

Contents

Installing the Plugin

Although the plugin's shared library is distributed with MariaDB by default, the plugin is not actually installed by MariaDB by default. There are two methods that can be used to install the plugin with MariaDB.

The first method can be used to install the plugin without restarting the server. You can install the plugin dynamically by executing INSTALL SONAME or INSTALL PLUGIN. For example:

```
INSTALL SONAME 'query_cache_info';
```

The second method can be used to tell the server to load the plugin when it starts up. The plugin can be installed this way by providing the --plugin-load or the --plugin-load-add options. This can be specified as a command-line argument to mysqld or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
plugin_load_add = query_cache_info
```

Uninstalling the Plugin

You can uninstall the plugin dynamically by executing UNINSTALL SONAME or UNINSTALL PLUGIN. For example:

```
UNINSTALL SONAME 'query_cache_info';
```

If you installed the plugin by providing the --plugin-load or the --plugin-load-add options in a relevant server option group in an option file, then those options should be removed to prevent the plugin from being loaded the next time the server is restarted.

Example

```
select statement_schema, statement_text, result_blocks_count,
  result_blocks_size from information_schema.query_cache_info;
+------------------+------------------+---------------------+--------------------+
| statement_schema | statement_text   | result_blocks_count | result_blocks_size |
+------------------+------------------+---------------------+--------------------+
| test             | select * from t1 |                   1 |                512 |
+------------------+------------------+---------------------+--------------------+
```

Versions

Version	Status	Introduced
1.1	Stable	MariaDB 10.1.13
1.1	Gamma	MariaDB 10.1.8
1.0	Gamma	MariaDB 10.0.10
1.0	Alpha	MariaDB 5.5.31

Options

query_cache_info

- **Description:** Controls how the server should treat the plugin when the server starts up.
  - Valid values are:
    - `OFF` - Disables the plugin without removing it from the mysql.plugins table.
    - `ON` - Enables the plugin. If the plugin cannot be initialized, then the server will still continue starting up, but the plugin will be disabled.
    - `FORCE` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error.
    - `FORCE_PLUS_PERMANENT` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error. In addition, the plugin cannot be uninstalled with UNINSTALL SONAME or

UNINSTALL PLUGIN while the server is running.
    - See Plugin Overview: Configuring Plugin Activation at Server Startup for more information.
- **Commandline:** `--query-cache-info=value`
- **Data Type:** `enumerated`
- **Default Value:** `ON`
- **Valid Values:** `OFF`, `ON`, `FORCE`, `FORCE_PLUS_PERMANENT`

5.4.11.6 Query Response Time Plugin

The `query_response_time` plugin creates the QUERY_RESPONSE_TIME table in the INFORMATION_SCHEMA database. The plugin also adds the SHOW QUERY_RESPONSE_TIME and FLUSH QUERY_RESPONSE_TIME statements.

The slow query log provides exact information about queries that take a long time to execute. However, sometimes there are a large number of queries that each take a very short amount of time to execute. This feature provides a tool for analyzing that information by counting and displaying the number of queries according to the the length of time they took to execute.

This feature is based on Percona's Response Time Distribution.

Contents

Installing the Plugin

This shared library actually consists of two different plugins:

- `QUERY_RESPONSE_TIME` - An INFORMATION_SCHEMA plugin that exposes statistics.
- `QUERY_RESPONSE_TIME_AUDIT` - audit plugin, collects statistics.

Both plugins need to be installed to get meaningful statistics.

Although the plugin's shared library is distributed with MariaDB by default, the plugin is not actually installed by MariaDB by default. There are two methods that can be used to install the plugin with MariaDB.

The first method can be used to install the plugin without restarting the server. You can install the plugin dynamically by executing INSTALL SONAME or INSTALL PLUGIN. For example:

```
INSTALL SONAME 'query_response_time';
```

The second method can be used to tell the server to load the plugin when it starts up. The plugin can be installed this way by providing the --plugin-load or the --plugin-load-add options. This can be specified as a command-line argument to mysqld or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
plugin_load_add = query_response_time
```

Uninstalling the Plugin

You can uninstall the plugin dynamically by executing UNINSTALL SONAME or UNINSTALL PLUGIN. For example:

```
UNINSTALL SONAME 'query_response_time';
```

If you installed the plugin by providing the --plugin-load or the --plugin-load-add options in a relevant server option group in an option file, then those options should be removed to prevent the plugin from being loaded the next time the server is restarted.

Response Time Distribution

The user can define time intervals that divide the range 0 to positive infinity into smaller intervals and then collect the number of commands whose execution times fall into each of those intervals.

Each interval is described as:

```
(range_base ^ n; range_base ^ (n+1)]
```

The range_base is some positive number (see Limitations). The interval is defined as the difference between two nearby powers of the range base.

For example, if the range base=10, we have the following intervals:

```
(0; 10 ^ -6], (10 ^ -6; 10 ^ -5], (10 ^ -5; 10 ^ -4], ...,
  (10 ^ -1; 10 ^1], (10^1; 10^2]...(10^7; positive infinity]
```

or

```
(0; 0.000001], (0.000001; 0.000010], (0.000010; 0.000100], ...,
  (0.100000; 1.0]; (1.0; 10.0]...(1000000; positive infinity]
```

For each interval, a count is made of the queries with execution times that fell into that interval.

You can select the range of the intervals by changing the range base. For example, for base range=2 we have the following intervals:

```
(0; 2 ^ -19], (2 ^ -19; 2 ^ -18], (2 ^ -18; 2 ^ -17], ...,
  (2 ^ -1; 2 ^1], (2 ^ 1; 2 ^ 2]...(2 ^ 25; positive infinity]
```

or

```
(0; 0.000001], (0.000001, 0.000003], ...,
  (0.25; 0.5], (0.5; 2], (2; 4]...(8388608; positive infinity]
```

Small numbers look strange (i.e., don't look like powers of 2), because we lose precision on division when the ranges are calculated at runtime. In the resulting table, you look at the high boundary of the range.

For example, you may see:

```
SELECT * FROM INFORMATION_SCHEMA.QUERY_RESPONSE_TIME;
+----------------+-------+----------------+
| TIME           | COUNT | TOTAL          |
+----------------+-------+----------------+
|       0.000001 |     0 |       0.000000 |
|       0.000010 |    17 |       0.000094 |
|       0.000100 |  4301 |       0.236555 |
|       0.001000 |  1499 |       0.824450 |
|       0.010000 | 14851 |      81.680502 |
|       0.100000 |  8066 |     443.635693 |
|       1.000000 |     0 |       0.000000 |
|      10.000000 |     0 |       0.000000 |
|     100.000000 |     1 |      55.937094 |
|    1000.000000 |     0 |       0.000000 |
|   10000.000000 |     0 |       0.000000 |
|  100000.000000 |     0 |       0.000000 |
| 1000000.000000 |     0 |       0.000000 |
| TOO LONG       |     0 | TOO LONG       |
+----------------+-------+----------------+
```

This means there were:

```
* 17 queries with 0.000001 < query execution time < = 0.000010 seconds; total execution time of
the 17 queries = 0.000094 seconds

* 4301 queries with 0.000010 < query execution time < = 0.000100 seconds; total execution time
of the 4301 queries = 0.236555 seconds

* 1499 queries with 0.000100 < query execution time < = 0.001000 seconds; total execution time
of the 1499 queries = 0.824450 seconds

* 14851 queries with 0.001000 < query execution time < = 0.010000 seconds; total execution time
of the 14851 queries = 81.680502 seconds

* 8066 queries with 0.010000 < query execution time < = 0.100000 seconds; total execution time
of the 8066 queries = 443.635693 seconds

* 1 query with 10.000000 < query execution time < = 100.0000 seconds; total execution time of
the 1 query = 55.937094 seconds
```

Using the Plugin

Using the Information Schema Table

You can get the distribution by querying the the QUERY_RESPONSE_TIME table in the INFORMATION_SCHEMA database. For example:

```
SELECT * FROM INFORMATION_SCHEMA.QUERY_RESPONSE_TIME;
```

You can also write more complex queries. For example:

```
SELECT c.count, c.time,
(SELECT SUM(a.count) FROM INFORMATION_SCHEMA.QUERY_RESPONSE_TIME as a
   WHERE a.count != 0) as query_count,
(SELECT COUNT(*)     FROM INFORMATION_SCHEMA.QUERY_RESPONSE_TIME as b
  WHERE b.count != 0) as not_zero_region_count,
(SELECT COUNT(*)     FROM INFORMATION_SCHEMA.QUERY_RESPONSE_TIME) as region_count
FROM INFORMATION_SCHEMA.QUERY_RESPONSE_TIME as c
  WHERE c.count > 0;
```

Note: If query_response_time_stats is set to `ON` , then the execution times for these two SELECT queries will also be collected.

Using the SHOW Statement

As an alternative to the QUERY_RESPONSE_TIME table in the INFORMATION_SCHEMA database, you can also use the SHOW QUERY_RESPONSE_TIME statement. For example:

```
SHOW QUERY_RESPONSE_TIME;
```

Flushing Plugin Data

Flushing the plugin data does two things:

- Clears the collected times from the QUERY_RESPONSE_TIME table in the INFORMATION_SCHEMA database.
- Reads the value of query_response_time_range_base and uses it to set the range base for the table.

Plugin data can be flushed with the FLUSH QUERY_RESPONSE_TIME statement. For example:

```
FLUSH QUERY_RESPONSE_TIME;
```

Setting the query_response_time_flush system variable has the same effect. For example:

```
SET GLOBAL query_response_time_flush=1;
```

Versions

Version	Status	Introduced
1.0	Stable	MariaDB 10.1.13
1.0	Gamma	MariaDB 10.0.10
1.0	Alpha	MariaDB 10.0.4

System Variables

query_response_time_flush

- **Description:** Updating this variable flushes the statistics and re-reads query_response_time_range_base.
- **Commandline:** None
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

query_response_time_range_base

- **Description:** Select base of log for `QUERY_RESPONSE_TIME` ranges. WARNING: variable change takes affect only after flush.
- **Commandline:** `--query-response-time-range-base=#`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `10`
- **Range:** `2` to `1000`

---

query_response_time_exec_time_debug

- **Description:** Pretend queries take this many microseconds. When 0 (the default) use the actual execution time.
  - This system variable is only available when the plugin is a debug build.
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `0`
- **Range:** `0` to `31536000`

---

query_response_time_stats

- **Description:** Enable or disable query response time statistics collecting
- **Commandline:** `query-response-time-stats[={0|1}]`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

Options

query_response_time

- **Description:** Controls how the server should treat the plugin when the server starts up.
  - Valid values are:
    - `OFF` - Disables the plugin without removing it from the mysql.plugins table.
    - `ON` - Enables the plugin. If the plugin cannot be initialized, then the server will still continue starting up, but the plugin will be disabled.

        - `FORCE` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error.
        - `FORCE_PLUS_PERMANENT` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error. In addition, the plugin cannot be uninstalled with UNINSTALL SONAME or UNINSTALL PLUGIN while the server is running.
    - See Plugin Overview: Configuring Plugin Activation at Server Startup for more information.
- **Commandline:** `--query-response-time=value`
- **Data Type:** `enumerated`
- **Default Value:** `ON`
- **Valid Values:** `OFF`, `ON`, `FORCE`, `FORCE_PLUS_PERMANENT`

---

query_response_time_audit

- **Description:** Controls how the server should treat the plugin when the server starts up.
    - Valid values are:
        - `OFF` - Disables the plugin without removing it from the mysql.plugins table.
        - `ON` - Enables the plugin. If the plugin cannot be initialized, then the server will still continue starting up, but the plugin will be disabled.
        - `FORCE` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error.
        - `FORCE_PLUS_PERMANENT` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error. In addition, the plugin cannot be uninstalled with UNINSTALL SONAME or UNINSTALL PLUGIN while the server is running.
    - See Plugin Overview: Configuring Plugin Activation at Server Startup for more information.
- **Commandline:** `--query-response-time-audit=value`
- **Data Type:** `enumerated`
- **Default Value:** `ON`
- **Valid Values:** `OFF`, `ON`, `FORCE`, `FORCE_PLUS_PERMANENT`

5.4.11.7 SQL Error Log Plugin

Contents

The `SQL_ERROR_LOG` plugin collects errors sent to clients in a log file defined by sql_error_log_filename, so that they can later be analyzed. The log file can be rotated if sql_error_log_rotate is set.

Errors are logged as they happen and an error will be logged even if it was handled by a condition handler and was never technically *sent* to the client.

Comments are also logged, which can make the log easier to search. But this is only possible if the client does not strip the comments away. For example, mysql command-line client only leaves comments when started with the --comments option.

Installing the Plugin

Although the plugin's shared library is distributed with MariaDB by default, the plugin is not actually installed by MariaDB by default. There are two methods that can be used to install the plugin with MariaDB.

The first method can be used to install the plugin without restarting the server. You can install the plugin dynamically by executing INSTALL SONAME or INSTALL PLUGIN. For example:

```
INSTALL SONAME 'sql_errlog';
```

The second method can be used to tell the server to load the plugin when it starts up. The plugin can be installed this way by providing the --plugin-load or the --plugin-load-add options. This can be specified as a command-line argument to mariadbd or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
plugin_load_add = sql_errlog
```

Uninstalling the Plugin

You can uninstall the plugin dynamically by executing UNINSTALL SONAME or UNINSTALL PLUGIN. For example:

```
UNINSTALL SONAME 'sql_errlog';
```

If you installed the plugin by providing the --plugin-load or the --plugin-load-add options in a relevant server option group in an option file, then those options should be removed to prevent the plugin from being loaded the next time the server is restarted.

Things logged

The current log format before 10.10 is:

```
Time User Error_code: Error_message : Query
```

Starting from 10.11 the format is:

```
Time User Type Error_code: Error_message : Query
```

Each separated by a space or : as above

Option	Description	Version
Time	Time (YYYY-MM-DD hh-mm-ss)	5.5.22
User	privilege_user [login_user_name] @ hostname [ip]	5.5.22
Type	ERROR or WARNING	10.11.6
Error_code	OS error, MariaDB storage engine code (120-199) or MariaDB internal error code (1000-)	5.5.22
Query	Query text	5.5.22

Example of logs

```
2023-10-31 15:54:37 root[root] @ localhost [] ERROR 1146: Table 'test.t_doesnt_exist' doesn't exi
2023-10-31 15:54:37 root[root] @ localhost [] ERROR 1064: You have an error in your SQL syntax; c
2023-10-31 15:54:37 root[root] @ localhost [] ERROR 1146: Table 'test.temptab' doesn't exist : SE
2023-11-01 11:31:15 [monty] @ storm [192.168.0.12] ERROR 1051: Unknown table 'test.t1' : drop tab
```

Example usage

```
install plugin SQL_ERROR_LOG soname 'sql_errlog';
Query OK, 0 rows affected (0.00 sec)

use test;
Database changed

set sql_mode='STRICT_ALL_TABLES,NO_ENGINE_SUBSTITUTION';
Query OK, 0 rows affected (0.00 sec)

CREATE TABLE foo2 (id int) ENGINE=WHOOPSIE;
ERROR 1286 (42000): Unknown storage engine 'WHOOPSIE'
\! cat data/sql_errors.log
2013-03-19  9:38:40 msandbox[msandbox] @ localhost [] ERROR 1286: Unknown storage engine
'WHOOPSIE' : CREATE TABLE foo2 (id int) ENGINE=WHOOPSIE
```

Versions

Version	Status	Introduced
1.0	Stable	MariaDB 10.1.13
1.0	Gamma	MariaDB 10.0.10
1.0	Alpha	MariaDB 5.5.22

System Variables

sql_error_log

- **Description:** Controls how the server should treat the plugin when the server starts up.
  - Valid values are:
    - `OFF` - Disables the plugin without removing it from the mysql.plugins table.
    - `ON` - Enables the plugin. If the plugin cannot be initialized, then the server will still continue starting up, but the plugin will be disabled.
    - `FORCE` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error.
    - `FORCE_PLUS_PERMANENT` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error. In addition, the plugin cannot be uninstalled with UNINSTALL SONAME or UNINSTALL PLUGIN while the server is running.
  - See Plugin Overview: Configuring Plugin Activation at Server Startup for more information.
- **Commandline:** `--sql-error-log=value`
- **Data Type:** `enumerated`
- **Default Value:** `ON`
- **Valid Values:** `OFF`, `ON`, `FORCE`, `FORCE_PLUS_PERMANENT`

---

sql_error_log_filename

- **Description:** The name of the logfile. Rotation of it will be named like *`sql_error_log_filename`*`.001`
- **Commandline:** `--sql-error-log-filename=value`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `string`
- **Default Value:** `sql_errors.log`

---

sql_error_log_rate

- **Description:** The rate of logging. `SET sql_error_log_rate=300;` means that one of 300 errors will be written to the log.
  If `sql_error_log_rate` is `0` the logging is disabled.
  The default rate is `1` (every error is logged).
- **Commandline:** `--sql-error-log-rate=#`
- **Scope:** Global

- **Dynamic:** Yes
- **Data Type:** `numeric`
- **Default Value:** `1`

---

sql_error_log_rotate

- **Description:** This is the 'write-only' variable. Assigning TRUE to this variable forces the log rotation.
- **Commandline:** `--sql-error-log-rotate={0|1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`

---

sql_error_log_rotations

- **Description:** The number of rotations. When rotated, the current log file is stored and the new empty one created. The sql_error_log_rotations logs are stored, older are removed.
  The default number of rotations is `9` .
- **Commandline:** `--sql-error-log-rotations`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `9`
- **Range:** `1` to `999`

---

sql_error_log_size_limit

- **Description:** The limitation for the size of the log file. After reaching the specified limit, the log file is rotated. 1M limit set by default.
- **Commandline:** `--sql-error-log-size-limit=#`
- **Scope:** Global
- **Dynamic:** No
- **Data Type:** `numeric`
- **Default Value:** `1000000`
- **Range:** `100` to `9223372036854775807`

---

sql_error_log_warnings

- **Description:** If set, log warnings in addition to errors.
- **Commandline:** `--sql-error-log-warnings={0,1}`
- **Scope:** Global
- **Dynamic:** Yes
- **Data Type:** `boolean`
- **Default Value:** `OFF`
- **Introduced:** MariaDB 10.11.6

3.3.4.6.6 User Statistics

5.4.11.9 User Variables Plugin

The `user_variables` plugin creates the USER_VARIABLES table in the INFORMATION_SCHEMA database. This table contains information about user-defined variables.

Contents

Installing the Plugin

In current versions, the `user_variables` plugin is statically linked into the server by default, so it does not need to be installed.

Prior to MariaDB 10.2.6, although the plugin's shared library is distributed with MariaDB by default, the plugin was not actually installed by MariaDB by default. There are two methods that can be used to install the plugin with MariaDB.

The first method can be used to install the plugin without restarting the server. You can install the plugin dynamically by executing INSTALL SONAME or INSTALL PLUGIN. For example:

```
INSTALL SONAME 'user_variables';
```

The second method can be used to tell the server to load the plugin when it starts up. The plugin can be installed this way by providing the --plugin-load or the --plugin-load-add options. This can be specified as a command-line argument to mysqld or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
plugin_load_add = user_variables
```

Uninstalling the Plugin

You can uninstall the plugin dynamically by executing UNINSTALL SONAME or UNINSTALL PLUGIN. For example:

```
UNINSTALL SONAME 'user_variables';
```

If you installed the plugin by providing the --plugin-load or the --plugin-load-add options in a relevant server option group in an option file, then those options should be removed to prevent the plugin from being loaded the next time the server is restarted.

Examples

```
SELECT * FROM information_schema.USER_VARIABLES ORDER BY VARIABLE_NAME;
+---------------+----------------+---------------+--------------------+
| VARIABLE_NAME | VARIABLE_VALUE | VARIABLE_TYPE | CHARACTER_SET_NAME |
+---------------+----------------+---------------+--------------------+
| var           | 0              | INT           | utf8               |
| var2          | abc            | VARCHAR       | utf8               |
+---------------+----------------+---------------+--------------------+
```

Flushing User-Defined Variables

User-defined variables are reset and the Information Schema table emptied with the FLUSH USER_VARIABLES statement.

```
SET @str = CAST(123 AS CHAR(5));

SELECT * FROM information_schema.USER_VARIABLES ORDER BY VARIABLE_NAME;
+---------------+----------------+---------------+--------------------+
| VARIABLE_NAME | VARIABLE_VALUE | VARIABLE_TYPE | CHARACTER_SET_NAME |
+---------------+----------------+---------------+--------------------+
| str           | 123            | VARCHAR       | utf8mb3            |
+---------------+----------------+---------------+--------------------+

FLUSH USER_VARIABLES;

SELECT * FROM information_schema.USER_VARIABLES ORDER BY VARIABLE_NAME;
Empty set (0.000 sec)
```

Versions

Version	Status	Introduced
1.0	Stable	MariaDB 10.3.13
1.0	Gamma	MariaDB 10.2.6
1.0	Alpha	MariaDB 10.2.0

Options

`user_variables`

- **Description:** Controls how the server should treat the plugin when the server starts up.
  - Valid values are:
    - `OFF` - Disables the plugin without removing it from the mysql.plugins table.
    - `ON` - Enables the plugin. If the plugin cannot be initialized, then the server will still continue starting up, but the plugin will be disabled.
    - `FORCE` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error.
    - `FORCE_PLUS_PERMANENT` - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error. In addition, the plugin cannot be uninstalled with UNINSTALL SONAME or UNINSTALL PLUGIN while the server is running.
  - See Plugin Overview: Configuring Plugin Activation at Server Startup for more information.
- **Commandline:** `--user-variables=value`
- **Data Type:** `enumerated`
- **Default Value:** `ON`
- **Valid Values:** `OFF`, `ON`, `FORCE`, `FORCE_PLUS_PERMANENT`

5.4.11.10 Disks Plugin

The `DISKS` plugin creates the DISKS table in the INFORMATION_SCHEMA database. This table shows metadata about disks on the system.

Before MariaDB 10.4.7, MariaDB 10.3.17 and MariaDB 10.2.26, this plugin did **not** check user privileges. When it is enabled, **any** user can query the `INFORMATION_SCHEMA.DISKS` table and see all the information it provides.

Since MariaDB 10.4.7, MariaDB 10.3.17 and MariaDB 10.2.26, it required the FILE privilege.

The plugin only works on Linux.

Contents

Installing the Plugin

Although the plugin's shared library is distributed with MariaDB by default, the plugin is not actually installed by MariaDB by default. There are two methods that can be used to install the plugin with MariaDB.

The first method can be used to install the plugin without restarting the server. You can install the plugin dynamically by executing INSTALL SONAME or INSTALL PLUGIN. For example:

```
INSTALL SONAME 'disks';
```

The second method can be used to tell the server to load the plugin when it starts up. The plugin can be installed this way by providing the --plugin-load or the --plugin-load-add options. This can be specified as a command-line argument to mysqld or it can be specified in a relevant server option group in an option file. For example:

```
[mariadb]
...
plugin_load_add = disks
```

Uninstalling the Plugin

You can uninstall the plugin dynamically by executing UNINSTALL SONAME or UNINSTALL PLUGIN. For example:

```
UNINSTALL SONAME 'disks';
```

If you installed the plugin by providing the --plugin-load or the --plugin-load-add options in a relevant server option group in an option file, then those options should be removed to prevent the plugin from being loaded the next time the server is restarted.

Example

```
SELECT * FROM information_schema.DISKS;

+-----------+-------+----------+---------+-----------+
| Disk      | Path  | Total    | Used    | Available |
+-----------+-------+----------+---------+-----------+
| /dev/vda1 | /     | 26203116 | 2178424 |  24024692 |
| /dev/vda1 | /boot | 26203116 | 2178424 |  24024692 |
| /dev/vda1 | /etc  | 26203116 | 2178424 |  24024692 |
+-----------+-------+----------+---------+-----------+
```

Versions

Version	Status	Introduced
1.1	Stable	MariaDB 10.4.7, MariaDB 10.3.17, MariaDB 10.2.26, MariaDB 10.1.41
1.0	Beta	MariaDB 10.3.6, MariaDB 10.2.14, MariaDB 10.1.32

Options

disks

- **Description:** Controls how the server should treat the plugin when the server starts up.
  - Valid values are:
    - OFF - Disables the plugin without removing it from the mysql.plugins table.
    - ON - Enables the plugin. If the plugin cannot be initialized, then the server will still continue starting up, but the plugin will be disabled.
    - FORCE - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error.
    - FORCE_PLUS_PERMANENT - Enables the plugin. If the plugin cannot be initialized, then the server will fail to start with an error. In addition, the plugin cannot be uninstalled with UNINSTALL SONAME or UNINSTALL PLUGIN while the server is running.

    - See Plugin Overview: Configuring Plugin Activation at Server Startup for more information.
- **Commandline:** `--disks=value`
- **Data Type:** `enumerated`
- **Default Value:** `ON`
- **Valid Values:** `OFF`, `ON`, `FORCE`, `FORCE_PLUS_PERMANENT`

5.4.11.11 Compression Plugins

MariaDB starting with 10.7.0
Compressions plugins were added in a MariaDB 10.7.0 preview release.

Contents

The various MariaDB storage engines, such as InnoDB, RocksDB, Mroonga, can use different compression libraries.

Before MariaDB 10.7.0, each separate library would have to be compiled in in order to be available for use, resulting in numerous runtime/rpm/deb dependencies, most of which would never be used by users.

From MariaDB 10.7.0, five additional MariaDB compression libraries (besides the default zlib) are available as plugins (note that these affect InnoDB and Mroonga only; RocksDB still uses the compression algorithms from its own library):

- bzip2
- lzma
- lz4
- lzo
- snappy

Installing

Depending on how MariaDB was installed, the libraries may already be available for installation, or may first need to be installed as .deb or .rpm packages, for example:

```
apt-get install mariadb-plugin-provider-lz4
```

Once available, install as a plugin, for example:

```
INSTALL SONAME 'provider_lz4';
```

The compression algorithm can then be used, for example, in InnoDB compression:

```
SET GLOBAL innodb_compression_algorithm = lz4;
```

Upgrading

When upgrading from a release without compression plugins, if a non-zlib compression algorithm was used, those tables will be unreadable until the appropriate compression library is installed. mariadb-upgrade should be run. The `--force` option (to run mariadb-check) or `mariadb-check` itself will indicate any problems with compression, for example:

```
Warning  : MariaDB tried to use the LZMA compression, but its provider plugin is not loaded

Error    : Table 'test.t' doesn't exist in engine

status   : Operation failed
```

or

```
Error    : Table test/t is compressed with lzma, which is not currently loaded.
  Please load the lzma provider plugin to open the table

error    : Corrupt
```

In this case, the appropriate compression plugin should be installed, and the server restarted.

6 Training & Tutorials

This section provides tutorials for those who want to learn about MariaDB and related software.

Beginner MariaDB Articles
Tutorials for newcomers and beginners to MariaDB.

Basic MariaDB Articles
Basic tutorials -- more advanced than beginner.

Intermediate MariaDB Articles
Intermediate level tutorials for MariaDB developers and administrators.

Advanced MariaDB Articles
Tutorial articles for advanced MariaDB developers and administrators.

Books on MariaDB

Beginner Books
List of books on MariaDB for newcomers and beginners.

Intermediate and Advanced Books
List of books on MariaDB for intermediate and advanced developers and administrators.

Books on MariaDB Code
List of books on coding MariaDB Server and plugins.

6.1 Beginner MariaDB Articles

These tutorial articles were written for those who very little about databases and nothing about MariaDB. They are articles for newcomers and beginners.

A MariaDB Primer
A 10-minute primer on using MariaDB.

MariaDB Basics
Basic article on using MariaDB.

Getting Data from MariaDB
Extensive tutorial on using the SELECT statement.

Adding and Changing Data in MariaDB
Tutorial on using INSERT and UPDATE statements.

Altering Tables in MariaDB
Tutorial on using the ALTER TABLE statement.

Changing Times in MariaDB
Tutorial on using various time and date functions in MariaDB.

Doing Time with MariaDB
Tutorial about temporal data types and functions.

Importing Data into MariaDB
Tutorial on using the LOAD DATA INFILE statement.

Making Backups with mariadb-dump
Tutorial article on how to make back-ups with mariadb-dump.

MariaDB String Functions
Extensive tutorial on how to use several string functions.

Restoring Data from Dump Files

Tutorial on how to restore data from a mariadb-dump backup.

Basic SQL Statements

Basic SQL statements for structuring and manipulating data.

Connecting to MariaDB

Connecting to MariaDB with the basic connection parameters.

External Tutorials

Links to external MariaDB, MySQL and SQL tutorials.

Useful MariaDB Queries

Quick reference of commonly-used MariaDB queries.

6.1.1 A MariaDB Primer

Contents

This primer is designed to teach you the basics of getting information into and out of an existing MariaDB database using the mariadb command-line client program. It's not a complete reference and will not touch on any advanced topics. It is just a quick jump-start into using MariaDB.

Logging into MariaDB

Log into your MariaDB server from the command-line like so:

```
mariadb -u user_name -p -h ip_address db_name
```

Replace *user_name* with your database username. Replace *ip_address* with the host name or address of your server. If you're accessing MariaDB from the same server you're logged into, then don't include `-h` and the *ip_address*. Replace *db_name* with the name of the database you want to access (such as *test*, which sometimes comes already created for testing purposes - note that Windows does not create this database, and some setups may also have removed the `test` database by running mariadb-secure-installation, in which case you can leave the *db_name* out).

When prompted to enter your password, enter it. If your login is successful you should see something that looks similar to this:

```
MariaDB [test]>
```

This is where you will enter in all of your SQL statements. More about those later. For now, let's look at the components of the prompt: The "MariaDB" part means you that you are connected to a MariaDB database server. The word between the brackets is the name of your default database, the *test* database in this example.

The Basics of a Database

To make changes to a database or to retrieve data, you will need to enter an SQL statement. SQL stands for Structured Query Language. An SQL statement that requests data is called a query. Databases store information in tables. They're are similar to spreadsheets, but much more efficient at managing data.

Note that the *test* database may not contain any data yet. If you want to follow along with the primer, copy and paste the following into the mariadb client. This will create the tables we will use and add some data to them. Don't worry about understanding them yet; we'll get to that later.

```
CREATE DATABASE IF NOT EXISTS test;

USE test;

CREATE TABLE IF NOT EXISTS books (
  BookID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
  Title VARCHAR(100) NOT NULL,
  SeriesID INT, AuthorID INT);

CREATE TABLE IF NOT EXISTS authors
(id INT NOT NULL PRIMARY KEY AUTO_INCREMENT);

CREATE TABLE IF NOT EXISTS series
(id INT NOT NULL PRIMARY KEY AUTO_INCREMENT);

INSERT INTO books (Title,SeriesID,AuthorID)
VALUES('The Fellowship of the Ring',1,1),
      ('The Two Towers',1,1), ('The Return of the King',1,1),
      ('The Sum of All Men',2,2), ('Brotherhood of the Wolf',2,2),
      ('Wizardborn',2,2), ('The Hobbbit',0,1);
```

Notice the semi-colons used above. The mariadb client lets you enter very complex SQL statements over multiple lines. It won't send an SQL statement until you type a semi-colon and hit [Enter].

Let's look at what you've done so far. Enter the following:

```
SHOW TABLES;

+---------------+
| Tables_in_test |
+---------------+
| authors        |
| books          |
| series         |
+---------------+
3 rows in set (0.00 sec)
```

Notice that this displays a list of the tables in the database. If you didn't already have tables in your `test` database, your results should look the same as above. Let's now enter the following to get information about one of these tables:

```
DESCRIBE books;

+----------+-------------+------+-----+---------+----------------+
| Field    | Type        | Null | Key | Default | Extra          |
+----------+-------------+------+-----+---------+----------------+
| BookID   | int(11)     | NO   | PRI | NULL    | auto_increment |
| Title    | varchar(100) | NO   |     | NULL    |                |
| SeriesID | int(11)     | YES  |     | NULL    |                |
| AuthorID | int(11)     | YES  |     | NULL    |                |
+----------+-------------+------+-----+---------+----------------+
```

The main bit of information of interest to us is the *Field* column. The other columns provide useful information about the structure and type of data in the database, but the *Field* column gives us the names, which is needed to retrieve data from the table.

Let's retrieve data from the `books` table. We'll do so by executing a SELECT statement like so:

```
SELECT * FROM books;

+--------+---------------------------+----------+----------+
| BookID | Title                     | SeriesID | AuthorID |
+--------+---------------------------+----------+----------+
|      1 | The Fellowship of the Ring |        1 |        1 |
|      2 | The Two Towers             |        1 |        1 |
|      3 | The Return of the King     |        1 |        1 |
|      4 | The Sum of All Men         |        2 |        2 |
|      5 | Brotherhood of the Wolf    |        2 |        2 |
|      6 | Wizardborn                 |        2 |        2 |
|      7 | The Hobbbit                |        0 |        1 |
+--------+---------------------------+----------+----------+
7 rows in set (0.00 sec)
```

This SQL statement or query asks the database to show us all of the data in the `books` table. The wildcard ('`*`') character indicates to select all columns.

Inserting Data

Suppose now that we want to add another book to this table. We'll add the book, *Lair of Bones*. To insert data into a table, you would use the INSERT statement. To insert information on a book, we would enter something like this:

```
INSERT INTO books (Title, SeriesID, AuthorID)
VALUES ("Lair of Bones", 2, 2);

Query OK, 1 row affected (0.00 sec)
```

Notice that we put a list of columns in parentheses after the name of the table, then we enter the keyword `VALUES` followed by a list of values in parentheses--in the same order as the columns were listed. We could put the columns in a different order, as long as the values are in the same order as we list the columns. Notice the message that was returned indicates that the execution of the SQL statement went fine and one row was entered.

Execute the following SQL statement again and see what results are returned:

```
SELECT * FROM books;
```

You should see the data you just entered on the last row of the results. In looking at the data for the other books, suppose we notice that the title of the seventh book is spelled wrong. It should be spelled *The Hobbit*, not *The Hobbbit*. We will need to update the data for that row.

Modifying Data

To change data in a table, you will use the UPDATE statement. Let's change the spelling of the book mentioned above. To do this, enter the following:

```
UPDATE books
SET Title = "The Hobbit"
WHERE BookID = 7;

Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Notice the syntax of this SQL statement. The `SET` clause is where you list the columns and the values to set them. The `WHERE` clause says that you want to update only rows in which the `BookID` column has a value of `7`, of which there are only one. You can see from the message it returned that one row matched the `WHERE` clause and one row was changed. There are no warnings because everything went fine. Execute the SELECT from earlier to see that the data changed.

As you can see, using MariaDB isn't very difficult. You just have to understand the syntax of SQL since it doesn't allow for typing mistakes or things in the wrong order or other deviations.

See Also

- MariaDB Basics

6.1.2 MariaDB Basics

Contents

Connecting to MariaDB

MariaDB is a database system, a database server. To interface with the MariaDB server, you can use a client program, or you can write a program or script with one of the popular programming languages (e.g., PHP) using an API (Application Programming Interface) to interface with the MariaDB server. For the purposes of this article, we will focus on using the

default client that comes with MariaDB called `mariadb`. With this client, you can either enter queries from the command-line, or you can switch to a terminal, that is to say, monitor mode. To start, we'll use the latter.

From the Linux command-line, you would enter the following to log in as the root user and to enter monitor mode:

```
mariadb -u root -p -h localhost
```

The `-u` option is for specifying the user name. You would replace `root` here if you want to use a different user name. This is the MariaDB user name, not the Linux user name. The password for the MariaDB user `root` will probably be different from the Linux user `root`. Incidentally, it's not a good security practice to use the `root` user unless you have a specific administrative task to perform for which only `root` has the needed privileges.

The `-p` option above instructs the `mariadb` client to prompt you for the password. If the password for the `root` user hasn't been set yet, then the password is blank and you would just hit [Enter] when prompted. The `-h` option is for specifying the host name or the IP address of the server. This would be necessary if the client is running on a different machine than the server. If you've secure-shelled into the server machine, you probably won't need to use the host option. In fact, if you're logged into Linux as `root`, you won't need the user option—the `-p` is all you'll need. Once you've entered the line above along with the password when prompted, you will be logged into MariaDB through the client. To exit, type quit or exit and press [Enter].

Creating a Structure

In order to be able to add and to manipulate data, you first have to create a database structure. Creating a database is simple. You would enter something like the following from within the mariadb client:

```
CREATE DATABASE bookstore;

USE bookstore;
```

This very minimal, first SQL statement will create a sub-directory called bookstore on the Linux filesystem in the directory which holds your MariaDB data files. It won't create any data, obviously. It'll just set up a place to add tables, which will in turn hold data. The second SQL statement above will set this new database as the default database. It will remain your default until you change it to a different one or until you log out of MariaDB.

The next step is to begin creating tables. This is only a little more complicated. To create a simple table that will hold basic data on books, we could enter something like the following:

```
CREATE TABLE books (
isbn CHAR(20) PRIMARY KEY,
title VARCHAR(50),
author_id INT,
publisher_id INT,
year_pub CHAR(4),
description TEXT );
```

This SQL statement creates the table books with six fields, or rather columns. The first column (isbn) is an identification number for each row—this name relates to the unique identifier used in the book publishing business. It has a fixed-width character type of 20 characters. It will be the primary key column on which data will be indexed. The column data type for the book title is a variable width character column of fifty characters at most. The third and fourth columns will be used for identification numbers for the author and the publisher. They are integer data types. The fifth column is used for the publication year of each book. The last column is for entering a description of each book. It's a TEXT data type, which means that it's a variable width column and it can hold up to 65535 bytes of data for each row. There are several other data types that may be used for columns, but this gives you a good sampling.

To see how the table we created looks, enter the following SQL statement:

```
DESCRIBE books;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| isbn        | char(20)    | NO   | PRI | NULL    |       |
| title       | varchar(50) | YES  |     | NULL    |       |
| author_id   | int(11)     | YES  |     | NULL    |       |
| publisher_id | int(11)    | YES  |     | NULL    |       |
| year_pub    | char(4)     | YES  |     | NULL    |       |
| description | text        | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
```

To change the settings of a table, you can use the ALTER TABLE statement. I'll cover that statement in another article. To delete a table completely (including its data), you can use the DROP TABLE statement, followed by the table name. Be careful with this statement since it's not reversible.

The next table we'll create for our examples is the authors table to hold author information. This table will save us from having to enter the author's name and other related data for each book written by each author. It also helps to ensure consistency of data: there's less chance of inadvertent spelling deviations.

```
CREATE TABLE authors
(author_id INT AUTO_INCREMENT PRIMARY KEY,
name_last VARCHAR(50),
name_first VARCHAR(50),
country VARCHAR(50) );
```

We'll join this table to the books table as needed. For instance, we would use it when we want a list of books along with their corresponding authors' names. For a real bookstore's database, both of these tables would probably have more columns. There would also be several more tables. For the examples that follow, these two tables as they are will be enough.

Minor Items

Before moving on to the next step of adding data to the tables, let me point out a few minor items that I've omitted mentioning. SQL statements end with a semi-colon (or a \G). You can spread an SQL statement over multiple lines. However, it won't be passed to the server by the client until you terminate it with a semi-colon and hit [Enter]. To cancel an SQL statement once you've started typing it, enter `\c` and press [Enter].

As a basic convention, reserved words are printed in all capital letters. This isn't necessary, though. MariaDB is case-insensitive with regards to reserved words. Database and table names, however, are case-sensitive on Linux. This is because they reference the related directories and files on the filesystem. Column names aren't case sensitive since they're not affected by the filesystem, per se. As another convention, we use lower-case letters for structural names (e.g., table names). It's a matter of preference for deciding on names.

Entering Data

The primary method for entering data into a table is to use the INSERT statement. As an example, let's enter some information about an author into the authors table. We'll do that like so:

```
INSERT INTO authors
(name_last, name_first, country)
VALUES('Kafka', 'Franz', 'Czech Republic');
```

This will add the name and country of the author Franz Kafka to the authors table. We don't need to give a value for the author_id since that column was created with the AUTO_INCREMENT option. MariaDB will automatically assign an identification number. You can manually assign one, especially if you want to start the count at a higher number than 1 (e.g., 1000). Since we are not providing data for all of the columns in the table, we have to list the columns for which we are giving data and in the order that the data is given in the set following the VALUES keyword. This means that we could give the data in a different order.

For an actual database, we would probably enter data for many authors. We'll assume that we've done that and move on to entering data for some books. Below is an entry for one of Kafka's books:

```
INSERT INTO books
(title, author_id, isbn, year_pub)
VALUES('The Castle', '1', '0805211063', '1998');
```

This adds a record for Kafka's book, *The Castle*. Notice that we mixed up the order of the columns, but it still works because both sets agree. We indicate that the author is Kafka by giving a value of 1 for the author_id. This is the value that was assigned by MariaDB when we entered the row for Kafka earlier. Let's enter a few more books for Kafka, but by a different method:

```
INSERT INTO books
(title, author_id, isbn, year_pub)
VALUES('The Trial', '1', '0805210407', '1995'),
('The Metamorphosis', '1', '0553213695', '1995'),
('America', '1', '0805210644', '1995');
```

In this example, we've added three books in one statement. This allows us to give the list of column names once. We also give the keyword `VALUES` only once, followed by a separate set of values for each book, each contained in parentheses and separated by commas. This cuts down on typing and speeds up the process. Either method is fine and both have their

advantages. To be able to continue with our examples, let's assume that data on thousands of books has been entered. With that behind us, let's look at how to retrieve data from tables.

Retrieving Data

The primary method of retrieving data from tables is to use a SELECT statement. There are many options available with the SELECT statement, but you can start simply. As an example, let's retrieve a list of book titles from the books table:

```
SELECT title
FROM books;
```

This will display all of the rows of books in the table. If the table has thousands of rows, MariaDB will display thousands. To limit the number of rows retrieved, we could add a LIMIT clause to the SELECT statement like so:

```
SELECT title
FROM books
LIMIT 5;
```

This will limit the number of rows displayed to five. To be able to list the author's name for each book along with the title, you will have to join the books table with the authors table. To do this, we can use the JOIN clause like so:

```
SELECT title, name_last
FROM books
JOIN authors USING (author_id);
```

Notice that the primary table from which we're drawing data is given in the `FROM` clause. The table to which we're joining is given in the JOIN clause along with the commonly named column (i.e., author_id) that we're using for the join.

To retrieve the titles of only books written by Kafka based on his name (not the author_id), we would use the `WHERE` clause with the SELECT statement. This would be entered like the following:

```
SELECT title AS 'Kafka Books'
FROM books
JOIN authors USING (author_id)
WHERE name_last = 'Kafka';

+-------------------+
| Kafka Books       |
+-------------------+
| The Castle        |
| The Trial         |
| The Metamorphosis |
| America           |
+-------------------+
```

This statement will list the titles of Kafka books stored in the database. Notice that I've added the `AS` parameter next to the column name title to change the column heading in the results set to Kafka Books. This is known as an alias. Looking at the results here, we can see that the title for one of Kafka's books is incorrect. His book Amerika is spelled above with a "c" in the table instead of a "k". This leads to the next section on changing data.

Changing & Deleting Data

In order to change existing data, a common method is to use the UPDATE statement. When changing data, though, we need to be sure that we change the correct rows. In our example, there could be another book with the title *America* written by a different author. Since the key column isbn has only unique numbers and we know the ISBN number for the book that we want to change, we can use it to specify the row.

```
UPDATE books
SET title = 'Amerika'
WHERE isbn = '0805210644';
```

This will change the value of the title column for the row specified. We could change the value of other columns for the same row by giving the column = value for each, separated by commas.

If we want to delete a row of data, we can use the DELETE statement. For instance, suppose that our fictitious bookstore has decided no longer to carry books by John Grisham. By first running a SELECT statement, we determine the identification number for the author to be 2034. Using this author identification number, we could enter the following:

```
DELETE FROM books
WHERE author_id = '2034';
```

This statement will delete all rows from the table books for the author_id given. To do a clean job of it, we'll have to do the same for the authors table. We would just replace the table name in the statement above; everything else would be the same.

Conclusion

This is a very basic primer for using MariaDB. Hopefully, it gives you the idea of how to get started with MariaDB. Each of the SQL statements mentioned here have several more options and clauses each. We will cover these statements and others in greater detail in other articles. For now, though, you can learn more about them from experimenting and by further reading of the Knowledge Base online documentation. A downloadable PDF of much of the documentation is available here.

6.1.3 Getting Data from MariaDB

Contents

The simplest way to retrieve data from MariaDB is to use the SELECT statement. Since the SELECT statement is an essential SQL statement, it has many options available with it. It's not necessary to know or use them all—you could execute very basic SELECT statements if that satisfies your needs. However, as you use MariaDB more, you may need more powerful SELECT statements. In this article we will go through the basics of SELECT and will progress to more involved SELECT statements;we will move from the beginner level to the more intermediate and hopefully you will find some benefit from this article regardless of your skill level. For absolute beginners who are just starting with MariaDB, you may want to read the MariaDB Basics article.

Basic Elements

The basic, minimal elements of the SELECT statement call for the keyword `SELECT`, of course, the columns to select or to retrieve, and the table from which to retrieve rows of data. Actually, for the columns to select, we can use the asterisk as a wildcard to select all columns in a particular table. Using a database from a fictitious bookstore, we might enter the following SQL statement to get a list of all columns and rows in a table containing information on books:

```
SELECT * FROM books;
```

This will retrieve all of the data contained in the `books` table. If we want to retrieve only certain columns, we would list them in place of the asterisk in a comma-separated list like so:

```
SELECT isbn, title, author_id
FROM books;
```

This narrows the width of the results set by retrieving only three columns, but it still retrieves all of the rows in the table. If the table contains thousands of rows of data, this may be more data than we want. If we want to limit the results to just a few books, say five, we would include what is known as a LIMIT clause:

```
SELECT isbn, title, author_id
FROM books
LIMIT 5;
```

This will give us the first five rows found in the table. If we want to get the next ten found, we would add a starting point parameter just before the number of rows to display, separated by a comma:

```
SELECT isbn, title, author_id
FROM books
LIMIT 5, 10;
```

Selectivity and Order

The previous statements have narrowed the number of columns and rows retrieved, but they haven't been very selective. Suppose that we want only books written by a certain author, say Dostoevsky. Looking in the authors table we find that his author identification number is 4729. Using a `WHERE` clause, we can retrieve a list of books from the database for this particular author like so:

```
SELECT isbn, title
FROM books
WHERE author_id = 4729
LIMIT 5;
```

I removed the author_id from the list of columns to select, but left the basic LIMIT clause in because we want to point out that the syntax is fairly strict on ordering of clauses and flags. You can't enter them in any order. You'll get an error in return.

The SQL statements we've looked at thus far will display the titles of books in the order in which they're found in the database. If we want to put the results in alphanumeric order based on the values of the title column, for instance, we would add an ORDER BY clause like this:

```
SELECT isbn, title
FROM books
WHERE author_id = 4729
ORDER BY title ASC
LIMIT 5;
```

Notice that the ORDER BY clause goes after the `WHERE` clause and before the LIMIT clause. Not only will this statement display the rows in order by book title, but it will retrieve only the first five based on the ordering. That is to say, MariaDB will first retrieve all of the rows based on the `WHERE` clause, order the data based on the ORDER BY clause, and then display a limited number of rows based on the LIMIT clause. Hence the reason for the order of clauses. You may have noticed that we slipped in the ASC flag. It tells MariaDB to order the rows in ascending order for the column name it follows. It's not necessary, though, since ascending order is the default. However, if we want to display data in descending order, we would replace the flag with `DESC`. To order by more than one column, additional columns may be given in the ORDER BY clause in a comma separated list, each with the `ASC` or `DESC` flags if preferred.

Friendlier and More Complicated

So far we've been working with one table of data containing information on books for a fictitious bookstore. A database will usually have more than one table, of course. In this particular database, there's also one called authors in which the name and other information on authors is contained. To be able to select data from two tables in one SELECT statement, we will have to tell MariaDB that we want to join the tables and will need to provide a join point. This can be done with a JOIN clause as shown in the following SQL statement, with the results following it:

```
SELECT isbn, title,
CONCAT(name_first, ' ', name_last) AS author
FROM books
JOIN authors USING (author_id)
WHERE name_last = 'Dostoevsky'
ORDER BY title ASC
LIMIT 5;

+------------+-----------------------+-------------------+
| isbn       | title                 | author            |
+------------+-----------------------+-------------------+
|  0553212168 | Brothers Karamozov    | Fyodor Dostoevsky |
|  0679420290 | Crime & Punishment    | Fyodor Dostoevsky |
|  0553211757 | Crime & Punishment    | Fyodor Dostoevsky |
|  0192834118 | Idiot                 | Fyodor Dostoevsky |
|  067973452X | Notes from Underground | Fyodor Dostoevsky |
+------------+-----------------------+-------------------+
5 rows in set (0.00 sec)
```

Our SELECT statement is getting hefty, but it's the same one to which we've been adding. Don't let the clutter fluster you. Looking for the new elements, let's focus on the JOIN clause first. There are a few possible ways to construct a join. This method works if you're using a newer version of MariaDB and if both tables contain a column of the same name and value. Otherwise you'll have to redo the JOIN clause to look something like this:

```
...
JOIN authors ON author_id = row_id
...
```

This excerpt is based on the assumption that the key field in the authors table is not called author_id, but row_id instead. There's much more that can be said about joins, but that would make for a much longer article. If you want to learn more on joins, look at MariaDB's documentation page on JOIN syntax.

Looking again at the last full SQL statement above, you must have spotted the CONCAT() function that we added to the on-going example statement. This string function takes the values of the columns and strings given and pastes them together, to give one neat field in the results. We also employed the AS parameter to change the heading of the results set for the field to author. This is much tider. Since we joined the books and the authors tables together, we were able to search for books based on the author's last name rather than having to look up the author ID first. This is a much friendlier method, albeit more complicated. Incidentally, we can have MariaDB check columns from both tables to narrow our search. We would just add *column* = *value* pairs, separated by commas in the WHERE clause. Notice that the string containing the author's name is wrapped in quotes—otherwise, the string would be considered a column name and we'd get an error.

The name Dostoevsky is sometimes spelled Dostoevskii, as well as a few other ways. If we're not sure how it's spelled in the authors table, we could use the LIKE operator instead of the equal-sign, along with a wildcard. If we think the author's name is probably spelled either of the two ways mentioned, we could enter something like this:

```
SELECT isbn, title,
CONCAT(name_first, ' ', name_last) AS author
FROM books
JOIN authors USING (author_id)
WHERE name_last LIKE 'Dostoevsk%'
ORDER BY title ASC
LIMIT 5;
```

This will match any author last name starting with Dostoevsk. Notice that the wildcard here is not an asterisk, but a percent-sign.

Some Flags

There are many flags or parameters that can be used in a SELECT statement. To list and explain all of them with examples would make this a very lengthy article. The reality is that most people never use some of them anyway. So, let's take a look at a few that you may find useful as you get more involved with MariaDB or if you work with large tables on very active servers.

The first flag that may be given, it goes immediately after the `SELECT` keyword, is `ALL`. By default, all rows that meet the requirements of the various clauses given are selected, so this isn't necessary. If instead we would only want the first occurrence of a particular criteria to be displayed, we could add the DISTINCT option. For instance, for authors like Dostoevsky there will be several printings of a particular title. In the results shown earlier you may have noticed that there were two copies of *Crime & Punishment* listed, however they have different ISBN numbers and different publishers. Suppose that for our search we only want one row displayed for each title. We could do that like so:

```
SELECT DISTINCT isbn, title
FROM books
JOIN authors USING (author_id)
WHERE name_last = 'Dostoevsky'
ORDER BY title;
```

We've thinned out the ongoing SQL statement a bit for clarity. This statement will result in only one row displayed for *Crime & Punishment* and it will be the first one found.

If we're retrieving data from an extremely busy database, by default any other SQL statements entered simultaneously which are changing or updating data will be executed before a SELECT statement. SELECT statements are considered to be of lower priority. However, if we would like a particular SELECT statement to be given a higher priority, we can add the keyword HIGH_PRIORITY. Modifying the previous SQL statement for this factor, we would enter it like this:

```
SELECT DISTINCT HIGH_PRIORITY isbn, title
FROM books
JOIN authors USING (author_id)
WHERE name_last = 'Dostoevsky'
ORDER BY title;
```

You may have noticed in the one example earlier in which the results are shown, that there's a status line displayed that specifies the number of rows in the results set. This is less than the number of rows that were found in the database that met the statement's criteria. It's less because we used a LIMIT clause. If we add the SQL_CALC_FOUND_ROWS flag just before the column list, MariaDB will calculate the number of columns found even if there is a LIMIT clause.

```
SELECT SQL_CALC_FOUND_ROWS isbn, title
FROM books
JOIN authors USING (author_id)
WHERE name_last = 'Dostoevsky'
LIMIT 5;
```

To retrieve this information, though, we will have to use the FOUND_ROWS() function like so:

```
SELECT FOUND_ROWS();

+--------------+
| FOUND_ROWS() |
+--------------+
|           26 |
+--------------+
```

This value is temporary and will be lost if the connection is terminated. It cannot be retrieved by any other client session. It relates only to the current session and the value for the variable when it was last calculated.

Conclusion

There are several more parameters and possibilities for the SELECT statement that we had to skip to keep this article a reasonable length. A popular one that we left out is the GROUP BY clause for calculating aggregate data for columns (e.g., an average). There are several flags for caching results and a clause for exporting a results set to a text file. If you would like to learn more about SELECT and all of the options available, look at the on-line documentation for SELECT statements.

6.1.4 Adding and Changing Data in MariaDB

Contents

There are several ways to add and to change data in MariaDB. There are a few SQL statements that you can use, each with a few options. Additionally, there are twists that you can do by mixing SQL statements together with various clauses. In this article, we will explore the ways in which data can be added and changed in MariaDB.

Adding Data

To add data to a table in MariaDB, you will need to use the INSERT statement. Its basic, minimal syntax is the command INSERT followed by the table name and then the keyword VALUES with a comma separated list of values contained in parentheses:

```
INSERT table1
VALUES('text1','text2','text3');
```

In this example, text is added to a table called table1, which contains only three columns —the same number of values that we're inserting. The number of columns must match. If you don't want to insert data into all of the columns of a table, though, you could name the columns desired:

```
INSERT INTO table1
(col3, col1)
VALUES('text3','text1');
```

Notice that the keyword `INTO` was added here. This is optional and has no effect on MariaDB. It's only a matter of grammatical preference. In this example we not only name the columns, but we list them in a different order. This is acceptable to MariaDB. Just be sure to list the values in the same order. If you're going to insert data into a table and want to specify all of the values except one (say the key column since it's an auto-incremented one), then you could just give a value of DEFAULT to keep from having to list the columns. Incidentally, you can give the column names even if you're naming all of them. It's just unnecessary unless you're going to reorder them as we did in this last example.

When you have many rows of data to insert into the same table, it can be more efficient to insert all of the rows in one SQL

statement. Multiple row insertions can be done like so:

```
INSERT IGNORE
INTO table2
VALUES('id1','text','text'),
('id2','text','text'),
('id2','text','text');
```

Notice that the keyword VALUES is used only once and each row is contained in its own set of parentheses and each set is separated by commas. We've added an intentional mistake to this example: We are attempting to insert three rows of data into table2 for which the first column happens to be a `UNIQUE` key field. The third row entered here has the same identification number for the key column as the second row. This would normally result in an error and none of the three rows would be inserted. However, since the statement has an `IGNORE` flag, duplicates will be ignored and not inserted, but the other rows will still be inserted. So, the first and second rows above will be inserted and the third one won't.

Priority

An INSERT statement takes priority over read statements (i.e., SELECT statements). An INSERT will lock the table and force other clients to wait until it's finished. On a busy MariaDB server that has many simultaneous requests for data, this could cause users to experience delays when you run a script that performs a series of INSERT statements. If you don't want user requests to be put on hold and you can wait to insert the data, you could use the `LOW_PRIORITY` flag:

```
INSERT LOW_PRIORITY
INTO table1
VALUES('text1','text2','text3');
```

The `LOW_PRIORITY` flag will put the INSERT statement in queue, waiting for all current and pending requests to be completed before it's performed. If new requests are made while a low priority statement is waiting, then they are put ahead of it in the queue. MariaDB does not begin to execute a low priority statement until there are no other requests waiting. Once the transaction begins, though, the table is locked and any other requests for data from the table that come in after it starts must wait until it's completed. Because it locks the table, low priority statements will prevent simultaneous insertions from other clients even if you're dealing with a MyISAM table. Incidentally, notice that the `LOW_PRIORITY` flag comes before the `INTO`.

One potential inconvenience with an `INSERT LOW_PRIORITY` statement is that the client will be tied up waiting for the statement to be completed successfully. So if you're inserting data into a busy server with a low priority setting using the mariadb client, your client could be locked up for minutes, maybe hours depending on how busy your server is at the time. As an alternative either to making other clients with read requests wait or to having your client wait, you can use the DELAYED flag instead of the `LOW_PRIORITY` flag:

```
INSERT DELAYED
INTO table1
VALUES('text1','text2','text3');
```

MariaDB will take the request as a low priority one and put it on its list of tasks to perform when it has a break. However, it will immediately release the client so that the client can go on to enter other SQL statements or even exit. Another advantage of this method is that multiple `INSERT DELAYED` requests are batched together for block insertion when there is a gap, making the process potentially faster than `INSERT LOW_PRIORITY`. The flaw in this choice, however, is that the client is never told if a delayed insertion is successfully made or not. The client is informed of error messages when the statement is entered—the statement has to be valid before it will be queued—but it's not told of problems that occur after it's accepted. This brings up another flaw: delayed insertions are stored in the server's memory. So if the MariaDB daemon (mariadbd) dies or is manually killed, then the transactions are lost and the client is not notified of the failure. So DELAYED is not always a good alternative.

Contingent Additions

As an added twist to INSERT, you can combine it with a SELECT statement. Suppose that you have a table called employees which contains employee information for your company. Suppose further that you have a column to indicate whether an employee is on the company's softball team. However, you one day decide to create a separate database and table for the softball team's data that someone else will administer. To get the database ready for the new administrator, you have to copy some data for team members to the new table. Here's one way you can accomplish this task:

```
INSERT INTO softball_team
(last, first, telephone)
SELECT name_last, name_first, tel_home
FROM company.employees
WHERE softball='Y';
```

In this SQL statement the columns in which data is to be inserted into are listed, then the complete SELECT statement follows with the appropriate WHERE clause to determine if an employee is on the softball team. Since we're executing this statement from the new database and since the table employees is in a separate database called company, we have to specify it as you see here. By the way, INSERT...SELECT statements cannot be performed on the same table.

Replacement Data

When you're adding massive amounts of data to a table that has a key field, as mentioned earlier, you can use the `IGNORE` flag to prevent duplicates from being inserted, but still allow unique rows to be entered. However, there may be times when you actually want to replace the rows with the same key fields with the new ones. In such a situation, instead of using INSERT you can use a REPLACE statement:

```
REPLACE LOW_PRIORITY
INTO table2 (id, col1, col2)
VALUES('id1','text','text'),
('id2','text','text'),
('id3','text','text');
```

Notice that the syntax is the same as an INSERT statement. The flags all have the same effect, as well. Also, multiple rows may be inserted, but there's no need for the `IGNORE` flag since duplicates won't happen—the originals are just overwritten. Actually, when a row is replaced, it's first deleted completely and the new row is then inserted. Any columns without values in the new row will be given the default values for the columns. None of the values of the old row are kept. Incidentally, REPLACE will also allow you to combine it with a SELECT statement as we saw with the INSERT statement earlier.

Updating Data

If you want to change the data contained in existing records, but only for certain columns, then you would need to use an UPDATE statement. The syntax for UPDATE is a little bit different from the syntax shown before for INSERT and REPLACE statements:

```
UPDATE LOW_PRIORITY table3
SET col1 = 'text-a', col2='text-b'
WHERE id < 100;
```

In the SQL statement here, we are changing the value of the two columns named individually using the `SET` clause. Incidentally, the `SET` clause optionally can be used in INSERT and REPLACE statements, but it eliminates the multiple row option. In the statement above, we're also using a `WHERE` clause to determine which records are changed: only rows with an id that has a value less than 100 are updated. Notice that the `LOW_PRIORITY` flag can be used with this statement, too. The `IGNORE` flag can be used, as well.

A useful feature of the UPDATE statement is that it allows the use of the current value of a column to update the same column. For instance, suppose you want to add one day to the value of a date column where the date is a Sunday. You could do the following:

```
UPDATE table5
SET col_date = DATE_ADD(col_date, INTERVAL 1 DAY)
WHERE DAYOFWEEK(col_date) = 1;
```

For rows where the day of the week is Sunday, the DATE_ADD() function will take the value of col_date before it's updated and add one day to it. MariaDB will then take this sum and set col_date to it.

There are a couple more twists that you can now do with the UPDATE statement: if you want to update the rows in a specific order, you can add an ORDER BY clause. You can also limit the number of rows that are updated with a LIMIT clause. Below is an example of both of these clauses:

```
UPDATE LOW_PRIORITY table3
SET col1='text-a', col2='text-b'
WHERE id < 100
ORDER BY col3 DESC
LIMIT 10;
```

The ordering can be descending as indicated here by the `DESC` flag, or ascending with either the ASC flag or by just leaving it out, as ascending is the default. The LIMIT clause, of course, limits the number of rows affected to ten here.

If you want to refer to multiple tables in one UPDATE statement, you can do so like this:

```
UPDATE table3, table4
SET table3.col1 = table4.col1
WHERE table3.id = table4.id;
```

Here we see a join between the two tables named. In table3, the value of col1 is set to the value of the same column in table4 where the values of id from each match. We're not updating both tables here; we're just accessing both. We must specify the table name for each column to prevent an ambiguity error. Incidentally, ORDER BY and LIMIT clauses aren't allowed with multiple table updates.

There's another combination that you can do with the INSERT statement that we didn't mention earlier. It involves the UPDATE statement. When inserting multiple rows of data, if you want to note which rows had potentially duplicate entries and which ones are new, you could add a column called status and change it's value accordingly with a statement like this one:

```
INSERT IGNORE INTO table1
(id, col1, col2, status)
VALUES('1012','text','text','new'),
('1025,'text','text','new'),
('1030','text','text','new')
ON DUPLICATE KEY
UPDATE status = 'old';
```

Because of the `IGNORE` flag, errors will not be generated, duplicates won't be inserted or replaced, but the rest will be added. Because of the ON DUPLICATE KEY, the column status of the original row will be set to old when there are duplicate entry attempts. The rest will be inserted and their status set to new.

Conclusion

As you can see from some of these SQL statements, MariaDB offers you quite a few ways to add and to change data. In addition to these methods, there are also some bulk methods of adding and changing data in a table. You could use the LOAD DATA INFILE statement and the mariadb-dump command-line utility. These methods are covered in another article on Importing Data into MariaDB.

6.1.5 Altering Tables in MariaDB

Contents

Despite a MariaDB developer's best planning, occasionally one needs to change the structure or other aspects of tables. This is not very difficult, but some developers are unfamiliar with the syntax for the functions used in MariaDB to accomplish this. And some changes can be very frustrating. In this article we'll explore the ways to alter tables in MariaDB and we'll give some precautions about related potential data problems.

Before Beginning

For the examples in this article, we will refer to a database called `db1` containing a table called `clients`. The `clients` table is for keeping track of client names and addresses. To start off, we'll enter a DESCRIBE statement to see what the table looks like:

```
DESCRIBE clients;

+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| cust_id     | int(11)     |      | PRI | 0       |       |
| name        | varchar(25) | YES  |     | NULL    |       |
| address     | varchar(25) | YES  |     | NULL    |       |
| city        | varchar(25) | YES  |     | NULL    |       |
| state       | char(2)     | YES  |     | NULL    |       |
| zip         | varchar(10) | YES  |     | NULL    |       |
| client_type | varchar(4)  | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
```

This is a very simple table that will hold very little information. However, it's sufficient for the examples here in which we will change several of its columns. Before doing any structural changes to a table in MariaDB, especially if it contains data, one should make a backup of the table to be changed. There are a few ways to do this, but some choices may not be permitted by your web hosting company. Even if your database is on your own server, though, the mariadb-dump utility is typically the best tool for making and restoring backups in MariaDB, and it's generally permitted by web hosting companies. To backup the clients table with mariadb-dump, we will enter the following from the command-line:

```
mariadb-dump --user='username' --password='password' --add-locks db1 clients > clients.sql
```

As you can see, the username and password are given on the first line. On the next line, the `--add-locks` option is used to lock the table before backing up and to unlock automatically it when the backup is finished. There are many other options in mariadb-dump that could be used, but for our purposes this one is all that's necessary. Incidentally, this statement can be entered in one line from the shell (i.e., not from the `mariadb` client), or it can be entered on multiple lines as shown here by using the back-slash (i.e., `/`) to let the shell know that more is to follow. On the third line above, the database name is given, followed by the table name. The redirect (i.e., `>`) tells the shell to send the results of the dump to a text file called `clients.sql` in the current directory. A directory path could be put in front of the file name to create the file elsewhere. If the table should need to be restored, the following can be run from the shell:

```
mariadb --user='username' --password='password' db1 < clients.sql
```

Notice that this line does not use the `mariadb-dump` utility. It uses the `mariadb` client from the outside, so to speak. When the dump file (`clients.sql`) is read into the database, it will delete the `clients` table and it's data in MariaDB before restoring the backup copy with its data. So be sure that users haven't added data in the interim. In the examples in this article, we are assuming that there isn't any data in the tables yet.

Basic Addition and More

In order to add a column to an existing MariaDB table, one would use the ALTER TABLE statement. To demonstrate, suppose that it has been decided that there should be a column for the client's account status (i.e., active or inactive). To make this change, the following is entered:

```
ALTER TABLE clients
ADD COLUMN status CHAR(2);
```

This will add the column `status` to the end with a fixed width of two characters (i.e., *AC* for active and *IA* for inactive). In looking over the table again, it's decided that another field for client apartment numbers or the like needs to be added. That data could be stored in the address column, but it would better for it to be in a separate column. An ALTER TABLE statement could be entered like above, but it will look tidier if the new column is located right after the address column. To do this, we'll use the `AFTER` option:

```
ALTER TABLE clients
ADD COLUMN address2 varchar(25)
AFTER address;
```

By the way, to add a column to the first position, you would replace the last line of the SQL statement above to read like this:

```
...
FIRST;
```

Before moving on, let's take a look at the table's structure so far:

```
DESCRIBE clients;

+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| cust_id     | int(11)     |      | PRI | 0       |       |
| name        | varchar(25) | YES  |     | NULL    |       |
| address     | varchar(25) | YES  |     | NULL    |       |
| address2    | varchar(25) | YES  |     | NULL    |       |
| city        | varchar(25) | YES  |     | NULL    |       |
| state       | char(2)     | YES  |     | NULL    |       |
| zip         | varchar(10) | YES  |     | NULL    |       |
| client_type | varchar(4)  | YES  |     | NULL    |       |
| status      | char(2)     | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
```

Changing One's Mind

After looking over the above table display, it's decided that it might be better if the status column has the choices of 'AC' and 'IA' enumerated. To make this change, we'll enter the following SQL statement:

```
ALTER TABLE clients
CHANGE status status enum('AC','IA');
```

Notice that the column name status is specified twice. Although the column name isn't being changed, it still must be respecified. To change the column name (from `status` to `active`), while leaving the enumerated list the same, we specify the new column name in the second position:

```
ALTER TABLE clients
CHANGE status active ENUM('AC','IA');
```

Here we have the current column name and then the new column name, along with the data type specifications (i.e., `ENUM`), even though the result is only a name change. With the `CHANGE` clause everything must be stated, even items that are not to be changed.

In checking the table structure again, more changes are decided on: The column address is to be renamed to address1 and changed to forty characters wide. Also, the enumeration of active is to have 'yes' and 'no' choices. The problem with changing enumerations is that data can be clobbered in the change if one isn't careful. We've glossed over this possibility before because we are assuming that clients is empty. Let's take a look at how the modifications suggested could be made with the table containing data:

```
ALTER TABLE clients
CHANGE address address1 varchar(40),
MODIFY active enum('yes','no','AC','IA');

UPDATE clients
SET active = 'yes'
WHERE active = 'AC';

UPDATE clients
SET active = 'no'
WHERE active = 'IA';

ALTER TABLE clients
MODIFY active enum('yes','no');
```

The first SQL statement above changes address and modifies active in preparation for the transition. Notice the use of a `MODIFY` clause. It works the same as `CHANGE`, but it is only used for changing data types and not column names. Therefore, the column name isn't respecified. Notice also that there is a comma after the CHANGE clause. You can string several `CHANGE` and `MODIFY` clauses together with comma separators. We've enumerated both the new choices and the old ones to be able to migrate the data. The two UPDATE statements are designed to adjust the data accordingly and the last ALTER TABLE statement is to remove the old enumerated choices for the status column.

In talking to the boss, we find out that the `client_type` column isn't going to be used. So we enter the following in MariaDB:

```
ALTER TABLE clients
DROP client_type;
```

This deletes `client_type` and its data, but not the whole table, obviously. Nevertheless, it is a permanent and non-reversible action; there won't be a confirmation request when using the mariadb client. This is how it is with all MariaDB DROP statements and clauses. So be sure that you want to delete an element and its data before using a `DROP`. As mentioned earlier, be sure that you have a backup of your tables before doing any structured changes.

The Default

You may have noticed that the results of the DESCRIBE statements shown before have a heading called 'Default' and just about all of the fields have a default value of NULL. This means that there are no default values and a null value is allowed and will be used if a value isn't specified when a row is created. To be able to specify a default value other than NULL, an ALTER TABLE statement can be entered with a `SET` clause. Suppose we're located in Louisiana and we want a default value of 'LA' for state since that's where our clients are usually located. We would enter the following to set the default:

```
ALTER TABLE clients
ALTER state SET DEFAULT 'LA';
```

Notice that the second line starts with `ALTER` and not `CHANGE`. If we change our mind about having a default value for state, we would enter the following to reset it back to NULL (or whatever the initial default value would be based on the data type):

```
ALTER TABLE clients
ALTER state DROP DEFAULT;
```

This particular `DROP` doesn't delete data, by the way.

Indexes

One of the most irritating tasks in making changes to a table for newcomers is dealing with indexes. If they try to rename a column that is indexed by only using an ALTER TABLE statement like we used earlier, they will get a frustrating and confusing error message:

```
ALTER TABLE clients
CHANGE cust_id client_id INT
PRIMARY KEY;

ERROR 1068: Multiple primary key defined
```

If they're typing this column change from memory, they will wear themselves out trying different deviations thinking that they remembered the syntax wrong. What most newcomers to MariaDB don't seem to realize is that the index is separate from the indexed column. To illustrate, let's take a look at the index for clients by using the SHOW INDEX statement:

```
SHOW INDEX FROM clientsG;

*************************** 1. row ***************************
           Table: clients
      Non_unique: 0
        Key_name: PRIMARY
    Seq_in_index: 1
     Column_name: cust_id
       Collation: A
     Cardinality: 0
        Sub_part: NULL
          Packed: NULL
         Comment:
1 row in set (0.00 sec)
```

The text above shows that behind the scenes there is an index associated with `cust_id`. The column `cust_id` is not the index. Incidentally, the G at the end of the SHOW INDEX statement is to display the results in portrait instead of landscape format. Before the name of an indexed column can be changed, the index related to it must be eliminated. The index is not automatically changed or deleted. Therefore, in the example above, MariaDB thinks that the developer is trying to create another primary key index. So, a `DROP` clause for the index must be entered first and then a `CHANGE` for the column name can be made along with the establishing of a new index:

```
ALTER TABLE clients
DROP PRIMARY KEY,
CHANGE cust_id
client_id INT PRIMARY KEY;
```

The order of these clauses is necessary. The index must be dropped before the column can be renamed. The syntax here is for a `PRIMARY KEY`. There are other types of indexes, of course. To change a column that has an index type other than a `PRIMARY KEY`. Assuming for a moment that `cust_id` has a `UNIQUE` index, this is what we would enter to change its name:

```
ALTER TABLE clients
DROP UNIQUE cust_id
CHANGE cust_id
client_id INT UNIQUE;
```

Although the index type can be changed easily, MariaDB won't permit you to do so when there are duplicate rows of data and when going from an index that allows duplicates (e.g., `INDEX`) to one that doesn't (e.g., `UNIQUE`). If you actually do want to eliminate the duplicates, though, you can add the `IGNORE` flag to force the duplicates to be deleted:

```
ALTER IGNORE TABLE clients
DROP INDEX cust_id
CHANGE cust_id
client_id INT UNIQUE;
```

In this example, we're not only changing the indexed column's name, but we're also changing the index type from `INDEX` to `UNIQUE`. And, again, the `IGNORE` flag tells MariaDB to ignore any records with duplicate values for `cust_id`.

Renaming & Shifting Tables

The previous sections covered how to make changes to columns in a table. Sometimes you may want to rename a table. To change the name of the `clients` table to client_addresses we enter this:

```
RENAME TABLE clients
TO client_addresses;
```

The RENAME TABLE statement will also allows a table to be moved to another database just by adding the receiving database's name in front of the new table name, separated by a dot. Of course, you can move a table without renaming it. To move the newly named `client_addresses` table to the database db2, we enter this:

```
RENAME TABLE client_addresses
TO db2.client_addresses;
```

Finally, with tables that contain data (excluding InnoDB tables), occasionally it's desirable to resort the data within the table. Although the ORDER BY clause in a SELECT statement can do this on the fly as needed, sometimes developers want to do this somewhat permanently to the data within the table based on a particular column or columns. It can be done by entering the following:

```
ALTER TABLE client_addresses
ORDER BY city, name;
```

Notice that we're sorting by the city first and then by the client's name. Now when the developer enters a SELECT statement without an ORDER BY clause, the results are already ordered by the default of city and then name, at least until more data is added to the table.

This is not applicable to InnoDB tables, the default, which are ordered according to the clustered index, unless the primary key is defined on the specific columns.

Summation

Good planning is certainly important in developing a MariaDB database. However, as you can see, MariaDB is malleable enough that it can be reshaped without much trouble. Just be sure to make a backup before restructuring a table and be sure to check your work and the data when you're finished. With all of this in mind, you should feel comfortable in creating tables since they don't have to be perfect from the beginning.

6.1.6 Changing Times in MariaDB

Contents

The article entitled, Doing Time with MariaDB dealt with time and date columns in MariaDB and how to selectively retrieve and format time and date elements. This article will go a little further by exploring special functions that are available in MariaDB to modify time and date.

The Nature of Time

For most of us, there is a morning and an afternoon in each day. Days are measured in either two twelve-hour blocks or one twenty-four-hour block. There are twelve months in a year, with each month consisting of thirty or thirty-one days. The only exception is the month of February which contains twenty-eight days usually, but once every four years it contains twenty-nine. While this all may be rather natural, putting it into a computer program can make it seem very unnatural and frustrating.

For the scenario in this article we have a MariaDB database in which customers enter work requests through the web. When they enter a trouble ticket, a record is entered into a MariaDB table called, tickets. This record contains several fields, one of which is the date that the ticket was entered called ticket_date. Another contains the time the ticket was entered. It's called simply, entered. Yet another column is called promised; it's the time that the customer was promised that their problem would be resolved. Both the entered and the promised columns are time data type columns. The value of entered is determined from the current time of the server. The value of promised is determined by adding a number of hours to the value of entered, depending on the urgency of the ticket set by the customer. For instance, tickets marked "ASAP" are to be completed within two hours according to our company's policy. This all works nicely in testing, but occasionally customers create tickets at odd times and on odd days.

Around the Clock

Setting aside the potential problems for a moment, let's look at a simple example of how we might add tickets. Suppose we wanted to write a CGI script (in Perl or PHP) that will allow users to create tickets on-line any time. We might use the following SQL statement in our script:

```
INSERT INTO tickets
(client_id, urgency, trouble,
ticket_date, entered, promised)
VALUES('$client_id', '$urgency', '$trouble',
CURDATE(), CURTIME(),
SEC_TO_TIME(TIME_TO_SEC(CURTIME()) + 7200));
```

If you're unfamiliar with INSERT statements and the use of script variables (e.g., $client_id), you may want to go back and read an earlier article (MariaDB Basics) in this series which explains both. For the purposes of this article, however, let's focus on the minor formula in the SQL statement above for calculating the promised time, the last line. The TIME_TO_SEC( ) function converts a time to seconds so that a calculation may be performed. In this case, the current time is converted to seconds. The formula above then adds 7200 seconds (which is two hours) to that. In order to insert the seconds sum into a time column (i.e., promised), it needs to be converted to a time format. Hence, the calculation is wrapped up in the SEC_TO_TIME( ) function.

As nice as the SQL statement above is, a problem arises when a customer runs it at 11:00 p.m (or 23:00 in MariaDB time) and the promised time is to be two hours later. The SQL statement above will calculate a promised time of 25:00. What time is that in human or computer terms? As humans, we know that it's meant to be 1:00 a.m., but MariaDB will need this clarified. One solution would be to place the time formula above inside of an IF clause in MariaDB. To do this, the last line of the SQL statement would be replaced with these lines:

```
...
IF((TIME_TO_SEC(CURTIME()) + 7200) < 86400,
SEC_TO_TIME(TIME_TO_SEC(CURTIME()) + 7200),
SEC_TO_TIME((TIME_TO_SEC(CURTIME()) + 7200) - 86400)));
```

The first element in the IF clause is the test. The second piece is the value used if the test passes. The third is the value if the test fails. So, if the total seconds is less than 86,400 (i.e., the number of seconds in one day), then the total seconds of the current time, converted to the time format is to be used. Otherwise, the total seconds of the current time minus 86,400 seconds, converted to the time format is to be used. Incidentally, there's an extra closing parenthesis at the end of this SQL

statement excerpt because there was an opening one as part of the `VALUES` clause that's not shown here. Although the statement above works, it's a bit excessive and can be accomplished a little more succinctly if one reconsiders the purpose of the IF clause.

What we're trying to determine in the IF clause is the number of seconds into the day in which the work was promised to be done, meaning the excess amount of time of the day (i.e., one hour). For such a calculation, the modulo division operator (i.e., the `%`) can be used. The modulo division operator will give the remainder of a division. For instance, the result of `SELECT 14 % 5;` is `4`. That is to say, 5 goes into 14 two complete times with 4 left over. As another example, the result of `SELECT 3 % 5;` is 3; that is to say, 5 goes into 3 zero times with 3 left over. Using this arithmetic operator in the time formula above, we can eliminate the IF clause and use the following to accomplish our task:

```
...
SEC_TO_TIME((TIME_TO_SEC(CURTIME()) + 7200) % 86400));
```

If the current time is 23:00, then the time in seconds will be 82,800. The formula above will add 7200 to 82,800 to make 90,000 seconds. The modulo division operator will divide 86,400 into 90,000 one time, giving a remainder of 3600 seconds. The SEC_TO_TIME function will then convert 3600 seconds to one hour or 1:00 a.m.

Today or Tomorrow?

There is a problem with the results from the formula at the end of the previous section. If the customer is promised 1:00 a.m., is that time today or tomorrow? Again, as humans we know that since the promised time must be after the entered time, it must be 1:00 a.m. on the following day. Since computers don't make these assumptions, though, we'll have to make some adjustments to the tickets table and the SQL statement. To be able to record the date and time in each column, we'll first change the column types of entered and promised from time to datetime. We'll do the following SQL statements to migrate the data and to clean up the table:

```
ALTER TABLE tickets,
CHANGE COLUMN entered entered_old TIME,
CHANGE COLUMN promised promised_old TIME,
ADD COLUMN entered DATETIME,
ADD COLUMN promised DATETIME;

UPDATE tickets
SET entered = CONCAT(ticket_date, ' ', entered_old),
promised = CONCAT(ticket_date, ' ', promised_old);

ALTER TABLE tickets,
DROP COLUMN entered_old,
DROP COLUMN promised_old,
DROP COLUMN ticket_date;
```

The first SQL statement above alters the table to change the names of the time columns temporarily and to add the new columns with datetime types. If we were instead just to change the existing time columns to datetime types without this two step process, the data would be clobbered and reset to all zeros. The next SQL statement copies the values of the ticket_date column and pastes it together with the value of one of the old time columns to come up with the new date and time value for the entered and promised dates and times. The flaw in this statement, of course, is that it doesn't deal with the problems with some promised times that the previous layout caused. In fact, it reinforces it by giving a 1:00 a.m. promised time the date of the entered time. This will either have to be fixed manually if it's important to the developer, or with a script that will compare the two time columns. Either way, it's a little out of the scope of this article, so we'll move on. The last SQL statement above deletes the old time columns and the old date column now that the data has been migrated. By the way, it's a good practice to backup the data before altering a table. Also, you probably would run a SELECT statement before the last SQL statement above to check the migrated data before dropping the old columns.

Having changed the column types, we can now use the function DATE_ADD( ), which can deal with times that exceed twenty-four hours so that the problem with times straddling the midnight hour won't reoccur. Therefore, our on-going SQL statement becomes this:

```
INSERT INTO tickets
(client_id, urgency, trouble,
entered, promised)
VALUES('$client_id', '$urgency', '$trouble',
NOW(),
DATE_ADD(NOW(), INTERVAL 2 HOUR));
```

First notice that the field ticket_date was eliminated and CURTIME() was replaced with NOW( ), which provides the date and time in one. In the last line we see DATE_ADD( ): an interval of two hours is added to the date and time now (or rather when the record is created). If the time rolls into the next day, then the date is advanced by one and the correct hour is set accordingly.

The DATE_ADD( ) function will also allow for the addition of minutes. The directive `HOUR` would be replaced with `MINUTE`. To add both hours and minutes (e.g., two hours and thirty minutes), the last line of the SQL statement above could read like this:

```
...
DATE_ADD(NOW(), INTERVAL '2:30' HOUR_MINUTE));
```

If the time in which the statement is run is 11:00 p.m., the result would be 1:30 a.m. on the next day.

Around the Calendar

The dilemma that can occur with calculations involving hours that wrap around the clock, can similarly occur with calculations involving days that roll into a new month. This problem was fairly easy to resolve with an arithmetic operator when dealing with a constant like the number of seconds in a day. However, a formula to deal with the various number of days in each month would be very lengthy. For instance, if we were simply to add five days to the date February 27, we would get February 32. Imagine trying to create an SQL statement to figure out whether that's supposed to be March 1, 2, 3, or 4--depending on whether the previous month is a regular month with 30 or 31 days, or the one irregular month with 28 or 29 days, depending on the year.

Fortunately (as you probably have already guessed), DATE_ADD( ) will solve the month dilemma, as well. If instead of promising that tickets will be resolved within a couple hours of the time they are entered, we promise resolution within five days, the SQL statement would look like this:

```
INSERT INTO tickets
(client_id, urgency, trouble,
entered, promised)
VALUES('$client_id', '$urgency', '$trouble',
NOW(),
DATE_ADD(NOW(), INTERVAL 5 DAY));
```

If this statement is run on February 27, then the value of promised would be March 3 or 4, depending on whether it is a leap year. Which one will be determined by the DATE_ADD( ) function, requiring no fancy formula.

Just as hours and minutes can be mixed with DATE_ADD( ), days and hours can be mixed, as well. To make the value of promised two days and six hours from now, the last line of the SQL statement above would read like this:

```
...
DATE_ADD(NOW(), INTERVAL '2 6' DAY_HOUR));
```

The function DATE_ADD( ) will also allow the addition of months and of years. For instance, to increase the date by one year and two months, the SQL statement would be adjusted to look like this:

```
...
DATE_ADD(NOW(), INTERVAL '1 2' YEAR_MONTH));
```

This increases the year by one and the month by two. These intervals have no effect on time or day values, though. So, if the value of NOW( ) is `2017-09-15 23:00`, then the value of promised would become 2018-11-15 23:00, regardless of whether next year is a leap year and regardless of the number of days in each intervening month.

Stepping Back

It stands to reason that if one wants to add days to the current date, then one will want to subtract days in an equally agreeable manner. For subtracting days we can still use the DATE_ADD function. Just put a negative sign in front of the interval value like this:

```
...
DATE_ADD(NOW(), INTERVAL -5 DAY));
```

This will give a value five days before the current date. An alternative would be to use the DATE_SUB( ) function which subtracts from the date given. The above amendment (subtracting five days from the current date) could be entered like so:

```
...
DATE_SUB(NOW(), INTERVAL 5 DAY));
```

Notice that the 5 is not preceded by a negative sign. If it were, it would have the effect of adding five days.

Conclusion

This article along with the previous one on time and date in MariaDB in no way exhaust the topic. There are many more functions and tricks to manipulating temporal values in MariaDB, not to mention what can be done with the extension of a script using a programming language like PHP. Plus, new functions are occasionally being added to MariaDB.

6.1.7 Doing Time with MariaDB

Contents

The recording of date and time in a MariaDB database is a very common requirement. For gathering temporal data, one needs to know which type of columns to use in a table. More importantly is knowing how to record chronological data and how to retrieve it in various formats. Although this is a seemingly basic topic, there are many built-in time functions that can be used for more accurate SQL statements and better formatting of data. In this article we will explore these various aspects of how to do time with MariaDB.

About Time

Since date and time are only numeric strings, they can be stored in a regular character column. However, by using temporal data type columns, you can make use of several built-in functions offered by MariaDB. Currently, there are five temporal data types available: `DATE`, `TIME`, `DATETIME`, `TIMESTAMP`, and `YEAR`. The `DATE` column type is for recording the date only and is basically in this format: `yyyy-mm-dd`. The `TIME` column type is for recording time in this format: `hhh:mm:ss`. To record a combination of date and time, there is the `DATETIME` column type: `yyyy-mm-dd hh:mm:ss`. The `TIMESTAMP` column is similar to `DATETIME`, but it's a little limited in its range of allowable time. It starts at the Unix epoc time (i.e., 1970-01-01) and ends at the end of 2037. Finally, the `YEAR` data type is for recording only the year in a column: `yy` or `yyyy`. For the examples in this article, `DATE`, `TIME`, and `DATETIME` columns will be used. The database that will be referenced is for a fictitious psychiatry practice that keeps track of its patients and billable hours in MariaDB.

Telling Time

To record the current date and time in a MariaDB table, there are a few built-in functions that may be used. First, to record the date there are the functions CURRENT_DATE and CURDATE( ) (depending on your style), which both produce the same results (e.g., 2017-08-01). Notice that CURDATE( ) requires parentheses and the other does not. With many functions a column name or other variables are placed inside of the parentheses to get a result. With functions like CURDATE( ), there is nothing that may go inside the parenthesis. Since these two functions retrieve the current date in the format of the `DATE` column type, they can be used to fill in a `DATE` column when inserting a row:

```
INSERT INTO billable_work
(doctor_id, patient_id, session_date)
VALUES('1021', '1256', CURRENT_DATE);
```

The column *session_date* is a `DATE` column. Notice that there are no quotes around the date function. If there were it would be taken as a literal value rather than a function. Incidentally, I've skipped discussing how the table was set up. If you're not familiar with how to set up a table, you may want to read the MariaDB Basics article. To see what was just recorded by the INSERT statement above, the following may be entered (results follow):

```
SELECT rec_id, doctor_id,
patient_id, session_date
FROM billable_work
WHERE rec_id=LAST_INSERT_ID();

+--------+-----------+------------+--------------+
| rec_id | doctor_id | patient_id | session_date |
+--------+-----------+------------+--------------+
|   2462 | 1021      | 1256       | 2017-08-23   |
+--------+-----------+------------+--------------+
```

Notice in the billable_work table that the primary key column (i.e., `rec_id`) is an automatically generated and incremental

number column (i.e., `AUTO_INCREMENT`). As long as another record is not created or the user does not exit from the mariadb client or otherwise end the session, the LAST_INSERT_ID( ) function will retrieve the value of the `rec_id` for the last record entered by the user.

To record the time of an appointment for a patient in a time data type column, CURRENT_TIME or CURTIME( ) are used in the same way to insert the time. The following is entered to update the row created above to mark the starting time of the appointment—another SELECT statement follows with the results:

```
UPDATE billable_work
SET session_time=CURTIME()
WHERE rec_id='2462';

SELECT patient_id, session_date, session_time
FROM billable_work
WHERE rec_id='2462';

+------------+--------------+--------------+
| patient_id | session_date | session_time |
+------------+--------------+--------------+
| 1256       | 2017-08-23   | 10:30:23     |
+------------+--------------+--------------+
```

The column session_time is a time column. To record the date and time together in the same column, CURRENT_TIMESTAMP or SYSDATE( ) or NOW( ) can be used. All three functions produce the same time format: `yyyy-mm-dd hh:mm:ss`. Therefore, the column's data type would have to be `DATETIME` to use them.

How to get a Date

Although MariaDB records the date in a fairly agreeable format, you may want to present the date when it's retrieved in a different format. Or, you may want to extract part of the date, such as only the day of the month. There are many functions for reformatting and selectively retrieving date and time information. To start off with, let's select a column with a data type of `DATE` and look at the functions available for retrieving each component. To extract the year, there's the YEAR( ) function. For extracting just the month, the MONTH( ) function could be called upon. And to grab the day of the month, DAYOFMONTH( ) will work. Using the record entered above, here's what an SQL statement and its results would look like in which the session date is broken up into separate parts, but in a different order:

```
SELECT MONTH(session_date) AS Month,
DAYOFMONTH(session_date) AS Day,
YEAR(session_date) AS Year
FROM billable_work
WHERE rec_id='2462';

+-------+------+------+
| Month | Day  | Year |
+-------+------+------+
|     8 |   23 | 2017 |
+-------+------+------+
```

For those who aren't familiar with the keyword `AS`, it's used to label a column's output and may be referenced within an SQL statement. Splitting up the elements of a date can be useful in analyzing a particular element. If the bookkeeper of the fictitious psychiatry office needed to determine if the day of the week of each session was on a Saturday because the billing rate would be higher (time and a half), the DAYOFWEEK( ) function could be used. To spice up the examples, let's wrap the date function up in an IF( ) function that tests for the day of the week and sets the billing rate accordingly.

```
SELECT patient_id AS 'Patient ID',
session_date AS 'Date of Session',
IF(DAYOFWEEK(session_date)=6, 1.5, 1.0)
  AS 'Billing Rate'
FROM billable_work
WHERE rec_id='2462';

+------------+-----------------+--------------+
| Patient ID | Date of Session | Billing Rate |
+------------+-----------------+--------------+
|       1256 |    2017-08-23   |          1.5 |
+------------+-----------------+--------------+
```

Since we've slipped in the IF( ) function, we should explain it's format. The test condition is listed first within the parentheses. In this case, the test is checking if the session date is the sixth day of the week. Then, what MariaDB should

display is given if the test passes, followed by the result if it fails.

Similar to the DAYOFWEEK( ) function, there's also WEEKDAY( ). The only difference is that for DAYOFWEEK( ) the first day of the week is Sunday—with WEEKDAY( ) the first day is Monday. Both functions represent the first day with 0 and the last with `6`. Having *Saturday* and *Sunday* symbolized by `5` and `6` can be handy in constructing an IF statement that has a test component like "`WEEKDAY(session_date) > 4`" to determine if a date is a weekend day. This is cleaner than testing for values of `0` and `6`.

There is a function for determining the day of the year: DAYOFYEAR( ). It's not used often, but it is available if you ever need it. Occasionally, though, knowing the quarter of a year for a date can be useful for financial accounting. Rather than set up a formula in a script to determine the quarter, the QUARTER( ) function can do this easily. For instance, suppose an accountant wants a list of a doctor's sessions for each patient for the previous quarter. These three SQL statements could be entered in sequence to achieve the results that follow:

```
SET @LASTQTR:=IF((QUARTER(CURDATE())-1)=0,
4, QUARTER(CURDATE())-1);

SET @YR:=IF(@LASTQTR=4,
YEAR(NOW())-1, YEAR(NOW()));

SELECT patient_id AS 'Patient ID',
COUNT(session_time)
  AS 'Number of Sessions'
FROM billable_work
WHERE QUARTER(session_date) = @LASTQTR
  AND YEAR(session_date) = @YR
  AND doctor_id='1021'
GROUP BY patient_id
ORDER BY patient_id LIMIT 5;

+------------+--------------------+
| Patient ID | Number of Sessions |
+------------+--------------------+
| 1104       |                 10 |
| 1142       |                  7 |
| 1203       |                 18 |
| 1244       |                  6 |
| 1256       |                 12 |
+------------+--------------------+
```

This example is the most complicated so far. But it's not too difficult to understand if we pull it apart. The first SQL statement sets up a user variable containing the previous quarter (i.e., 1, 2, 3, or 4). This variable will be needed in the other two statements. The IF( ) clause in the first statement checks if the quarter of the current date minus one is zero. It will equal zero when it's run during the first quarter of a year. During a first quarter, of course, the previous quarter is the fourth quarter of the previous year. So, if the equation equals zero, then the variable `@LASTQTR` is set to `4`. Otherwise, `@LASTQTR` is set to the value of the current quarter minus one. The second statement is necessary to ensure that the records for the correct year are selected. So, if `@LASTQTR` equals four, then `@YR` needs to equal last year. If not, `@YR` is set to the current year. With the user variables set to the correct quarter and year, the SELECT statement can be entered. The COUNT( ) function counts the number of appointments that match the `WHERE` clause for each patient based on the GROUP BY clause. The `WHERE` clause looks for sessions with a quarter that equals `@LASTQTR` and a year that equals `@YR`, as well as the doctor's identification number. In summary, what we end up with is a set of SQL statements that retrieve the desired information regardless of which quarter or year it's entered.

What is the Time?

The last section covered how to retrieve pieces of a date column. Now let's look at how to do the same with a time column. To extract just the hour of a time saved in MariaDB, the HOUR( ) function could be used. For the minute and second, there's MINUTE( ) and SECOND( ). Let's put them all together in one straightforward SELECT statement:

```
SELECT HOUR(session_time) AS Hour,
MINUTE(session_time) AS Minute,
SECOND(session_time) AS Second
FROM billable_work
WHERE rec_id='2462';

+------+--------+--------+
| Hour | Minute | Second |
+------+--------+--------+
|   10 |     30 |     00 |
+------+--------+--------+
```

Date & Time Combined

All of the examples given so far have involved separate columns for date and time. The EXTRACT( ) function, however, will allow a particular component to be extracted from a combined column type (i.e., `DATETIME` or `TIMESTAMP`). The format is `EXTRACT(date_type FROM date_column)` where *date_type* is the component to retrieve and *date_column* is the name of the column from which to extract data. To extract the year, the *date_type* would be `YEAR`; for month, `MONTH` is used; and for day, there's `DAY`. To extract time elements, `HOUR` is used for hour, `MINUTE` for minute, and `SECOND` for second. Although that's all pretty simple, let's look at an example. Suppose the table billable_work has a column called `appointment` (a `datetime` column) that contains the date and time for which the appointment was scheduled (as opposed to the time it actually started in `session_time`). To get the hour and minute for a particular date, the following SQL statement will suffice:

```
SELECT patient_name AS Patient,
EXTRACT(HOUR FROM appointment) AS Hour,
EXTRACT(MINUTE FROM appointment) AS Minute
FROM billable_work, patients
WHERE doctor_id='1021'
  AND EXTRACT(MONTH FROM appointment)='8'
  AND EXTRACT(DAY FROM appointment)='30'
  AND billable_work.patient_id =
    patients.patient_id;
```

This statement calls upon another table (`patients`) which holds patient information such as their names. It requires a connecting point between the tables (i.e., the `patient_id` from each table). If you're confused on how to form relationships between tables in a SELECT statement, you may want to go back and read the Getting Data from MariaDB article. The SQL statement above would be used to retrieve the appointments for one doctor for one day, giving results like this:

```
+-------------------+------+--------+
| Patient           | Hour | Minute |
+-------------------+------+--------+
| Michael Zabalaoui |   10 |     00 |
| Jerry Neumeyer    |   11 |     00 |
| Richard Stringer  |   13 |     30 |
| Janice Sogard     |   14 |     30 |
+-------------------+------+--------+
```

In this example, the time elements are separated and they don't include the date. With the EXTRACT( ) function, however, you can also return combined date and time elements. There is `DAY_HOUR` for the day and hour; there's `DAY_MINUTE` for the day, hour, and minute; `DAY_SECOND` for day, hour, minute, and second; and `YEAR_MONTH` for year and month. There are also some time only combinations: `HOUR_MINUTE` for hour and minute; `HOUR_SECOND` for hour, minute, and second; and `MINUTE_SECOND` for minute and second. However, there's not a `MONTH_DAY` to allow the combining of the two extracts in the `WHERE` clause of the last SELECT statement above. Nevertheless, we'll modify the example above and use the `HOUR_MINUTE` date_type to retrieve the hour and minute in one resulting column. It would only require the second and third lines to be deleted and replaced with this:

```
...
EXTRACT(HOUR_MINUTE FROM appointment)
  AS Appointment
...

+-------------------+-------------+
| Patient           | Appointment |
+-------------------+-------------+
| Michael Zabalaoui |        1000 |
| Jerry Neumeyer    |        1100 |
| Richard Stringer  |        1330 |
| Janice Sogard     |        1430 |
+-------------------+-------------+
```

The problem with this output, though, is that the times aren't very pleasing looking. For more natural date and time displays, there are a few simple date formatting functions available and there are the DATE_FORMAT( ) and TIME_FORMAT( ) functions.

Fine Time Pieces

The simple functions that we mentioned are used for reformatting the output of days and months. To get the date of patient

sessions for August, but in a more wordier format, MONTHNAME( ) and DAYNAME( ) could be used:

```
SELECT patient_name AS Patient,
CONCAT(DAYNAME(appointment), ' - ',
  MONTHNAME(appointment), ' ',
  DAYOFMONTH(appointment), ', ',
  YEAR(appointment)) AS Appointment
FROM billable_work, patients
WHERE doctor_id='1021'
  AND billable_work.patient_id =
    patients.patient_id
  AND appointment>'2017-08-01'
  AND appointment<'2017-08-31'
LIMIT 1;

+------------------+----------------------------+
| Patient          | Appointment                |
+------------------+----------------------------+
| Michael Zabalaoui | Wednesday - August 30, 2017 |
+------------------+----------------------------+
```

In this statement the CONCAT( ) splices together the results of several date functions along with spaces and other characters. The EXTRACT( ) function was eliminated from the `WHERE` clause and instead a simple numeric test for sessions in August was given. Although EXTRACT( ) is fairly straightforward, this all can be accomplished with less typing by using the DATE_FORMAT( ) function.

The DATE_FORMAT( ) function has over thirty options for formatting the date to your liking. Plus, you can combine the options and add your own separators and other text. The syntax is `DATE_FORMAT(date_column, 'options & characters')` . As an example, let's reproduce the last SQL statement by using the DATE_FORMAT( ) function for formatting the date of the appointment and for scanning for appointments in July only:

```
SELECT patient_name AS Patient,
DATE_FORMAT(appointment, '%W - %M %e, %Y')
  AS Appointment
FROM billable_work, patients
WHERE doctor_id='1021'
  AND billable_work.patient_id =
    patients.patient_id
  AND DATE_FORMAT(appointment, '%c') = 8
LIMIT 1;
```

This produces the exact same output as above, but with a more succinct statement. The option `%W` gives the name of the day of the week. The option `%M` provides the month's name. The option `%e` displays the day of the month (`%d` would work, but it left-pads single-digit dates with zeros). Finally, `%Y` is for the four character year. All other elements within the quotes (i.e., the spaces, the dash, and the comma) are literal characters for a nicer display.

With DATE_FORMAT( ), time elements of a field also can be formatted. For instance, suppose we also wanted the hour and minute of the appointment. We would only need to change the second line of the SQL statement above (to save space, patient_name was eliminated):

```
SELECT
DATE_FORMAT(appointment, '%W - %M %e, %Y at %r')
  AS Appointment
...

+-----------------------------------------+
| Appointment                             |
+-----------------------------------------+
| Wednesday - August 30, 2017 at 02:11:19 AM |
+-----------------------------------------+
```

The word at was added along with the formatting option `%r` which gives the time with AM or PM at the end.

Although it may be a little confusing at first, once you've learned some of the common formatting options, DATE_FORMAT( ) is much easier to use than EXTRACT( ). There are many more options to DATE_FORMAT( ) that we haven't mentioned. For a complete list of the options available, see the DATE_FORMAT( ) documentation page.

Clean up Time

In addition to DATE_FORMAT( ), MariaDB has a comparable built-in function for formating only time: TIME_FORMAT( ).

The syntax is the same and uses the same options as DATE_FORMAT( ), except only the time related formatting options apply. As an example, here's an SQL statement that a doctor might use at the beginning of each day to get a list of her appointments for the day:

```
SELECT patient_name AS Patient,
TIME_FORMAT(appointment, '%l:%i %p')
  AS Appointment
FROM billable_work, patients
WHERE doctor_id='1021'
  AND billable_work.patient_id =
    patients.patient_id
  AND DATE_FORMAT(appointment, '%Y-%m-%d') =
    CURDATE();

+------------------+-------------+
| Patient          | Appointment |
+------------------+-------------+
| Michael Zabalaoui |    10:00 AM |
| Jerry Neumeyer    |    11:00 AM |
| Richard Stringer  |    01:30 PM |
| Janice Sogard     |    02:30 PM |
+------------------+-------------+
```

The option `%l` provides the hours 01 through 12. The `%p` at the end indicates (with the AM or PM) whether the time is before or after noon. The `%i` option gives the minute. The colon and the space are for additional display appeal. Of course, all of this can be done exactly the same way with the DATE_FORMAT( ) function. As for the DATE_FORMAT( ) component in the WHERE clause here, the date is formatted exactly as it will be with CURDATE( ) (i.e., 2017-08-30) so that they may be compared properly.

Time to End

Many developers use PHP, Perl, or some other scripting language with MariaDB. Sometimes developers will solve retrieval problems with longer scripts rather than learn precisely how to extract temporal data with MariaDB. As you can see in several of the examples here (particularly the one using the QUARTER( ) function), you can accomplish a great deal within MariaDB. When faced with a potentially complicated SQL statement, try creating it in the mariadb client first. Once you get what you need (under various conditions) and in the format desired, then copy the statement into your script. This practice can greatly help you improve your MariaDB statements and scripting code.

6.1.8 Importing Data into MariaDB

Contents

When a MariaDB developer first creates a MariaDB database for a client, often times the client has already accumulated data in other, simpler applications. Being able to convert data easily to MariaDB is critical. In the previous two articles of this MariaDB series, we explored how to set up a database and how to query one. In this third installment, we will introduce some methods and tools for bulk importing of data into MariaDB. This isn't an overly difficult task, but the processing of large amounts of data can be intimidating for a newcomer and and as a result it can be a barrier to getting started with MariaDB. Additionally, for intermediate developers, there are many nuances to consider for a clean import, which is especially important for automating regularly scheduled imports. There are also restraints to deal with that may be imposed on a developer when using a web hosting company.

Foreign Data Basics

Clients sometimes give developers raw data in formats created by simple database programs like MS Access ®. Since non-technical clients don't typically understand database concepts, new clients often give me their initial data in Excel spreadsheets. Let's first look at a simple method for importing data. The simplest way to deal with incompatible data in any format is to load it up in its original software and to export it out to a delimited text file. Most applications have the ability to export data to a text format and will allow the user to set the delimiters. We like to use the bar (i.e., `|` , a.k.a. pipe) to separate fields and the line-feed to separate records.

For the examples in this article, we will assume that a fictitious client's data was in Excel and that the exported text file will be named `prospects.txt`. It contains contact information about prospective customers for the client's sales department, located on the client's intranet site. The data is to be imported into a MariaDB table called `prospect_contact`, in a database called `sales_dept`. To make the process simpler, the order and number of columns in MS Excel ® (the format of the data provided by the client) should be the same as the table into which the data is going to be imported. So if prospect_contact has columns that are not included in the spreadsheet, one would make a copy of the spreadsheet and add the missing columns and leave them blank. If there are columns in the spreadsheet that aren't in `prospect_contact`, one would either add them to the MariaDB table, or, if they're not to be imported, one would delete the extra columns from the spreadsheet. One should also delete any headings and footnotes from the spreadsheet. After this is completed then the data can be exported. Since this is Unix Review, we'll skip how one would export data in Excel and assume that the task was accomplished easily enough using its export wizard.

The next step is to upload the data text file to the client's web site by FTP. It should be uploaded in ASCII mode. Binary mode may send binary hard-returns for row-endings. Also, it's a good security habit to upload data files to non-public directories. Many web hosting companies provide virtual domains with a directory like `/public_html`, which is the document root for the Apache web server; it typically contains the site's web pages. In such a situation, `/` is a virtual root containing logs and other files that are inaccessible to the public. We usually create a directory called `tmp` in the virtual root directory to hold data files temporarily for importing into MariaDB. Once that's done, all that's required is to log into MariaDB with the mariadb client as an administrative user (if not root, then a user with `FILE` privileges), and run the proper SQL statement to import the data.

Loading Data Basics

The LOAD DATA INFILE statement is the easiest way to import data from a plain text file into MariaDB. Below is what one would enter in the mariadb client to load the data in the file called prospects.txt into the table `prospect_contact`:

```
LOAD DATA INFILE '/tmp/prospects.txt'
INTO TABLE prospect_contact
FIELDS TERMINATED BY '|';
```

Before entering the statement above, the MariaDB session would, of course, be switched to the sales_dept database with a USE statement. It is possible, though, to specify the database along with the table name (e.g., `sales_dept.prospect_contact`). If the server is running Windows, the forward slashes are still used for the text file's path, but a drive may need to be specified at the beginning of the path: '`c:/tmp/prospects.txt`'. Notice that the SQL statement above has `|` as the field delimiter. If the delimiter was [TAB]—which is common—then one would replace `|` with `\t` here. A line-feed (`\n`) isn't specified as the record delimiter since it's assumed. If the rows start and end with something else, though, then they will need to be stated. For instance, suppose the rows in the text file start with a double-quote and end with a double-quote and a Windows hard-return (i.e., a return and a line-feed). The statement would need to read like this:

```
LOAD DATA INFILE '/tmp/prospects.txt'
INTO TABLE prospect_contact
FIELDS TERMINATED BY '|'
LINES STARTING BY '"'
TERMINATED BY '"\r\n';
```

Notice that the starting double-quote is inside of single-quotes. If one needs to specify a single-quote as the start of a line, one could either put the one single-quote within double-quotes or one could escape the inner single-quote with a back-slash, thus telling MariaDB that the single-quote that follows is to be taken literally and is not part of the statement, per se:

```
...
LINES STARTING BY '\''
...
```

Duplicate Rows

If the table prospect_contact already contains some of the records that are about to be imported from prospects.txt (that is to say, records with the same primary key), then a decision should be made as to what MariaDB is to do about the duplicates. The SQL statement, as it stands above, will cause MariaDB to try to import the duplicate records and to create duplicate rows in prospect_contact for them. If the table's properties are set not to allow duplicates, then MariaDB will kick out errors. To get MariaDB to replace the duplicate existing rows with the ones being imported in, one would add the `REPLACE` just before the `INTO TABLE` clause like this:

```
LOAD DATA INFILE '/tmp/prospects.txt'
REPLACE INTO TABLE prospect_contact
FIELDS TERMINATED BY '|'
LINES STARTING BY '"'
TERMINATED BY '"\n';
```

To import only records for prospects that are not already in prospect_contact, one would substitute REPLACE with the `IGNORE` flag. This instructs MariaDB to ignore records read from the text file that already exist in the table.

Live Data

For importing data into a table while it's in use, table access needs to be addressed. If access to the table by other users may not be interrupted, then a `LOW_PRIORITY` flag can be added to the LOAD DATA INFILE statement. This tells MariaDB that the loading of this data is a low priority. One would only need to change the first line of the SQL statement above to set its priority to low:

```
LOAD DATA LOW_PRIORITY INFILE '/tmp/prospects.txt'
...
```

If the `LOW_PRIORITY` flag isn't included, the table will be locked temporarily during the import and other users will be prevented from accessing it.

Being Difficult

I mentioned earlier that uploading of the text file should not be done in binary mode so as to avoid the difficulties associated with Windows line endings. If this is unavoidable, however, there is an easy way to import binary row-endings with MariaDB. One would just specify the appropriate hexadecimals for a carriage-return combined with a line-feed (i.e., `CRLF`) as the value of `TERMINATED BY` :

```
...
TERMINATED BY 0x0d0a;
```

Notice that there are intentionally no quotes around the binary value. If there were, MariaDB would take the value for text and not a binary code. The semi-colon is not part of the value; it's the SQL statement terminator.

Earlier we also stated that the first row in the spreadsheet containing the column headings should be deleted before exporting to avoid the difficulty of importing the headings as a record. It's actually pretty easy to tell MariaDB to just skip the top line. One would add the following line to the very end of the LOAD DATA INFILE statement:

```
...
IGNORE 1 LINES;
```

The number of lines for MariaDB to ignore can, of course, be more than one.

Another difficulty arises when some Windows application wizards export data with each field surrounded by double-quotes, as well as around the start and end of records. This can be a problem when a field contains a double-quote. To deal with this, some applications use back-slash (\) to escape embedded double-quotes, to indicate that a particular double-quote is not a field ending but part of the field's content. However, some applications will use a different character (like a pound-sign) to escape embedded quotes. This can cause problems if MariaDB isn't prepared for the odd escape-character. MariaDB will think the escape character is actually text and the embedded quote-mark, although it's escaped, is a field ending. The unenclosed text that follows will be imported into the next column and the remaining columns will be one column off, leaving the last column not imported. As maddening as this can be, it's quite manageable in MariaDB by adding an `ENCLOSED BY` and an `ESCAPED BY` clause:

```
LOAD DATA LOW_PRIORITY INFILE '/tmp/prospects.txt'
REPLACE INTO TABLE prospect_contact
FIELDS TERMINATED BY '"'
ENCLOSED BY '"' ESCAPED BY '#'
LINES STARTING BY '"'
TERMINATED BY '"\n'
IGNORE 1 LINES;
```

In the *Foreign Data Basics* section above, we said that the columns in the spreadsheet should be put in the same order and quantity as the receiving table. This really isn't necessary if MariaDB is cued in as to what it should expect. To illustrate, let's assume that prospect_contact has four columns in the following order: `row_id` , `name_first` , `name_last` , `telephone` . Whereas, the spreadsheet has only three columns, differently named, in this order: `Last Name` , `First Name` ,

`Telephone`. If the spreadsheet isn't adjusted, then the SQL statement will need to be changed to tell MariaDB the field order:

```
LOAD DATA LOW_PRIORITY INFILE '/tmp/prospects.txt'
REPLACE INTO TABLE sales_dept.prospect_contact
FIELDS TERMINATED BY 0x09
ENCLOSED BY '"' ESCAPED BY '#'
TERMINATED BY 0x0d0a
IGNORE 1 LINES
(name_last, name_first, telephone);
```

This SQL statement tells MariaDB the name of each table column associated with each spreadsheet column in the order that they appear in the text file. From there it will naturally insert the data into the appropriate columns in the table. As for columns that are missing like row_id, MariaDB will fill in those fields with the default value if one has been supplied in the table's properties. If not, it will leave the field as NULL. Incidentally, we slipped in the binary `[TAB] (0x09)` as a field delimiter.

mariadb-import

For some clients and for certain situations it may be of value to be able to import data into MariaDB without using the mariadb client. This could be necessary when constructing a shell script to import text files on an automated, regular schedule. To accomplish this, the mariadb-import (`mysqlimport` before MariaDB 10.5) utility may be used as it encompasses the LOAD DATA INFILE statement and can easily be run from a script. So if one wants to enter the involved SQL statement at the end of the last section above, the following could be entered from the command-line (i.e., not in the mariadb client):

```
mariadb-import --user='marie_dyer' --password='angelle1207' \
--fields-terminated-by=0x09 --lines-terminated-by=0x0d0a \
--replace --low-priority --fields-enclosed-by='"' \
 --fields-escaped-by='#' --ignore-lines='1' --verbose \
--columns='name_last, name_first, telephone' \
sales_dept '/tmp/prospect_contact.txt'
```

Although this statement is written over several lines here, it either has to be on the same line when entered or a space followed by a back-slash has to be entered at the end of each line (as seen here) to indicate that more follows. Since the above is entered at the command-line prompt, the user isn't logged into MariaDB. Therefore the first line contains the user name and password for mariadb-import to give to MariaDB. The password itself is optional, but the directive `--password` (without the equal sign) isn't. If the password value is not given in the statement, then the user will be prompted for it. Notice that the order of directives doesn't matter after the initial command, except that the database and file name go last. Regarding the file name, its prefix must be the same as the table—the dot and the extension are ignored. This requires that `prospects.txt` be renamed to `prospect_contact.txt`. If the file isn't renamed, then MariaDB would create a new table called prospects and the `--replace` option would be pointless. After the file name, incidentally, one could list more text files, separated by a space, to process using mariadb-import. We've added the `--verbose` directive so as to be able to see what's going on. One probably would leave this out in an automated script. By the way, `--low-priority` and `--ignore-lines` are available.

Web Hosting Restraints

Some web hosting companies do not allow the use of LOAD DATA INFILE or mariadb-import statements due to security vulnerabilities in these statements for them. To get around this, some extra steps are necessary to avoid having to manually enter the data one row at a time. First, one needs to have MariaDB installed on one's local workstation. For simplicity, we'll assume this is done and is running Linux on the main partition and MS Windows® on an extra partition. Recapping the on-going example of this article based on these new circumstances, one would boot up into Windows and start MS Excel®, load the client's spreadsheet into it and then run the export wizard as before—saving the file prospects.txt to the 'My Documents' directory. Then one would reboot into Linux and mount the Windows partition and copy the data text file to `/tmp` in Linux, locally. Next one would log into the local (not the client's) MariaDB server and import the text file using a LOAD DATA INFILE as we've extensively outline above. From there one would exit MariaDB and export the data out of MariaDB using the mariadb-dump utility locally, from the command-line like this:

```
mariadb-dump --user='root' --password='geronimo' sales_dept prospect_contact > /tmp/prospects.sql
```

This creates an interesting text file complete with all of the SQL commands necessary to insert the data back into MariaDB one record, one INSERT at a time. When you run mariadb-import, it's very educational to open up it in a text editor to see what it generates.

After creating this table dump, one would upload the resulting file (in ASCII mode) to the `/tmp` directory on the client's web server. From the command prompt on the client's server one would enter the following:

```
mariadb --user='marie_dyer' --password='angelle12107' sales_dept < '/tmp/prospects.sql'
```

This line along with the mariadb-dump line show above are simple approaches. Like the Windows application wizard, with mariadb-dump one can specify the format of the output file and several other factors. One important factor related to the scenario used in this article is the CREATE TABLE statement that will be embedded in the mariadb-dump output file. This will fail and kick out an error because of the existing table prospect_contact in the client's database. To limit the output to only INSERT statements and no CREATE TABLE statements, the mariadb-dump line would look like this:

```
mariadb-dump -u marie_dyer -p --no-create-info sales_dept prospect_contact > /tmp/prospects.sql
```

Notice that we've used acceptable abbreviations for the user name and the password directives. Since the password was given here, the user will be prompted for it.

The mariadb-dump utility usually works pretty well. However, one feature it's lacking at this time is a `REPLACE` flag as is found in the LOAD DATA INFILE statement and with the mariadb-import tool. So if a record already exists in the `prospect_contact`, it won't be imported. Instead it will kick out an error message and stop at that record, which can be a mess if one has imported several hundred rows and have several hundred more to go. One easy fix for this is to open up `prospects.sql` in a text editor and do a search on the word `INSERT` and replace it with `REPLACE`. The syntax of both of these statements are the same, fortunately. So one would only need to replace the keyword for new records to be inserted and for existing records to be replaced.

Concluding Observations and Admissions

It's always amazing to me how much can be involved in the simplest of statements in MariaDB. MariaDB is deceptively powerful and feature rich. One can keep the statements pretty minimal or one can develop a fairly detailed, single statement to allow for accuracy of action. There are many other aspects of importing data into MariaDB that we did not address—in particular dealing with utilities. We also didn't talk about the Perl modules that could be used to convert data files. These can be useful in scripting imports. There are many ways in which one can handle importing data. Hopefully, this article has presented most of the basics and pertinent advanced details that may be of use to most MariaDB developers.

6.1.9 Making Backups with mariadb-dump

Contents

One of the best utilities to use to make a backup copy of a server's MariaDB's data is mariadb-dump (previously called mysqldump, which still works as a symlink). It comes with MariaDB, so it costs you nothing more. Best of all it doesn't require you to shut down MariaDB services to make a backup. It works very simply: it retrieves the data and schema from each database and table and builds a data text file outside of MariaDB. This data text file (known as a dump file) will contain the SQL statements necessary to reconstruct the databases and data. If you were to open a dump file generated by mariadb-dump, you would see CREATE TABLE statements and a multitude of INSERT statements, one for each row of data.

Backing Up Everything

To export all of the databases in MariaDB using mariadb-dump, the following would be entered from the filesystem command-line:

```
mariadb-dump -u admin_backup -p -x -A > /data/backup/dbs.sql
```

The first set of options here (`-u admin_backup -p`) tell MariaDB that this utility is to be executed by the user *admin_backup* and that the user needs to be prompted for a password, which will have to be typed in on the next line when asked. Incidentally, although you might be tempted to just use the root user, you should create a special administrative user as we're using here. If the dump is to be executed by cron by way of a shell script, this option can be changed to `-pmypwd`, where *mypwd* is the password—there's no space between the `-p` and the password. The `-x` option has MariaDB lock all of the tables before performing the backup. The lock won't be released until the process is finished. To bundle INSERT statements together for each table, we've added the `-e` option. This extended insert option will cause the dump file to be

smaller and allow any possible future restores to be executed faster. The `-A` option specifies that all databases are to be exported. Finally, the greater-than sign is a shell redirect of the standard output (STDOUT) to the path and file named after it.

The example given for backing up all database is the short hand version. The convention is migrating to longer options, not the single letter options. In fact, some are being deprecated and won't be available in the future. So, the above could and should be entered like this:

```
mariadb-dump --user=admin_backup --password --lock-tables --all-databases > /data/backup/dbs.sql
```

The longer option names are easier to follow and to remember. Again, if the backup is to be executed by a shell script, the user's password should be listed: `--password=mypwd`. Notice that the equal-sign is added when the password is given with the long option name.

Just One Database

Backing up all of the databases at once with mariadb-dump may result in one large dump file. This could be take longer to complete the backup and make restoration a bit cumbersome later. Therefore, it might be more useful to stagger backups based on databases, making for possbily several smaller files. You could then backup larger databases during slower traffic times. You might also backup critical databases or ones that are changed much during slower times of the day so that you don't dimish user interaction.

To export only one database and not all, enter something like the following from the command-line:

```
mariadb-dump --user=admin_backup --password --lock-tables --databases db1 > /data/backup/db1.sql
```

The only significant difference in this line is that the `-A` option has been replaced with `-B` and the database to be exported has been given. To export multiple databases, just enter them after the `-B` option, separated by spaces (e.g., `-B db1 db2`).

Dumping Tables

For very large databases, you may want to backup the data based on tables rather than the whole database. You could backup weekly an entire database and then only backup daily individual tables for which data changes often. To backup just one table, the following could be entered from the command line:

```
mariadb-dump --user=admin_backup --password --lock-tables db1 table1 > /data/backup/db1_table1.sq
```

First notice that the `--databases` option has not been included in the line above. The utility assumes that the first name given is a database and the second name is a table name and not another database. To backup multiple tables from a database, just list them after the database name, separated by spaces (e.g., db1 table1 table2).

Conclusion

As you can see from this article, mariadb-dump is easy to use and very powerful. In fact, it can clobber your data if you're not careful. Therefore, you should practice using it on a test database—a test server even—a few times until you're comfortable with making backups and restoring them. Don't wait until you've lost your data and in a panic to restore your data to find out that you haven't been backing up your data properly or that you don't know how to fine tune data restoration. Develop some skills in advance and in a safe and controlled way. To learn how to restore dump files, see Restoring Data From Dump Files.

Other References

- Devart backup tutorial

6.1.10 MariaDB String Functions

Contents

MariaDB has many built-in functions that can be used to manipulate strings of data. With these functions, one can format data, extract certain characters, or use search expressions. Good developers should be aware of the string functions that are available. Therefore, in this article we will go through several string functions, grouping them by similar features, and provide examples of how they might be used.

Formatting

There are several string functions that are used to format text and numbers for nicer display. A popular and very useful function for pasting together the contents of data fields with text is the CONCAT() function. As an example, suppose that a table called contacts has a column for each sales contact's first name and another for the last name. The following SQL statement would put them together:

```
SELECT CONCAT(name_first, ' ', name_last)
AS Name
FROM contacts;
```

This statement will display the first name, a space, and then the last name together in one column. The AS clause will change the column heading of the results to Name.

A less used concatenating function is CONCAT_WS(). It will put together columns with a separator between each. This can be useful when making data available for other programs. For instance, suppose we have a program that will import data, but it requires the fields to be separated by vertical bars. We could just export the data, or we could use a SELECT statement like the one that follows in conjunction with an interface written with an API language like Perl:

```
SELECT CONCAT_WS('|', col1, col2, col3)
FROM table1;
```

The first element above is the separator. The remaining elements are the columns to be strung together.

If we want to format a long number with commas every three digits and a period for the decimal point (e.g., 100,000.00), we can use the function FORMAT() like so:

```
SELECT CONCAT('$', FORMAT(col5, 2))
FROM table3;
```

In this statement, the CONCAT() will place a dollar sign in front of the numbers found in the `col5` column, which will be formatted with commas by FORMAT(). The `2` within the FORMAT() stipulates two decimal places.

Occasionally, one will want to convert the text from a column to either all upper-case letters or all lower-case letters. In the example that follows, the output of the first column is converted to upper-case and the second to lower-case:

```
SELECT UCASE(col1),
LCASE(col2)
FROM table4;
```

When displaying data in forms, it's sometimes useful to pad the data displayed with zeros or dots or some other filler. This can be necessary when dealing with VARCHAR columns where the width varies to help the user to see the column limits. There are two functions that may be used for padding: LPAD() and RPAD().

```
SELECT RPAD(part_nbr, 8, '.') AS 'Part Nbr.',
LPAD(description, 15, '_') AS Description
FROM catalog;
```

In this SQL statement, dots are added to the right end of each part number. So a part number of "H200" will display as "H200....", but without the quotes. Each part's description will have under-scores preceding it. A part with a description of "brass hinge" will display as "brass hinge".

If a column is a CHAR data-type, a fixed width column, then it may be necessary to trim any leading or trailing spaces from displays. There are a few functions to accomplish this task. The LTRIM() function will eliminate any leading spaces to the left. So " `H200` " becomes " `H200` ". For columns with trailing spaces, spaces on the right, RTRIM() will work: " `H500` "

becomes " `H500` ". A more versatile trimming function, though, is TRIM(). With it one can trim left, right or both. Below are a few examples:

```
SELECT TRIM(LEADING '.' FROM col1),
TRIM(TRAILING FROM col2),
TRIM(BOTH '_' FROM col3),
TRIM(col4)
FROM table5;
```

In the first TRIM() clause, the padding component is specified; the leading dots are to be trimmed from the output of `col1`. The trailing spaces will be trimmed off of `col2` —space is the default. Both leading and trailing under-scores are trimmed from `col3` above. Unless specified, BOTH is the default. So leading and trailing spaces are trimmed from `col4` in the statement here.

Extracting

When there is a need to extract specific elements from a column, MariaDB has a few functions that can help. Suppose a column in the table contacts contains the telephone numbers of sales contacts, including the area-codes, but without any dashes or parentheses. The area-code of each could be extracted for sorting with the LEFT() and the telephone number with the RIGHT() function.

```
SELECT LEFT(telephone, 3) AS area_code,
RIGHT(telephone, 7) AS tel_nbr
FROM contacts
ORDER BY area_code;
```

In the LEFT() function above, the column telephone is given along with the number of characters to extract, starting from the first character on the left in the column. The RIGHT() function is similar, but it starts from the last character on the right, counting left to capture, in this statement, the last seven characters. In the SQL statement above, area_code is reused to order the results set. To reformat the telephone number, it will be necessary to use the SUBSTRING() function.

```
SELECT CONCAT('(', LEFT(telephone, 3), ') ',
SUBSTRING(telephone, 4, 3), '-',
MID(telephone, 7)) AS 'Telephone Number'
FROM contacts
ORDER BY LEFT(telephone, 3);
```

In this SQL statement, the CONCAT() function is employed to assemble some characters and extracted data to produce a common display for telephone numbers (e.g., (504) 555-1234). The first element of the CONCAT() is an opening parenthesis. Next, a LEFT() is used to get the first three characters of telephone, the area-code. After that a closing parenthesis, along with a space is added to the output. The next element uses the SUBSTRING() function to extract the telephone number's prefix, starting at the fourth position, for a total of three characters. Then a dash is inserted into the display. Finally, the function MID() extracts the remainder of the telephone number, starting at the seventh position. The functions MID() and SUBSTRING() are interchangeable and their syntax are the same. By default, for both functions, if the number of characters to capture isn't specified, then it's assumed that the remaining ones are to be extracted.

Manipulating

There are a few functions in MariaDB that can help in manipulating text. One such function is REPLACE(). With it every occurrence of a search parameter in a string can be replaced. For example, suppose we wanted to replace the title *Mrs.* with *Ms.* in a column containing the person's title, but only in the output. The following SQL would do the trick:

```
SELECT CONCAT(REPLACE(title, 'Mrs.', 'Ms.'),
' ', name_first, ' ', name_last) AS Name
FROM contacts;
```

We're using the ever handy CONCAT() function to put together the contact's name with spaces. The REPLACE() function extracts each title and replaces *Mrs.* with *Ms.*, where applicable. Otherwise, for all other titles, it displays them unchanged.

If we want to insert or replace certain text from a column (but not all of its contents), we could use the INSERT() function in conjunction with the LOCATE() function. For example, suppose another contacts table has the contact's title and full name in one column. To change the occurrences of Mrs. to Ms., we could not use REPLACE() since the title is embedded in this example. Instead, we would do the following:

```
SELECT INSERT(name, LOCATE(name, 'Mrs.'), 4, 'Ms.')
FROM contacts;
```

The first element of the INSERT() function is the column. The second element which contains the LOCATE() is the position in the string that text is to be inserted. The third element is optional; it states the number of characters to overwrite. In this case, Mrs. which is four characters is overwritten with Ms. (the final element), which is only three. Incidentally, if 0 is specified, then nothing is overwritten, text is inserted only. As for the LOCATE() function, the first element is the column and the second the search text. It returns the position within the column where the text is found. If it's not found, then 0 is returned. A value of 0 for the position in the INSERT() function negates it and returns the value of name unchanged.

On the odd chance that there is a need to reverse the content of a column, there's the REVERSE() function. You would just place the column name within the function. Another minor function is the REPEAT() function. With it a string may be repeated in the display:

```
SELECT REPEAT(col1, 2)
FROM table1;
```

The first component of the function above is the string or column to be repeated. The second component states the number of times it's to be repeated.

Expression Aids

The function CHAR_LENGTH() is used to determine the number of characters in a string. This could be useful in a situation where a column contains different types of information of specific lengths. For instance, suppose a column in a table for a college contains identification numbers for students, faculty, and staff. If student identification numbers have eight characters while others have less, the following will count the number of student records:

```
SELECT COUNT(school_id)
AS 'Number of Students'
FROM table8
WHERE CHAR_LENGTH(school_id)=8;
```

The COUNT() function above counts the number of rows that meet the condition of the `WHERE` clause.

In a SELECT statement, an ORDER BY clause can be used to sort a results set by a specific column. However, if the column contains IP addresses, a simple sort may not produce the desired results:

```
SELECT ip_address
FROM computers WHERE server='Y'
ORDER BY ip_address LIMIT 3;

+-------------+
| ip_address  |
+-------------+
| 10.0.1.1    |
| 10.0.11.1   |
| 10.0.2.1    |
+-------------+
```

In the limited results above, the IP address 10.0.2.1 should be second. This happens because the column is being sorted lexically and not numerically. The function INET_ATON() will solve this sorting problem.

```
SELECT ip_address
FROM computers WHERE server='Y'
ORDER BY INET_ATON(ip_address) LIMIT 3;
```

Basically, the INET_ATON() function will convert IP addresses to regular numbers for numeric sorting. For instance, if we were to use the function in the list of columns in a SELECT statement, instead of the `WHERE` clause, the address 10.0.1.1 would return 167772417, 10.0.11.1 will return 167774977, and 10.0.2.1 the number 167772673. As a complement to INET_ATON(), the function INET_ATON() will translate these numbers back to their original IP addresses.

MariaDB is fairly case insensitive, which usually is fine. However, to be able to check by case, the STRCMP() function can be used. It converts the column examined to a string and makes a comparison to the search parameter.

```
SELECT col1, col2
FROM table6
WHERE STRCMP(col3, 'text')=0;
```

If there is an exact match, the function STRCMP() returns 0. So if `col3` here contains "Text", it won't match. Incidentally, if `col3` alphabetically is before the string to which it's compared, a `-1` will be returned. If it's after it, a `1` is returned.

When you have list of items in one string, the SUBSTRING_INDEX() can be used to pull out a sub-string of data. As an example, suppose we have a column which has five elements, but we want to retrieve just the first two elements. This SQL statement will return them:

```
SELECT SUBSTRING_INDEX(col4, '|', 2)
FROM table7;
```

The first component in the function above is the column or string to be picked apart. The second component is the delimiter. The third is the number of elements to return, counting from the left. If we want to grab the last two elements, we would use a negative two to instruct MariaDB to count from the right end.

Conclusion

There are more string functions available in MariaDB. A few of the functions mentioned here have aliases or close alternatives. There are also functions for converting between ASCII, binary, hexi-decimal, and octal strings. And there are also string functions related to text encryption and decryption that were not mentioned. However, this article has given you a good collection of common string functions that will assist you in building more powerful and accurate SQL statements.

6.1.11 Restoring Data from Dump Files

Contents

If you lose your data in MariaDB, but have been using mariadb-dump (previously called mysqldump) to make regular backups of your data in MariaDB, you can use the dump files to restore your data. This is the point of the back-ups, after all. To restore a dump file, it's just a matter of having the mariadb client execute all of the SQL statements that the file contains. There are some things to consider before restoring from a dump file, so read this section all of the way through before restoring. One simple and perhaps clumsy method to restore from a dump file is to enter something like the following:

```
mariadb --user admin_restore --password < /data/backup/db1.sql
```

Again, this is not using mariadb-dump. The mariadb-dump utility is only for making back-up copies, not restoring databases. Instead, you would use the mariadb client, which will read the dump file's content in order to batch execute the SQL statements that it contains. Notice that the redirect for `STDOUT` is not used here, but the redirect for the standard input (`STDIN`); the less-than sign is used since the dump file is an input source. Also, notice that in this example a database isn't specified. That's given within the dump file. You may want to stop MariaDB before doing a restore, and then start it again when done.

Restoring One Table

The problem with restoring from a dump file is that you may overwrite tables or databases that you wish you hadn't. For instance, your dump file might be a few days old and only one table may have been lost. If you restore all of the databases or all of the tables in a database, you would be restoring the data back to it's state at the time of the backup, a few days before. This could be quite a disaster. This is why dumping by database and table can be handy. However, that could be cumbersome.

A simple and easy method of limiting a restoration would be to create temporarily a user who only has privileges for the table you want to restore. You would enter a GRANT statement like this:

```
GRANT SELECT
ON db1.* TO 'admin_restore_temp'@'localhost'
IDENTIFIED BY 'its_pwd';

GRANT ALL ON db1.table1
TO 'admin_restore_temp'@'localhost';
```

These two SQL statements allow the temporary user to have the needed SELECT privileges on all of the tables of `db1` and `ALL` privileges for the `table1` table. Now when you restore the dump file containing the whole `db1` database, only `table1` will be replaced with the back-up copy. Of course, MariaDB will generate errors. To overlook the errors and to proceed with the restoration of data where no errors are generated (i.e., `table1`), use the `--force` option. Here's what you would enter at the command-line for this situation:

```
mariadb --user admin_restore_temp --password --force < /data/backup/db1.sql
```

6.1.12 Basic SQL Queries: A Quick SQL Cheat Sheet

Contents

This page lists the most important SQL statements and contains links to their documentation pages. If you need a basic tutorial on how to use the MariaDB database server and how to execute simple commands, see A MariaDB Primer.

Also see Common MariaDB Queries for examples of commonly-used queries.

Defining How Your Data Is Stored

- CREATE DATABASE is used to create a new, empty database.
- DROP DATABASE is used to completely destroy an existing database.
- USE is used to select a default database.
- CREATE TABLE is used to create a new table, which is where your data is actually stored.
- ALTER TABLE is used to modify an existing table's definition.
- DROP TABLE is used to completely destroy an existing table.
- DESCRIBE shows the structure of a table.

Manipulating Your Data

- SELECT is used when you want to read (or select) your data.
- INSERT is used when you want to add (or insert) new data.
- UPDATE is used when you want to change (or update) existing data.
- DELETE is used when you want to remove (or delete) existing data.
- REPLACE is used when you want to add or change (or replace) new or existing data.
- TRUNCATE is used when you want to empty (or delete) all data from the template.

Transactions

- START TRANSACTION is used to begin a transaction.
- COMMIT is used to apply changes and end transaction.
- ROLLBACK is used to discard changes and end transaction.

A Simple Example

```
CREATE DATABASE mydb;
USE mydb;
CREATE TABLE mytable ( id INT PRIMARY KEY, name VARCHAR(20) );
INSERT INTO mytable VALUES ( 1, 'Will' );
INSERT INTO mytable VALUES ( 2, 'Marry' );
INSERT INTO mytable VALUES ( 3, 'Dean' );
SELECT id, name FROM mytable WHERE id = 1;
UPDATE mytable SET name = 'Willy' WHERE id = 1;
SELECT id, name FROM mytable;
DELETE FROM mytable WHERE id = 1;
SELECT id, name FROM mytable;
DROP DATABASE mydb;
SELECT count(1) from mytable; gives the number of records in the table
```

The first version of this article was copied, with permission, from http://hashmysql.org/wiki/Basic_SQL_Statements on 2012-10-05.

6.1.13 Connecting to MariaDB

Contents

This article covers connecting to MariaDB and the basic connection parameters. If you are completely new to MariaDB, take a look at A MariaDB Primer first.

In order to connect to the MariaDB server, the client software must provide the correct connection parameters. The client software will most often be the mariadb client, used for entering statements from the command line, but the same concepts apply to any client, such as a graphical client, a client to run backups such as mariadb-dump, etc. The rest of this article assumes that the mariadb command line client is used.

If a connection parameter is not provided, it will revert to a default value.

For example, to connect to MariaDB using only default values with the mariadb client, enter the following from the command line:

```
mariadb
```

In this case, the following defaults apply:

- The host name is `localhost`.
- The user name is either your Unix login name, or `ODBC` on Windows.
- No password is sent.
- The client will connect to the server with the default socket, but not any particular database on the server.

These defaults can be overridden by specifying a particular parameter to use. For example:

```
mariadb -h 166.78.144.191 -u username -ppassword database_name
```

In this case:

- `-h` specifies a host. Instead of using `localhost`, the IP `166.78.144.191` is used.
- `-u` specifies a user name, in this case `username`
- `-p` specifies a password, `password`. Note that for passwords, unlike the other parameters, there cannot be a space between the option (`-p`) and the value (`password`). It is also not secure to use a password in this way, as other users on the system can see it as part of the command that has been run. If you include the `-p` option, but leave out the password, you will be prompted for it, which is more secure.
- The database name is provided as the first argument after all the options, in this case `database_name`.
- It will connect with the default tcp_ip port, 3306

Connection Parameters

host

```
--host=name
-h name
```

Connect to the MariaDB server on the given host. The default host is `localhost`. By default, MariaDB does not permit remote logins - see Configuring MariaDB for Remote Client Access.

password

```
--password[=passwd]
-p[passwd]
```

The password of the MariaDB account. It is generally not secure to enter the password on the command line, as other users on the system can see it as part of the command that has been run. If you include the `-p` or `--password` option, but leave out the password, you will be prompted for it, which is more secure.

pipe

```
--pipe
-W
```

On Windows systems that have been started with the `--enable-named-pipe` option, use this option to connect to the server using a named pipe.

port

```
--port=num
-P num
```

The TCP/IP port number to use for the connection. The default is `3306`.

protocol

```
--protocol=name
```

Specifies the protocol to be used for the connection for the connection. It can be one of `TCP`, `SOCKET`, `PIPE` or `MEMORY` (case-insensitive). Usually you would not want to change this from the default. For example on Unix, a Unix socket file (`SOCKET`) is the default protocol, and usually results in the quickest connection.

- `TCP`: A TCP/IP connection to a server (either local or remote). Available on all operating systems.
- `SOCKET`: A Unix socket file connection, available to the local server on Unix systems only. If socket is not specified with --socket, in a config file or with the environment variable `MYSQL_UNIX_PORT` then the default `/tmp/mysql.sock` will be used.
- `PIPE`. A named-pipe connection (either local or remote). Available on Windows only.
- `MEMORY`. Shared-memory connection to the local server on Windows systems only.

shared-memory-base-name

```
--shared-memory-base-name=name
```

Only available on Windows systems in which the server has been started with the `--shared-memory` option, this specifies the shared-memory name to use for connecting to a local server. The value is case-sensitive, and defaults to `MARIADB`.

socket

```
--socket=name
-S name
```

For connections to localhost, this specifies either the Unix socket file to use (default `/tmp/mysql.sock`), or, on Windows where the server has been started with the `--enable-named-pipe` option, the name (case-insensitive) of the named pipe to use (default `MARIADB`).

TLS Options

A brief listing is provided below. See Secure Connections Overview and TLS System Variables for more detail.

ssl

```
--ssl
```

Enable TLS for connection (automatically enabled with other TLS flags). Disable with ' `--skip-ssl` '

ssl-ca

```
--ssl-ca=name
```

CA file in PEM format (check OpenSSL docs, implies `--ssl`).

ssl-capath

```
--ssl-capath=name
```

CA directory (check OpenSSL docs, implies `--ssl`).

ssl-cert

```
--ssl-cert=name
```

X509 cert in PEM format (implies `--ssl`).

ssl-cipher

```
--ssl-cipher=name
```

TLS cipher to use (implies `--ssl`).

ssl-key

```
--ssl-key=name
```

X509 key in PEM format (implies `--ssl`).

ssl-crl

```
--ssl-crl=name
```

Certificate revocation list (implies `--ssl`).

ssl-crlpath

```
--ssl-crlpath=name
```

Certificate revocation list path (implies `--ssl`).

ssl-verify-server-cert

```
--ssl-verify-server-cert
```

Verify server's "Common Name" in its cert against hostname used when connecting. This option is disabled by default.

user

```
--user=name
-u name
```

The MariaDB user name to use when connecting to the server. The default is either your Unix login name, or `ODBC` on Windows. See the GRANT command for details on creating MariaDB user accounts.

Option Files

It's also possible to use option files (or configuration files) to set these options. Most clients read option files. Usually, starting a client with the `--help` option will display which files it looks for as well as which option groups it recognizes.

6.1.14 External Tutorials

Contents

Here are some links to external MariaDB, MySQL and SQL tutorials that may be of interest

MariaDB-Focused Tutorials

- Tutorialspoint MariaDB tutorial for beginners
- MariaDB Tutorial by "Tech on the net"
- Learn MySQL / MariaDB for Beginners
- MariaDB Tutorial from javatpoint

MySQL-Focused Tutorials (But Should Work For MariaDB Too)

- mysqltutorial. Site with a lot of MySQL usage information and also how to connect to MySQL from different programming languages.
- MySQL Tutorial for Beginners Learn in 7 Days
- MySQL Tutorial from javatpoint
- MySQL Tutorial from w3resource
- MySQL by Examples for Beginners
- MySQL Tutorial – A Beginner's Guide To Learn MySQL
- Php/MySQL Tutorial from tizag
- PHP & MySQL Tutorial from siteground

General SQL Tutorials

- w3schools general SQL tutorial
- sqltutorial. Helps you get started with SQL quickly and effectively through many practical examples.

6.1.15 Useful MariaDB Queries

Contents

This page is intended to be a quick reference of commonly-used and/or useful queries in MariaDB.

Creating a Table

```
CREATE TABLE t1 ( a INT );
CREATE TABLE t2 ( b INT );
CREATE TABLE student_tests (
 name CHAR(10), test CHAR(10),
 score TINYINT, test_date DATE
);
```

See CREATE TABLE for more.

Inserting Records

```
INSERT INTO t1 VALUES (1), (2), (3);
INSERT INTO t2 VALUES (2), (4);

INSERT INTO student_tests
 (name, test, score, test_date) VALUES
 ('Chun', 'SQL', 75, '2012-11-05'),
 ('Chun', 'Tuning', 73, '2013-06-14'),
 ('Esben', 'SQL', 43, '2014-02-11'),
 ('Esben', 'Tuning', 31, '2014-02-09'),
 ('Kaolin', 'SQL', 56, '2014-01-01'),
 ('Kaolin', 'Tuning', 88, '2013-12-29'),
 ('Tatiana', 'SQL', 87, '2012-04-28'),
 ('Tatiana', 'Tuning', 83, '2013-09-30');
```

See INSERT for more.

Using AUTO_INCREMENT

The AUTO_INCREMENT attribute is used to automatically generate a unique identity for new rows.

```
CREATE TABLE student_details (
 id INT NOT NULL AUTO_INCREMENT, name CHAR(10),
 date_of_birth DATE, PRIMARY KEY (id)
);
```

When inserting, the id field can be omitted, and is automatically created.

```
INSERT INTO student_details (name,date_of_birth) VALUES
 ('Chun', '1993-12-31'),
 ('Esben','1946-01-01'),
 ('Kaolin','1996-07-16'),
 ('Tatiana', '1988-04-13');

SELECT * FROM student_details;
+----+---------+---------------+
| id | name    | date_of_birth |
+----+---------+---------------+
|  1 | Chun    | 1993-12-31    |
|  2 | Esben   | 1946-01-01    |
|  3 | Kaolin  | 1996-07-16    |
|  4 | Tatiana | 1988-04-13    |
+----+---------+---------------+
```

See AUTO_INCREMENT for more.

Querying from two tables on a common value

```
SELECT * FROM t1 INNER JOIN t2 ON t1.a = t2.b;
```

This kind of query is called a join - see JOINS for more.

Finding the Maximum Value

```
SELECT MAX(a) FROM t1;
+--------+
| MAX(a) |
+--------+
|      3 |
+--------+
```

See the MAX() function for more, as well as Finding the maximum value and grouping the results below for a more practical example.

Finding the Minimum Value

```
SELECT MIN(a) FROM t1;
+--------+
| MIN(a) |
+--------+
|      1 |
+--------+
```

See the MIN() function for more.

Finding the Average Value

```
SELECT AVG(a) FROM t1;
+--------+
| AVG(a) |
+--------+
| 2.0000 |
+--------+
```

See the AVG() function for more.

Finding the Maximum Value and Grouping the Results

```
SELECT name, MAX(score) FROM student_tests GROUP BY name;
+---------+------------+
| name    | MAX(score) |
+---------+------------+
| Chun    |         75 |
| Esben   |         43 |
| Kaolin  |         88 |
| Tatiana |         87 |
+---------+------------+
```

See the MAX() function for more.

Ordering Results

```
SELECT name, test, score FROM student_tests ORDER BY score DESC;
+---------+--------+-------+
| name    | test   | score |
+---------+--------+-------+
| Kaolin  | Tuning |    88 |
| Tatiana | SQL    |    87 |
| Tatiana | Tuning |    83 |
| Chun    | SQL    |    75 |
| Chun    | Tuning |    73 |
| Kaolin  | SQL    |    56 |
| Esben   | SQL    |    43 |
| Esben   | Tuning |    31 |
+---------+--------+-------+
```

See ORDER BY for more.

Finding the Row with the Minimum of a Particular Column

In this example, we want to find the lowest test score for any student.

```
SELECT name,test, score FROM student_tests WHERE score=(SELECT MIN(score) FROM student);
+-------+--------+-------+
| name  | test   | score |
+-------+--------+-------+
| Esben | Tuning |    31 |
+-------+--------+-------+
```

Finding Rows with the Maximum Value of a Column by Group

This example returns the best test results of each student:

```
SELECT name, test, score FROM student_tests st1 WHERE score = (
  SELECT MAX(score) FROM student st2 WHERE st1.name = st2.name
);
+---------+--------+-------+
| name    | test   | score |
+---------+--------+-------+
| Chun    | SQL    |    75 |
| Esben   | SQL    |    43 |
| Kaolin  | Tuning |    88 |
| Tatiana | SQL    |    87 |
+---------+--------+-------+
```

Calculating Age

The TIMESTAMPDIFF function can be used to calculate someone's age:

```
SELECT CURDATE() AS today;
+------------+
| today      |
+------------+
| 2014-02-17 |
+------------+

SELECT name, date_of_birth, TIMESTAMPDIFF(YEAR,date_of_birth,'2014-08-02') AS age
  FROM student_details;
+---------+---------------+------+
| name    | date_of_birth | age  |
+---------+---------------+------+
| Chun    | 1993-12-31    |   20 |
| Esben   | 1946-01-01    |   68 |
| Kaolin  | 1996-07-16    |   18 |
| Tatiana | 1988-04-13    |   26 |
+---------+---------------+------+
```

See TIMESTAMPDIFF() for more.

Using User-defined Variables

This example sets a user-defined variable with the average test score, and then uses it in a later query to return all results above the average.

```
SELECT @avg_score:= AVG(score) FROM student_tests;
+-------------------------+
| @avg_score:= AVG(score) |
+-------------------------+
|            67.000000000 |
+-------------------------+

SELECT * FROM student_tests WHERE score > @avg_score;
+---------+--------+-------+------------+
| name    | test   | score | test_date  |
+---------+--------+-------+------------+
| Chun    | SQL    |    75 | 2012-11-05 |
| Chun    | Tuning |    73 | 2013-06-14 |
| Kaolin  | Tuning |    88 | 2013-12-29 |
| Tatiana | SQL    |    87 | 2012-04-28 |
| Tatiana | Tuning |    83 | 2013-09-30 |
+---------+--------+-------+------------+
```

User-defined variables can also be used to add an incremental counter to a resultset:

```
SET @count = 0;

SELECT @count := @count + 1 AS counter, name, date_of_birth FROM student_details;
+---------+---------+---------------+
| counter | name    | date_of_birth |
+---------+---------+---------------+
|       1 | Chun    | 1993-12-31    |
|       2 | Esben   | 1946-01-01    |
|       3 | Kaolin  | 1996-07-16    |
|       4 | Tatiana | 1988-04-13    |
+---------+---------+---------------+
```

See User-defined Variables for more.

View Tables in Order of Size

Returns a list of all tables in the database, ordered by size:

```
SELECT table_schema as `DB`, table_name AS `Table`,
  ROUND(((data_length + index_length) / 1024 / 1024), 2) `Size (MB)`
  FROM information_schema.TABLES
  ORDER BY (data_length + index_length) DESC;

+--------------------+---------------------------------------+-----------+
| DB                 | Table                                 | Size (MB) |
+--------------------+---------------------------------------+-----------+
| wordpress          | wp_simple_history_contexts            |      7.05 |
| wordpress          | wp_posts                              |      6.59 |
| wordpress          | wp_simple_history                     |      3.05 |
| wordpress          | wp_comments                           |      2.73 |
| wordpress          | wp_commentmeta                        |      2.47 |
| wordpress          | wp_simple_login_log                   |      2.03 |
...
```

Removing Duplicates

This example assumes there's a unique ID, but that all other fields are identical. In the example below, there are 4 records, 3 of which are duplicates, so two of the three duplicates need to be removed. The intermediate SELECT is not necessary, but demonstrates what is being returned.

```
CREATE TABLE t (id INT, f1 VARCHAR(2));

INSERT INTO t VALUES (1,'a'), (2,'a'), (3,'b'), (4,'a');

SELECT * FROM t t1, t t2 WHERE t1.f1=t2.f1 AND t1.id<>t2.id AND t1.id=(
  SELECT MAX(id) FROM t tab WHERE tab.f1=t1.f1
);
+------+------+------+------+
| id   | f1   | id   | f1   |
+------+------+------+------+
|    4 | a    |    1 | a    |
|    4 | a    |    2 | a    |
+------+------+------+------+

DELETE FROM t WHERE id IN (
  SELECT t2.id FROM t t1, t t2 WHERE t1.f1=t2.f1 AND t1.id<>t2.id AND t1.id=(
    SELECT MAX(id) FROM t tab WHERE tab.f1=t1.f1
  )
);
Query OK, 2 rows affected (0.120 sec)

SELECT * FROM t;
+------+------+
| id   | f1   |
+------+------+
|    3 | b    |
|    4 | a    |
+------+------
```

6.2 Basic MariaDB Articles

These articles are at a basic level. They are more advanced than beginners and less advanced than intermediate developers and administrators.

Basic SQL Debugging

An introductory tutorial on debugging MariaDB.

Configuring MariaDB for Remote Client Access

How to configure MariaDB for remote client access.

Creating & Using Views

A tutorial on creating and using views.

Getting Started with Indexes

Extensive tutorial on creating indexes for tables.

Joining Tables with JOIN Clauses

An introductory tutorial on using the JOIN clause.

The Essentials of an Index

Explains the basics of a table index.

Troubleshooting Connection Issues

Common problems when trying to connect to MariaDB.

6.2.1 Basic SQL Debugging

Contents

Designing Queries

Following a few conventions makes finding errors in queries a lot easier, especially when you ask for help from people who might know SQL, but know nothing about your particular schema. A query easy to read is a query easy to debug. Use whitespace to group clauses within the query. Choose good table and field aliases to add clarity, not confusion. Choose the syntax that supports the query's meaning.

Using Whitespace

A query hard to read is a query hard to debug. White space is free. New lines and indentation make queries easy to read, particularly when constructing a query inside a scripting language, where variables are interspersed throughout the query.

There is a syntax error in the following. How fast can you find it?

```
SELECT u.id, u.name, alliance.ally FROM users u JOIN alliance ON
(u.id=alliance.userId) JOIN team ON (alliance.teamId=team.teamId
WHERE team.teamName='Legionnaires' AND u.online=1 AND ((u.subscription='paid'
AND u.paymentStatus='current') OR u.subscription='free') ORDER BY u.name;
```

Here's the same query, with correct use of whitespace. Can you find the error faster?

```
SELECT
    u.id
    , u.name
    , alliance.ally
FROM
    users u
    JOIN alliance ON (u.id = alliance.userId)
    JOIN team ON (alliance.teamId = team.teamId
WHERE
    team.teamName = 'Legionnaires'
    AND u.online = 1
    AND (
        (u.subscription = 'paid' AND u.paymentStatus = 'current')
        OR
        u.subscription = 'free'
    )
ORDER BY
    u.name;
```

Even if you don't know SQL, you might still have caught the missing ')' following team.teamId.

The exact formatting style you use isn't so important. You might like commas in the select list to follow expressions, rather than precede them. You might indent with tabs or with spaces. Adherence to some particular form is not important. Legibility is the only goal.

Table and Field Aliases

Aliases allow you to rename tables and fields for use within a query. This can be handy when the original names are very long, and is required for self joins and certain subqueries. However, poorly chosen aliases can make a query harder to debug, rather than easier. Aliases should reflect the original table name, not an arbitrary string.

Bad:

```
SELECT *
FROM
    financial_reportQ_1 AS a
    JOIN sales_renderings AS b ON (a.salesGroup = b.groupId)
    JOIN sales_agents AS c ON (b.groupId = c.group)
WHERE
    b.totalSales > 10000
    AND c.id != a.clientId
```

As the list of joined tables and the WHERE clause grow, it becomes necessary to repeatedly look back to the top of the query to see to which table any given alias refers.

Better:

```
SELECT *
FROM
    financial_report_Q_1 AS frq1
    JOIN sales_renderings AS sr ON (frq1.salesGroup = sr.groupId)
    JOIN sales_agents AS sa ON (sr.groupId = sa.group)
WHERE
    sr.totalSales > 10000
    AND sa.id != frq1.clientId
```

Each alias is just a little longer, but the table initials give enough clues that anyone familiar with the database only need see the full table name once, and can generally remember which table goes with which alias while reading the rest of the query.

Placing JOIN conditions

The manual warns against using the JOIN condition (that is, the ON clause) for restricting rows. Some queries, particularly those using implicit joins, take the opposite extreme - all join conditions are moved to the WHERE clause. In consequence, the table relationships are mixed with the business logic.

Bad:

```
SELECT *
FROM
    family,
    relationships
WHERE
    family.personId = relationships.personId
    AND relationships.relation = 'father'
```

Without digging through the WHERE clause, it is impossible to say what links the two tables.

Better:

```
SELECT *
FROM
    family
    JOIN relationships ON (family.personId = relationships.personId)
WHERE
    relationships.relation = 'father'
```

The relation between the tables is immediately obvious. The WHERE clause is left to limit rows in the result set.

Compliance with such a restriction negates the use of the comma operator to join tables. It is a small price to pay. Queries should be written using the explicit JOIN keyword anyway, and the two should never be mixed (unless you like rewriting all your queries every time a new version changes operator precedence).

Finding Syntax Errors

Syntax errors are among the easiest problems to solve. MariaDB provides an error message showing the exact point where the parser became confused. Check the query, including a few words before the phrase shown in the error message. Most syntax and parsing errors are obvious after a second look, but some are more elusive, especially when the error text seems empty, points to a valid keyword, or seems to error on syntax that appears exactly correct.

Interpreting the Empty Error

Most syntax errors are easy to interpret. The error generally details the exact source of the trouble. A careful look at the query, with the error message in mind, often reveals an obvious mistake, such as mispelled field names, a missing 'AND', or an extra closing parenthesis. Sometimes the error is a little less helpful. A frequent, less-than-helpful message:

```
ERROR 1064: You have an error in your SQL syntax; check the manual that corresponds to your
MariaDB server version for the right syntax to use near ' ' at line 1
```

The empty ' ' can be disheartening. Clearly there is an error, but where? A good place to look is at the end of the query. The ' ' suggests that the parser reached the end of the statement while still expecting some syntax token to appear.

Check for missing closers, such as ' and):

```
SELECT * FROM someTable WHERE field = 'value
```

Look for incomplete clauses, often indicated by an exposed comma:

```
SELECT * FROM someTable WHERE field = 1 GROUP BY id,
```

Checking for keywords

MariaDB allows table and field names and aliases that are also reserved words. To prevent ambiguity, such names must be enclosed in backticks (`):

```
SELECT * FROM actionTable WHERE `DELETE` = 1;
```

If the syntax error is shown near one of your identifiers, check if it appears on the reserved word list.

A text editor with color highlighting for SQL syntax helps to find these errors. When you enter a field name, and it shows up in the same color as the SELECT keyword, you know something is amiss. Some common culprits:

- **DESC** is a common abbreviation for "description" fields. It means "descending" in a MariaDB **ORDER** clause.
- **DATE**, **TIME**, and **TIMESTAMP** are all common field names. They are also field types.
- **ORDER** appears in sales applications. MariaDB uses it to specify sorting for results.

Some keywords are so common that MariaDB makes a special allowance to use them unquoted. My advice: don't. If it's a keyword, quote it.

Version specific syntax

As MariaDB adds new features, the syntax must change to support them. Most of the time, old syntax will work in newer versions of MariaDB. One notable exception is the change in precedence of the comma operator relative to the JOIN keyword in version 5.0. A query that used to work, such as

```
SELECT * FROM a, b JOIN c ON a.x = c.x;
```

will now fail.

More common, however, is an attempt to use new syntax in an old version. Web hosting companies are notoriously slow to upgrade MariaDB, and you may find yourself using a version several years out of date. The result can be very frustrating when a query that executes flawlessly on your own workstation, running a recent installation, fails completely in your production environment.

This query fails in any version of MySQL prior to 4.1, when subqueries were added to the server:

```
SELECT * FROM someTable WHERE someId IN (SELECT id FROM someLookupTable);
```

This query fails in some early versions of MySQL, because the JOIN syntax did not originally allow an ON clause:

```
SELECT * FROM tableA JOIN tableB ON tableA.x = tableB.y;
```

Always check the installed version of MariaDB, and read the section of the manual relevant for that version. The manual usually indicates exactly when particular syntax became available for use.

The initial version of this article was copied, with permission, from http://hashmysql.org/wiki/Basic_Debugging on 2012-10-05.

6.2.2 Configuring MariaDB for Remote Client Access

Contents

Some MariaDB packages bind MariaDB to 127.0.0.1 (the loopback IP address) by default as a security measure using the bind-address configuration directive. Old MySQL packages sometimes disabled TCP/IP networking altogether using the skip-networking directive. Before going in to how to configure these, let's explain what each of them actually does:

- skip-networking is fairly simple. It just tells MariaDB to run without any of the TCP/IP networking options.
- bind-address requires a little bit of background information. A given server usually has at least two networking interfaces (although this is not required) and can easily have more. The two most common are a *Loopback* network device and a physical *Network Interface Card* (NIC) which allows you to communicate with the network. MariaDB is bound to the loopback interface by default because it makes it impossible to connect to the TCP port on the server from a remote host (the bind-address must refer to a local IP address, or you will receive a fatal error and MariaDB will not start). This of course is not desirable if you want to use the TCP port from a remote host, so you must remove this bind-address directive or replace it either 0.0.0.0 to listen on all interfaces, or the address of a specific public interface.

MariaDB starting with 10.11

Multiple comma-separated addresses can now be given to `bind_address` to allow the server to listen on more than one specific interface while not listening on others.

If bind-address is bound to 127.0.0.1 (localhost), one can't connect to the MariaDB server from other hosts or from the same host over TCP/IP on a different interface than the loopback (127.0.0.1). This for example will not work (connecting with a hostname that points to a local IP of the host):

```
(/my/maria-10.11) ./client/mariadb --host=myhost --protocol=tcp --port=3306 test
ERROR 2002 (HY000): Can't connect to MySQL server on 'myhost' (115)
(/my/maria-10.11) telnet myhost 3306
Trying 192.168.0.11...
telnet: connect to address 192.168.0.11: Connection refused
```

Using 'localhost' works when binding with `bind_address` :

```
(my/maria-10.11) ./client/mariadb --host=localhost --protocol=tcp --port=3306 test
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MariaDB monitor.  Commands end with ; or \g.
...
```

Finding the Defaults File

To enable MariaDB to listen to remote connections, you need to edit your defaults file. See Configuring MariaDB with my.cnf for more detail.

Common locations for defaults files:

```
  * /etc/my.cnf                              (*nix/BSD)
  * $MYSQL_HOME/my.cnf                       (*nix/BSD) *Most Notably /etc/mysql/my.cnf
  * SYSCONFDIR/my.cnf                        (*nix/BSD)
  * DATADIR\my.ini                           (Windows)
```

You can see which defaults files are read and in which order by executing:

```
shell> mariadbd --help --verbose
mariadbd  Ver 10.11.5-MariaDB for linux-systemd on x86_64 (MariaDB Server)
Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Starts the MariaDB database server.

Usage: ./mariadbd [OPTIONS]

Default options are read from the following files in the given order:
/etc/my.cnf /etc/mysql/my.cnf ~/.my.cnf
```

The last line shows which defaults files are read.

Editing the Defaults File

Once you have located the defaults file, use a text editor to open the file and try to find lines like this under the [mysqld] section:

```
[mysqld]
   ...
   skip-networking
   ...
   bind-address = <some ip-address>
   ...
```

(The lines may not be in this order, and the order doesn't matter.)

If you are able to locate these lines, make sure they are both commented out (prefaced with hash (#) characters), so that they look like this:

```
[mysqld]
   ...
   #skip-networking
   ...
   #bind-address = <some ip-address>
   ...
```

(Again, the order of these lines don't matter)

Alternatively, just add the following lines **at the end** of your .my.cnf (notice that the file name starts with a dot) file in your home directory or alternative **last** in your /etc/my.cnf file.

```
[mysqld]
skip-networking=0
skip-bind-address
```

This works as one can have any number of [mysqld] sections.

Save the file and restart the mariadbd daemon or service (see Starting and Stopping MariaDB).

You can check the options mariadbd is using by executing:

```
shell> ./sql/mariadbd --print-defaults
./sql/mariadbd would have been started with the following arguments:
--bind-address=127.0.0.1 --innodb_file_per_table=ON --server-id=1 --skip-bind-address ...
```

It doesn't matter if you have the original --bind-address left as the later --skip-bind-address will overwrite it.

Granting User Connections From Remote Hosts

Now that your MariaDB server installation is setup to accept connections from remote hosts, we have to add a user that is allowed to connect from something other than 'localhost' (Users in MariaDB are defined as 'user'@'host', so 'chadmaynard'@'localhost' and 'chadmaynard'@'1.1.1.1' (or 'chadmaynard'@'server.domain.local') are different users that can have completely different permissions and/or passwords.

To create a new user:

- log into the mariadb command line client (or your favorite graphical client if you wish)

```
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 36
Server version: 5.5.28-MariaDB-mariadb1~lucid mariadb.org binary distribution

Copyright (c) 2000, 2012, Oracle, Monty Program Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

- if you are interested in viewing any existing remote users, issue the following SQL statement on the mysql.user table:

```
SELECT User, Host FROM mysql.user WHERE Host <> 'localhost';
+--------+-----------+
| User   | Host      |
+--------+-----------+
| daniel | %         |
| root   | 127.0.0.1 |
| root   | ::1       |
| root   | gandalf   |
+--------+-----------+
4 rows in set (0.00 sec)
```

(If you have a fresh install, it is normal for no rows to be returned)

Now you have some decisions to make. At the heart of every grant statement you have these things:

- list of allowed privileges
- what database/tables these privileges apply to
- username
- host this user can connect from
- and optionally a password

It is common for people to want to create a "root" user that can connect from anywhere, so as an example, we'll do just that, but to improve on it we'll create a root user that can connect from anywhere on my local area network (LAN), which has addresses in the subnet 192.168.100.0/24. This is an improvement because opening a MariaDB server up to the Internet and granting access to all hosts is bad practice.

```
GRANT ALL PRIVILEGES ON *.* TO 'root'@'192.168.100.%'
  IDENTIFIED BY 'my-new-password' WITH GRANT OPTION;
```

(% is a wildcard)

For more information about how to use GRANT, please see the GRANT page.

At this point we have accomplished our goal and we have a user 'root' that can connect from anywhere on the 192.168.100.0/24 LAN.

Port 3306 is Configured in Firewall

One more point to consider whether the firewall is configured to allow incoming request from remote clients:

On RHEL and CentOS 7, it may be necessary to configure the firewall to allow TCP access to MariaDB from remote hosts. To do so, execute both of these commands:

```
firewall-cmd --add-port=3306/tcp
firewall-cmd --permanent --add-port=3306/tcp
```

Caveats

- If your system is running a software firewall (or behind a hardware firewall or NAT) you must allow connections

destined to TCP port that MariaDB runs on (by default and almost always 3306).

- To undo this change and not allow remote access anymore, simply remove the `skip-bind-address` line or uncomment the bind-address line in your defaults file. The end result should be that you should have in the output from `./sql/mariadbd --print-defaults` the option `--bind-address=127.0.0.1` and no `--skip-bind-address`.

The initial version of this article was copied, with permission, from http://hashmysql.org/wiki/Remote_Clients_Cannot_Connect on 2012-10-30.

6.2.3 Creating & Using Views

Contents

A Tutorial Introduction

Up-front warning: This is the beginning of a very basic tutorial on views, based on my experimentation with them. This tutorial assumes that you've read the appropriate tutorials up to and including More Advanced Joins (or that you understand the concepts behind them). This page is intended to give you a general idea of how views work and what they do, as well as some examples of when you could use them.

Requirements for This Tutorial

In order to perform the SQL statements in this tutorial, you will need access to a MariaDB database and you will need the CREATE TABLE and CREATE VIEW privileges on this table.

The Employee Database

First, we need some data we can perform our optimizations on, so we'll recreate the tables from the More Advanced Joins tutorial, to provide us with a starting point. If you have already completed that tutorial and have this database already, you can skip ahead.

First, we create the table that will hold all of the employees and their contact information:

```
CREATE TABLE `Employees` (
  `ID` TINYINT(3) UNSIGNED NOT NULL AUTO_INCREMENT,
  `First_Name` VARCHAR(25) NOT NULL,
  `Last_Name` VARCHAR(25) NOT NULL,
  `Position` VARCHAR(25) NOT NULL,
  `Home_Address` VARCHAR(50) NOT NULL,
  `Home_Phone` VARCHAR(12) NOT NULL,
  PRIMARY KEY (`ID`)
) ENGINE=MyISAM;
```

Next, we add a few employees to the table:

```
INSERT INTO `Employees` (`First_Name`, `Last_Name`, `Position`, `Home_Address`, `Home_Phone`)
VALUES
  ('Mustapha', 'Mond', 'Chief Executive Officer', '692 Promiscuous Plaza', '326-555-3492'),
  ('Henry', 'Foster', 'Store Manager', '314 Savage Circle', '326-555-3847'),
  ('Bernard', 'Marx', 'Cashier', '1240 Ambient Avenue', '326-555-8456'),
  ('Lenina', 'Crowne', 'Cashier', '281 Bumblepuppy Boulevard', '328-555-2349'),
  ('Fanny', 'Crowne', 'Restocker', '1023 Bokanovsky Lane', '326-555-6329'),
  ('Helmholtz', 'Watson', 'Janitor', '944 Soma Court', '329-555-2478');
```

Now, we create a second table, containing the hours which each employee clocked in and out during the week:

```
CREATE TABLE `Hours` (
  `ID` TINYINT(3) UNSIGNED NOT NULL,
  `Clock_In` DATETIME NOT NULL,
  `Clock_Out` DATETIME NOT NULL
) ENGINE=MyISAM;
```

Finally, although it is a lot of information, we add a full week of hours for each of the employees into the second table that we created:

```
INSERT INTO `Hours`
VALUES ('1', '2005-08-08 07:00:42', '2005-08-08 17:01:36'),
  ('1', '2005-08-09 07:01:34', '2005-08-09 17:10:11'),
  ('1', '2005-08-10 06:59:56', '2005-08-10 17:09:29'),
  ('1', '2005-08-11 07:00:17', '2005-08-11 17:00:47'),
  ('1', '2005-08-12 07:02:29', '2005-08-12 16:59:12'),
  ('2', '2005-08-08 07:00:25', '2005-08-08 17:03:13'),
  ('2', '2005-08-09 07:00:57', '2005-08-09 17:05:09'),
  ('2', '2005-08-10 06:58:43', '2005-08-10 16:58:24'),
  ('2', '2005-08-11 07:01:58', '2005-08-11 17:00:45'),
  ('2', '2005-08-12 07:02:12', '2005-08-12 16:58:57'),
  ('3', '2005-08-08 07:00:12', '2005-08-08 17:01:32'),
  ('3', '2005-08-09 07:01:10', '2005-08-09 17:00:26'),
  ('3', '2005-08-10 06:59:53', '2005-08-10 17:02:53'),
  ('3', '2005-08-11 07:01:15', '2005-08-11 17:04:23'),
  ('3', '2005-08-12 07:00:51', '2005-08-12 16:57:52'),
  ('4', '2005-08-08 06:54:37', '2005-08-08 17:01:23'),
  ('4', '2005-08-09 06:58:23', '2005-08-09 17:00:54'),
  ('4', '2005-08-10 06:59:14', '2005-08-10 17:00:12'),
  ('4', '2005-08-11 07:00:49', '2005-08-11 17:00:34'),
  ('4', '2005-08-12 07:01:09', '2005-08-12 16:58:29'),
  ('5', '2005-08-08 07:00:04', '2005-08-08 17:01:43'),
  ('5', '2005-08-09 07:02:12', '2005-08-09 17:02:13'),
  ('5', '2005-08-10 06:59:39', '2005-08-10 17:03:37'),
  ('5', '2005-08-11 07:01:26', '2005-08-11 17:00:03'),
  ('5', '2005-08-12 07:02:15', '2005-08-12 16:59:02'),
  ('6', '2005-08-08 07:00:12', '2005-08-08 17:01:02'),
  ('6', '2005-08-09 07:03:44', '2005-08-09 17:00:00'),
  ('6', '2005-08-10 06:54:19', '2005-08-10 17:03:31'),
  ('6', '2005-08-11 07:00:05', '2005-08-11 17:02:57'),
  ('6', '2005-08-12 07:02:07', '2005-08-12 16:58:23');
```

Working with the Employee Database

In this example, we are going to assist Human Resources by simplifying the queries that their applications need to perform. At the same time, it's going to enable us to abstract their queries from the database, which allows us more flexibility in maintaining it.

Filtering by Name, Date and Time

In the previous tutorial, we looked at a JOIN query that displayed all of the lateness instances for a particular employee. In this tutorial, we are going to abstract that query somewhat to provide us with all lateness occurrences for all employees, and then standardize that query by making it into a view.

Our previous query looked like this:

```
SELECT
  `Employees`.`First_Name`,
  `Employees`.`Last_Name`,
  `Hours`.`Clock_In`,
  `Hours`.`Clock_Out`
FROM `Employees`
INNER JOIN `Hours` ON `Employees`.`ID` = `Hours`.`ID`
WHERE `Employees`.`First_Name` = 'Helmholtz'
AND DATE_FORMAT(`Hours`.`Clock_In`, '%Y-%m-%d') >= '2005-08-08'
AND DATE_FORMAT(`Hours`.`Clock_In`, '%Y-%m-%d') <= '2005-08-12'
AND DATE_FORMAT(`Hours`.`Clock_In`, '%H:%i:%S') > '07:00:59';
```

The result:

```
+------------+-----------+---------------------+---------------------+
| First_Name | Last_Name | Clock_In            | Clock_Out           |
+------------+-----------+---------------------+---------------------+
| Helmholtz  | Watson    | 2005-08-09 07:03:44 | 2005-08-09 17:00:00 |
| Helmholtz  | Watson    | 2005-08-12 07:02:07 | 2005-08-12 16:58:23 |
+------------+-----------+---------------------+---------------------+
```

Refining Our Query

The previous example displays to us all of Heimholtz's punch-in times that were after seven AM. We can see here that Heimholz has been late twice within this reporting period, and we can also see that in both instances, he either left exactly on time or he left early. Our company policy, however, dictates that late instances must be made up at the end of one's shift, so we want to exclude from our report anyone whose clock-out time was greater than 10 hours and one minute after their clock-in time.

```
SELECT
  `Employees`.`First_Name`,
  `Employees`.`Last_Name`,
  `Hours`.`Clock_In`,
  `Hours`.`Clock_Out`,
  (TIMESTAMPDIFF(MINUTE,`Hours`.`Clock_Out`,`Hours`.`Clock_In`) + 601) as Difference
FROM `Employees`
INNER JOIN `Hours` USING (`ID`)
WHERE DATE_FORMAT(`Hours`.`Clock_In`, '%Y-%m-%d') >= '2005-08-08'
AND DATE_FORMAT(`Hours`.`Clock_In`, '%Y-%m-%d') <= '2005-08-12'
AND DATE_FORMAT(`Hours`.`Clock_In`, '%H:%i:%S') > '07:00:59'
AND TIMESTAMPDIFF(MINUTE,`Hours`.`Clock_Out`,`Hours`.`Clock_In`) > -601;
```

This gives us the following list of people who have violated our attendance policy:

```
+------------+-----------+---------------------+---------------------+------------+
| First_Name | Last_Name | Clock_In            | Clock_Out           | Difference |
+------------+-----------+---------------------+---------------------+------------+
| Mustapha   | Mond      | 2005-08-12 07:02:29 | 2005-08-12 16:59:12 |          4 |
| Henry      | Foster    | 2005-08-11 07:01:58 | 2005-08-11 17:00:45 |          2 |
| Henry      | Foster    | 2005-08-12 07:02:12 | 2005-08-12 16:58:57 |          4 |
| Bernard    | Marx      | 2005-08-09 07:01:10 | 2005-08-09 17:00:26 |          1 |
| Lenina     | Crowne    | 2005-08-12 07:01:09 | 2005-08-12 16:58:29 |          3 |
| Fanny      | Crowne    | 2005-08-11 07:01:26 | 2005-08-11 17:00:03 |          2 |
| Fanny      | Crowne    | 2005-08-12 07:02:15 | 2005-08-12 16:59:02 |          4 |
| Helmholtz  | Watson    | 2005-08-09 07:03:44 | 2005-08-09 17:00:00 |          4 |
| Helmholtz  | Watson    | 2005-08-12 07:02:07 | 2005-08-12 16:58:23 |          4 |
+------------+-----------+---------------------+---------------------+------------+
```

The Utility of Views

We can see in the previous example that there have been several instances of employees coming in late and leaving early. Unfortunately, we can also see that this query is getting needlessly complex. Having all of this SQL in our application not only creates more complex application code, but also means that if we ever change the structure of this table we're going to have to change what is becoming a somewhat messy query. This is where views begin to show their usefulness.

Creating the Employee Tardiness View

Creating a view is almost exactly the same as creating a SELECT statement, so we can use our previous SELECT statement in the creation of our new view:

```
CREATE SQL SECURITY INVOKER VIEW Employee_Tardiness AS
SELECT
   `Employees`.`First_Name`,
   `Employees`.`Last_Name`,
   `Hours`.`Clock_In`,
   `Hours`.`Clock_Out`,
(TIMESTAMPDIFF(MINUTE,`Hours`.`Clock_Out`,`Hours`.`Clock_In`) + 601) as Difference
FROM `Employees`
INNER JOIN `Hours` USING (`ID`)
WHERE DATE_FORMAT(`Hours`.`Clock_In`, '%Y-%m-%d') >= '2005-08-08'
AND DATE_FORMAT(`Hours`.`Clock_In`, '%Y-%m-%d') <= '2005-08-12'
AND DATE_FORMAT(`Hours`.`Clock_In`, '%H:%i:%S') > '07:00:59'
AND TIMESTAMPDIFF(MINUTE,`Hours`.`Clock_Out`,`Hours`.`Clock_In`) > -601;
```

Note that the first line of our query contains the statement 'SQL SECURITY INVOKER' - this means that when the view is accessed, it runs with the same privileges that the person accessing the view has. Thus, if someone without access to our Employees table tries to access this view, they will get an error.

Other than the security parameter, the rest of the query is fairly self explanatory. We simply run 'CREATE VIEW <view-name> AS' and then append any valid SELECT statement, and our view is created. Now if we do a SELECT from the view, we can see we get the same results as before, with much less SQL:

```
SELECT * FROM Employee_Tardiness;
```

```
+------------+-----------+---------------------+---------------------+------------+
| First_Name | Last_Name | Clock_In            | Clock_Out           | Difference |
+------------+-----------+---------------------+---------------------+------------+
| Mustapha   | Mond      | 2005-08-12 07:02:29 | 2005-08-12 16:59:12 |          5 |
| Henry      | Foster    | 2005-08-11 07:01:58 | 2005-08-11 17:00:45 |          3 |
| Henry      | Foster    | 2005-08-12 07:02:12 | 2005-08-12 16:58:57 |          5 |
| Bernard    | Marx      | 2005-08-09 07:01:10 | 2005-08-09 17:00:26 |          2 |
| Lenina     | Crowne    | 2005-08-12 07:01:09 | 2005-08-12 16:58:29 |          4 |
| Fanny      | Crowne    | 2005-08-09 07:02:12 | 2005-08-09 17:02:13 |          1 |
| Fanny      | Crowne    | 2005-08-11 07:01:26 | 2005-08-11 17:00:03 |          3 |
| Fanny      | Crowne    | 2005-08-12 07:02:15 | 2005-08-12 16:59:02 |          5 |
| Helmholtz  | Watson    | 2005-08-09 07:03:44 | 2005-08-09 17:00:00 |          5 |
| Helmholtz  | Watson    | 2005-08-12 07:02:07 | 2005-08-12 16:58:23 |          5 |
+------------+-----------+---------------------+---------------------+------------+
```

Now we can even perform operations on the table, such as limiting our results to just those with a Difference of at least five minutes:

```
SELECT * FROM Employee_Tardiness WHERE Difference >=5;
```

```
+------------+-----------+---------------------+---------------------+------------+
| First_Name | Last_Name | Clock_In            | Clock_Out           | Difference |
+------------+-----------+---------------------+---------------------+------------+
| Mustapha   | Mond      | 2005-08-12 07:02:29 | 2005-08-12 16:59:12 |          5 |
| Henry      | Foster    | 2005-08-12 07:02:12 | 2005-08-12 16:58:57 |          5 |
| Fanny      | Crowne    | 2005-08-12 07:02:15 | 2005-08-12 16:59:02 |          5 |
| Helmholtz  | Watson    | 2005-08-09 07:03:44 | 2005-08-09 17:00:00 |          5 |
| Helmholtz  | Watson    | 2005-08-12 07:02:07 | 2005-08-12 16:58:23 |          5 |
+------------+-----------+---------------------+---------------------+------------+
```

Other Uses of Views

Aside from just simplifying our application's SQL queries, there are also other benefits that views can provide, some of which are only possible by using views.

Restricting Data Access

For example, even though our Employees database contains fields for Position, Home Address, and Home Phone, our query does not allow for these fields to be shown. This means that in the case of a security issue in the application (for example, an SQL injection attack, or even a malicious programmer), there is no risk of disclosing an employee's personal

information.

Row-level Security

We can also define separate views to include a specific WHERE clause for security; for example, if we wanted to restrict a department head's access to only the staff that report to him, we could specify his identity in the view's CREATE statement, and he would then be unable to see any other department's employees, despite them all being in the same table. If this view is writeable and it is defined with the CASCADE clause, this restriction will also apply to writes. This is actually the only way to implement row-level security in MariaDB, so views play an important part in that area as well.

Pre-emptive Optimization

We can also define our views in such a way as to force the use of indexes, so that other, less-experienced developers don't run the risk of running un-optimized queries or JOINs that result in full-table scans and extended locks. Expensive queries, queries that SELECT *, and poorly thought-out JOINs can not only slow down the database entirely, but can cause inserts to fail, clients to time out, and reports to error out. By creating a view that is already optimized and letting users perform their queries on that, you can ensure that they won't cause a significant performance hit unnecessarily.

Abstracting Tables

When we re-engineer our application, we sometimes need to change the database to optimize or accommodate new or removed features. We may, for example, want to normalize our tables when they start getting too large and queries start taking too long. Alternately, we may be installing a new application with different requirements alongside a legacy application. Unfortunately, database redesign will tend to break backwards-compatibility with previous applications, which can cause obvious problems.

Using views, we can change the format of the underlying tables while still presenting the same table format to the legacy application. Thus, an application which demands username, hostname, and access time in string format can access the same data as an application which requires firstname, lastname, user@host, and access time in Unix timestamp format.

Summary

Views are an SQL feature that can provide a lot of versatility in larger applications, and can even simplify smaller applications further. Just as stored procedures can help us abstract out our database logic, views can simplify the way we access data in the database, and can help un-complicate our queries to make application debugging easier and more efficient.

The initial version of this article was copied, with permission, from http://hashmysql.org/wiki/Views_(Basic) on 2012-10-05.

3.3.3.2 Getting Started with Indexes

6.2.5 Joining Tables with JOIN Clauses

In the absence of a more tutorial-level document, here is a simple example of three basic JOIN types, which you can experiment with in order to see what the different joins accomplish:

```
CREATE TABLE t1 ( a INT );
CREATE TABLE t2 ( b INT );
INSERT INTO t1 VALUES (1), (2), (3);
INSERT INTO t2 VALUES (2), (4);
SELECT * FROM t1 INNER JOIN t2 ON t1.a = t2.b;
SELECT * FROM t1 CROSS JOIN t2;
SELECT * FROM t1 LEFT JOIN t2 ON t1.a = t2.b;
SELECT * FROM t2 LEFT JOIN t1 ON t1.a = t2.b;
```

The first two SELECTs are (unfortunately) commonly written with an older form:

```
SELECT * FROM t1, t2 WHERE t1.a = t2.b;
SELECT * FROM t1, t2;
```

What you can see from this is that an **INNER JOIN** produces a result set containing only rows that have a match, in both tables (t1 and t2), for the specified join condition(s).

A **CROSS JOIN** produces a result set in which every row in each table is joined to every row in the other table; this is also called a **cartesian product**. In MariaDB the CROSS keyword can be omitted, as it does nothing. Any JOIN without an ON clause is a CROSS JOIN.

The **LEFT JOIN** is an **outer join**, which produces a result set with all rows from the table on the "left" (t1); the values for the columns in the other table (t2) depend on whether or not a match was found. If no match is found, all columns from that table are set to NULL for that row.

The **RIGHT JOIN** is similar to the LEFT JOIN, though its resultset contains all rows from the right table, and the left table's columns will be filled with NULLs when needed.

JOINs can be concatenated to read results from three or more tables.

Here is the output of the various SELECT statements listed above:

```
SELECT * FROM t1 INNER JOIN t2 ON t1.a = t2.b;
------ ------
| a    | b    |
------ ------
|    2 |    2 |
------ ------
1 row in set (0.00 sec)


SELECT * FROM t1 CROSS JOIN t2;
------ ------
| a    | b    |
------ ------
|    1 |    2 |
|    2 |    2 |
|    3 |    2 |
|    1 |    4 |
|    2 |    4 |
|    3 |    4 |
------ ------
6 rows in set (0.00 sec)


SELECT * FROM t1 LEFT JOIN t2 ON t1.a = t2.b;
------ ------
| a    | b    |
------ ------
|    1 | NULL |
|    2 |    2 |
|    3 | NULL |
------ ------
3 rows in set (0.00 sec)


SELECT * FROM t2 LEFT JOIN t1 ON t1.a = t2.b;
------ ------
| b    | a    |
------ ------
|    2 |    2 |
|    4 | NULL |
------ ------
2 rows in set (0.00 sec)
```

That should give you a bit more understanding of how JOINS work!

Other References

- JOINs Tutorial with Examples
- How to write complex queries

6.2.6 The Essentials of an Index

Imagine you've created a table with the following rows (this is the same table as used in the More Advanced Joins tutorial).

```
+----+------------+-----------+-------------------------+----------------------------+------------
| ID | First_Name | Last_Name | Position                | Home_Address               | Home_Phone
+----+------------+-----------+-------------------------+----------------------------+------------
|  1 | Mustapha   | Mond      | Chief Executive Officer | 692 Promiscuous Plaza      | 326-555-349
|  2 | Henry      | Foster    | Store Manager           | 314 Savage Circle          | 326-555-384
|  3 | Bernard    | Marx      | Cashier                 | 1240 Ambient Avenue        | 326-555-845
|  4 | Lenina     | Crowne    | Cashier                 | 281 Bumblepuppy Boulevard  | 328-555-234
|  5 | Fanny      | Crowne    | Restocker               | 1023 Bokanovsky Lane       | 326-555-632
|  6 | Helmholtz  | Watson    | Janitor                 | 944 Soma Court             | 329-555-247
+----+------------+-----------+-------------------------+----------------------------+------------
```

Now, imagine you've been asked to return the home phone of Fanny Crowne. Without indexes, the only way to do it is to go through every row until you find the matching first name and surname. Now imagine there are millions of records and you can see that, even for a speedy database server, this is highly inefficient.

The answer is to sort the records. If they were stored in alphabetical order by surname, even a human could quickly find a record amongst a large number. But we can't sort the entire record by surname. What if we want to also look a record by ID, or by first name? The answer is to create separate indexes for each column we wish to sort by. An index simply contains the sorted data (such as surname), and a link to the original record.

For example, an index on Last_Name:

```
+-----------+----+
| Last_Name | ID |
+-----------+----+
| Crowne    |  4 |
| Crowne    |  5 |
| Foster    |  2 |
| Marx      |  3 |
| Mond      |  1 |
| Watson    |  6 |
+-----------+----+
```

and an index on Position

```
+-------------------------+----+
| Position                | ID |
+-------------------------+----+
| Cashier                 |  3 |
| Cashier                 |  4 |
| Chief Executive Officer |  1 |
| Janitor                 |  6 |
| Restocker               |  5 |
| Store Manager           |  2 |
+-------------------------+----+
```

would allow you to quickly find the phone numbers of all the cashiers, or the phone number of the employee with the surname Marx, very quickly.

Where possible, you should create an index for each column that you search for records by, to avoid having the server read every row of a table.

See CREATE INDEX and Getting Started with Indexes for more information.

6.2.7 Troubleshooting Connection Issues

Contents

If you are completely new to MariaDB and relational databases, you may want to start with the MariaDB Primer. Also, make sure you understand the connection parameters discussed in the Connecting to MariaDB article.

There are a number of common problems that can occur when connecting to MariaDB.

Server Not Running in Specified Location

If the error you get is something like:

```
mariadb -uname -p -uname -p
ERROR 2002 (HY000): Can't connect to local MySQL server through
  socket '/var/run/mysqld/mysqld.sock' (2 "No such file or directory")
```

or

```
mariadb -uname -p --port=3307 --protocol=tcp
ERROR 2003 (HY000): Can't connect to MySQL server on  'localhost'
  (111 "Connection refused")
```

the server is either not running, or not running on the specified port, socket or pipe. Make sure you are using the correct host, port, pipe, socket and protocol options, or alternatively, see Getting, Installing and Upgrading MariaDB, Starting and Stopping MariaDB or Troubleshooting Installation Issues.

The socket file can be in a non-standard path. In this case, the `socket` option is probably written in the my.cnf file. Check that its value is identical in the [mysqld] and [client] sections; if not, the client will look for a socket in a wrong place.

If unsure where the Unix socket file is running, it's possible to find this out, for example:

```
netstat -ln | grep mysqld
unix  2      [ ACC ]     STREAM     LISTENING     33209505 /var/run/mysqld/mysqld.sock
```

Unable to Connect from a Remote Location

Usually, the MariaDB server does not by default accept connections from a remote client or connecting with tcp and a hostname and has to be configured to permit these.

```
(/my/maria-10.4) ./client/mysql --host=myhost --protocol=tcp --port=3306 test
ERROR 2002 (HY000): Can't connect to MySQL server on 'myhost' (115)
(/my/maria-10.4) telnet myhost 3306
Trying 192.168.0.11...
telnet: connect to address 192.168.0.11: Connection refused
(/my/maria-10.4) perror 115
OS error code 115:  Operation now in progress
```

To solve this, see Configuring MariaDB for Remote Client Access

Authentication Problems

Note that from MariaDB 10.4.3, the unix_socket authentication plugin is enabled by default on Unix-like systems. This uses operating system credentials when connecting to MariaDB via the local Unix socket file. See unix_socket authentication plugin for instructions on connecting and on switching to password-based authentication as well as Authentication from MariaDB 10.4 for an overview of the MariaDB 10.4 changes..

Authentication is granted to a particular username/host combination. `user1'@'localhost'` , for example, is not the same as `user1'@'166.78.144.191'` . See the GRANT article for details on granting permissions.

Passwords are hashed with PASSWORD function. If you have set a password with the SET PASSWORD statement, the PASSWORD function must be used at the same time. For example, `SET PASSWORD FOR 'bob'@'%.loc.gov' = PASSWORD('newpass')` rather than just `SET PASSWORD FOR 'bob'@'%.loc.gov' = 'newpass'` ;

Problems Exporting Query Results

If you can run regular queries, but get an authentication error when running the SELECT ... INTO OUTFILE, SELECT ...

INTO DUMPFILE or LOAD DATA INFILE statements, you do not have permission to write files to the server. This requires the FILE privilege. See the GRANT article.

Access to the Server, but not to a Database

If you can connect to the server, but not to a database, for example:

```
USE test;
ERROR 1044 (42000): Access denied for user 'ian'@'localhost' to database 'test'
```

or can connect to a particular database, but not another, for example `mariadb -uname -p -u name db1` works but not `mariadb -uname -p -u name db2`, you have not been granted permission for the particular database. See the GRANT article.

Option Files and Environment Variables

It's possible that option files or environment variables may be providing incorrect connection parameters. Check the values provided in any option files read by the client you are using (see mysqld Configuration Files and Groups and the documentation for the particular client you're using - see Clients and Utilities).

Option files can usually be suppressed with `no-defaults` option, for example:

```
mariadb-import --no-defaults ...
```

Unable to Connect to a Running Server / Lost root Password

If you are unable to connect to a server, for example because you have lost the root password, you can start the server without using the privilege tables by running the --skip-grant-tables option, which gives users full access to all tables. You can then run FLUSH PRIVILEGES to resume using the grant tables, followed by SET PASSWORD to change the password for an account.

localhost and %

You may have created a user with something like:

```
CREATE USER melisa identified by 'password';
```

This creates a user with the '%' wildcard host.

```
select user,host from mysql.user where user='melisa';
+--------+------+
| user   | host |
+--------+------+
| melisa | %    |
+--------+------+
```

However, you may still be failing to login from localhost. Some setups create anonymous users, including localhost. So the following records exist in the user table:

```
select user,host from mysql.user where user='melisa' or user='';
+--------+-----------+
| user   | host      |
+--------+-----------+
| melisa | %         |
|        | localhost |
+--------+-----------+
```

Since you are connecting from localhost, the anonymous credentials, rather than those for the 'melisa' user, are used. The solution is either to add a new user specific to localhost, or to remove the anonymous localhost user.

6.3 Intermediate MariaDB Articles

These are articles for intermediate level MariaDB developers and administrators.

Database Theory

Articles on hierarchical, network and relational databases.

Starting and Stopping MariaDB

Articles related to starting and stopping MariaDB Server.

6.3.1 Database Theory

Just as perhaps we take movie special effects for granted until we see what state of the art was in previous eras, so we can't fully appreciate the power of relational databases without seeing what preceded them.

Relational databases allow any table to relate to any other table through means of common fields. It is a highly flexible system, and most modern databases are relational.

Introduction to Relational Databases

Brief introduction to the concept of a relational database.

Exploring Early Database Models

Before relational databases there were a number of other models

Understanding the Hierarchical Database Model

The earliest model was the hierarchical database model, resembling an upside-down tree.

Understanding the Network Database Model

A progression from the hierarchical model designed to solve some of its problems

Understanding the Relational Database Model

The relational database model was a huge leap forward from the network data...

Relational Databases: Basic Terms

The relational database model uses certain terms to describe its components

Relational Databases: Table Keys

A key, or index, unlocks access to the tables

Relational Databases: Foreign Keys

Foreign keys are the primary key in a foreign table

Relational Databases: Views

Views are virtual tables

Database Design

Articles about the database design process

Database Normalization

Normalization is a powerful tool for designing databases

ACID: Concurrency Control with Transactions

Ensuring data integrity.

6.3.1.1 Introduction to Relational Databases

Contents

What is a Database?

The easiest way to understand a database is as a collection of related files. Imagine a file (either paper or digital) of sales orders in a shop. Then there's another file of products, containing stock records. To fulfil an order, you'd need to look up the product in the order file and then look up and adjust the stock levels for that particular product in the product file. A database

and the software that controls the database, called a *database management system* (DBMS), helps with this kind of task.

Most databases today are *relational* databases, named such because they deal with tables of data related by a common field. For example, Table 1 below shows the product table, and Table 2 shows the invoice table. As you can see, the relation between the two tables is based on the common field `product_code`. Any two tables can relate to each other simply by having a field in common.

Table 1

Product_code	Description	Price
A416	Nails, box	$0.14
C923	Drawing pins, box	$0.08

Table 2

Invoice_code	Invoice_line	*Product_code*	Quantity
3804	1	A416	10
3804	2	C923	15

Database Terminology

Let's take a closer look at the previous two tables to see how they are organized:

- Each table consists of many *rows* and *columns*.
- Each new row contains data about one single *entity* (such as one product or one order line). This is called a *record*. For example, the first row in Table 1 is a record; it describes the A416 product, which is a box of nails that costs fourteen cents. The terms *row* and *record* are interchangeable.
- Each column (also called an *attribute*) contains one piece of data that relates to the record, called a *tuple*. Examples of attributes are the quantity of an item sold or the price of a product. An attribute, when referring to a database table, is called a *field*. For example, the data in the *Description* column in Table 1 are fields. The terms *attribute* and *field* are interchangeable.

Given this kind of structure, the database gives you a way to manipulate this data: SQL. SQL (structured query language) is a powerful way to search for records or make changes. Almost all DBMSs use SQL, although many have added their own enhancements to it. This means that when you learn SQL while using MariaDB, almost all of it is not specific to MariaDB and can be used with other relational databases as well, such as PostgreSQL, MySQL, Oracle and SQL Server. MariaDB was originally-created as a drop-in replacement to MySQL, so MariaDB and MySQL are particularly close.

6.3.1.2 Exploring Early Database Models

Before the advent of databases, the only way to store data was from unrelated files. Programmers had to go to great lengths to extract the data, and their programs had to perform complex parsing and relating.

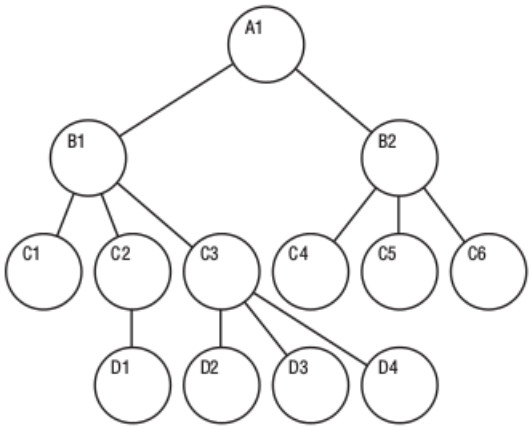
Languages such as Perl, with its powerful regular expressions ideal for processing text, have made the job a lot easier than before; however, accessing the data from files is still a challenging task. Without a standard way to access data, systems are more prone to errors, are slower to develop, and are more difficult to maintain. Data redundancy (where data is duplicated unnecessarily) and poor data integrity (where data is not changed in all locations, leading to wrong or outdated data being supplied) are frequent consequences of the file access method of data storage. For these reasons, database management systems (DBMSs) were developed to provide a standard and reliable way to access and update data. They provide an intermediary layer between the application and the data, and the programmer is able to concentrate on developing the application, rather than worrying about data access issues.

A *database model* is a logical model concerned with how the data is represented. Instead of database designers worrying about the physical storage of data, the database model allows them to look at a higher, more conceptual level, reducing the gap between the real-world problem for which the application is being developed and the technical implementation.

There are a number of database models. The next two articles cover two common models; the hierarchical database model and the network database model. After that comes the one MariaDB, along with most modern DBMSs uses, the relational model.

6.3.1.3 Understanding the Hierarchical Database Model

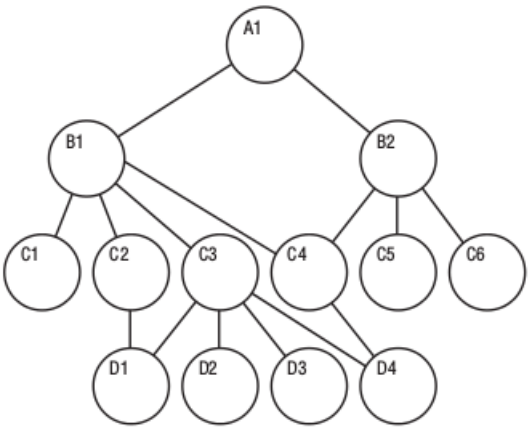
The earliest model was the hierarchical database model, resembling an upside-down tree. Files are related in a parent-child manner, with each parent capable of relating to more than one child, but each child only being related to one parent. Most of you will be familiar with this kind of structure—it's the way most file systems work. There is usually a root, or top-level, directory that contains various other directories and files. Each subdirectory can then contain more files and directories, and so on. Each file or directory can only exist in one directory itself—it only has one parent. As you can see in the image below *A1* is the root directory, and its children are *B1* and *B2*. *B1* is a parent to *C1*, *C2*, and *C3*, which in turn has children of its own.



This model, although being a vast improvement on dealing with unrelated files, has some serious disadvantages. It represents one-to-many relationships well (one parent has many children; for example, one company branch has many employees), but it has problems with many-to-many relationships. Relationships such as that between a product file and an orders file are difficult to implement in a hierarchical model. Specifically, an order can contain many products, and a product can appear in many orders. Also, the hierarchical model is not flexible because adding new relationships can result in wholesale changes to the existing structure, which in turn means all existing applications need to change as well. This is not fun when someone has forgotten a table and wants it added to the system shortly before the project is due to launch! And developing the applications is complex because the programmer needs to know the data structure well in order to traverse the model to access the needed data. As you've seen in the earlier chapters, when accessing data from two related tables, you only need to know the fields you require from those two tables. In the hierarchical model, you'd need to know the entire chain between the two. For example, to relate data from *A1* and *D4*, you'd need to take the route: *A1*, *B1*, *C3* and *D4*.

6.3.1.4 Understanding the Network Database Model

The network database model was a progression from the hierarchical database model and was designed to solve some of that model's problems, specifically the lack of flexibility. Instead of only allowing each child to have one parent, this model allows each child to have multiple parents (it calls the children *members* and the parents *owners*). It addresses the need to model more complex relationships such as the orders/parts many-to-many relationship mentioned in the hierarchical article. As you can see in the figure below, *A1* has two members, *B1* and *B2*. *B1.* is the owner of *C1*, *C2*, *C3* and *C4*. However, in this model, *C4* has two owners, *B1* and *B2*.



Of course, this model has its problems, or everyone would still be using it. It is more difficult to implement and maintain, and, although more flexible than the hierarchical model, it still has flexibility problems, Not all relations can be satisfied by assigning another owner, and the programmer still has to understand the data structure well in order to make the model efficient.

6.3.1.5 Understanding the Relational Database Model

The relational database model was a huge leap forward from the network database model. Instead of relying on a parent-child or owner-member relationship, the relational model allows any file to be related to any other by means of a common field. Suddenly, the complexity of the design was greatly reduced because changes could be made to the database schema without affecting the system's ability to access data. And because access was not by means of paths to and from files, but from a direct relationship between files, new relations between these files could easily be added.

In 1970, when E.F. Codd developed the model, it was thought to be impractical. The increased ease of use comes at a large performance penalty, and the hardware in those days was not able to implement the model. Since then, of course, hardware has taken huge strides to where today, even the simplest computers can run sophisticated relational database management systems.

Relational databases go hand-in-hand with the development of SQL. The simplicity of SQL - where even a novice can learn to perform basic queries in a short period of time - is a large part of the reason for the popularity of the relational model.

The two tables below relate to each other through the *product_code* field. Any two tables can relate to each other simply by creating a field they have in common.

Table 1

Product_code	Description	Price
A416	Nails, box	$0.14
C923	Drawing pins, box	$0.08

Table 2

Invoice_code	Invoice_line	*Product_code*	Quantity
3804	1	A416	10
3804	2	C923	15

6.3.1.6 Relational Databases: Basic Terms

The relational database model uses certain terms to describe its components:

- *Data* are the values kept in the database. On their own, the data means very little. `CA 684-213` is an example of data in a DMV (Division of Motor Vehicles) database.
- *Information* is processed data. For example, `CA 684-213` is the car registration number of a car belonging to Lyndon Manson, in a DMV database.
- A *database* is a collection of *tables*, also called *entities*.
- Each table is made up of *records* (the horizontal rows in the table, also called *tuples*). Each record should be unique, and can be stored in any order in the table.
- Each record is made up of *fields* (which are the vertical columns of the table, also called *attributes*). Basically, a record is one fact (for example, one customer or one sale).
- These fields can be of various *types*. MariaDB has many types (see Data Types for a list), but generally types fall into three kinds: character, numeric, and date. For example, a customer name is a character field, a customer's birthday is a date field, and a customer's number of children is a numeric field.
- The range of allowed values for a field is called the *domain* (also called a *field specification*). For example, a credit card field may be limited to only the values `Mastercard`, `Visa` and `Amex`.
- A field is said to contain a *null* value when it contains nothing at all. Null fields can create complexities in calculations and have consequences for data accuracy. For this reason, many fields are specifically set not to contain null values.
- A *key* accesses specific records in a table.
- An *index* is a mechanism to improve the performance of a database. Indexes are often confused with keys. Indexes are, strictly speaking, part of the physical structure, and keys are part of the logical structure. You'll often see the terms used interchangeably, however, including throughout this Knowledge Base.
- A *view* is a virtual table made up of a subset of the actual tables.
- A *one-to-one* (1:1) relationship is where for each instance of the first table in a relationship, only one instance of the second table exists, An example of this would be a case where a chain of stores carries a vending machine. Each vending machine can only be in one store, and each store carries only one vending machine.

- A *one-to-many* (1:N) relationship is where for each instance of the first table in a relationship, many instances of the second table exist. This is a common kind of relationship. An example is the relationship between a sculptor and their sculptures. Each sculptor may have created many sculptures, but each sculpture has been created by only one sculptor.

- A *many-to-many* (M:N) relationship occurs where, for each instance of the first table, there are many instances of the second table, and for each instance of the second table, there are many instances of the first. For example, a student can have many lecturers, and a lecturer can have many students.

- A *mandatory* relationship exists where for each instance of the first table in a relationship, one or more instances of the second *must* exist. For example, for a music group to exist, there must exist at least one musician in that group.
- An *optional* relationship is where for each instance of the first table in a relationship, there *may* exist instances of the second. For example, if an author can be listed in the database without having written a book (in other words, a prospective author), that relationship is optional. The reverse isn't necessarily true though. For example, for a book to be listed, it must have an author.
- *Data integrity* refers to the condition where data is accurate, valid, and consistent. An example of poor integrity would be if a customer telephone number is stored differently in two different locations. Another is where a course record contains a reference to a lecturer who is no longer present at the school. Database normalization is a technique that assists you to minimize the risk of these sorts of problems.

6.3.1.7 Relational Databases: Table Keys

A *key*, or *index*, as the term itself indicates, unlocks access to the tables. If you know the key, you know how to identify specific records and the relationships between the tables.

Each key consists of one or more fields, or field prefix. The order of columns in an index is significant. Each key has a name.

A *candidate key* is a field, or combination of fields, that uniquely identifies a record. It cannot contain a null value, and its value must be unique. (With duplicates, you would no longer be identifying a unique record).

A *primary key* (PK) is a candidate key that has been designated to identify unique records in the table throughout the database structure.

A *surrogate key* is a primary key that contains unique values automatically generated by the database system - usually, integer numbers. A surrogate key has no meaning, except uniquely identifying a record. This is the most common type of primary key.

For example, see the following table:

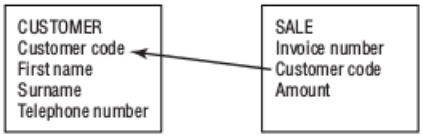
Customer_code	First_name	Surname	Telephone_number
1	Ariane	Edison	448-2143
2	Georgei	Edison	448-2142
3	Abbas	Edison	9231-5312

At first glance, there are two possible candidate keys for this table. Either *customer_code* or a combination of *first_name*, *surname* and *telephone_number* would suffice. It is always better to choose the candidate key with the least number of fields for the primary key, so you would choose *customer_code* in this example (note that it is a surrogate key). Upon reflection, there is also the possibility of the second combination not being unique. The combination of *first_name*, *surname* and *telephone_number* could in theory be duplicated, such as where a father has a son of the same name who is contactable at the same telephone number. This system would have to expressly exclude this possibility for these three fields to be considered for the status of primary key.

There may be many Ariane Edisons, but you avoid confusion by assigning each a unique number. Once a primary key has been created, the remaining candidates are labeled as *alternate keys*.

6.3.1.8 Relational Databases: Foreign Keys

You already know that a relationship between two tables is created by assigning a common field to the two tables (see Relational Databases: Table Keys). This common field must be a primary key to one table. Consider a relationship between a *customer* table and a *sale* table. The relationship is not much good if instead of using the primary key, *customer_code*, in the *sale* table, you use another field that is not unique, such as the customer's first name. You would be unlikely to know for sure which customer made the sale in that case. So, in the table below, *customer_code* is called the *foreign_key* in the *sale* table; in other words, it is the primary key in a foreign table.



Foreign keys allow for something called *referential integrity*. What this means is that if a foreign key contains a value, this value refers to an existing record in the related table. For example, take a look at the tables below:

Lecturer table

Code	First Name	Surname
1	Anne	Cohen
2	Leonard	Clark
3	Vusi	Cave

Course table

Course Title	Lecturer Code
Introduction to Programming	1
Information Systems	2
Systems Software	3

Referential integrity exists here, as all the lecturers in the *course* table exist in the *lecturer* table. However, let's assume Anne Cohen leaves the institution, and you remove her from the lecturer table. In a situation where referential integrity is not enforced, she would be removed from the lecturer table, but not from the course table, as shown below:

Lecturer table

Code	First Name	Surname
2	Leonard	Clark
3	Vusi	Cave

Course table

Course Title	Lecturer Code
Introduction to Programming	1
Information Systems	2
Systems Software	3

Now, when you look up who lectures *Introduction to Programming*, you are sent to a non-existent record. This is called poor data intregrity.

Foreign keys also allow *cascading* deletes and updates. For example, if Anne Cohen leaves, taking the Introduction to Programming Course with her, all trace of her can be removed from both the *lecturer* and *course* table using one statement. The delete *cascades* through the relevant tables, removing all relevant records.

Foreign keys can also contain null values, indicating that no relationship exists.

6.3.1.9 Relational Databases: Views

Views are virtual tables. They are only a structure, and contain no data. Their purpose is to allow a user to see a subset of the actual data. A view can consist of a subset of one table. For example, the *student view*, below, is a subset of the *student table*.

Student View
First name
Surname
Grade

Student Table
Student_id
First name
Surname
Grade
Address
Telephone

This view could be used to allow other students to see their fellow student's marks but not allow them access to personal information.

Alternatively, a view could be a combination of a number of tables, such as the view below:

Student View
First name
Surname
Grade

Student Table
Student_id
First name
Surname
Address
Telephone

Course Table
Course_id
Course description

Grade Table
Student_id
Course_id
Grade

Views are also useful for security. In larger organizations, where many developers may be working on a project, views allow developers to access only the data they need. What they don't need, even if it is in the same table, is hidden from them, safe from being seen or manipulated. It also allows queries to be simplified for developers. For example, without the view, a developer would have to retrieve the fields in the view with the following sort of query

```
SELECT first_name, surname, course_description, grade FROM student, grade, course
  WHERE grade.student_id = student.student_id AND grade.course_id = course.course_id
```

With the view, a developer could do the same with the following:

```
SELECT first_name, surname, course_description, grade FROM student_grade_view
```

Much simpler for a junior developer who hasn't yet learned to do joins, and it's just less hassle for a senior developer too!

For more use cases, see the Views Tutorial.

6.3.1.10 Database Design

Articles about the database design process

Database Design: Overview

Databases exist because of the need to change data into information

Database Lifecycle

Like everything else, databases have a finite lifespan

Database Design Phase 1: Analysis

Defining problems, possibilities, constraints, objectives and agreeing on the scope

Database Design Phase 2: Conceptual Design

Requirements identified in the previous phase are used as the basis to develop the new system

Database Design Phase 2: Logical and Physical Design

After conceptual design, it's time to convert to the logical and physical design

Database Design Phase 3: Implementation

Install the DBMS and load the data

Database Design Phase 4: Testing

Testing performance, security, and integrity of the data

Database Design Phase 5: Operation

Rolling out for everyday use

Database Design Phase 6: Maintenance

Maintaining indexes, optimizing tables, adding and removing users, changing passwords

Database Design Example Phase 1: Analysis

Walking through the database design process with a step-by-step example

Database Design Example Phase 2: Design

Poet Circle logical design and identifying the initial entities

Database Design Example Phase 3: Implementation

Creating the Poet's Circle tables

Database Design Example Phases 4-6: Testing, Operation and Maintenance

Testing, rolling out and maintenance of the Poet's Circle database.

6.3.1.10.1 Database Design: Overview

Databases exist because of the need to change data into information. Data are the raw and unprocessed facts. Information is obtained by processing the data into something useful. For example, millions of names and telephone numbers in a phone book are data. Information is the telephone number of the fire department when your house is burning down.

A database is a large repository of facts, designed in such a way that processing the facts into information is easy. If the phone book was structured in a less convenient way, such as with names and numbers placed in chronological order according to when the numbers were issued, converting the data into information would be much more difficult. Not knowing when the fire department was issued their latest number, you could search for days, and by the time you find the number your house would be a charred pile of ash. So, it's a good thing the phone book was designed as it was.

A database is much more flexible; a similar set of data to what's in a phone book could be ordered by MariaDB according to name, telephone number, address as well as chronologically. But databases are of course more complex, containing many different kinds of information. People, job titles and a company's products can all mingle to provide complex information. But

this complexity makes the design of databases more complex as well. Poor design could make for slow queries, or it could even make certain kinds of information impossible to reach. This section of the Knowledge Base features articles about good database design, specifically:

- The database lifecycle
- Entity-relationship modeling
- Common mistakes in database design
- Real-world example: creating a publishing tracking system
- Concurrency control with transactions

6.3.1.10.2 Database Lifecycle

> This article follows on from Database Design: Overview.

Like everything else, databases have a finite lifespan. They are born in a flush of optimism and make their way through life achieving fame, fortune, and peaceful anonymity, or notoriety as the case may be, before fading out once more. Even the most successful database at some time is replaced by another, more flexible and up-to-date structure, and so begins life anew. Although exact definitions differ, there are generally six stages of the database lifecycle.

Analysis

The analysis phase is where the stakeholders are interviewed and any existing system is examined to identify problems, possibilities and constraints. The objectives and scope of the new system are determined.

Design

The design phase is where a conceptual design is created from the previously determined requirements, and a logical and physical design are created that will ready the database for implementation.

Implementation

The implementation phase is where the database management system (DBMS) is installed, the databases are created, and the data are loaded or imported.

Testing

The testing phase is where the database is tested and fine-tuned, usually in conjunction with the associated applications.

Operation

The operation phase is where the database is working normally, producing information for its users.

Maintenance

The maintenance phase is where changes are made to the database in response to new requirements or changed operating conditions (such as heavier load).

Database development is not independent of systems development, often being one component of the greater systems development process. The stages of systems development basically mirror the stages of a database lifecycle but are a superset. Whereas database design deals with designing the system to store the data, systems design is also concerned with the processes that will impact on the data.

6.3.1.10.3 Database Design Phase 1: Analysis

> This article follows on from Database Lifecycle.

Your existing system can no longer cope. It's time to move on. Perhaps the existing paper system is generating too many errors, or the old Perl script based on flat files can no longer handle the load. Or perhaps an existing news database is struggling under its own popularity and needs an upgrade. This is the stage where the existing system is reviewed.

Depending on the size of the project, the designer may be an individual, responsible for the database implementation and coding, or may be a whole team of analysts. For now, the term *designer* will represent all these possibilities.

The following are the steps in the Analysis Phase.

1. Analyze the organization
2. Define any problems, possibilities or constraints
3. Define the objectives
4. Agree on the scope

When reviewing a system, the designer needs to look at the bigger picture - not just the hardware or existing table structures, but the whole situation of the organization calling for the redesign. For example, a large bank with centralized management would have a different structure and a different way of operating from a decentralized media organization, where anyone can post news onto a website. This may seem trivial, but understanding the organization you're building the database for is vital. to designing a good database for it. The same demands in the bank and media organizations should lead to different designs because the organizations are different. In other words, a solution that was constructed for the bank cannot be unthinkingly implemented for the media organization, even when the situation seems similar. A culture of central control at the bank may mean that news posted on the bank website has to be moderated and authorized by central management, or may require the designer to keep detailed audit trails of who modified what and when. On the flip-side, the media organization may be more laissez-faire and will be happy with news being modified by any authorized editor.

Understanding an organization's culture helps the designers ask the right questions. The bank may not ask for an audit trail, it may simply expect it; and when the time comes to roll out the implementation, the audit trail would need to be patched on, requiring more time and resources.

Once you understand the organization structure, you can question the users of any existing system as to what their problems and needs are, as well as what constraints will exist then. You need to question different role players, as each can add new understanding as to what the database may need. For example, the media organization's marketing department may need detailed statistics about the times of day certain articles are read. You may also be alerted to possible future requirements. Perhaps the editorial department is planning to expand the website, which will give them the staff to cross-link web articles. Keeping this future requirement in mind could make it easier to add the cross-linking feature when the time comes.

Constraints can include hardware ("We have to use our existing database server") or people ("We only have one data capturer on shift at any one time"). Constraints also refer to the limitations on values. For example, a student's grade in a university database may not be able to go beyond 100 percent, or the three categories of seats in a theatre database are small, medium and large.

It is rarely sufficient to rely on one level of management, or an individual, to supply objectives and current problems, except in the smallest of organizations. Top management may be paying for the database design, but lower levels will need to use it, and their input is probably even more important for a successful design.

Of course, although anything is possible given infinite time and money, this is (usually) never forthcoming. Determining scope, and formalizing it, is an important part of the project. If the budget is for one month's work but the ideal solution requires three, the designer must make clear these constraints and agree with the project owners on which facets are not going to be implemented.

6.3.1.10.4 Database Design Phase 2: Conceptual Design

This article follows on from Database Design Phase 1: Analysis.

Contents

The design phase is where the requirements identified in the previous phase are used as the basis to develop the new system. Another way of putting it is that the business understanding of the data structures is converted to a technical understanding. The *what* questions ("What data are required? What are the problems to be solved?") are replaced by the *how* questions ("How will the data be structured? How is the data to be accessed?")

This phase consists of three parts: the conceptual design, the logical design and the physical design. Some methodologies merge the logical design phase into the other two phases. This section is not aimed at being a definitive discussion of database design methodologies (there are whole books written on that!); rather it aims to introduce you to the topic.

Conceptual design

The purpose of the conceptual design phase is to build a conceptual model based upon the previously identified requirements, but closer to the final physical model. A commonly-used conceptual model is called an *entity-relationship* model.

Entities and attributes

Entities are basically people, places, or things you want to keep information about. For example, a library system may have the *book*, *library* and *borrower* entities. Learning to identify what should be an entity, what should be a number of entities, and what should be an *attribute* of an entity takes practice, but there are some good rules of thumb. The following questions can help to identify whether something is an entity:

- Can it vary in number independently of other entities? For example, *person height* is probably not an entity, as it cannot vary in number independently of *person*. It is not fundamental, so it cannot be an entity in this case.
- Is it important enough to warrant the effort of maintaining. For example *customer* may not be important for a small grocery store and will not be an entity in that case, but it will be important for a video store, and will be an entity in that case.
- Is it its own thing that cannot be separated into subcategories? For example, a car-rental agency may have different criteria and storage requirements for different kinds of vehicles. *Vehicle* may not be an entity, as it can be broken up into *car* and *boat*, which are the entities.
- Does it list a type of thing, not an instance? The video game *blow-em-up 6* is not an entity, rather an instance of the *game* entity.
- Does it have many associated facts? If it only contains one attribute, it is unlikely to be an entity. For example, *city* may be an entity in some cases, but if it contains only one attribute, *city name*, it is more likely to be an attribute of another entity, such as *customer*.

The following are examples of entities involving a university with possible attributes in parentheses.

- **Course** (name, code, course prerequisites)
- **Student** (first_name, surname, address, age)
- **Book** (title, ISBN, price, quantity in stock)

An instance of an entity is one particular occurrence of that entity. For example, the student Rudolf Sono is one instance of the student entity. There will probably be many instances. If there is only one instance, consider whether the entity is warranted. The top level usually does not warrant an entity. For example, if the system is being developed for a particular university, *university* will not be an entity because the whole system is for that one university. However, if the system was developed to track legislation at all universities in the country, then *university* would be a valid entity.

Relationships

Entities are related in certain ways. For example, a borrower may belong to a library and can take out books. A book can be found in a particular library. Understanding what you are storing data about, and how the data relate, leads you a large part of the way to a physical implementation in the database.

There are a number of possible relationships:

Mandatory

For each instance of entity A, there must exist one or more instances of entity B. This does not necessarily mean that for each instance of entity B, there must exist one or more instances of entity A. Relationships are optional or mandatory in one direction only, so the A-to-B relationship can be optional, while the B-to-A relationship is mandatory.

Optional

For each instance of entity A, there may or may not exist instances of entity B.

One-to-one (1:1)

This is where for each instance of entity A, there exists one instance of entity B, and vice-versa. If the relationship is optional, there can exist zero or one instances, and if the relationship is mandatory, there exists one and only one instance of the associated entity.

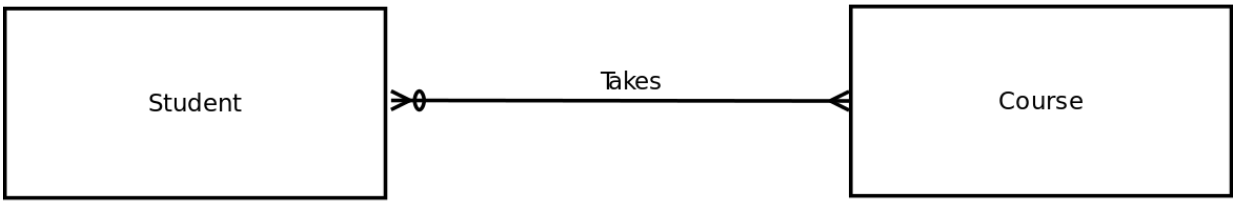
One-to-many (1:M)

For each instance of entity A, many instances of entity B can exist, which for each instance of entity B, only one instance of entity A exists. Again, these can be optional or mandatory relationships.

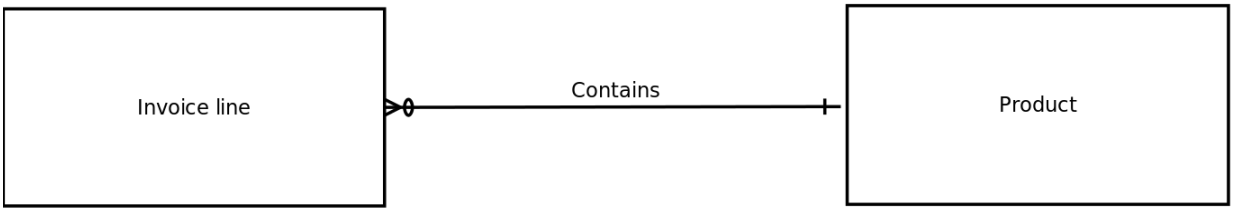
Many-to-many (M:N)

For each instance of entity A, many instances of entity B can exist, and vice versa. These can be optional or mandatory relationships.

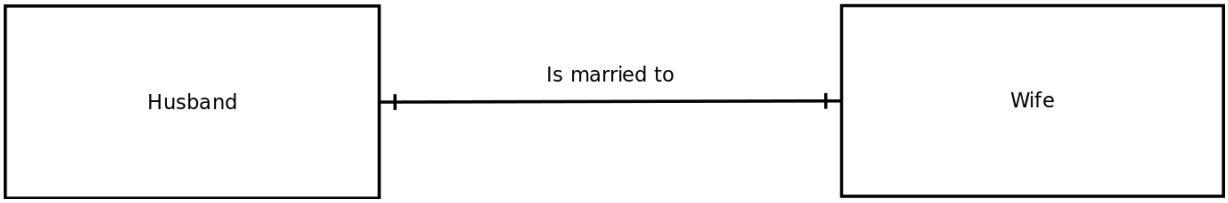
There are numerous ways of showing these relationships. The image below shows *student* and *course* entities. In this case, each student must have registered for at least one course, but a course does not necessarily have to have students registered. The student-to-course relationship is mandatory, and the course-to-student relationship is optional.



The image below shows *invoice_line* and *product* entities. Each invoice line must have at least one product (but no more than one); however each product can appear on many invoice lines, or none at all. The *invoice_line-to-product* relationship is mandatory, while the *product-to-invoice_line* relationship is optional.



The figure below shows husband and wife entities. In this system (others are of course possible), each husband must have one and only one wife, and each wife must have one, and only one, husband. Both relationships are mandatory.



An entity can also have a relationship with itself. Such an entity is called a *recursive entity*. Take a *person* entity. If you're interested in storing data about which people are brothers, you wlll have an "is brother to" relationship. In this case, the relationship is an M:N relationship.

Conversely, a *weak entity* is an entity that cannot exist without another entity. For example, in a school, the *scholar* entity is related to the weak entity *parent/guardian*. Without the scholar, the parent or guardian cannot exist in the system. Weak entities usually derive their primary key, in part or in totality, from the associated entity. *parent/guardian* could take the primary key from the scholar table as part of its primary key (or the entire key if the system only stored one parent/guardian per scholar).

The term *connectivity* refers to the relationship classification.

The term *cardinality* refers to the specific number of instances possible for a relationship. *Cardinality limits* list the minimum and maximum possible occurrences of the associated entity. In the husband and wife example, the cardinality limit is (1,1), and in the case of a student who can take between one and eight courses, the cardinality limits would be represented as (1,8).

Developing an entity-relationship diagram

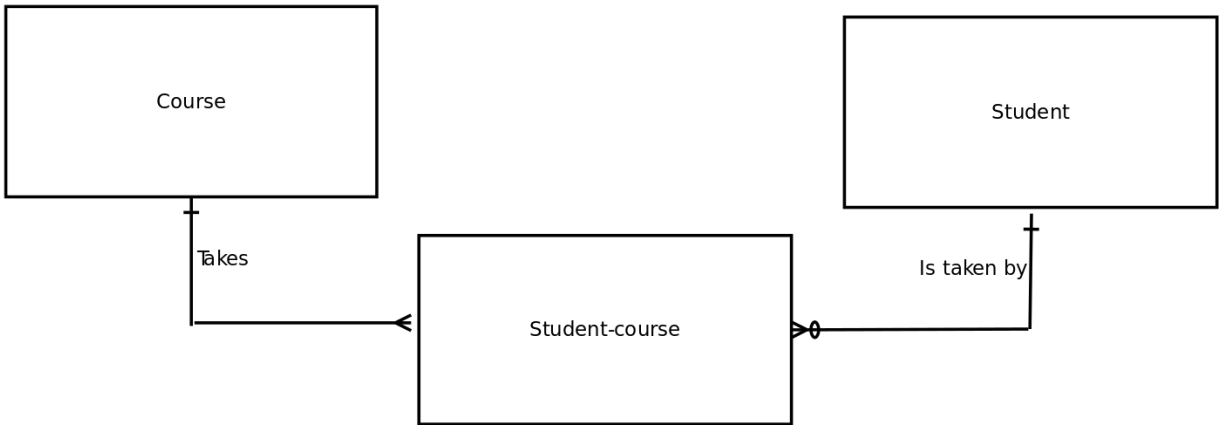
An entity-relationship diagram models how the entities relate to each other. It's made up of multiple relationships, the kind shown in the examples above. In general, these entities go on to become the database tables.

The first step in developing the diagram is to identify all the entities in the system. In the initial stage, it is not necessary to identify the attributes, but this may help to clarify matters if the designer is unsure about some of the entities. Once the entities are listed, relationships between these entities are identified and modeled according to their type: one-to-many, optional and so on. There are many software packages that can assist in drawing an entity-relationship diagram, but any graphical package should suffice.

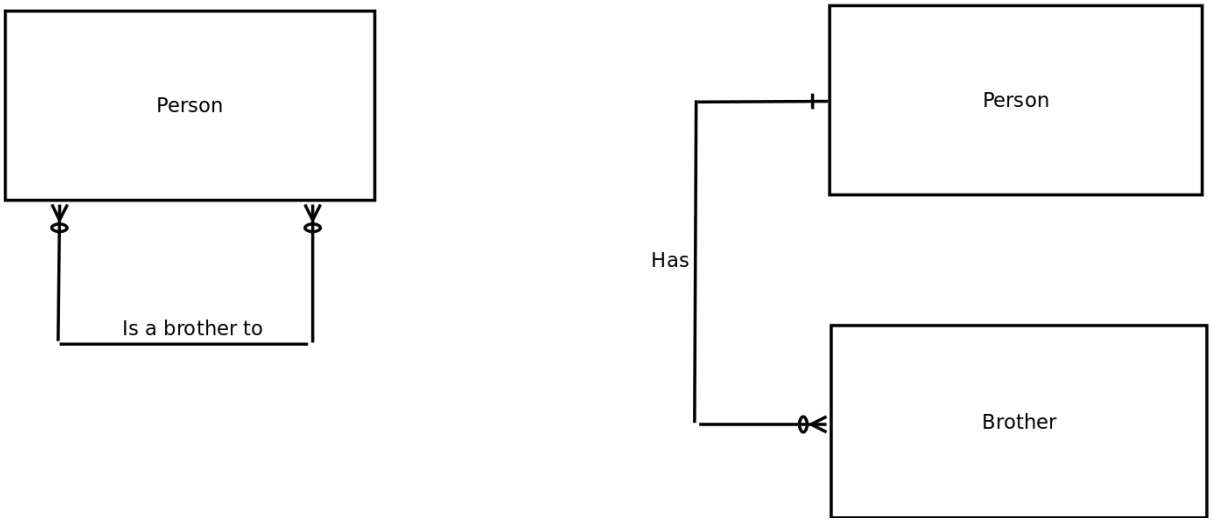
Once the initial entity-relationship diagram has been drawn, it is often shown to the stakeholders. Entity-relationship diagrams are easy for non-technical people to understand, especially when guided through the process. This can help identify any errors that have crept in. Part of the reason for modeling is that models are much easier to understand than pages of text, and they are much more likely to be viewed by stakeholders, which reduces the chances of errors slipping through to the next stage, when they may be more difficult to fix.

It is important to remember that there is no one right or wrong answer. The more complex the situation, the more possible designs that will work. Database design is an acquired skill, though, and more experienced designers will have a good idea of what works and of possible problems at a later stage, having gone through the process before.

Once the diagram has been approved, the next stage is to replace many-to-many relationships with two one-to-many relationships. A DBMS cannot directly implement many-to-many relationships, so they are decomposed into two smaller relationships. To achieve this, you have to create an *intersection*, or *composite* entity type. Because intersection entities are less "real-world" than ordinary entities, they are sometimes difficult to name. In this case, you can name them according to the two entities being intersected. For example, you can intersect the many-to-many relationship between *student* and *course* by a *student-course* entity.



The same applies even if the entity is recursive. The person entity that has an M:N relationship "is brother to" also needs an intersection entity. You can come up with a good name for the intersection entity in this case: *brother*. This entity would contain two fields, one for each person of the brother relationship — in other words, the primary key of the first brother and the primary key of the other brother.



6.3.1.10.5 Database Design Phase 2: Logical and Physical Design

This article follows on from Database Design Phase 2: Conceptual Design .

Contents

Overview

Once the conceptual design is finalized, it's time to convert this to the logical and physical design. Usually, the DBMS is chosen at this stage, depending on the requirements and complexity of the data structures. Strictly speaking, the logical design and the physical design are two separate stages, but are often merged into one. They overlap because most current DBMSs (including MariaDB) match logical records to physical records on disk on a 1:1 basis.

Each entity will become a database table, and each attribute will become a field of this table. Foreign keys can be created if the DBMS supports them and the designer decides to implement them. If the relationship is mandatory, the foreign key must be defined as *NOT NULL*, and if it's optional, the foreign key can allow nulls. For example, because of the invoice line-to-product relationship in the previous example, the product code field is a foreign key in the invoice to line table. Because the invoice line must contain a product, the field must be defined as *NOT NULL*. The default MariaDB storage engine, XtraDB, does support foreign key constraints, but some storage engines, such as MyISAM do not. The *ON DELETE CASCADE* and *ON DELETE RESTRICT* clauses are used to support foreign keys. *ON DELETE RESTRICT* means that records cannot be deleted unless all records associated with the foreign key are also deleted. In the invoice line-to-product case, *ON DELETE RESTRICT* in the invoice line table means that if a product is deleted, the deletion will not take place unless all associated invoice lines with that product are deleted as well. This avoids the possibility of an invoice line existing that points to a non-existent product. *ON DELETE CASCADE* achieves a similar effect, but more automatically (and more dangerously!). If the foreign key was declared with *ON CASCADE DELETE*, associated invoice lines would automatically be deleted if a product was deleted. *ON UPDATE CASCADE* is similar to *ON DELETE CASCADE* in that all foreign key references to a primary key are updated when the primary key is updated.

Normalizing your tables is an important step when designing the database. This process helps avoid data redundancy and improves your data integrity.

Novice database designers usually make a number of common errors. If you've carefully identified entities and attributes and you've normalized your data, you'll probably avoid these errors.

Common errors

- Keep unrelated data in different tables. People who are used to using spreadsheets often make this mistake because they are used to seeing all their data in one two-dimensional table. A relational database is much more powerful; don't 'hamstring' it in this way.
- Don't store values you can calculate. Let's say you're interested three numbers: /A, B and the product of A and B (A*B). Don't store the product. It wastes space and can easily be calculated if you need it. And it makes your database more difficult to maintain: If you change A, you also have to change all of the products as well. Why waste your database's efforts on something you can calculate when you need it?
- Does your design cater to all the conditions you've analyzed? In the heady rush of creating an entity-relationship diagram, you can easily overlook a condition. Entity-relationship diagrams are usually better at getting stakeholders to spot an incorrect rule than spot a missing one. The business logic is as important as the database logic and is more likely to be overlooked. For example, it's easy to spot that you cannot have a sale without an associated customer, but have you built in that the customer cannot be approved for a sale of less than $500 if another approved customer has not recommended them?
- Are your attributes, which are about to become field names, well chosen? Fields should be clearly named. For example, if you use *f1* and *f2* instead of *surname* and *first_name*, the time saved in less typing will be lost in looking up the correct spelling of the field, or in mistakes where a developer thought *f1* was the first name, and *f2* the surname. Similarly, try to avoid the same names for different fields. If six tables have a primary key of *code*, you're making life unnecessarily difficult. Rather, use more descriptive terms, such as *sales_code* or *customer_code*.
- Don't create too many relationships. Almost every table in a system can be related by some stretch of the imagination, but there's no need to do this. For example, a tennis player belongs to a sports club. A sports club belongs to a region. The tennis players then also belong to a region, but this relationship can be derived through the sports club, so there's no need to add another foreign key (except to achieve performance benefits for certain kinds of queries). Normalizing can help you avoid this sort of problem (and even when you're trying to optimize for speed, it's usually better to normalize and then consciously denormalize rather than not normalize at all).
- Conversely, have you catered to all relations? Do all relations from your entity-relationship diagram appear as common fields in your table structures? Have you covered all relations? Are all many-to-many relationships broken up into two one-to-many relationships, with an intersection entity?
- Have you listed all constraints? Constraints include a gender that can only be *m* or *f*, ages of schoolchildren that cannot exceed twenty, or email addresses that need to have an @ sign and at least one period (.; don't take these limits for granted. At some stage the system you will need to implement them, and you're either going to forget to do so, or have to go back and gather more data if you don't list these up front.
- Are you planning to store too much data? Should a customer be asked to supply their eye color, favorite kind of fish, and names of their grandparents if they are simply trying to register for an online newsletter? Sometimes stakeholders want too much information from their customers. If the user is outside the organization, they may not have a voice in the design process, but they should always be thought of foremost. Consider also the difficulty and time taken to capture all the data. If a telephone operator needs to take all this information down before making a sale, imagine how much slower they will be. Also consider the impact data has on database speed. Larger tables are generally slower to access, and unnecessary BLOB, TEXT and VARCHAR fields lead to record and table fragmentation.
- Have you combined fields that should be separate? Combining first name and surname into one field is a common beginner mistake. Later you'll realise that sorting names alphabetically is tricky if you've stored them as *John Ellis* and *Alfred Ntombela*. Keep distinct data discrete.
- Has every table got a primary key? There had better be a good reason for leaving out a primary key. How else are you going to identify a unique record quickly? Consider that an index speeds up access time tremendously, and when kept small it adds very little overhead. Also, it's usually better to create a new field for the primary key rather than take

existing fields. First name and surname may be unique in your current database, but they may not always be. Creating a system-defined primary key ensures it will always be unique.

- Give some thought to your other indexes. What fields are likely to be used in this condition to access the table? You can always create more fields later when you test the system, but add any you think you need at this stage.
- Are your foreign keys correctly placed? In a one-to-many relationship, the foreign key appears in the *many* table, and the associated primary key in the *one* table. Mixing these up can cause errors.
- Do you ensure referential integrity? Foreign keys should not relate to a primary key in another table that no longer exists.
- Have you covered all character sets you may need? German letters, for example, have an expanded character set, and if the database is to cater for German users it will have to take this into account. Similarly, dates and currency formats should be carefully considered if the system is to be international
- Is your security sufficient? Remember to assign the minimum permissions you can. Do not allow anyone to view a table if they do not need to do so. Allowing malicious users view data, even if they cannot change it, is often the first step in for an attacker.

6.3.1.10.6 Database Design Phase 3: Implementation

This article follows on from Database Design Phase 2: Logical and Physical Design.

The implementation phase is where you install the DBMS on the required hardware, optimize the database to run best on that hardware and software platform, and create the database and load the data. The initial data could be either new data captured directly or existing data imported from a MariaDB database or another DBMS. You also establish database security in this phase and give the various users that you've identified access applicable to their requirements. Finally, you also initiate backup plans in this phase.

The following are steps in the implementation phase:

1. Install the DBMS.
2. Tune the setup variables according to the hardware, software and usage conditions.
3. Create the database and tables.
4. Load the data.
5. Set up the users and security.
6. Implement the backup regime.

6.3.1.10.7 Database Design Phase 4: Testing

This article follows on from Database Design Phase 3: Implementation.

The testing phase is where the performance, security, and integrity of the data are tested. Usually this will occur in conjunctions with the applications that have been developed. You test the performance under various loads conditions to see how the database handles multiple concurrent connections or high volumes of updating and reading. Are the reports generated quickly enough? For example, an application designed with the old MyISAM storage engine may prove to be too slow because the impact of the updates was underestimated. The storage engine may have to be changed to XtraDB in response.

Data integrity also needs to be tested, as the application may have logical flaws that result in transactions being lost or other inaccuracies. Further, security needs to be tested to ensure that users can access and change only the data they should.

The logical or physical designs may have to be modified. Perhaps new indexes are required (which the tester may discover after careful use of MariaDB's EXPLAIN statement, for example).

The testing and fine-tuning process is an iterative one, with multiple tests performed and changes implemented.

The following are the steps in the testing phase:

1. Test the performance
2. Test the security
3. Test the data integrity
4. Fine-tune the parameters or modify the logical or physical designs in response to the tests.

6.3.1.10.8 Database Design Phase 5: Operation

This article follows on from Database Design Phase 4: Testing.

The operation phase takes place when the testing is complete and the database is ready to be rolled out for everyday use. The users of the system begin to operate the system, load data, read reports and so on. Inevitably, problems come to light. The designers need to carefully manage the database's scope at this stage, as users may expect all their desires to be pandered to. Poor database designers may find themselves extending the project well beyond their initial time estimate, and the situation may also become unpleasant if the scope has not been clearly defined and agreed upon. Project owners will feel wronged if their needs are not met, and the database designers will feel overworked and underpaid. Even when scope has been well managed, there will always be new requirements, These then lead to the next stage.

There are numerous strategies for implementing a rollout. The low-key approach often works well, where the relatively low number of users in the early stage make bug fixing easy. Hugely publicized rollouts often end with egg on the stakeholder's faces, as the best testers of all, the users, invariably find unforeseen bugs, which is best done away from the spotlight. Alternatively, rollouts can occur in a distributed manner, where a pilot branch or office is selected, and when the system has proven its stability, it's rolled out to the remaining branches.

The following are the steps in the operation phase:

1. Hand over operation of the database to the users.
2. Make any final changes based on the problems discovered by users.

6.3.1.10.9 Database Design Phase 6: Maintenance

This article follows on from Database Design Phase 5: Operation.

The database maintenance phase incorporates general maintenance, such as maintaining the indexes, optimizing the tables, adding and removing users, and changing passwords, as well as backups and restoration of backups in case of a failure. New requirements also start to be requested, and this may result in new fields, or new tables, being created.

As the new system and organization changes, the existing database becomes less and less sufficient to meet the organization's needs. For example, the media organization may be amalgamated with media bodies from other countries, requiring integration of many data sources, or the volumes and staff may expand (or reduce) dramatically. Eventually, there comes a time, whether it's 10 months after completion or 10 years, when the database system needs to be replaced. The maintenance of the existing database begins to drain more and more resources, and the effort to create a new design is matched by the current maintenance effort. As this point, the database is coming to the end of its life, and a new project begins life in the Analysis phase.

The following are the steps in the maintenance phase:

1. Maintain the indexes
2. Maintain the tables
3. Maintain the users
4. Change passwords
5. Backup
6. Restore backups
7. Change the design to meet new requirements

6.3.1.10.10 Database Design Example Phase 1: Analysis

This article follows on from Database Design Phase 6: Maintenance.

Real-world example: creating a publishing tracking system

Now let's walk through the database design process with a step-by-step example. The Poet's Circle is a publisher that publishes poetry and poetry anthologies. It is keen to develop a new system that tracks poets, poems, anthologies and sales. The following sections show the steps taken from the initial analysis to the final, working database.

Poet's circle database phase 1: analysis

The following information is gleaned from speaking to the various stakeholders at Poet's Circle. They want to develop a database system to track the poets they have recorded, the poems they write, the publications they appear in, as well as the sales to customers that these publications make.

The designer asks various questions to get more detailed information, such as "What is a poet, as far as the system goes? Does Poet's Circle keep track of poets even if they haven't written or published poems? Are publications recorded even before there are any associated poems? Does a publication consist of one poem, or many? Are potential customer's details recorded?" The following summarizes the responses in our example:

- Poet's Circle is a publisher that bases its choices of publications on an active poetry community on its website. If enough of the community wants a poem published, Poet's Circle will do so.
- A poet can be anybody who wants to be a poet, not necessarily someone who has a poem captured in the system or someone who has even written a poem.
- Poems can be submitted through a web interface, by email or on paper.
- All captured poems are written by an associated poet, whose details are already in the system. There can be no poems submitted and stored without a full set of details of the poet.
- A publication can be a single poem, a poetry anthology, or a work of literary criticism.
- Customers can sign up through a web interface and may order publications at that point in time, or express interest in receiving updates for possible later purchases.
- Sales of publications are made to customers whose details are stored in the system. There are no anonymous sales.
- A single sale can be for one publication, but many publications can also be purchased at the same time. If more than one customer is involved in this sale, Poet's Circle treats it as more than one sale. Each customer has their own sale.
- Not all publications make sales — some may be special editions, and others simply never sell any copies.

6.3.1.10.11 Database Design Example Phase 2: Design

> This article follows on from Database Design Example Phase 1: Analysis.

Based on the provided information, you can begin your logical design and should be able to identify the initial entities:

- Poet
- Poem
- Publication
- Sale
- Customer

The Poet's Circle is not an entity, or even of instance an a *publisher* entity. Only if the system were developed for many publishers would *publisher* be a valid entity.

Neither *website* nor *poetry community* are entities. There is only one website, and anyway, a website is merely a means of producing data to populate the database. There is also only one poetry community as far as this system is concerned, and there is not much you'd want to store about it.

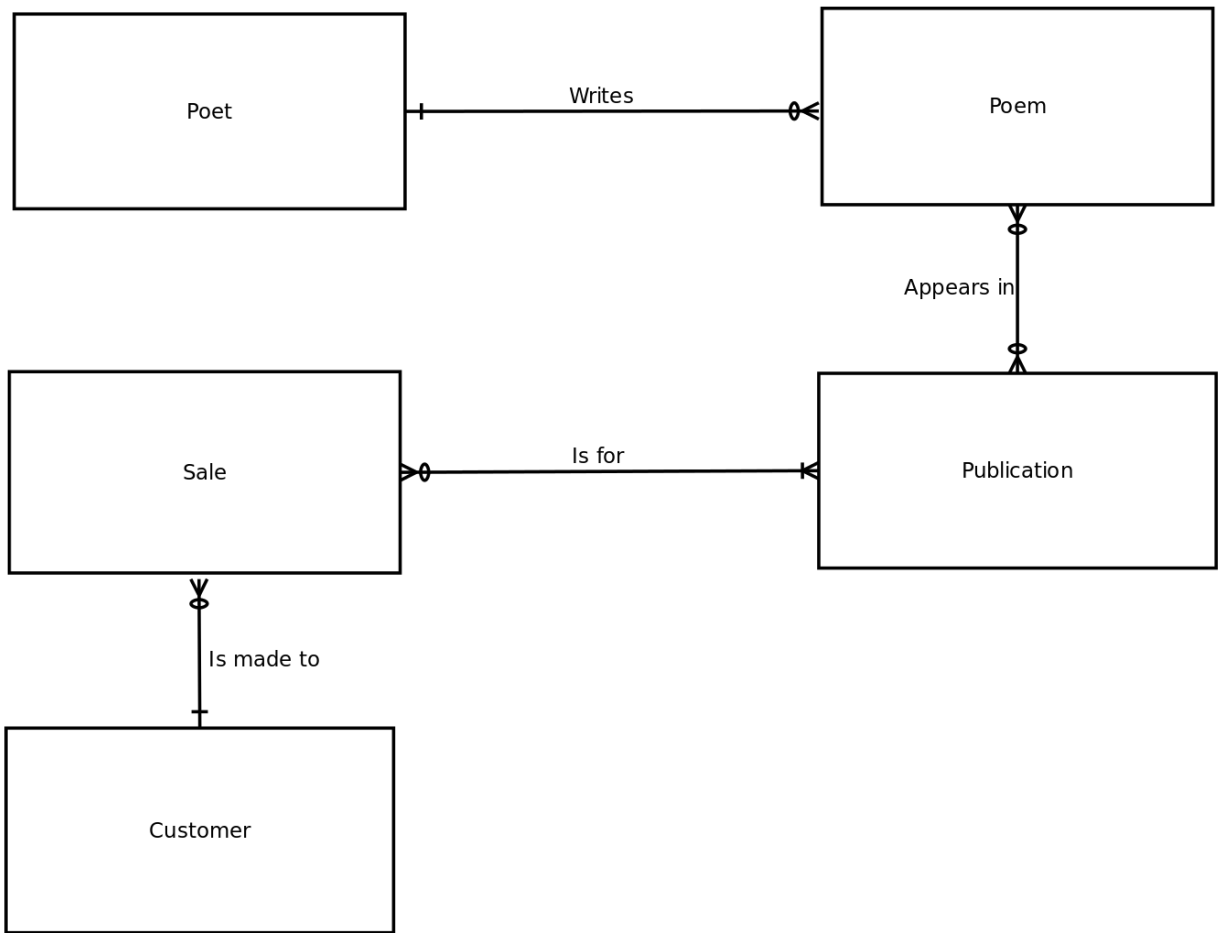
Next, you need to determine the relationship between these entities. You can identify the following:

- A poet can write many poems. The analysis identified the fact that a poet can be stored in the system even if there are no associated poems. Poems may be captured at a later point in time, or the poet may still be a potential poet. Conversely, many poets could conceivably write a poem, though the poem must have been written by at least one poet.
- A publication may contain many poems (an anthology) or just one. It can also contain no poems (poetry criticism for example). A poem may or may not appear in a publication.
- A sale must be for at least one publication, but it may be for many. A publication may or may not have made any sales.
- A customer may be made for many sales, or none at all. A sale is only made for one and only one customer.

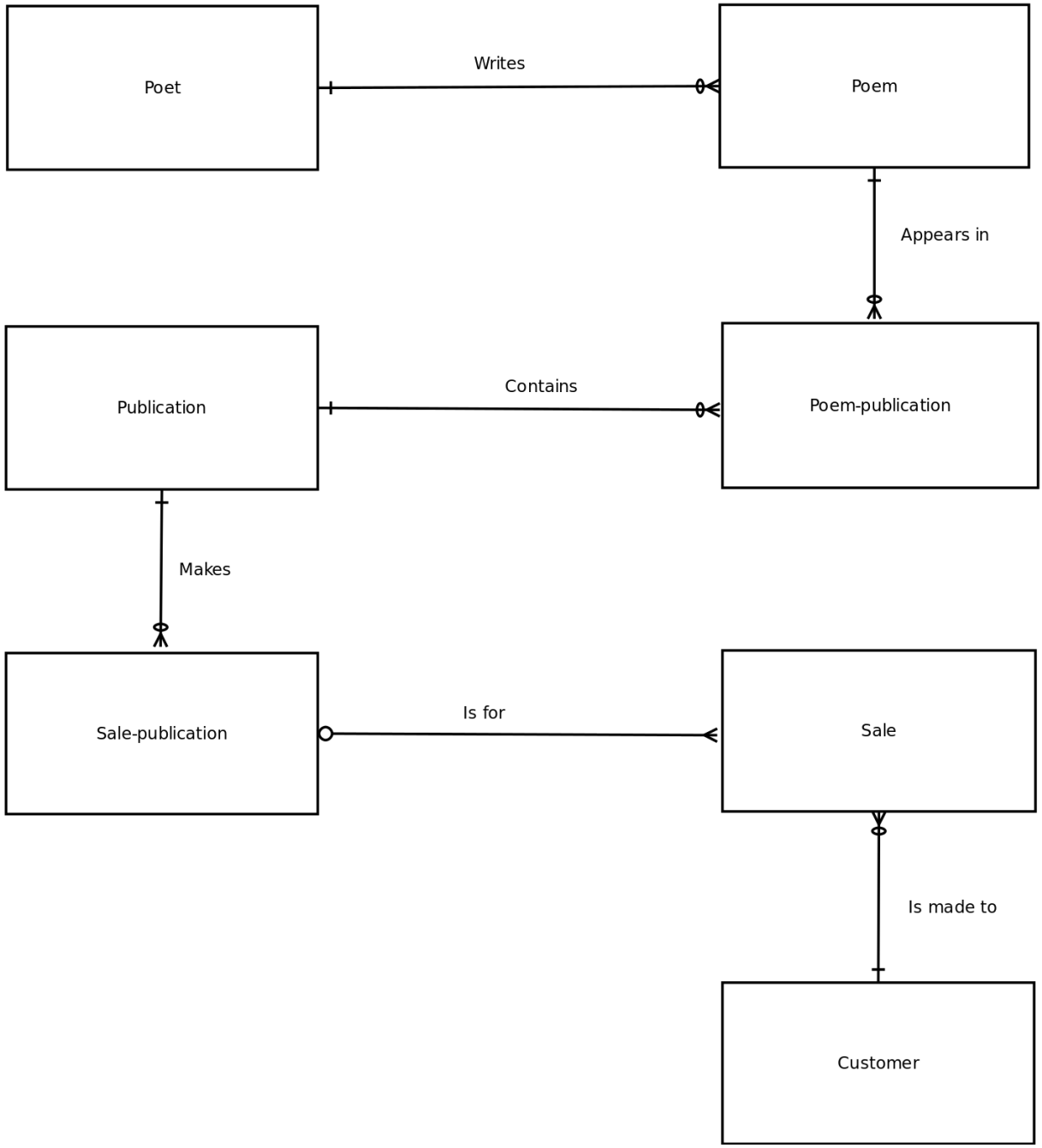
You can identify the following attributes:

- **Poet**: first name, surname, address, email address
- **Poem**: poem title, poem contents
- **Publication**: title, price
- **Sales**: date, amount
- **Customer**: first name, surname, address, email address

Based on these entities and relationships, you can construct the entity-relationship diagram shown below:



There are two many-to-many relationships in the figure above. These need to be converted into one-to-many relationships before you can implement them in a DBMS. After doing so, the intersection entities *poem-publication* and *sale-publication* are created.



Now, to begin the logical and physical design, you need to add attributes that can create the relationship between the entities and specify primary keys. You do what's usually best, and create new, unique, primary keys. The following tables show the structures for the tables created from each of the entities:

Poet table

Field	Definition
poet code	primary key, integer
first name	character (30)
surname	character (40)
address	character (100)
postcode	character (20)
email address	character (254)

Poem table

Field	Definition
poem code	primary key, integer

poem title	character(50)
poem contents	text
poet code	foreign key, integer

Poem-publication table

Field	Definition
poem code	joint primary key, foreign key, integer
publication code	joint primary key, foreign key, integer

Publication table

Field	Definition
publication code	primary key, integer
title	character(100)
price	numeric(5.2)

Sale-publication table

Field	Definition
sale code	joint primary key, foreign key, integer
publication code	joint primary key, foreign key, integer

Sale table

Field	Definition
sale code	primary key, integer
date	date
amount	numeric(10.2)
customer code	foreign key, integer

Customer table

Field	Definition
customer code	primary key, integer
first name	character (30)
surname	character (40)
address	character (100)
postcode	character (20)
email address	character (254)

MariaDB will have no problem with this, and is selected as the DBMS. Existing hardware and operating system platforms are also selected. The following section looks at the implementation and the SQL statements used to create the MariaDB tables.

6.3.1.10.12 Database Design Example Phase 3: Implementation

This article follows on from Database Design Example Phase 2: Design .

With the design complete, it's time to install MariaDB and run the CREATE statements, as follows:

```
CREATE DATABASE poets_circle;

CREATE TABLE poet (
  poet_code INT NOT NULL,
  first_name VARCHAR(30),
  surname VARCHAR(40),
  address VARCHAR(100),
  postcode VARCHAR(20),
  email VARCHAR(254),
  PRIMARY KEY(poet_code)
);

CREATE TABLE poem(
  poem_code INT NOT NULL,
  title VARCHAR(50),
  contents TEXT,
  poet_code INT NOT NULL,
  PRIMARY KEY(poem_code),
  INDEX(poet_code),
  FOREIGN KEY(poet_code) REFERENCES poet(poet_code)
);

CREATE TABLE publication(
  publication_code INT NOT NULL,
  title VARCHAR(100),
  price MEDIUMINT UNSIGNED,
  PRIMARY KEY(publication_code)
);

CREATE TABLE poem_publication(
  poem_code INT NOT NULL,
  publication_code INT NOT NULL,
  PRIMARY KEY(poem_code, publication_code),
  INDEX(publication_code),
  FOREIGN KEY(poem_code) REFERENCES poem(poem_code),
  FOREIGN KEY(publication_code) REFERENCES publication(publication_code)
);

CREATE TABLE sales_publication(
  sales_code INT NOT NULL,
  publication_code INT NOT NULL,
  PRIMARY KEY(sales_code, publication_code)
);

CREATE TABLE customer(
  customer_code INT NOT NULL,
  first_name VARCHAR(30),
  surname VARCHAR(40),
  address VARCHAR(100),
  postcode VARCHAR(20),
  email VARCHAR(254),
  PRIMARY KEY(customer_code)
);

CREATE TABLE sale(
  sale_code INT NOT NULL,
  sale_date DATE,
  amount INT UNSIGNED,
  customer_code INT NOT NULL,
  PRIMARY KEY(sale_code),
  INDEX(customer_code),
  FOREIGN KEY(customer_code) REFERENCES customer(customer_code)
);
```

6.3.1.10.13 Database Design Example Phases 4-6: Testing, Operation and Maintenance

This article follows on from Database Design Example Phase 3: Implementation.

Once the database is ready the application programs have been rolled out, it's time for the testing to begin. While the other phases of the database lifecycle can occur reasonably independently of the systems development process, part of the testing phase is how all the components run together.

Load testing may indicate that MariaDB has not been set up to handle the expected 600 concurrent connections, and the configuration file needs to be changed. Other tests may indicate that in certain circumstances, duplicate key errors are received, as the locking mechanism is not uniformly implemented, and the application does not handle locking correctly. The application needs to be fixed. Backups also need to be tested, as well as the ability to smoothly restore from backup with a minimum of downtime.

> Testing is one of the most neglected and critical phases. A designer or manager who does not properly account for testing is simply incompetent. No matter how tiny your system, make sure you allocate time for thorough testing, and time for fixing the inevitable bugs.

Once testing is complete, the system can be rolled out. You decide on a low-key rollout and give a few selected poets access to the website to upload their poems. You discover other problems. Some poets upload poems using character sets you haven't catered for, and you need to make a few tweaks to ensure these are handled correctly.

Soon enough, the system is rolled out completely. Maintenance, though, is a never-ending task, and with the immense popularity of the system, and with large numbers of updates and deletes, the system tends to become fragmented. The administrator regularly needs to take care of this, and, of course, the inevitable disk failure leads to an all-night restore session, and much thankfulness for the ease of use of mariadb-dump.

6.3.1.11 Database Normalization

This section introduces you to a powerful tool for designing databases: normalization.

Database Normalization Overview

A sample system going through the process of normalization

Database Normalization: 1st Normal Form

Moving from unnormalized to 1st normal form

Database Normalization: 2nd Normal Form

From 1st to 2nd normal form

Database Normalization: 3rd Normal Form

From 2nd to 3rd normal form

Database Normalization: Boyce-Codd Normal Form

Beyond 3rd normal form with Boyce-Codd normal form

Database Normalization: 4th Normal Form

Beyond Boyce-Codd normal form with 4th normal form

Database Normalization: 5th Normal Form and Beyond

Normal forms beyond 4th are mainly of academic interest

Understanding Denormalization

Denormalization is the process of reversing the transformations made during...

6.3.1.11.1 Database Normalization Overview

Contents

Developed in the 1970's by E.F. Codd, database normalization is standard requirement of many database designs. Normalization is a technique that can help you avoid data anomalies and other problems with managing your data. It consists of transforming a table through various stages: *1st normal form*, *2nd normal form*, *3rd normal form*, and beyond.

It aims to:

- Eliminate data redundancies (and therefore use less space)
- Make it easier to make changes to data, and avoid anomalies when doing so
- Make referential integrity constraints easier to enforce
- Produce an easily comprehensible structure that closely resembles the situation the data represents, and allows for growth

Let's begin by creating a sample set of data. You'll walk through the process of normalization first without worrying about the theory, to get an understanding of the reasons you'd want to normalize. Once you've done that, we'll introduce the theory and the various stages of normalization, which will make the whole process described below much simpler the next time you do it.

Imagine you are working on a system that records plants placed in certain locations, and the soil descriptions associated with them.

The location:

- Location Code: 11
- Location name: Kirstenbosch Gardens

contains the following three plants:

- Plant code: 431
- Plant name: Leucadendron
- Soil category: A
- Soil description: Sandstone

- Plant code: 446
- Plant name: Protea
- Soil category: B
- Soil description: Sandstone/Limestone

- Plant code: 482
- Plant name: Erica
- Soil category: C
- Soil description: Limestone

The location:

- Location Code: 12
- Location name: Karbonkelberg Mountains

contains the following two plants:

- Plant code: 431
- Plant name: Leucadendron
- Soil category: A
- Soil description: Sandstone

- Plant code: 449
- Plant name: Restio
- Soil category: B
- Soil description: Sandstone/Limestone

Tables in a relational database are in a grid, or table format (MariaDB, like most modern DBMSs is a relational database), so let's rearrange this data in the form of a tabular report:

Plant data displayed as a tabular report

Location code	Location name	Plant code	Plant name	Soil category	Soil description
11	Kirstenbosch Gardens	431	Leaucadendron	A	Sandstone
		446	Protea	B	Sandstone/limestone
		482	Erica	C	Limestone
12	Karbonkelberg Mountains	431	Leucadendron	A	Sandstone
		449	Restio	B	Sandstone/limestone

How are you to enter this data into a table in the database? You could try to copy the layout you see above, resulting in a table something like the below. The null fields reflect the fields where no data was entered.

Trying to create a table with plant data

Location code	Location name	Plant code	Plant name	Soil category	Soil description
11	Kirstenbosch Gardens	431	Leucadendron	A	Sandstone
NULL	NULL	446	Protea	B	Sandstone/limestone
NULL	NULL	482	Erica	C	Limestone
1 2	Karbonkelberg Mountains	431	Leucadendron	A	Sandstone
NULL	NULL	449	Restio	B	Sandstone/limestone

This table is not much use, though. The first three rows are actually a group, all belonging to the same location. If you take the third row by itself, the data is incomplete, as you cannot tell the location the Erica is to be found. Also, with the table as it stands, you cannot use the location code, or any other fields, as a primary key (remember, a primary key is a field, or list of fields, that uniquely identify one record). There is not much use in having a table if you can't uniquely identify each record in it.

So, the solution is to make sure each table row can stand alone, and is not part of a group, or set. To achieve this, remove the groups, or sets of data, and make each row a complete record in its own right, which results in the table below.

Each record stands alone

Location code	Location name	Plant code	Plant name	Soil category	Soil description
11	Kirstenbosch Gardens	431	Leucadendron	A	Sandstone
11	Kirstenbosch Gardens	446	Protea	B	Sandstone/limestone
11	Kirstenbosch Gardens	482	Erica	C	Limestone
12	Karbonkelberg Mountains	431	Leucadendron	A	Sandstone
12	Karbonkelberg Mountains	449	Restio	B	Sandstone/limestone

Notice that the location code cannot be a primary key on its own. It does not uniquely identify a row of data. So, the primary key must be a combination of *location code* and *plant code*. Together these two fields uniquely identify a row of data. Think about it. You would never add the same plant type more than once to a particular location. Once you have the fact that it occurs in that location, that's enough. If you want to record quantities of plants at a location - for this example, you're just interested in the spread of plants - you don't need to add an entire new record for each plant; rather, just add a quantity field. If for some reason you would be adding more than one instance of a plant/location combination, you'd need to add something else to the key to make it unique.

So, now the data can go in table format, but there are still problems with it. The table stores the information that code *11* refers to *Kirstenbosch Gardens* three times! Besides the waste of space, there is another serious problem. Look carefully at the data below.

Data anomaly

Location code	Location name	Plant code	Plant name	Soil category	Soil description
11	Kirstenbosch Gardens	431	Leucadendron	A	Sandstone
11	Kirstenbosh Gardens	446	Protea	B	Sandstone/limestone

11	Kirstenbosch Gardens	482	Erica	C	Limestone
12	Karbonkelberg Mountains	431	Leucadendron	A	Sandstone
12	Karbonkelberg Mountains	449	Restio	B	Sandstone/limestone

Did you notice anything strange? Congratulations if you did! *Kirstenbosch* is misspelled in the second record. Now imagine trying to spot this error in a table with thousands of records! By using the structure in the table above, the chances of data anomalies increases dramatically.

The solution is simple. You remove the duplication. What you are doing is looking for partial dependencies - in other words, fields that are dependent on a part of a key and not the entire key. Because both the location code and the plant code make up the key, you look for fields that are dependent only on location code or on plant name.

There are quite a few fields where this is the case. Location name is dependent on location code (plant code is irrelevant in determining project name), and plant name, soil code, and soil name are all dependent on plant number. So, take out all these fields, as shown in the table below:

Removing the fields not dependent on the entire key

Location code	Plant code
11	431
11	446
11	482
12	431
12	449

Clearly you can't remove the data and leave it out of your database completely. You take it out, and put it into a new table, consisting of the fields that have the partial dependency and the fields on which they are dependent. For each of the *key* fields in the partial dependency, you create a new table (in this case, both are already part of the primary key, but this doesn't always have to be the case). So, you identified *plant name*, *soil description* and *soil category* as being dependent on *plant code*. The new table will consist of *plant code*, as a key, as well as plant name, soil category and soil description, as shown below:

Creating a new table with location data

Plant code	Plant name	Soil category	Soil description
431	Leucadendron	A	Sandstone
446	Protea	B	Sandstone/limestone
482	Erica	C	Limestone
431	Leucadendron	A	Sandstone
449	Restio	B	Sandstone/limestone

You do the same process with the location data, shown below:

Creating a new table with location data

Location code	Location name
11	Kirstenbosch Gardens
12	Karbonkelberg Mountains

See how these tables remove the earlier duplication problem? There is only one record that contains *Kirstenbosch Gardens*, so the chances of noticing a misspelling are much higher. And you aren't wasting space storing the name in many different records. Notice that the location code and plant code fields are repeated in two tables. These are the fields that create the relation, allowing you to associate the various plants with the various locations. Obviously there is no way to remove the duplication of these fields without losing the relation altogether, but it is far more efficient storing a small code repeatedly than a large piece of text.

But the table is still not perfect. There is still a chance for anomalies to slip in. Examine the table below carefully:

Another anomaly

Plant code	Plant name	Soil category	Soil description
431	Leucadendron	A	Sandstone
446	Protea	B	Sandstone/limestone
482	Erica	C	Limestone
431	Leucadendron	A	Sandstone
449	Restio	B	Sandstone

The problem in the table above is that the Restio has been associated with Sandstone, when in fact, having a soil category of B, it should be a mix of sandstone and limestone (the soil category determines the soil description in this example). Once again you're storing data redundantly. The *soil category* to *soil description* relationship is being stored in its entirety for each plant. As before, the solution is to take out this excess data and place it in its own table. What you are in fact doing at this stage is looking for transitive relationships, or relationships where a nonkey field is dependent on another nonkey field. *Soil description*, although in one sense dependent on plant code (it did seem to be a partial dependency when we looked at it in the previous step), is actually dependent on *soil category*. So, *soil description* must be removed. Once again, take it out and place it in a new table, along with its actual key (*soil category*) as shown in the tables below:

Plant data after removing the soil description

Plant code	Plant name	Soil category
431	Leucadendron	A
446	Protea	B
482	Erica	C
449	Restio	B

Creating a new table with the soil description

Soil category	Soil description
A	Sandstone
B	Sandstone/limestone
C	Limestone

You've cut down on the chances of anomalies once again. It is now impossible to mistakenly assume soil category B is associated with anything but a mix of sandstone and limestone. The soil description to soil category relationships are stored in only one place - the new *soil* table, where you can be much more certain they are accurate.

Often, when you're designing a system you don't yet have a complete set of test data available, and it's not necessary if you understand how the data relates. This article has used the tables and their data to demonstrate the consequences of storing data in tables that were not normalized, but without them you have to rely on dependencies between fields, which is the key to database normalization.

The following articles will describe the process more formally, starting with moving from unnormalized data (or zero normal form) to first normal form.

6.3.1.11.2 Database Normalization: 1st Normal Form

> This article follows on from the Database Normalization Overview.

At first, the data structure was as follows:

- Location code
- Location name
- 1-n plant numbers (1-n is a shorthand for saying there are many occurrences of this field. In other words, it is a repeating group).

- 1-n plant names
- 1-n soil categories
- 1-n soil descriptions

This is a completely unnormalized structure - in other words, it is in *zero normal form* So, to begin the normalization process, you start by moving from zero normal form to 1st normal form.

Tables are in 1st normal form if they follow these rules:

- There are no repeating groups.
- All the key attributes are defined.
- All attributes are dependent on the primary key.

What this means is that data must be able to fit into a tabular format, where each field contains one value. This is also the stage where the primary key is defined. Some sources claim that defining the primary key is not necessary for a table to be in first normal form, but usually it's done at this stage and is necessary before we can progress to the next stage. Theoretical debates aside, you'll have to define your primary keys at this point.

> Although not always seen as part of the definition of 1st normal form, the principle of atomicity is usually applied at this stage as well. This means that all columns must contain their smallest parts, or be indivisible. A common example of this is where someone creates a *name* field, rather than *first name* and *surname* fields. They usually regret it later.

So far, the plant example has no keys, and there are repeating groups. To get it into 1st normal form, you'll need to define a primary key and change the structure so that there are no repeating groups; in other words, each row / column intersection contains one, and only one, value. Without this, you cannot put the data into the ordinary two-dimensional table that most databases require. You define location code and plant code as the primary key together (neither on its own can uniquely identify a record), and replace the repeating groups with a single-value attribute. After doing this, you are left with the structure shown in the table below (the primary key is in italics):

Plant location table
Location code
Location name
Plant code
Plant name
Soil category
Soil description

This table is now in 1st normal formal. The process for turning a table into 2nd normal form is continued in the next article.

6.3.1.11.3 Database Normalization: 2nd Normal Form

Contents

> This article follows on from Database Normalization: 1st Normal Form.

After converting to first normal form, the following table structure was achieved:

Plant location table
Location code
Location name
Plant code
Plant name
Soil category

Soil description

Is this in 2nd normal form?

A table is in 2nd normal form if:

- it is in 1st normal form
- it includes no partial dependencies (where an attribute is only dependent on part of a primary key)

> For an attribute to be only dependent on part of the primary key, the primary key must consist of more than one field. If the primary key contains only one field, the table is automatically in 2nd normal form if it is in 1st normal form

Let's examine all the fields. *Location name* is only dependent on *location code*. *Plant name*, *soil category*, and *soil description* are only dependent on *plant code* (this assumes that each plant only occurs in one soil type, which is the case in this example). So you remove each of these fields and place them in a separate table, with the key being that part of the original key on which they are dependent. For example, with *plant name*, the key is *plant code*. This leaves you with the tables below:

Plant location table with partial dependencies removed

Plant location table
Plant code
Location code

Table resulting from fields dependent on plant code

Plant table
Plant code
Plant name
Soil category
Soil description

Table resulting from fields dependent on location code

Location table
Location code
Location name

The resulting tables are now in 2nd normal form. The process for turning a table into 3rd normal form is continued in the next article.

6.3.1.11.4 Database Normalization: 3rd Normal Form

> This article follows on from Database Normalization: 2nd Normal Form.

After converting to second normal form, the following table structure was achieved:

Plant location table
Plant code
Location code

Plant table
Plant code

Plant name
Soil category
Soil description

Location table
Location code
Location name

Are these tables in 3rd normal form?

A table is in 3rd normal form if:

- it is in 2nd normal form
- it contains no transitive dependencies (where a non-key attribute is dependent on the primary key through another non-key attribute)

> If a table only contains one non-key attribute, it is obviously impossible for a non-key attribute to be dependent on another non-key attribute. Any tables where this is the case that are in 2nd normal form are then therefore also in 3rd normal form.

As only the plant table has more than one non-key attribute, you can ignore the others because they are in 3rd normal form already. All fields are dependent on the primary key in some way, since the tables are in second normal form. But is this dependency on another non-key field? *Plant name* is not dependent on either *soil category* or *soil description*. Nor is *soil category* dependent on either *soil description* or *plant name*.

However, *soil description* is dependent on *soil category*. You use the same procedure as before, removing it, and placing it in its own table with the attribute that it was dependent on as the key. You are left with the tables below:

Plant location table remains unchanged

Plant location table
Plant code
Location code

Plant table with soil description removed

Plant table
Plant code
Plant name
Soil category

The new soil table

Soil table
Soil category
Soil description

Location table remains unchanged

Location table
Location code
Location name

All of these tables are now in 3rd normal form. 3rd normal form is usually sufficient for most tables because it avoids the most common kind of data anomalies. It's suggested getting most tables you work with to 3rd normal form before you implement them, as this will achieve the aims of normalization listed in Database Normalization Overview in the vast majority of cases.

The normal forms beyond this, such as Boyce-Codd normal form and 4th normal form, are rarely useful for business applications. In most cases, tables in 3rd normal form are already in these normal forms anyway. But any skilful database practitioner should know the exceptions, and be able to normalize to the higher levels when required.

The next article covers Boyce-Codd normal form.

6.3.1.11.5 Database Normalization: Boyce-Codd Normal Form

Contents

This article follows on from Database Normalization: 3rd Normal Form

E.F. Codd and R.F. Boyce, two of the people instrumental in the development of the database model, have been honored by the name of this normal form. E.F. Codd developed and expanded the relational model, and also developed normalization for relational models in 1970, while R.F. Boyce was one of the creators of Structured Query Language (then called SEQUEL).

In spite of some resources stating the contrary, Boyce-Codd normal form is not the same as 4th normal form. Let's look at an example of data anomalies, which are presented in 3rd normal form and solved by transforming into Boyce-Codd normal form, before defining it.

Table containing data about the student, course, and instructor relationship

Student Course Instructor table
Student
Course
Instructor

Assume that the following is true for the table above:

- Each instructor takes only one course
- Each course can have one or more instructors
- Each student only has one instructor per course
- Each student can take one or more courses

What would the key be? None of the fields on their own would be sufficient to uniquely identify a records, so you have to use two fields. Which two should you use?

Perhaps *student* and *instructor* seem like the best choice, as that would allow you to determine the *course*. Or you could use *student* and *course*, which would determine the *instructor*. For now, let's use *student* and *course* as the key:

Using student and course as the key

Student Course Instructor table
Student
Course
Instructor

What normal form is this table in? It's in 1st normal form, as it has a key and no repeating groups. It's also in 2nd normal form, as the instructor is dependent on both other fields (students have many courses, and therefore instructors, and courses have many instructors). Finally, it's also in 3rd normal form, as there is only one non-key attribute.

But there are still some data anomalies. Look at the data sample below:

More data anomalies

Student	Course	Instructor
Conrad Pienaar	Biology	Nkosizana Asmal
Dingaan Fortune	Mathematics	Kader Dlamini
Gerrie Jantjies	Science	Helen Ginwala
Mark Thobela	Biology	Nkosizana Asmal
Conrad Pienaar	Science	Peter Leon
Alicia Ncita	Science	Peter Leon
Quinton Andrews	Mathematics	Kader Dlamini

The fact that Peter Leon teaches science is stored redundantly, as are Kader Dlamini with mathematics and Nkosizana Asmal with biology. The problem is that the *instructor* determines the *course*. Or put another, *course* is determined by *instructor*. The table conforms to 3rd normal form rules because no non-key attribute is dependent upon a non-key attribute! Again, you use the familiar method of removing this field and placing it into another table, along with its key:

Student Instructor table after removing Course

Student Course Instructor table
Student
Instructor

After removing the *course* field, the primary key needs to include both remaining fields to uniquely identify a record.

Resulting Instructor table

Student Course Instructor table
Instructor
Course

Although we had chosen course as part of the primary key in the original table, the instructor determines the course, which is why we make it the primary key in this table. As you can see, the redundancy problem has been solved.

Thus, a table is in Boyce-Codd normal form if:

- it is in 3rd normal form
- each determinant is a candidate key

That sounds scary! For most people new to database design, these are new terms. If you followed along with the example above, however, the terms will soon become clear:

- a *determinant* is an attribute that determines the value of another attribute.
- a *candidate key* is either the key or an alternate key (in other words, the attribute could be a key for that table)

In the initial table, *instructor* is not a candidate key (alone it cannot uniquely identify the record), yet it determines the course, so the table is not in Boyce-Codd normal form.

Let's look at the example again, and see what happens if you chose student and instructor as the key. What normal form is the table in this time?

Using student and instructor as the key

Student Course Instructor table
Student
Instructor
Course

Once again it's in 1st normal form because there is a primary key and there are no repeating groups. This time, though, it's

not in 2nd normal form because *course* is determined by only part of the key: the instructor. By removing *course* and its key, instructor, you get the structure shown below:

Removing course

Student Instructor table
Student
Instructor

Creating a new table with course

Student Course Instructor table
Instructor
Course

Either way you do it, by making sure the tables are normalized into Boyce-Codd normal form, you get the same two resulting tables. It's usually the case that when there are alternate fields to choose as a key, it doesn't matter which ones you choose initially because after normalizing you get the same results either way.

6.3.1.11.6 Database Normalization: 4th Normal Form

Contents

This article is intended to be read after the Boyce-Codd normal form article.

Let's look at the situation where redundancies can creep in even though a table is in Boyce-Codd normal form. Let's take the student / instructor / course example used in that article, but change one of the initial assumptions.

Assume that the following is true for the tables below:

- Each instructor takes only one course
- Each course can have one or more instructors
- Each student can have several instructors per course (this is different to the previous example)
- Each student can take one or more courses

Student Course Instructor data, with several instructors per course

More data anomalies

Student	Course	Instructor
Conrad Pienaar	Biology	Nkosizana Asmal
Dingaan Fortune	Mathematics	Kader Dlamini
Gerrie Jantjies	Science	Helen Ginwala
Mark Thobela	Biology	Nkosizana Asmal
Conrad Pienaar	Science	Peter Leon
Alicia Ncita	Science	Peter Leon
Quinton Andrews	Mathematics	Kader Dlamini
Dingaan Fortune	Mathematics	Helen Ginwala

The data is the same as before, except that Helen Ginwala is teaching science to Gerrie Jantjies as well as mathematics to Dingaan Fortune, and Dingaan Fortune is being taught by both Helen Ginwala and Kader Dlamini for mathematics.

The only possible key is a combination of all three attributes, as shown below. No other combination will uniquely identify a particular record.

Three attributes as key

Student Course Instructor table
Student
Instructor
Course

But this still has some potentially anomalous behavior. The fact that Kader Dlamini teaches mathematics is still stored more than once, as is the fact that Dingaan Thobela takes mathematics. The real problem is that the table stores more than one kind of fact: that of student-to-course relationship, as well as that of a student-to-instructor relationship. You can avoid this, as always, by separating the data into two tables, as shown below:

Creating a table for the student to instructor relationship

Student Instructor table
Student
Instructor

Creating a table for the student to course relationship

Student Instructor table
Student
Course

This situation exists when you have multiple multivalued dependencies. A multivalued dependency exists between two attributes when, for each value of the first attribute, there are one or more associated values of the second attribute. For each value of *student*, there were many values of *course*. This is the first multivalued dependency. Then, for each value of *student*, there are one or more associated values of *instructor*. This is the second multivalued dependency.

Thus, a table is in 4th normal form if:

- it is in Boyce-Codd normal form
- it does not contain more than one multivalued dependency

6.3.1.11.7 Database Normalization: 5th Normal Form and Beyond

Contents

This article follows on from the 4th normal form article.

There are normal forms beyond 4th that are mainly of academic interest, as the problems they exist to solve rarely appear in practice. This series won't discuss then in detail, but for those interested, the following example provides a taste.

The sales rep example

Sales rep	Company	Product

Felicia Powers	Exclusive	Books
Afzal Ignesund	Wordsworth	Magazines
Felicia Powers	Exclusive	Magazines

Usually you would store this data in one table, as you need all three records to see which combinations are valid. *Afzal Ignesund* sells magazines for *Wordsworth*, but not necessarily books. *Felicia Powers* happens to sell both books and magazines for *Exclusive*. However, let's add another condition. If a sales rep sells a certain product, and they sell it for a particular company, then they must sell that product for that company.

Let's look at a larger data set adhering to this condition:

Looking at a larger set of data

Sales rep	Company	Product
Felicia Powers	Exclusive	Books
Felicia Powers	Exclusive	Magazines
Afzal Ignesund	Wordsworth	Books
Felicia Powers	Wordsworth	Books
Felicia Powers	Wordsworth	Magazines

Now, with this extra dependency, you could normalize the table above into three separate tables without losing any facts, as shown below:

Creating a table with Sales rep and Product

Sales rep	Product
Felicia Powers	Books
Felicia Powers	Magazines
Afzal Ignesund	Books

Creating a table with Sales rep and Company

Sales rep	Company
Felicia Powers	Exclusive
Felicia Powers	Wordsworth
Afzal Ignesund	Wordsworth

Creating a table with Company and Product

Company	Product
Exclusive	Books
Exclusive	Magazines
Wordsworth	Books
Wordsworth	Magazines

Basically, a table is in 5th normal form if it cannot be made into any smaller tables with different keys (most tables can obviously be made into smaller tables with the same key!).

Beyond 5th normal form you enter the heady realms of domain key normal form, a kind of theoretical ideal. Its practical use to a database designer os similar to that of infinity to a bookkeeper - i.e. it exists in theory but is not going to be used in practice. Even the most demanding owner is not going to expect that of the bookkeeper!

For those interested in pursuing this academic and highly theoretical topic further, I suggest obtaining a copy of *An Introduction to Database Systems* by C.J. Date, at the time of writing in its 8th edition, or *Relational Theory for Computer Professionals* by the same author.

6.3.1.11.8 Understanding Denormalization

Denormalization is the process of reversing the transformations made during normalization for performance reasons. It's a topic that stirs controversy among database experts; there are those who claim the cost is too high and never denormalize, and there are those that tout its benefits and routinely denormalize.

For proponents of denormalization, the thinking is as follows: normalization creates more tables as you proceed towards higher normal forms, but more tables mean there are more joins to be made when data is retrieved, which in turn slows down your queries. For that reason, to improve the performance of certain queries, you can override the advantages to data integrity and return the data structure to a lower normal form.

A practical approach makes sense, taking into account the limitations of SQL and MariaDB in particular, but being cautious not to needless denormalize. Here are some suggestions:

- if your performance with a normalized structure is acceptable, you should not denormalize.
- if your performance is unacceptable, make sure normalizing will cause it to become acceptable. There are very likely to be other alternatives, such as better hardware, load balancing, etc. It's hard to undo structural changes later.
- be sure you are willing to trade decreased data integrity for the increase in performance.
- consider possible future scenario, where applications may place different requirements on the data. Denormalizing to enhance performance of a specific application makes your data structure dependent on that application, when in an ideal situation it will be application-independent.

The table below introduces a common structure where it may not be in your best interests to denormalize. Which normal form is it in?

Customer table
ID
First name
Surname
Address line 1
Address line 2
Town
Zip code

It must be in 1st normal form because it has a primary key and there are no repeating groups. It must be in 2nd normal form because there's only one key, so there cannot be any partial dependencies. And 3rd normal form? Are there any transitive dependencies? It looks like it. *Zip Code* is probably determined by the town attribute. To transform it into 3rd normal form, you should take out *Zi..p code*, putting it in a separate table with town as the key. In most cases, though, this is not worth doing. So although this table is not in 3rd normal form, separating the table is not worth the trouble. The more tables you have, the more joins you need to do, which slows the system down. The reason you normalize at all is to reduce the size of the tables by removing redundant data, and doing do can often speed up the system.

But you also need to look at how your tables are used. *Town* and *Zip code* would almost always be returned together, as part of the address. In most cases, the small amount of space you save by removing the duplicate town/zip code combinations would not offset the slowing down of the system because of the extra joins. In some situations, this may be useful, perhaps where you need to sort addresses according to zip codes or towns for many thousands of customers, and the distribution of data means that a query to the new, smaller table can return the results substantially quicker. In the end, experienced database designers can go beyond rigidly following the steps, as they understand how the data will be used. And that is something only experience can teach you. Normalization is just a helpful set of steps that most often produces an efficient table structure, and not a rule for database design.

> There are some scary database designs out there, almost always because of not normalizing rather than too much normalization. So if you're unsure, normalize!

6.3.1.12 ACID: Concurrency Control with Transactions

Database requests happen in linear fashion, one after another. When many users are accessing a database, or one user has a related set of requests to run, it becomes important to ensure that the results remain consistent. To achieve this, you use *transactions*, which are groups of database requests that are processed as a whole. Put another way, they are logical units of work.

To ensure data integrity, transactions need to adhere to four conditions: atomicity, consistency, isolation and durability

(ACID).

Atomicity

Atomicity means the entire transaction must complete. If this is not the case, the entire transaction is aborted. This ensures that the database can never be left with partially completed transactions, which lead to poor data integrity. If you remove money out of one bank account, for example, but the second request fails and the system cannot place the money in another bank, both requests must fail. The money cannot simply be lost, or taken from one account without going into the other.

Consistency

Consistency refers to the state the data is in when certain conditions are met. For example, one rule may be that each invoice must relate to a customer in the customer table. These rules may be broken during the course of a transaction if, for example the invoice is inserted without a related customer, which is added at a later stage in the transaction. These temporary violations are not visible outside of the transaction, and will always be resolved by the time the transaction is complete.

Isolation

Isolation means that any data being used during the processing of one transaction cannot be used by another transaction until the first transaction is complete. For example, if two people deposit $100 into another account with a balance of $900, the first transaction must add $100 to $900, and the second must add $100 to $1000. If the second transaction reads the $900 before the first transaction has completed, both transactions will seem to succeed, but $100 will have gone missing. The second transaction must wait until it alone is accessing the data.

Durability

Durability refers to the fact that once data from a transaction has been committed, its effects will remain, even after a system failure. While a transaction is under way, the effects are not persistent. If the database crashes, backups will always restore it to a consistent state prior to the transaction commencing. Nothing a transaction does should be able to change this fact.

2.1.6 Starting and Stopping MariaDB

6.4 Advanced MariaDB Articles

Tutorial articles for advanced MariaDB developers and administrators.

Development Articles

Articles of interest to MariaDB developers.

There are 2 related questions.

6.4.1 Development Articles

Articles of interest to MariaDB Developers

General Development Information

MariaDB Server Releases

Information about MariaDB Server releases, release policies and procedures.

MariaDB Internals Documentation

Documentation on the internal workings of MariaDB.

MariaDB Development Tools

Tools for developing MariaDB.

Debugging MariaDB

This section is for articles on debugging MariaDB.

Quality

This section collects articles related to MariaDB quality assurance efforts.

Outdated pages

Security Vulnerabilities Fixed in MariaDB

Security vulnerabilities (CVEs) fixed in MariaDB

Security Vulnerabilities Fixed in Oracle MySQL That Did Not Exist in MariaDB

Lists of CVEs fixed in MySQL but that were never present in MariaDB.

Uploading Package to PPA

After creating a Launchpad account: Docker build, cloning the MariaDB repo...

MariaDB Quality Development Rules

Those are quality-improving rules that everyone with a write access to the ...

There are 21 related questions.

6.4.1.1 MariaDB Internals Documentation

Documentation on the internal workings of MariaDB.

Contributing Code

Guidelines and procedures for contributing code to MariaDB.

Writing Plugins for MariaDB

Writing plugins for MariaDB.

Pluggable Authentication Overview

The authentication of users is delegated to plugins.

Information Schema plugins: SHOW and FLUSH statements

Information Schema plugins can support SHOW and FLUSH statements.

Query Optimizer

Articles about the MariaDB query optimizer

Using MariaDB with Your Programs (API)

Using MariaDB with your programs (API)

Storage Engine Development

Storage Engine Development.

Merging into MariaDB

How to merge various source trees into MariaDB

MariaDB Source Code Internals

Articles about MariaDB source code and related internals.

Encryption Plugin API

MariaDB uses plugins to handle key management and encryption of data.

Password Validation Plugin API

Allows the creation of password validation plugins to check user passwords as they are set.

There are 1 related questions.

6.4.1.1.1 Query Optimizer

Articles about the MariaDB query optimizer

Optimization Strategies
Various optimization strategies used by the query optimizer.

Optimizations for Derived Tables
Optimizations for derived tables, or subqueries in the FROM clause

Optimizer Trace
Produces a JSON trace with decision info taken by the optimizer during the optimization phase.

Statistics for Optimizing Queries
Different statistics provided by MariaDB to help you optimize your queries

Table Elimination
Resolving queries without accessing some of the tables the query refers to

Block-Based Join Algorithms
Algorithms that employ a join buffer for the first join before starting to look in the second.

Condition Selectivity Computation Internals
How the MariaDB optimizer computes condition selectivities.

Extended Keys
Optimization using InnoDB key components to generate more efficient execution plans.

MIN/MAX optimization
How MIN and MAX are optimized

Notes When an Index Cannot Be Used
Notes (low severity warnings) when an indexed column cannot use the index to lookup rows.

Optimizer Debugging With GDB
Useful things for debugging optimizer code with gdb.

Rowid Filtering Optimization
Rowid filtering is an optimization available from MariaDB 10.4.

The Optimizer Cost Model from MariaDB 11.0
The optimizer cost model in MariaDB 11.0.

6.4.1.1.1.1 Optimizer Trace

Optimizer Trace Overview
Produces a JSON trace with decision info taken by the optimizer during the optimization phase.

Optimizer Trace Guide
Guide to the structured log file showing what actions were taken by the query optimizer.

Basic Optimizer Trace Example
*MariaDB> set optimizer_trace='enabled=on'; MariaDB> select * from t1 where a<10; MariaDB> s*

How to Collect Large Optimizer Traces
Steps for collecting large optimizer traces.

Optimizer Trace for Developers
This article describes guidelines for what/how to write to Optimizer Trace ...

Optimizer Trace Resources

Optimizer Trace Walkthrough talk at MariaDB Fest 2020: https://mariadb.org/...

6.4.1.1.1.1.1 Optimizer Trace Overview

MariaDB starting with 10.4.3

Optimizer Trace was introduced in MariaDB 10.4.3.

Contents

Usage

This feature produces a trace as a JSON document for any SELECT/UPDATE/DELETE containing information about decisions taken by the optimizer during the optimization phase (choice of table access method, various costs, transformations, etc). This feature helps to explain why some decisions were taken by the optimizer and why some were rejected.

Associated System Variables

- optimizer_trace='enabled=on/off'
  - Default value is off
- optimizer_trace_max_mem_size= value
  - Default value: 1048576

INFORMATION_SCHEMA.OPTIMIZER_TRACE

Each connection stores a trace from the last executed statement. One can view the trace by reading the Information Schema OPTIMIZER_TRACE table.

Structure of the optimizer trace table:

```
SHOW CREATE TABLE INFORMATION_SCHEMA.OPTIMIZER_TRACE \G
*************************** 1. row ***************************
       Table: OPTIMIZER_TRACE
Create Table: CREATE TEMPORARY TABLE `OPTIMIZER_TRACE` (
  `QUERY` longtext NOT NULL DEFAULT '',
  `TRACE` longtext NOT NULL DEFAULT '',
  `MISSING_BYTES_BEYOND_MAX_MEM_SIZE` int(20) NOT NULL DEFAULT 0,
  `INSUFFICIENT_PRIVILEGES` tinyint(1) NOT NULL DEFAULT 0
) ENGINE=Aria DEFAULT CHARSET=utf8 PAGE_CHECKSUM=0
```

Optimizer Trace Contents

See Optimizer Trace Guide for an overview of what one can find in the trace.

Traceable Queries

These include SELECT, UPDATE, DELETE as well as their multi-table variants and all of the preceding prefixed by EXPLAIN and ANALYZE.

Enabling Optimizer Trace

To enable optimizer trace run:

```
SET optimizer_trace='enabled=on';
```

Memory Usage

Each trace is stored as a string. It is extended (with realloc()) as the optimization progresses and appends data to it. The optimizer_trace_max_mem_size variable sets a limit on the total amount of memory used by the current trace. If this limit is reached, the current trace isn't extended (so it will be incomplete), and the MISSING_BYTES_BEYOND_MAX_MEM_SIZE column will show the number of bytes missing from this trace.

Privilege Checking

In complex scenarios where the query uses SQL SECURITY DEFINER views or stored routines, it may be that a user is denied from seeing the trace of its query because it lacks some extra privileges on those objects. In that case, the trace will be shown as empty and the INSUFFICIENT_PRIVILEGES column will show "1".

Limitations

Currently, only one trace is stored. It is not possible to trace the sub-statements of a stored routine; only the statement at the top level is traced.

6.4.1.1.1.1.2 Optimizer Trace Guide

Contents

Optimizer trace uses the JSON format. It is basically a structured log file showing what actions were taken by the query optimizer.

A Basic Example

Let's take a simple query:

```
MariaDB> explain select * from t1 where a<10;
+------+-------------+-------+-------+---------------+------+---------+------+------+-------------
| id   | select_type | table | type  | possible_keys | key  | key_len | ref  | rows | Extra
+------+-------------+-------+-------+---------------+------+---------+------+------+-------------
|    1 | SIMPLE      | t1    | range | a             | a    | 5       | NULL | 10   | Using index
+------+-------------+-------+-------+---------------+------+---------+------+------+-------------
```

One can see the full trace here. Taking only the component names, one gets:

```
MariaDB> select * from information_schema.optimizer_trace limit 1\G
*************************** 1. row ***************************
                            QUERY: select * from t1 where a<10
                            TRACE:
{
  "steps": [
    {
      "join_preparation": { ... }
    },
    {
      "join_optimization": {
        "select_id": 1,
        "steps": [
          { "condition_processing": { ... } },
          { "table_dependencies": [ ... ] },
          { "ref_optimizer_key_uses": [ ... ] },
          { "rows_estimation": [
              {
                "range_analysis": {
                   "analyzing_range_alternatives" : { ... },
                  "chosen_range_access_summary": { ... },
                },
                "selectivity_for_indexes" : { ... },
                "selectivity_for_columns" : { ... }
              }
            ]
          },
          { "considered_execution_plans": [ ... ] },
          { "attaching_conditions_to_tables": { ... } }
         ]
      }
    },
    {
      "join_execution": { ... }
    }
  ]
}
```

Trace Structure

For each SELECT, there are two "Steps":

- `join_preparation`
- `join_optimization`

Join preparation shows early query rewrites. `join_optmization` is where most of the query optimizations are done. They are:

- `condition_processing` - basic rewrites in WHERE/ON conditions.
- `ref_optimizer_key_uses` - Construction of possible ways to do ref and eq_ref accesses.
- `rows_estimation` - Consideration of range and index_merge accesses.
- `considered_execution_plans` - Join optimization itself, that is, choice of the join order.
- `attaching_conditions_to_tables` - Once the join order is fixed, parts of the WHERE clause are "attached" to tables to filter out rows as early as possible.

The above steps are for just one SELECT. If the query has subqueries, each SELECT will have these steps, and there will be extra steps/rewrites to handle the subquery construct itself.

Extracting Trace Components

If you are interested in some particular part of the trace, MariaDB has two functions that come in handy:

- JSON_EXTRACT extracts a part of JSON document
- JSON_DETAILED presents it in a user-readable way.

For example, the contents of the `analyzing_range_alternatives` node can be extracted like so:

```
MariaDB> select JSON_DETAILED(JSON_EXTRACT(trace, '$**.analyzing_range_alternatives'))
   ->   from INFORMATION_SCHEMA.OPTIMIZER_TRACE\G
*************************** 1. row ***************************
JSON_DETAILED(JSON_EXTRACT(trace, '$**.analyzing_range_alternatives')): [
    {
        "range_scan_alternatives":
        [
            {
                "index": "a_b_c",
                "ranges":
                [
                    "(1) <= (a,b) < (4,50)"
                ],
                "rowid_ordered": false,
                "using_mrr": false,
                "index_only": false,
                "rows": 4,
                "cost": 6.2509,
                "chosen": true
            }
        ],
        "analyzing_roworder_intersect":
        {
            "cause": "too few roworder scans"
        },
        "analyzing_index_merge_union": []
    }
]
```

Examples of Various Information in the Trace

Basic Rewrites

A lot of applications construct database query text on the fly, which sometimes means that the query has constructs that are repetitive or redundant. In most cases, the optimizer will be able to remove them. One can check the trace to be sure:

```
explain select * from t1 where not (col1 >= 3);
```

Optimizer trace will show:

```
"steps": [
  {
    "join_preparation": {
      "select_id": 1,
      "steps": [
        {
          "expanded_query": "select t1.a AS a,t1.b AS b,t1.col1 AS col1 from t1 where t1.col1 < 3
        }
```

Here, one can see that `NOT` was removed.

Similarly, one can also see that `IN(...)` with one element is the same as equality:

```
explain select * from t1 where col1  in (1);
```

will show

```
  "join_preparation": {
    "select_id": 1,
    "steps": [
      {
        "expanded_query": "select t1.a AS a,t1.b AS b,t1.col1 AS col1 from t1 where t1.col1 = 1"
```

On the other hand, converting an UTF-8 column to UTF-8 is not removed:

```
explain select * from t1 where convert(utf8_col using utf8) = 'hello';
```

will show

```
  "join_preparation": {
    "select_id": 1,
    "steps": [
      {
        "expanded_query": "select t1.a AS a,t1.b AS b,t1.col1 AS col1,t1.utf8_col AS utf8_col fro
        }
```

so redundant `CONVERT` calls should be used with caution.

VIEW Processing

MariaDB has two algorithms to handle VIEWs: merging and materialization. If you run a query that uses a VIEW, the trace will have either

```
          "view": {
            "table": "view1",
            "select_id": 2,
            "algorithm": "merged"
          }
```

or

```
        {
          "view": {
            "table": "view2",
            "select_id": 2,
            "algorithm": "materialized"
          }
        },
```

depending on which algorithm was used.

Range Optimizer - What Ranges Will Be Scanned

The MariaDB optimizer has a complex part called the Range Optimizer. This is a module that examines WHERE (and ON) clauses and constructs index ranges that need to be scanned to answer the query. The rules for constructing the ranges are quite complex.

An example: Consider a table

```
create table some_events (
  start_date date,
  end_date date,
  ...
  key (start_date, end_date)
);
```

and a query:

```
explain select * from some_events where start_date >= '2019-09-10' and end_date <= '2019-09-14';
+------+-------------+-------------+------+---------------+------+---------+------+------+-------
| id   | select_type | table       | type | possible_keys | key  | key_len | ref  | rows | Extra
+------+-------------+-------------+------+---------------+------+---------+------+------+-------
|    1 | SIMPLE      | some_events | ALL  | start_date    | NULL | NULL    | NULL | 1000 | Using
+------+-------------+-------------+------+---------------+------+---------+------+------+-------
```

One might think that the optimizer would be able to use the restrictions on both *start_date* and *end_date* to construct a narrow range to be scanned. But this is not so, one of the restrictions creates a left-endpoint range and the other one creates a right-endpoint range, hence they cannot be combined.

```
select
   JSON_DETAILED(JSON_EXTRACT(trace, '$**.analyzing_range_alternatives')) as trace
from information_schema.optimizer_trace\G
*************************** 1. row ***************************
trace: [
    {
        "range_scan_alternatives":
        [
            {
                "index": "start_date",
                "ranges":
                [
                    "(2019-09-10,NULL) < (start_date,end_date)"
                ],
...
```

the potential range only uses one of the bounds.

Ref Access Options

Index-based Nested-loops joins are called "ref access" in the MariaDB optimizer.

The optimizer analyzes the WHERE/ON conditions and collects all equality conditions that can be used by ref access using some index.

The list of conditions can be found in the `ref_optimizer_key_uses` node. (TODO example)

Join Optimization

The join optimizer's node is named `considered_execution_plans`.

The optimizer constructs the join orders in a left-to-right fashion. That is, if the query is a join of three tables:

```
select * from t1, t2, t3 where ...
```

then the optimizer will

- Pick the first table (say, it is t1),
- consider adding another table (say, t2), and construct a prefix "t1, t2"
- consider adding the third table (t3), and constructing a prefix "t1, t2, t3", which is a complete join plan Other join orders will be considered as well.

The basic operation here is: "given a join prefix of tables A,B,C ..., try adding table X to it". In JSON, it looks like this:

```
      {
        "plan_prefix": ["t1", "t2"],
        "table": "t3",
        "best_access_path": {
          "considered_access_paths": [
            {
              ...
            }
          ]
        }
      }
```

(search for `plan_prefix` followed by `table`).

If you are interested in how the join order of #t1,t2,t3# was constructed (or not constructed), you need to search for these patterns:

- `"plan_prefix":[], "table":"t1"`
- `"plan_prefix":["t1"], "table":"t2"`
- `"plan_prefix":["t1", "t2"], "table":"t3"`

6.4.1.1.1.1.3 Basic Optimizer Trace Example

```
MariaDB> set optimizer_trace='enabled=on';
```

```
MariaDB> select * from t1 where a<10;

MariaDB> select * from information_schema.optimizer_trace limit 1\G
*************************** 1. row ***************************
                            QUERY: select * from t1 where a<10
                            TRACE: {
  "steps": [
    {
      "join_preparation": {
        "select_id": 1,
        "steps": [
          {
            "expanded_query": "select t1.a AS a,t1.b AS b,t1.c AS c from t1 where t1.a < 10"
          }
        ]
      }
    },
    {
      "join_optimization": {
        "select_id": 1,
        "steps": [
          {
            "condition_processing": {
              "condition": "WHERE",
              "original_condition": "t1.a < 10",
              "steps": [
                {
                  "transformation": "equality_propagation",
                  "resulting_condition": "t1.a < 10"
                },
                {
                  "transformation": "constant_propagation",
                  "resulting_condition": "t1.a < 10"
                },
                {
                  "transformation": "trivial_condition_removal",
                  "resulting_condition": "t1.a < 10"
                }
              ]
            }
          },
          {
            "table_dependencies": [
              {
                "table": "t1",
                "row_may_be_null": false,
                "map_bit": 0,
                "depends_on_map_bits": []
              }
            ]
          },
          {
            "ref_optimizer_key_uses": []
          },
          {
            "rows_estimation": [
              {
                "table": "t1",
                "range_analysis": {
                  "table_scan": {
                    "rows": 1000,
                    "cost": 206.1
                  },
                  "potential_range_indexes": [
                    {
                      "index": "a",
                      "usable": true,
                      "key_parts": ["a"]
                    },
                    {
                      "index": "b",
                      "usable": false,
                      "cause": "not applicable"
                    }
                  ],
```

```
            "setup_range_conditions": [],
            "group_index_range": {
              "chosen": false,
              "cause": "no group by or distinct"
            },
            "analyzing_range_alternatives": {
              "range_scan_alternatives": [
                {
                  "index": "a",
                  "ranges": ["(NULL) < (a) < (10)"],
                  "rowid_ordered": false,
                  "using_mrr": false,
                  "index_only": false,
                  "rows": 10,
                  "cost": 13.751,
                  "chosen": true
                }
              ],
              "analyzing_roworder_intersect": {
                "cause": "too few roworder scans"
              },
              "analyzing_index_merge_union": []
            },
            "chosen_range_access_summary": {
              "range_access_plan": {
                "type": "range_scan",
                "index": "a",
                "rows": 10,
                "ranges": ["(NULL) < (a) < (10)"]
              },
              "rows_for_plan": 10,
              "cost_for_plan": 13.751,
              "chosen": true
            }
          }
        },
        {
          "selectivity_for_indexes": [
            {
              "index_name": "a",
              "selectivity_from_index": 0.01
            }
          ],
          "selectivity_for_columns": [],
          "cond_selectivity": 0.01
        }
      ]
    },
    {
      "considered_execution_plans": [
        {
          "plan_prefix": [],
          "table": "t1",
          "best_access_path": {
            "considered_access_paths": [
              {
                "access_type": "range",
                "resulting_rows": 10,
                "cost": 13.751,
                "chosen": true
              }
            ]
          }
        }
      ]
    },
    {
      "attaching_conditions_to_tables": {
        "original_condition": "t1.a < 10",
        "attached_conditions_computation": [],
        "attached_conditions_summary": [
          {
            "table": "t1",
            "attached": "t1.a < 10"
          }
```

```
              ]
            }
          }
        ]
      }
    },
    {
      "join_execution": {
        "select_id": 1,
        "steps": []
      }
    }
  ]
}
MISSING_BYTES_BEYOND_MAX_MEM_SIZE: 0
          INSUFFICIENT_PRIVILEGES: 0
```

6.4.1.1.1.1.4 How to Collect Large Optimizer Traces

Optimizer traces can be large for some queries.

In order to collect a large trace, you need to perform the following steps (using 128 MB as an example):

```
set global max_allowed_packet=128*1024*1024;
```

Reconnect specifying `--max-allowed-packet=128000000` for the client as well.

```
set optimizer_trace=1;
set optimizer_trace_max_mem_size=127*1024*1024;
```

Now, one can run the query and save the large trace.

6.4.1.1.1.1.5 Optimizer Trace for Developers

This article describes guidelines for what/how to write to Optimizer Trace when doing server development.

Basic considerations

The trace is a "structured log" of what was done by the optimizer. Prefer to do tracing as soon as a rewrite/decision is made (instead of having a separate trace_something() function).

Generally, a function should expect to find the trace in a state where we're writing an array. The rationale is that array elements are ordered, while object members are not (even if they come in a certain order in the JSON text). We're writing a log, so it's natural for different entries to form an array.

Typically you'll want to start an unnamed object, then use member names to show what kind of entry you're about to write:

```
[
  ...,  # Something before us
  {
    "my_new_rewrite": {
       "from": "foo",
       "to": "bar",
       ...
    }
  }
  ...
```

(TODO other considerations)

Making sure the trace is valid

`Json_writer_object` and `Json_writer_array` classes use RAII idiom and ensure that JSON objects and arrays are

"closed" in the reverse order they were started.

However, they do not ensure these constraints:

- JSON objects must have named members.
- JSON arrays must have unnamed members.

Tracing code has runtime checks for these. Attempt to write invalid JSON will cause assertion failure.

Test coverage

It is possible to run `mysql-test-run` with this argument

```
--mysqld=--optimizer_trace=enabled=on
```

This will run all tests with tracing on. As mentioned earlier, debug build will perform checks that we are not producing invalid trace.

The BuildBot instance at http://buildbot.askmonty.org/ also runs tests with this argument, see `mtr_opttrace` pass in kvm-fulltest and kvm-fulltest2.

Debugging

See optimizer-debugging-with-gdb/#printing-the-optimizer-trace for commands to print the trace for the current statement.

6.4.1.1.2 Using MariaDB with Your Programs (API)

Error Codes
MariaDB error codes and SQLSTATE codes

libMariaDB

libmysqld
The Embedded, Stand-Alone MariaDB Server.

Non-Blocking Client Library
Non-blocking client library documentation.

Progress Reporting
Progress reporting for long running commands.

There are 2 related questions.

6.4.1.1.2.1 Error Codes

MariaDB Error Codes
MariaDB error codes reference list.

Operating System Error Codes
Linux and Windows operating system error codes.

SQLSTATE
A string which identifies a condition's class and subclass

There are 2 related questions.

1.1.2.9 MariaDB Error Codes

6.4.1.1.2.1.2 Operating System Error Codes

Contents

Below is a partial list of more common Linux and Windows operating system error codes.

Linux Error Codes

The perror tool can be used to find the error message which is associated with a given error code.

Number	Error Code	Description
1	EPERM	Operation not permitted
2	ENOENT	No such file or directory
3	ESRCH	No such process
4	EINTR	Interrupted system call
5	EIO	I/O error
6	ENXIO	No such device or address
7	E2BIG	Argument list too long
8	ENOEXEC	Exec format error
9	EBADF	Bad file number
10	ECHILD	No child processes
11	EAGAIN	Try again
12	ENOMEM	Out of memory
13	EACCES	Permission denied
14	EFAULT	Bad address
15	ENOTBLK	Block device required
16	EBUSY	Device or resource busy
17	EEXIST	File exists
18	EXDEV	Cross-device link
19	ENODEV	No such device
20	ENOTDIR	Not a directory
21	EISDIR	Is a directory
22	EINVAL	Invalid argument
23	ENFILE	File table overflow
24	EMFILE	Too many open files
25	ENOTTY	Not a typewriter
26	ETXTBSY	Text file busy
27	EFBIG	File too large
28	ENOSPC	No space left on device
29	ESPIPE	Illegal seek
30	EROFS	Read-only file system
31	EMLINK	Too many links

32	EPIPE	Broken pipe
33	EDOM	Math argument out of domain of func
34	ERANGE	Math result not representable
35	EDEADLK	Resource deadlock would occur
36	ENAMETOOLONG	File name too long
37	ENOLCK	No record locks available
38	ENOSYS	Function not implemented
39	ENOTEMPTY	Directory not empty
40	ELOOP	Too many symbolic links encountered
42	ENOMSG	No message of desired type
43	EIDRM	Identifier removed
44	ECHRNG	Channel number out of range
45	EL2NSYNC	Level 2 not synchronized
46	EL3HLT	Level 3 halted
47	EL3RST	Level 3 reset
48	ELNRNG	Link number out of range
49	EUNATCH	Protocol driver not attached
50	ENOCSI	No CSI structure available
51	EL2HLT	Level 2 halted
52	EBADE	Invalid exchange
53	EBADR	Invalid request descriptor
54	EXFULL	Exchange full
55	ENOANO	No anode
56	EBADRQC	Invalid request code
57	EBADSLT	Invalid slot
59	EBFONT	Bad font file format
60	ENOSTR	Device not a stream
61	ENODATA	No data available
62	ETIME	Timer expired
63	ENOSR	Out of streams resources
64	ENONET	Machine is not on the network
65	ENOPKG	Package not installed
66	EREMOTE	Object is remote
67	ENOLINK	Link has been severed
68	EADV	Advertise error
69	ESRMNT	Srmount error
70	ECOMM	Communication error on send
71	EPROTO	Protocol error
72	EMULTIHOP	Multihop attempted
73	EDOTDOT	RFS specific error
74	EBADMSG	Not a data message
75	EOVERFLOW	Value too large for defined data type
76	ENOTUNIQ	Name not unique on network

77	EBADFD	File descriptor in bad state
78	EREMCHG	Remote address changed
79	ELIBACC	Can not access a needed shared library
80	ELIBBAD	Accessing a corrupted shared library
81	ELIBSCN	.lib section in a.out corrupted
82	ELIBMAX	Attempting to link in too many shared libraries
83	ELIBEXEC	Cannot exec a shared library directly
84	EILSEQ	Illegal byte sequence
85	ERESTART	Interrupted system call should be restarted
86	ESTRPIPE	Streams pipe error
87	EUSERS	Too many users
88	ENOTSOCK	Socket operation on non-socket
89	EDESTADDRREQ	Destination address required
90	EMSGSIZE	Message too long
91	EPROTOTYPE	Protocol wrong type for socket
92	ENOPROTOOPT	Protocol not available
93	EPROTONOSUPPORT	Protocol not supported
94	ESOCKTNOSUPPORT	Socket type not supported
95	EOPNOTSUPP	Operation not supported on transport endpoint
96	EPFNOSUPPORT	Protocol family not supported
97	EAFNOSUPPORT	Address family not supported by protocol
98	EADDRINUSE	Address already in use
99	EADDRNOTAVAIL	Cannot assign requested address
100	ENETDOWN	Network is down
101	ENETUNREACH	Network is unreachable
102	ENETRESET	Network dropped connection because of reset
103	ECONNABORTED	Software caused connection abort
104	ECONNRESET	Connection reset by peer
105	ENOBUFS	No buffer space available
106	EISCONN	Transport endpoint is already connected
107	ENOTCONN	Transport endpoint is not connected
108	ESHUTDOWN	Cannot send after transport endpoint shutdown
109	ETOOMANYREFS	Too many references: cannot splice
110	ETIMEDOUT	Connection timed out
111	ECONNREFUSED	Connection refused
112	EHOSTDOWN	Host is down
113	EHOSTUNREACH	No route to host
114	EALREADY	Operation already in progress
115	EINPROGRESS	Operation now in progress
116	ESTALE	Stale NFS file handle
117	EUCLEAN	Structure needs cleaning
118	ENOTNAM	Not a XENIX named type file
119	ENAVAIL	No XENIX semaphores available

120	EISNAM	Is a named type file
121	EREMOTEIO	Remote I/O error
122	EDQUOT	Quota exceeded
123	ENOMEDIUM	No medium found
124	EMEDIUMTYPE	Wrong medium type
125	ECANCELED	Operation Canceled
126	ENOKEY	Required key not available
127	EKEYEXPIRED	Key has expired
128	EKEYREVOKED	Key has been revoked
129	EKEYREJECTED	Key was rejected by service
130	EOWNERDEAD	Owner died
131	ENOTRECOVERABLE	State not recoverable

Windows Error Codes

For a complete list, see https://msdn.microsoft.com/en-us/library/ms681381.aspx

Number	Error Code	Description
1	ERROR_INVALID_FUNCTION	Incorrect function.
2	ERROR_FILE_NOT_FOUND	The system cannot find the file specified.
3	ERROR_PATH_NOT_FOUND	The system cannot find the path specified.
4	ERROR_TOO_MANY_OPEN_FILES	The system cannot open the file.
5	ERROR_ACCESS_DENIED	Access is denied.
6	ERROR_INVALID_HANDLE	The handle is invalid.
7	ERROR_ARENA_TRASHED	The storage control blocks were destroyed.
8	ERROR_NOT_ENOUGH_MEMORY	Not enough storage is available to process this command.
9	ERROR_INVALID_BLOCK	The storage control block address is invalid.
10	ERROR_BAD_ENVIRONMENT	The environment is incorrect.
11	ERROR_BAD_FORMAT	An attempt was made to load a program with an incorrect format.
12	ERROR_INVALID_ACCESS	The access code is invalid.
13	ERROR_INVALID_DATA	The data is invalid.
14	ERROR_OUTOFMEMORY	Not enough storage is available to complete this operation.
15	ERROR_INVALID_DRIVE	The system cannot find the drive specified.
16	ERROR_CURRENT_DIRECTORY	The directory cannot be removed.
17	ERROR_NOT_SAME_DEVICE	The system cannot move the file to a different disk drive.
18	ERROR_NO_MORE_FILES	There are no more files.
19	ERROR_WRITE_PROTECT	The media is write protected.
20	ERROR_BAD_UNIT	The system cannot find the device specified.
21	ERROR_NOT_READY	The device is not ready.
22	ERROR_BAD_COMMAND	The device does not recognize the command.
23	ERROR_CRC	Data error (cyclic redundancy check).
24	ERROR_BAD_LENGTH	The program issued a command but the command length is incorrect.

25	ERROR_SEEK	The drive cannot locate a specific area or track on the disk.
26	ERROR_NOT_DOS_DISK	The specified disk or diskette cannot be accessed.
27	ERROR_SECTOR_NOT_FOUND	The drive cannot find the sector requested.
28	ERROR_OUT_OF_PAPER	The printer is out of paper.
29	ERROR_WRITE_FAULT	The system cannot write to the specified device.
30	ERROR_READ_FAULT	The system cannot read from the specified device.
31	ERROR_GEN_FAILURE	A device attached to the system is not functioning.
32	ERROR_SHARING_VIOLATION	The process cannot access the file because it is being used by another process.
33	ERROR_LOCK_VIOLATION	The process cannot access the file because another process has locked a portion of the file.
34	ERROR_WRONG_DISK	The wrong diskette is in the drive. Insert %2 (Volume Serial Number: %3) into drive %1.
36	ERROR_SHARING_BUFFER_EXCEEDED	Too many files opened for sharing.
38	ERROR_HANDLE_EOF	Reached the end of the file.
39	ERROR_HANDLE_DISK_FULL	The disk is full.
87	ERROR_INVALID_PARAMETER	The parameter is incorrect.
112	ERROR_DISK_FULL	The disk is full.
123	ERROR_INVALID_NAME	The file name, directory name, or volume label syntax is incorrect.
1450	ERROR_NO_SYSTEM_RESOURCES	Insufficient system resources exist to complete the requested service.

6.4.1.1.2.1.3 SQLSTATE

SQLSTATE is a code which identifies SQL error conditions. It composed by five characters, which can be numbers or uppercase ASCII letters. An SQLSTATE value consists of a class (first two characters) and a subclass (last three characters).

There are three important standard classes. They all indicate in which logical group of errors the condition falls. They match to a particular keyword which can be used with DECLARE HANDLER. Also, the SQLSTATE class determines the default value for the MYSQL_ERRNO and MESSAGE_TEXT condition properties.

- '00' means 'success'. It can not be set in any way, and can only be read via the API.
- '01' contains all warnings, and matches to the SQLWARNING keyword. The default MYSQL_ERRNO is 1642 and default MESSAGE_TEXT is 'Unhandled user-defined warning condition'.
- '02' is the NOT FOUND class. The default MYSQL_ERRNO is 1643 and default MESSAGE_TEXT is 'Unhandled user-defined not found condition'.
- All other classes match the SQLEXCEPTION keyword. The default MYSQL_ERRNO is 1644 and default MESSAGE_TEXT is 'Unhandled user-defined exception condition'.

The subclass, if it is set, indicates a particular condition, or a particular group of conditions within the class. However the '000' sequence means 'no subclass'.

For example, if you try to SELECT from a table which does not exist, a 1109 error is produced, with a '42S02' SQLSTATE. '42' is the class and 'S02' is the subclass. This value matches to the SQLEXCEPTION keyword. When FETCH is called for a cursor which has already reached the end, a 1329 error is produced, with a '02000' SQLSTATE. The class is '02' and there is no subclass (because '000' means 'no subclass'). It can be handled by a NOT FOUND handlers.

The standard SQL specification says that classes beginning with 0, 1, 2, 3, 4, A, B, C, D, E, F and G are reserved for standard-defined classes, while other classes are vendor-specific. It also says that, when the class is standard-defined, subclasses starting with those characters (except for '000') are standard-defined subclasses, while other subclasses are vendor-defined. However, MariaDB and MySQL do not strictly obey this rule.

To read the SQLSTATE of a particular condition which is in the diagnostics area, the GET DIAGNOSTICS statement can be used: the property is called RETURNED_SQLSTATE. For user-defined conditions (SIGNAL and RESIGNAL statements), a SQLSTATE value must be set via the SQLSTATE clause. However, SHOW WARNINGS and SHOW ERRORS do not display the SQLSTATE.

For user-defined conditions, MariaDB and MySQL recommend the '45000' SQLSTATE class.

'HY000' is called the "general error": it is the class used for builtin conditions which do not have a specific SQLSTATE class.

A partial list of error codes and matching SQLSTATE values can be found in the page MariaDB Error Codes.

6.4.1.1.2.2 Progress Reporting

Contents

MariaDB supports progress reporting for some long running commands.

What is Progress Reporting?

Progress reporting means that:

- There is a `Progress` column in SHOW PROCESSLIST which shows the total progress (0-100%)
- INFORMATION_SCHEMA.PROCESSLIST has three columns which allow you to see in which process stage we are and how much of that stage is completed:
  - `STAGE`
  - `MAX_STAGE`
  - `PROGRESS` (within current stage).
- The client receives progress messages which it can display to the user to indicate how long the command will take.

We have separate progress reporting for stages because different stages take different amounts of time.

Supported Commands

Currently, the following commands can send progress report messages to the client:

- ALTER TABLE
- CREATE INDEX
- DROP INDEX
- LOAD DATA INFILE (not `LOAD DATA LOCAL INFILE`, as in that case we don't know the size of the file).

Some Aria storage engine operations also support progress messages:

- CHECK TABLE
- REPAIR TABLE
- ANALYZE TABLE
- OPTIMIZE TABLE

Limitations

Although the above commands support progress reporting, there are some limitations to what progress is reported. To be specific, when executing one of these commands against an InnoDB table with ALGORITHM=INPLACE (which is the default in MariaDB 10.0+), progress is only reported during the merge sort phase while reconstructing indexes.

Enabling and Disabling Progress Reporting

`mysqld` (the MariaDB server) automatically sends progress report messages to clients that support the new protocol, using the value of the progress_report_time variable. They are sent every max(`global.progress_report_time` , `progress_report_time`) seconds (by default 5). You can disable the sending of progress report messages to the client by setting either the local variable (affects only the current connection) or the global variable (affects all connections) to `0` .

If the extra column in `SHOW PROCESSLIST` gives you a compatibility problem, you can disable it by starting `mysqld` with the `--old` flag.

Clients Which Support Progress Reporting

- The `mariadb` command line client
- The `mytop` that comes with MariaDB has a `'%'` column which shows the progress.

Progress Reporting in the mysql Command Line Client

Progress reporting is enabled by default in the mariadb client. You can disable it with `--disable-progress-reports`. It is automatically disabled in batch mode.

When enabled, for every supported command you get a progress report like:

```
ALTER TABLE my_mail ENGINE=aria;
Stage: 1 of 2 'copy to tmp table'  5.37% of stage done
```

This is updated every progress_report_time seconds (the default is 5). If the global `progress_report_time` is higher, this will be used. You can also disable error reporting by setting the variable to `0`.

How to Add Support for Progress Reporting to a Client

You need to use the MariaDB 5.3 or later client library. You can check that the library supports progress reporting by doing:

```
#ifdef CLIENT_PROGRESS
```

To enable progress reporting to the client you need to add `CLIENT_PROGRESS` to the `connect_flag` in `mysql_real_connect()`:

```
mysql_real_connect(mysql, host, user, password,
                   database, opt_mysql_port, opt_mysql_unix_port,
                   connect_flag | CLIENT_PROGRESS);
```

Then you need to provide a callback function for progress reports:

```
static void report_progress(const MYSQL *mysql, uint stage, uint max_stage,
                            double progress, const char *proc_info,
                            uint proc_info_length);

mysql_options(&mysql, MYSQL_PROGRESS_CALLBACK, (void*) report_progress);
```

The above `report_progress` function will be called for each progress message.

This is the implementation used by `mysql.cc`:

```
uint last_progress_report_length;

static void report_progress(const MYSQL *mysql, uint stage, uint max_stage,
                            double progress, const char *proc_info,
                            uint proc_info_length)
{
  uint length= printf("Stage: %d of %d '%.*s' %6.3g%% of stage done",
                      stage, max_stage, proc_info_length, proc_info,
                      progress);
  if (length < last_progress_report_length)
    printf("%*s", last_progress_report_length - length, "");
  putc('\r', stdout);
  fflush(stdout);
  last_progress_report_length= length;
}
```

If you want only one number for the total progress, you can calculate it with:

```
double total_progress=
 ((stage -1) / (double) max_stage * 100.00 + progress / max_stage);
```

Note: `proc_info` is totally independent of stage. You can have many different `proc_info` values within a stage. The idea behind `proc_info` is to give the user more information about what the server is doing.

How to Add Support for Progress Reporting to a Storage Engine

The functions to use for progress reporting are:

```
void thd_progress_init(MYSQL_THD thd, unsigned int max_stage);
```

Initialize progress reporting with stages. This is mainly used for commands that are totally executed within the engine, like `CHECK TABLE`. You should not use this for operations that could be called by, for example, `ALTER TABLE` as this has already called the function.

`max_stage` is the number of stages your storage engine will have.

```
void thd_progress_report(MYSQL_THD thd, unsigned long long progress,
                         unsigned long long max_progress);
```

The above is used for reporting progress.

- `progress` is how much of the file/rows/keys you have gone through.
- `max_progress` is the max number of rows you will go through.

You can call this with varying numbers, but normally the ratio `progress/max_progress` should be increasing.

This function can be called even if you are not using stages, for example when enabling keys as part of `ALTER TABLE` or `ADD INDEX`.

```
void thd_progress_next_stage(MYSQL_THD thd);
```

To go to the next stage in a multi-stage process initiated by `thd_progress_init()`:

```
void thd_progress_end(MYSQL_THD thd);
```

End progress reporting; Sets 'Progress' back to 0 in `SHOW PROCESSLIST`.

```
const char *thd_proc_info(thd, 'stage name');
```

This sets the name of the current status/stage that is displayed in `SHOW PROCESSLIST` and in the client. It's recommended that you call this between stages and thus before `thd_progress_report()` and `thd_progress_next_stage()`.

This functions returns the last used `proc_info`. It's recommended that you restore `proc_info` to its original value when you are done processing.

Note: `thd_proc_info()` is totally independent of stage. You can have many different `proc_info` values within a stage to give the user more information about what is going on.

Examples to Look at in the MariaDB Source:

- `client/mysql.cc` for an example of how to use reporting.
- `libmysql/client.c:cli_safe_read()` to see how progress packets are handled in client
- `sql/protocol.cc::net_send_progress_packet()` for how progress packets are handled in server.

Format of Progress Packets

The progress packet is sent as an error packet with error number `65535`.

It contains the following data (in addition to the error header):

Option	Number of bytes	Other info

1	1	Number of strings. For future
Stage	1	Stage from 1 - Max_stage
Max_stage	1	Max number of stages
Progress	3	Progress in `% * 1000`
Status_length	1-2	Packet length of string in `net_field_length()` format
Status	Status_length	Status / Stage name

6.4.1.2 EXPLAIN FORMAT=JSON in MySQL

Contents

There are some things that we in MariaDB are not happy with in MySQL/Oracle's implementation of EXPLAIN FORMAT=JSON.

The most important issues are already fixed in MariaDB's EXPLAIN FORMAT=JSON implementation. See EXPLAIN FORMAT=JSON Differences From MySQL for details.

This page lists things are are not fixed yet.

Higher priority

- Better display for ORDER/GROUP BY (MDEV-6995)
- Better display for Batched Key Access plans (Plain join buffering is fixed already)

Nice to have

Show ranges being scanned

Currently, one can only find the ranges produced by the range optimizer by looking into optimizer_trace. It would be nice if EXPLAIN showed them, too

```
MySQL [dbt3sf1]> explain format=json select * from customer where c_acctbal < -1000 \G
*************************** 1. row ***************************
EXPLAIN: {
  "query_block": {
    "select_id": 1,
    "table": {
      "table_name": "customer",
      "access_type": "range",
      "possible_keys": [
        "c_acctbal",
        "i_c_acctbal_nationkey"
      ],
      "key": "c_acctbal",
      "used_key_parts": [
        "c_acctbal"
      ],
      "key_length": "9",
      "rows": 1,
      "filtered": 100,
      "index_condition": "(`dbt3sf1`.`customer`.`c_acctbal` < -(1000))"
    }
  }
}
```

Low priority

Filesort/priority queue

Neither version of EXPLAIN in 5.6 shows the "filesort with small limit" optimization. See MDEV-6430.

7 MariaDB Server Releases

You can find the release dates for all upcoming MariaDB Server releases here.

MariaDB Server 11.3
The current short-term development release series.

MariaDB Server 11.2
A current short-term stable series, maintained until November 2024.

MariaDB Server 11.1
A previous short-term stable release series, maintained until August 2024.

MariaDB Server 11.0
A previous short-term MariaDB stable release series, maintained until June 2024.

MariaDB Server 10.11
The current long-term MariaDB stable release series, maintained until February 2028.

MariaDB Server 10.10
A previous short-term MariaDB stable release series, no longer maintained.

MariaDB Server 10.9
A previous short-term MariaDB stable release series, no longer maintained.

MariaDB Server 10.8
A previous short-term MariaDB stable release series, no longer maintained.

MariaDB Server 10.7
A previous short-term MariaDB stable release series, no longer maintained.

MariaDB Server 10.6
A previous long-term MariaDB stable release series, maintained until July 2026.

MariaDB Server 10.5
A previous MariaDB stable release series, maintained until 24 June 2025.

MariaDB Server 10.4
A previous MariaDB stable release series, maintained until 18 June 2024.

MariaDB Server 10.3
A previous MariaDB stable release series, no longer maintained.

MariaDB Server 10.2
A previous MariaDB stable release series, no longer maintained.

MariaDB Server 10.1
A previous stable MariaDB release series, no longer maintained.

MariaDB Server 10.0
A previous stable MariaDB release series, no longer maintained.

MariaDB Server 5.5
A previous stable MariaDB release series, no longer maintained.

MariaDB Server 5.3
A previous stable MariaDB release series, no longer maintained.

MariaDB Server 5.2
A previous stable MariaDB release series, no longer maintained.

MariaDB Server 5.1
A previous stable MariaDB release series, no longer maintained.

Release Notes
Notes regarding MariaDB releases.

Changelogs
MariaDB Changelogs.

MariaDB Release Model
Current MariaDB Release Model (from 10.7 up) Releases happen four times a y...

MariaDB Release Criteria
Alpha, Beta, Gamma and Stable releases.

MariaDB Security Bug Fixing Policy
Bug fixing policy and how security issues are handled.

MariaDB Maintenance Policy
Information on the MariaDB Software Maintenance Policy.

MariaDB Platform Deprecation Policy
Information on MariaDB's Software Deprecation Policy and Schedule.

MariaDB Feature Deprecation Policy
MariaDB Server policy for removing deprecated features.

Release Process [obsolete]
The release process for MariaDB.

Release Coordinator [obsolete]
Project release coordinator

There are 4 related questions.

7.0.0.1 MariaDB Server 11.3

Changes and Improvements in MariaDB 11.3
Current Version: 11.3.1 | Status: RC | Release Date: 21 Nov 2023

Release Notes - MariaDB 11.3 Series
MariaDB 11.3 series release notes.

7.0.0.2 Changes and Improvements in MariaDB 11.3

The most recent release of MariaDB 11.3 is:
MariaDB 11.3.1 (RC) Download Now

Contents

MariaDB 11.3 is a current short-term development series, and will be maintained for one year after its Generally Available release. See Plans for MariaDB 11.3.

New Features & Improvements

Connection Redirection

- Added a redirect mechanism using the redirect_url system variable (MDEV-15935)
- At the moment client-side support is missing

InnoDB

- Shrink temporary tablespaces without restart by setting the innodb_truncate_temporary_tablespace_now system variable. (MDEV-28699)

Replication

- Add keywords "SQL_BEFORE_GTIDS" and "SQL_AFTER_GTIDS" for START SLAVE UNTIL (MDEV-27247). SQL_BEFORE_GTIDS stops the replica when it sees gtids of the option's argument list, without executing them.

SSL/TLS

- SSL is now enabled in the server by default. No configuration necessary, if no server certificate was provided a self-signed certificate will be automatically generated by the server. See Mission Impossible: Zero-Configuration SSL on mariadb.org (MDEV-31856).
- Clients now can validate self-signed server certificates if the mysql_native_password or ed25519 authentication is used and account password is not empty (MDEV-31855).
- Clients now require SSL and have --ssl-verify-server-cert enabled by default (MDEV-31857).
- Replication clients do that too, MASTER_SSL_VERIFY_SERVER_CERT is enabled by default.
- Use `--disable-ssl` or `--disable-ssl-verify-server-cert` to revert to the old behavior.
- Clients can use new command line options --tls-fp and --tls-fplist to verify the server certificate by its fingerprint

Data Types

- INET4 data types can now be cast into INET6 types (MDEV-31626)
- This means INET4 values can be compared with INET6 values and can be inserted into INET6 columns; the server can automatically convert INET4 value into INET6 as needed.

Functions

- Key derivation function KDF for generating good encryption keys for AES_ENCRYPT (MDEV-31474)

Date and Time

- DATE_FORMAT function can now print the current time zone abbreviation and current time zone offset from UTC with `%Z` and `%z` format specifiers. (MDEV-31684)

Optimizer

- Queries like UCASE(varchar_col)=... can now use an index on `varchar_col` if its collation is case insensitive. An optimizer_switch option, sargable_casefold=ON, has been added to enable this optimization. (MDEV-31496)

Privileges

- Add a new database-level privilege, `SHOW CREATE ROUTINE` that allows one to see the routine definition even if the user isn't the routine owner (MDEV-29167)

Processlist

- Added a `SENT_ROWS` column to the Information Schema PROCESSLIST table, as well as extended the display size for the columns in processlist to ensure that most results will fit in display (MDEV-3953)

Application-Time Periods

- Add views for periods in information_schema (MDEV-22597), in particular
- New view INFORMATION_SCHEMA.PERIODS
- New view INFORMATION_SCHEMA.KEY_PERIOD_USAGE
- New columns `IS_SYSTEM_TIME_PERIOD_START` and `IS_SYSTEM_TIME_PERIOD_END` in the INFORMATION_SCHEMA.COLUMNS view

OLD_MODE

- Setting a non-default old_mode value will now always issue a deprecation warning (MDEV-31811)

Mariabackup

- mariabackup --innobackupex mode has been deprecated (MDEV-31505)

Spider

- The Spider storage engine now supports table options instead of having to encode them in COMMENT/CONNECTION strings. When any table option is specified, Spider will ignore COMMENT/CONNECTION strings at the same table/partition/subpartition. A new variable spider_ignore_comments is introduced to ignore them globally at all levels (table/partition/subpartition). Another variable, spider_suppress_comment_ignored_warning, is introduced to suppress warnings when Spider ignores COMMENT/CONNECTION strings. (MDEV-28856)

Removed

The following deprecated features and system variables have been removed (MDEV-32104):

- debug (deprecated since MariaDB 5.5.37)
- sr_YU locale (deprecated since MariaDB 10.0.11)
- "engine_condition_pushdown" in optimizer_switch (deprecated since MariaDB 10.1.1)
- date_format, datetime_format, time_format, max_tmp_tables (deprecated since MariaDB 10.1.2)
- wsrep_causal_reads (deprecated since MariaDB 10.1.3)
- "parser" in mroonga table comment (deprecated since MariaDB 10.2.11)

Variables

- For a list of all new variables, see System Variables Added in MariaDB 11.3.

List of All MariaDB 11.3 Releases

Date	Release	Status	Release Notes	Changelog

21 Nov 2023	**MariaDB 11.3.1**	RC	Release Notes	Changelog
20 Sep 2023	**MariaDB 11.3.0**	Alpha	Release Notes	

7.0.0.3 Release Notes - MariaDB 11.3 Series

MariaDB 11.3.1 Release Notes

Status: Release Candidate (RC) | Release Date: 21 Nov 2023

MariaDB 11.3.0 Release Notes

Status: Alpha | Release Date: 20 September 2023

7.0.0.4 MariaDB 11.3.1 Release Notes

Download | Release Notes | Changelog | Overview of 11.3

Alternate download from mariadb.org

Release date: 21 Nov 2023

Do not use non-stable (non-GA) releases in production!

MariaDB 11.3 is a current short-term development series of MariaDB, and will be maintained for one year after its Generally Available release. It is an evolution of MariaDB 11.2 with several entirely new features.

MariaDB 11.3.1 is a ***Release Candidate (RC)*** release.

For an overview of MariaDB 11.3 see the What is MariaDB 11.3? page.

Thanks, and enjoy MariaDB!

Notable Items

InnoDB

- DROP INDEX followed by CREATE INDEX may corrupt data (MDEV-32132)
- ROW_FORMAT=COMPRESSED table: InnoDB: 2048 bytes should have been read. Only 0 bytes read. (MDEV-31875)
- Server aborts during alter operation when table doesn't have foreign index (MDEV-32527)
- row_merge_fts_doc_tokenize() handles FTS plugin parser inconsistently (MDEV-32578)
- InnoDB: tried to purge non-delete-marked record of an index on a virtual column prefix (MDEV-30024)
- lock_row_lock_current_waits counter in information_schema.innodb_metrics may become negative (MDEV-30658)
- SET GLOBAL innodb_max_purge_lag_wait=… hangs if innodb_read_only=ON (MDEV-31813)
- Auto-increment no longer works for explicit FTS_DOC_ID (MDEV-32017)
- Assertion `pos < table->n_def' failed in dict_table_get_nth_col (MDEV-32337)
- innochecksum man pages seem to be inconsistent with the binary (10.2.25) (MDEV-20583)
- innodb_compression_algorithm=0 (none) increments Innodb_num_pages_page_compression_error (MDEV-30825)
- wrong table name in innodb's "row too big" errors (MDEV-32128)
- Optimize is_file_on_ssd() to speedup opening tablespaces on Windows (MDEV-32228)
- Race condition between page write completion and log checkpoint (MDEV-32511)
- After crash recovery, Checksum mismatch + Failing assertion: !i || prev_id + 1 == space_id, (MDEV-31851)
- Deadlock due to log_free_check(), involving trx_purge_truncate_rseg_history() and trx_undo_assign_low() (MDEV-32049)
- Write-ahead logging is broken for freed pages (MDEV-32552)
- X-lock on supremum for prepared transaction for RR (MDEV-30165)
- Assertion failures in log_sort_flush_list upon crash recovery (MDEV-32029)
- Assertion `purge_sys.tail.trx_no <= purge_sys.rseg->last_trx_no()' (MDEV-30100)
- Assertion `index->is_btree() || index->is_ibuf()' failed in btr_search_guess_on_hash (MDEV-30802)
- InnoDB hang in buf_flush_wait_LRU_batch_end() (MDEV-32134)

- InnoDB may hang when running out of buffer pool (MDEV-32588)
- Corrupt index(es) on busy table when using FOREIGN KEY (MDEV-30531)
- InnoDB Recovery doesn't display encryption message when no encryption configuration passed (MDEV-31098)
- Wrong information about innodb_checksum_algorithm in the information_schema.SYSTEM_VARIABLES (MDEV-31473)
- InnoDB may fail to recover after being killed in fil_delete_tablespace() (MDEV-31826)
- Create separate tpool thread for async aio (MDEV-31095)
- UNDO logs still growing for write-intensive workloads (MDEV-32050)
  - Increase the default for innodb_purge_batch_size from 300 to 1000 (MDEV-32050)
  - Deprecate innodb_purge_rseg_truncate_frequency.
  - The motivation for introducing this in MySQL seems to have been to avoid stalls due to freeing undo log pages or truncating undo log tablespaces. In MariaDB, innodb_undo_log_truncate=ON should be a much lighter operation because it will not involve any log checkpoint. (MDEV-32050)
- Slow full index scan in 10.6 vs 10.5 for the (slow) I/O-bound case (MDEV-30986)
- LOAD DATA into InnoDB w/partitions: huge performance loss, affected 10.6+ (MDEV-31835)
- Disable read-ahead for temporary tablespace (MDEV-32145)
- Wrong result of: `WHERE inet6_column IN ('','::1')` (MDEV-31719)

Optimizer

- Crash when HAVING in a correlated subquery references columns in the outer query (MDEV-29731)
- Server crashes at TABLE::add_tmp_key (MDEV-32320)
- Server crashes inside filesort at my_decimal::to_binary (MDEV-32324)
- Assertion `bitmap_is_set(&m_part_info->read_partitions, m_part_spec.start_part)' failed in ha_partition::handle_ordered_index_scan (MDEV-24283)
- Crash when searching for the best split of derived table (MDEV-32064)
- Test case from opt_tvc.test fails with statement memory protection (MDEV-32225)
- Significant slowdown for query with many outer joins (MDEV-32351)
- test_if_skip_sort_order() should catch the join types JT_EQ_REF, JT_CONST and JT_SYSTEM and skip sort order for these (MDEV-32475)
- jointable materialization subquery optimization ignoring errors, then failing ASSERT. (MDEV-31983)
- Server crashes in JOIN::cleanup after erroneous query with view (MDEV-32164)
- Prepared statement return wrong result (missing row) (MDEV-9938)
- Assertion `range->rows >= s->found_records' failed in best_access_path (MDEV-32682)
- Raise notes when an index cannot be used on data type mismatch (MDEV-32203)

Replication

- rpl.rpl_parallel_temptable failure due to incorrect commit optimization of temptables (MDEV-10356)
- Lock wait timeout with INSERT-SELECT, autoinc, and statement-based replication (MDEV-31482)
- strings/ctype-ucs2.c:2336: my_vsnprintf_utf32: Assertion `(n % 4) == 0' failed in my_vsnprintf_utf32 on INSERT (MDEV-32249)
- Assertion fails in MDL_context::acquire_lock upon parallel replication of CREATE SEQUENCE (MDEV-31792)
- SHOW SLAVE STATUS Last_SQL_Errno Race Condition on Errored Slave Restart (MDEV-31177)
- seconds_behind_master is inaccurate for Delayed replication (MDEV-32265)
- detailize the semisync replication magic number error (MDEV-32365)
- Parallel replication deadlock victim preference code errorneously removed (MDEV-31655)
- Parallel replication lags because innobase_kill_query() may fail to interrupt a lock wait (MDEV-32096)
- Missed kill when the SQL driver thread goes to wait for parallel slave worker queues to drain (MDEV-29974)

Galera

- Assertion `` `state() == s_executing || state() == s_prepared || state() == s_committing || state() == s_must_abort || state() == s_replaying' `` failed. (MDEV-24912)
- Assertion `` `state() == s_executing || state() == s_preparing || state() == s_prepared || state() == s_must_abort || state() == s_aborting || state() == s_cert_failed || state() == s_must_replay' `` failed (MDEV-31285)
- wsrep_sst_mariabackup not working on FreeBSD (MDEV-31467)
- Galera library 26.4.16 fails with every server version (MDEV-32024)
- Galera node remains paused after interleaving FTWRLs (MDEV-32282)
- Failed to insert streaming client (MDEV-32051)
- When set at runtime, wsrep_sst_method accepts any value (MDEV-31470)
- galera needs packaging script changes to successfully build (MDEV-32642)
- replication breaks when using optimistic replication and replica is a galera node (MDEV-31833)
- McAfee database vulnerability scan caused MariaDB crash with signal 6 (system abort) (MDEV-27004)

- Assertion `mode_ == m_local || transaction_.is_streaming()' failed in int wsrep::client_state::bf_abort(wsrep::seqno) (MDEV-30217)

Data Definition

- MariaDB crash on calling function (MDEV-23902)
- ASAN errors in grn_obj_unlink / ha_mroonga::clear_indexes upon index operations (MDEV-31970)
- vcol circular references lead to stack overflow (MDEV-31112)
- OPTIMIZE TABLE crash (MDEV-28122)
- Server crashes in Alter_info::add_stat_drop_index upon CREATE TABLE (MDEV-32449)

Scripts and Clients

- mariadb-binlog -T/--table (mysqlbinlog) option (MDEV-25369)
- mariadb-admin (mysqladmin) wrong error with simple_password_check (MDEV-22418)
- mariadb-install-db shows warning on missing directory $pamtooldir/auth_pam_tool_dir (MDEV-32142)
- main.mysql_client_test, main.mysql_client_test_comp failed on ASAN build with error: 5888, status: 23, errno: 2 (MDEV-19369)
- mariadb-install-db (mysql_install_db) doesn't properly grant proxy privileges to all default root user accounts (MDEV-21194)

MariaBackup

- MariaBackup full backup failed with InnoDB: Failing assertion: success in storage/innobase/fil/fil0fil.cc line 657 (MDEV-18200)
- mbstream breaks page compression on XFS (MDEV-25734)

Character Sets, Data Types, Collations

- Prefix keys for CHAR work differently for MyISAM vs InnoDB (MDEV-30048)
- Inconsistent results of DISTINCT with NOPAD (MDEV-30050)
- Assertion `(length % 4) == 0' failed in my_lengthsp_utf32 on INSERT (MDEV-28835)
- Compressed varchar values lost on joins when sorting on columns from joined table(s) (MDEV-31724)
- UBSAN shift exponent X is too large for 64-bit type 'long long int' in sql/field.cc (MDEV-32226)
- Wrong bit encoding using COALESCE (MDEV-32244)
- utf8mb3_key_col=utf8mb4_value cannot be used for ref access (MDEV-32113)

Spider

- Spider UBSAN runtime error: applying non-zero offset x to null pointer in st_spider_param_string_parse::restore_delims (MDEV-31117)
- Segfault when setting spider_delete_all_rows to 0 and delete all rows of a spider table, ASAN heap-use-after-free in spider_db_delete_all_rows (MDEV-31996)
- ASAN errors in spider_fields::free_conn_holder or spider_create_group_by_handler (MDEV-28998)
- ASAN: heap-buffer-overflow & stack-buffer-overflow in spider_db_mbase_row::append_to_str | SIGSEGV's in memmove_avx_unaligned_erms from memcpy in Binary_string::q_append, in Static_binary_string::q_append and my_strntoull10rnd_8bit | Unknown error 12801 (MDEV-29502)

General

- binlog_do_db option breaks importing sql dumps (MDEV-29989)
- Crashes in MDL_key::mdl_key_init with lower-case-table-names=2 (MDEV-32025)
- getting error 'Illegal parameter data types row and bigint for operation '+' ' when using ITERATE in a FOR..DO (MDEV-32275)
- Assertion `arena_for_set_stmt== 0' failed in LEX::set_arena_for_set_stmt upon SET STATEMENT (MDEV-17711)
- main.mysqlcheck fails on ARM with ASAN use-after-poison in my_mb_wc_filename (MDEV-26494)
- Assertion failed: !pfs->m_idle || (state == PSI_SOCKET_STATE_ACTIVE) (MDEV-28561)
- MyISAM wrong server status flags (MDEV-28820)
- Server crashes in check_sequence_fields upon CREATE TABLE .. SEQUENCE=1 AS SELECT .. (MDEV-29771)
- slow log Rows_examined out of range (MDEV-30820)
- " rpm --setugids " breaks PAM authentication (MDEV-30904)
- incorrect examined rows in case of stored function usage (MDEV-31742)
- Compilation failing on MacOS (unknown warning option -Wno-unused-but-set-variable) (MDEV-31890)
- Server crash upon inserting into Mroonga table with compressed column (MDEV-31966)

- hash unique corrupts index on virtual blobs (MDEV-32012)
- insert into an empty table fails with hash unique (MDEV-32015)
- Valgrind/MSAN warnings in dynamic_column_update_move_left (MDEV-32140)
- Memory leak showed in MDEV-6146 test suite (MDEV-32223)
- Test from subselect.test fails with statement memory protection (MDEV-32245)
- Memory leak when executing PS for query with IN subquery (MDEV-32369)
- Allow the setting of Auto_increment on FK referenced columns (MDEV-32018)
- mariadb-upgrade fails with sql_safe_updates = on (MDEV-29914)
- Assertion `!(thd->server_status & (1U | 8192U))' failed in MDL_context::release_transactional_locks (MDEV-32541)
- Information schema leaks table names and structure to unauthorized users (MDEV-32500)
- Missing CHACHA20-POLY1305 support in WolfSSL (MDEV-31653)
- incorrect error about cyclic reference about JSON type virtual column (MDEV-32586)
- Disable TLS v1.0 and 1.1 for MariaDB (MDEV-31369)
- Better indication of refusing to start because of ProtectHome (MDEV-25177)
- Database upgrade fails: slow_log table (MDEV-27757)
- myrocks_hotbackup.1 and test suite files installed when engine is disabled (MDEV-29993)
- client_ed25519.dll isn't inluded for HeidiSQL. (MDEV-31315)
- Assertion `!m_null_value' failed in int FixedBinTypeBundle<FbtImpl>::cmp_item_fbt::compare or in cmp_item_inet6::compare (MDEV-27207)
- LeakSanitizer errors in get_quick_select or Assertion `status_var.local_memory_used == 0 || !debug_assert_on_not_freed_memory' failed (MDEV-32476)
- Update signal hander user info more compassion and correct url (MDEV-32535)
- maria-install-db fails on MacOS (MDEV-31871)
- MSAN / Valgrind errors in Item_func_like::get_mm_leaf upon query from partitioned table (MDEV-32388)
- MSAN / Valgrind errors in Item_func_like::get_mm_leaf with temporal field (MDEV-32531)
- ASAN errors in base_list_iterator::next / setup_table_map upon 2nd execution of PS (MDEV-32656)
- safe_mutex: Found wrong usage of mutex 'LOCK_thd_data' and 'wait_mutex' (MDEV-32728)

Changelog

For a complete list of changes made in MariaDB 11.3.1, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 11.3.1, see the MariaDB Foundation release announcement.

Do not use non-stable (non-GA) releases in production!

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.0.4.1 MariaDB 11.3.0 Release Notes

Release date: 20 September 2023

Do not use *alpha* releases in production!

MariaDB 11.3 is a current short-term development series of MariaDB, and will be maintained for one year after its Generally Available release. It is an evolution of MariaDB 11.2 with several entirely new features.

MariaDB 11.3.0 is a single preview release. Features are to be considered preview, and none are guaranteed to make it into MariaDB 11.3.

The preview is available as a container **quay.io/mariadb-foundation/mariadb-devel:11.3-preview**.

For an overview of MariaDB 11.3 see the What is MariaDB 11.3? page.

Thanks, and enjoy MariaDB!

Connection Redirection

- Added a redirect mechanism using the redirect_url system variable (MDEV-15935)
- At the moment client-side support is missing

InnoDB

- Shrink temporary tablespaces without restart by setting the innodb_truncate_temporary_tablespace_now system variable. (MDEV-28699)

Replication

- Add keywords "SQL_BEFORE_GTIDS" and "SQL_AFTER_GTIDS" for START SLAVE UNTIL (MDEV-27247). SQL_BEFORE_GTIDS stops the replica when it sees gtids of the option's argument list, without executing them.

SSL/TLS

- SSL is now enabled in the server by default. No configuration necessary, if no server certificate was provided a self-signed certificate will be automatically generated by the server. See Mission Impossible: Zero-Configuration SSL on mariadb.org (MDEV-31856).
- Clients now can validate self-signed server certificates if the mysql_native_password or ed25519 authentication is used and account password is not empty (MDEV-31855).
- Clients now require SSL and have --ssl-verify-server-cert enabled by default (MDEV-31857).
- Replication clients do that too, MASTER_SSL_VERIFY_SERVER_CERT is enabled by default.
- Use `--disable-ssl` or `--disable-ssl-verify-server-cert` to revert to the old behavior.
- Clients can use new command line options --tls-fp and --tls-fplist to verify the server certificate by its fingerprint

Data Types

- INET4 data types can now be cast into INET6 types (MDEV-31626)
- This means, INET4 values can be compared with INET6 values and can be inserted into INET6 columns, the server can automatically convert INET4 value into INET6 as needed.

Functions

- Key derivation function KDF for generating good encryption keys for AES_ENCRYPT (MDEV-31474)

Date and Time

- DATE_FORMAT function can now print the current time zone abbreviation and current time zone offset from UTC with `%Z` and `%z` format specifiers. (MDEV-31684)

Optimizer

- Queries like UCASE(varchar_col)=... can now use an index on `varchar_col` if its collation is case insensitive. An optimizer_switch option, sargable_casefold=ON, has been added to enable this optimization. (MDEV-31496)

Privileges

- Add a new database-level privilege, `SHOW CREATE ROUTINE` that allows to see the routine definition even if the user isn't the routine owner (MDEV-29167)

Processlist

- Added a `SENT_ROWS` column to the Information Schema PROCESSLIST table, as well as extended the display size for the columns in processlist to ensure that most results will fit in display (MDEV-3953)

Application-Time Periods

- Add views for periods in information_schema (MDEV-22597), in particular
- New view INFORMATION_SCHEMA.PERIODS
- New view INFORMATION_SCHEMA.KEY_PERIOD_USAGE
- New columns `IS_SYSTEM_TIME_PERIOD_START` and `IS_SYSTEM_TIME_PERIOD_END` in the INFORMATION_SCHEMA.COLUMNS view

OLD_MODE

- Setting a non-default old_mode value will now always issue a deprecation warning (MDEV-31811)

Mariabackup

- mariabackup --innobackupex mode has been deprecated (MDEV-31505)

Spider

- The Spider storage engine now supports table options instead of having to encode them in COMMENT/CONNECTION strings. When any table option is specified, Spider will ignore COMMENT/CONNECTION strings at the same table/partition/subpartition. A new variable spider_ignore_comments is introduced to ignore them globally at all levels (table/partition/subpartition). Another variable, spider_suppress_comment_ignored_warning, is introduced to suppress warnings when Spider ignores COMMENT/CONNECTION strings. (MDEV-28856)

Removed

The following deprecated features and system variables have been removed (MDEV-32104):

- debug (deprecated since MariaDB 5.5.37)
- sr_YU locale (deprecated since MariaDB 10.0.11)
- "engine_condition_pushdown" in optimizer_switch (deprecated since MariaDB 10.1.1)
- date_format, datetime_format, time_format, max_tmp_tables (deprecated since MariaDB 10.1.2)
- wsrep_causal_reads (deprecated since MariaDB 10.1.3)
- "parser" in mroonga table comment (deprecated since MariaDB 10.2.11)

7.0.0.5 MariaDB Server 11.2

Changes and Improvements in MariaDB 11.2

Current Version: 11.2.2 | Status: Stable (GA) | Release Date: 21 Nov 2023

Release Notes - MariaDB 11.2 Series

MariaDB 11.2 series release notes.

7.0.0.6 Changes and Improvements in MariaDB 11.2

The most recent release of MariaDB 11.2 is:
MariaDB 11.2.2 Stable (GA) Download Now
Alternate download from mariadb.org

Contents

MariaDB 11.2 is a current short-term release series, maintained until November 2024.

New Features & Improvements

Online Schema Change

- ALTER TABLE can now do most operations with `ALGORITHM=COPY, LOCK=NONE`, that is, in most cases, unless the algorithm and lock level are explicitly specified, `ALTER TABLE` will be performed using the COPY algorithm while simultaneously allowing concurrent DML statements on the altered table.

InnoDB

- The InnoDB system tablespace is now shrunk by reclaiming unused space at startup (MDEV-14795)

JSON

- JSON_TABLE now allows retrieval of the key when iterating on JSON objects (MDEV-30145)
- New functions JSON_OBJECT_FILTER_KEYS, JSON_OBJECT_TO_ARRAY and JSON_ARRAY_INTERSECT to check for JSON intersection (MDEV-26182)

Miscellaneous

- All binlog* variables are now visible as system variables, specifically binlog_do_db, binlog_ignore_db, binlog_row_event_max_size (MDEV-30188)
- ALTER TABLE IMPORT enhancement (MDEV-26137)
- Temporary tables are now displayed in the Information Schema TABLES Table, SHOW TABLES and SHOW TABLE STATUS (MDEV-12459)
- Stored programs: validation of stored program statements (MDEV-5816)
- Remove the deprecated old_alter_table variable (MDEV-30905)
- Extend AES_ENCRYPT() and AES_DECRYPT() to support an initialization vector and algorithm (MDEV-9069)

Variables

- For a list of all new variables, see System Variables Added in MariaDB 11.2.

List of All MariaDB 11.2 Releases

Date	Release	Status	Release Notes	Changelog
21 Nov 2023	**MariaDB 11.2.2**	Stable (GA)	Release Notes	Changelog
21 Aug 2023	**MariaDB 11.2.1**	RC	Release Notes	Changelog
20 Jun 2023	**MariaDB 11.2.0**	Alpha	Release Notes	

7.0.0.7 Release Notes - MariaDB 11.2 Series

MariaDB 11.2.2 Release Notes
Status: Stable (GA) | Release Date: 21 Nov 2023

MariaDB 11.2.1 Release Notes
Status: Release Candidate (RC) | Release Date: 21 Aug 2023

MariaDB 11.2.0 Release Notes
Status: Alpha | Release Date: 20 Jun 2023

7.0.0.7.1 MariaDB 11.2.2 Release Notes

Download | Release Notes | Changelog | Overview of 11.2

Alternate download from mariadb.org

Release date: 21 Nov 2023

MariaDB 11.2 is a current short-term stable series of MariaDB, maintained until November 2024. It is an evolution of MariaDB 11.1 with several entirely new features.

MariaDB 11.2.2 is a ***Stable (GA)*** release.

For an overview of MariaDB 11.1 see the What is MariaDB 11.2? page.

Thanks, and enjoy MariaDB!

Notable Items

InnoDB

- DROP INDEX followed by CREATE INDEX may corrupt data (MDEV-32132)
- ROW_FORMAT=COMPRESSED table: InnoDB: 2048 bytes should have been read. Only 0 bytes read. (MDEV-31875)
- Server aborts during alter operation when table doesn't have foreign index (MDEV-32527)
- row_merge_fts_doc_tokenize() handles FTS plugin parser inconsistently (MDEV-32578)
- InnoDB: tried to purge non-delete-marked record of an index on a virtual column prefix (MDEV-30024)
- lock_row_lock_current_waits counter in information_schema.innodb_metrics may become negative (MDEV-30658)
- SET GLOBAL innodb_max_purge_lag_wait=… hangs if innodb_read_only=ON (MDEV-31813)
- Auto-increment no longer works for explicit FTS_DOC_ID (MDEV-32017)
- Assertion `pos < table->n_def' failed in dict_table_get_nth_col (MDEV-32337)
- innochecksum man pages seem to be inconsistent with the binary (10.2.25) (MDEV-20583)
- innodb_compression_algorithm=0 (none) increments Innodb_num_pages_page_compression_error (MDEV-30825)
- wrong table name in innodb's "row too big" errors (MDEV-32128)
- Optimize is_file_on_ssd() to speedup opening tablespaces on Windows (MDEV-32228)
- Race condition between page write completion and log checkpoint (MDEV-32511)
- After crash recovery, Checksum mismatch + Failing assertion: !i || prev_id + 1 == space_id, (MDEV-31851)
- Deadlock due to log_free_check(), involving trx_purge_truncate_rseg_history() and trx_undo_assign_low() (MDEV-32049)
- Write-ahead logging is broken for freed pages (MDEV-32552)
- X-lock on supremum for prepared transaction for RR (MDEV-30165)
- Assertion failures in log_sort_flush_list upon crash recovery (MDEV-32029)
- Assertion `purge_sys.tail.trx_no <= purge_sys.rseg->last_trx_no()' (MDEV-30100)
- Assertion `index->is_btree() || index->is_ibuf()' failed in btr_search_guess_on_hash (MDEV-30802)
- InnoDB hang in buf_flush_wait_LRU_batch_end() (MDEV-32134)
- InnoDB may hang when running out of buffer pool (MDEV-32588)
- Corrupt index(es) on busy table when using FOREIGN KEY (MDEV-30531)
- InnoDB Recovery doesn't display encryption message when no encryption configuration passed (MDEV-31098)
- Wrong information about innodb_checksum_algorithm in the information_schema.SYSTEM_VARIABLES (MDEV-31473)
- InnoDB may fail to recover after being killed in fil_delete_tablespace() (MDEV-31826)
- Create separate tpool thread for async aio (MDEV-31095)

- UNDO logs still growing for write-intensive workloads (MDEV-32050)
  - Increase the default for innodb_purge_batch_size from 300 to 1000 (MDEV-32050)
  - Deprecate innodb_purge_rseg_truncate_frequency.
  - The motivation for introducing this in MySQL seems to have been to avoid stalls due to freeing undo log pages or truncating undo log tablespaces. In MariaDB, innodb_undo_log_truncate=ON should be a much lighter operation because it will not involve any log checkpoint. (MDEV-32050)
- Slow full index scan in 10.6 vs 10.5 for the (slow) I/O-bound case (MDEV-30986)
- LOAD DATA into InnoDB w/partitions: huge performance loss, affected 10.6+ (MDEV-31835)
- Disable read-ahead for temporary tablespace (MDEV-32145)
- Wrong result of: `WHERE inet6_column IN ('','::1')` (MDEV-31719)

Optimizer

- Crash when HAVING in a correlated subquery references columns in the outer query (MDEV-29731)
- Server crashes at TABLE::add_tmp_key (MDEV-32320)
- Server crashes inside filesort at my_decimal::to_binary (MDEV-32324)
- Assertion `bitmap_is_set(&m_part_info->read_partitions, m_part_spec.start_part)' failed in ha_partition::handle_ordered_index_scan (MDEV-24283)
- Crash when searching for the best split of derived table (MDEV-32064)
- Test case from opt_tvc.test fails with statement memory protection (MDEV-32225)
- Significant slowdown for query with many outer joins (MDEV-32351)
- test_if_skip_sort_order() should catch the join types JT_EQ_REF, JT_CONST and JT_SYSTEM and skip sort order for these (MDEV-32475)
- jointable materialization subquery optimization ignoring errors, then failing ASSERT. (MDEV-31983)
- Server crashes in JOIN::cleanup after erroneous query with view (MDEV-32164)
- Prepared statement return wrong result (missing row) (MDEV-9938)
- Assertion `range->rows >= s->found_records' failed in best_access_path (MDEV-32682)
- Raise notes when an index cannot be used on data type mismatch (MDEV-32203)

Replication

- rpl.rpl_parallel_temptable failure due to incorrect commit optimization of temptables (MDEV-10356)
- Lock wait timeout with INSERT-SELECT, autoinc, and statement-based replication (MDEV-31482)
- strings/ctype-ucs2.c:2336: my_vsnprintf_utf32: Assertion `(n % 4) == 0' failed in my_vsnprintf_utf32 on INSERT (MDEV-32249)
- Assertion fails in MDL_context::acquire_lock upon parallel replication of CREATE SEQUENCE (MDEV-31792)
- SHOW SLAVE STATUS Last_SQL_Errno Race Condition on Errored Slave Restart (MDEV-31177)
- seconds_behind_master is inaccurate for Delayed replication (MDEV-32265)
- detailize the semisync replication magic number error (MDEV-32365)
- Parallel replication deadlock victim preference code errorneously removed (MDEV-31655)
- Parallel replication lags because innobase_kill_query() may fail to interrupt a lock wait (MDEV-32096)
- Missed kill when the SQL driver thread goes to wait for parallel slave worker queues to drain (MDEV-29974)

Galera

- Assertion `state() == s_executing || state() == s_prepared || state() == s_committing || state() == s_must_abort || state() == s_replaying' failed. (MDEV-24912)
- Assertion `state() == s_executing || state() == s_preparing || state() == s_prepared || state() == s_must_abort || state() == s_aborting || state() == s_cert_failed || state() == s_must_replay' failed (MDEV-31285)
- wsrep_sst_mariabackup not working on FreeBSD (MDEV-31467)
- Galera library 26.4.16 fails with every server version (MDEV-32024)
- Galera node remains paused after interleaving FTWRLs (MDEV-32282)
- Failed to insert streaming client (MDEV-32051)
- When set at runtime, wsrep_sst_method accepts any value (MDEV-31470)
- galera needs packaging script changes to successfully build (MDEV-32642)
- replication breaks when using optimistic replication and replica is a galera node (MDEV-31833)
- McAfee database vulnerability scan caused MariaDB crash with signal 6 (system abort) (MDEV-27004)
- Assertion `mode_ == m_local || transaction_.is_streaming()' failed in `int wsrep::client_state::bf_abort(wsrep::seqno)` (MDEV-30217)

Data Definition

- MariaDB crash on calling function (MDEV-23902)
- ASAN errors in grn_obj_unlink / ha_mroonga::clear_indexes upon index operations (MDEV-31970)

- vcol circular references lead to stack overflow (MDEV-31112)
- OPTIMIZE TABLE crash (MDEV-28122)
- Server crashes in Alter_info::add_stat_drop_index upon CREATE TABLE (MDEV-32449)

Scripts and Clients

- mariadb-binlog -T/--table (mysqlbinlog) option (MDEV-25369)
- mariadb-admin (mysqladmin) wrong error with simple_password_check (MDEV-22418)
- mariadb-install-db shows warning on missing directory $pamtooldir/auth_pam_tool_dir (MDEV-32142)
- main.mysql_client_test, main.mysql_client_test_comp failed on ASAN build with error: 5888, status: 23, errno: 2 (MDEV-19369)
- mariadb-install-db (mysql_install_db) doesn't properly grant proxy privileges to all default root user accounts (MDEV-21194)

MariaBackup

- MariaBackup full backup failed with InnoDB: Failing assertion: success in storage/innobase/fil/fil0fil.cc line 657 (MDEV-18200)
- mbstream breaks page compression on XFS (MDEV-25734)

Character Sets, Data Types, Collations

- Prefix keys for CHAR work differently for MyISAM vs InnoDB (MDEV-30048)
- Inconsistent results of DISTINCT with NOPAD (MDEV-30050)
- Assertion `(length % 4) == 0' failed in my_lengthsp_utf32 on INSERT (MDEV-28835)
- Compressed varchar values lost on joins when sorting on columns from joined table(s) (MDEV-31724)
- UBSAN shift exponent X is too large for 64-bit type 'long long int' in sql/field.cc (MDEV-32226)
- Wrong bit encoding using COALESCE (MDEV-32244)
- utf8mb3_key_col=utf8mb4_value cannot be used for ref access (MDEV-32113)

Spider

- Spider UBSAN runtime error: applying non-zero offset x to null pointer in st_spider_param_string_parse::restore_delims (MDEV-31117)
- Segfault when setting spider_delete_all_rows to 0 and delete all rows of a spider table, ASAN heap-use-after-free in spider_db_delete_all_rows (MDEV-31996)
- ASAN errors in spider_fields::free_conn_holder or spider_create_group_by_handler (MDEV-28998)
- ASAN: heap-buffer-overflow & stack-buffer-overflow in `spider_db_mbase_row::append_to_str` | SIGSEGV's in `memmove_avx_unaligned_erms` from memcpy in `Binary_string::q_append`, in `Static_binary_string::q_append` and `my_strntoull10rnd_8bit` | Unknown error 12801 (MDEV-29502)

General

- binlog_do_db option breaks importing sql dumps (MDEV-29989)
- Crashes in MDL_key::mdl_key_init with lower-case-table-names=2 (MDEV-32025)
- getting error 'Illegal parameter data types row and bigint for operation '+' ' when using ITERATE in a FOR..DO (MDEV-32275)
- Assertion `arena_for_set_stmt== 0' failed in LEX::set_arena_for_set_stmt upon SET STATEMENT (MDEV-17711)
- main.mysqlcheck fails on ARM with ASAN use-after-poison in my_mb_wc_filename (MDEV-26494)
- Assertion failed: !pfs->m_idle || (state == PSI_SOCKET_STATE_ACTIVE) (MDEV-28561)
- MyISAM wrong server status flags (MDEV-28820)
- Server crashes in check_sequence_fields upon CREATE TABLE .. SEQUENCE=1 AS SELECT .. (MDEV-29771)
- slow log Rows_examined out of range (MDEV-30820)
- " `rpm --setugids` " breaks PAM authentication (MDEV-30904)
- incorrect examined rows in case of stored function usage (MDEV-31742)
- Compilation failing on MacOS (unknown warning option -Wno-unused-but-set-variable) (MDEV-31890)
- Server crash upon inserting into Mroonga table with compressed column (MDEV-31966)
- hash unique corrupts index on virtual blobs (MDEV-32012)
- insert into an empty table fails with hash unique (MDEV-32015)
- Valgrind/MSAN warnings in dynamic_column_update_move_left (MDEV-32140)
- Memory leak showed in MDEV-6146 test suite (MDEV-32223)
- Test from subselect.test fails with statement memory protection (MDEV-32245)
- Memory leak when executing PS for query with IN subquery (MDEV-32369)
- Allow the setting of Auto_increment on FK referenced columns (MDEV-32018)
- mariadb-upgrade fails with sql_safe_updates = on (MDEV-29914)

- Assertion `!(thd->server_status & (1U | 8192U))' failed in MDL_context::release_transactional_locks (MDEV-32541)
- Information schema leaks table names and structure to unauthorized users (MDEV-32500)
- Missing CHACHA20-POLY1305 support in WolfSSL (MDEV-31653)
- incorrect error about cyclic reference about JSON type virtual column (MDEV-32586)
- Disable TLS v1.0 and 1.1 for MariaDB (MDEV-31369)
- Better indication of refusing to start because of ProtectHome (MDEV-25177)
- Database upgrade fails: slow_log table (MDEV-27757)
- myrocks_hotbackup.1 and test suite files installed when engine is disabled (MDEV-29993)
- client_ed25519.dll isn't inluded for HeidiSQL. (MDEV-31315)
- Assertion `!m_null_value' failed in int FixedBinTypeBundle<FbtImpl>::cmp_item_fbt::compare or in cmp_item_inet6::compare (MDEV-27207)
- LeakSanitizer errors in get_quick_select or Assertion `status_var.local_memory_used == 0 || !debug_assert_on_not_freed_memory' failed (MDEV-32476)
- Update signal hander user info more compassion and correct url (MDEV-32535)
- maria-install-db fails on MacOS (MDEV-31871)
- MSAN / Valgrind errors in Item_func_like::get_mm_leaf upon query from partitioned table (MDEV-32388)
- MSAN / Valgrind errors in Item_func_like::get_mm_leaf with temporal field (MDEV-32531)
- ASAN errors in base_list_iterator::next / setup_table_map upon 2nd execution of PS (MDEV-32656)
- safe_mutex: Found wrong usage of mutex 'LOCK_thd_data' and 'wait_mutex' (MDEV-32728)

Docker Official Images

- Invert single and double quotes for sql command definitions in healthcheck.sh due to failure under sql_mode=ANSI_QUOTES - contribution by Dominik Häckel
- healthcheck.sh --no-defaults behaviour was corrected - reported by Dominik Häckel
- Added /docker-entrypoint-init.d for tar{,compression} from mariadb-backup - instructions
- Refactor `docker_mariadb_init` in the entrypoint for extending the MariaDB image
- CIS failure due to world-writable directory /var/run/mysqld, added sticky bit - reported by @ollie1
- Add PROXY privileges for root@MARIADB_ROOT_HOST - reported by Matthieu Gusmini
- healthcheck.sh added --galera_online test, to match what the mariadb-operator does.

Variables

- Added the note_verbosity system variable to manage notes when an index cannot be used.

Security

- Fixes for the following security vulnerabilities:
  - CVE-2023-22084

Changelog

For a complete list of changes made in MariaDB 11.2.2, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 11.2.2, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.0.7.2 MariaDB 11.2.1 Release Notes

The most recent release of MariaDB 11.2 is:
MariaDB 11.2.2 Stable (GA) Download Now

Alternate download from mariadb.org

Download 11.2.1 | Release Notes | Changelog | Overview of 11.2

Release date: 21 Aug 2023

Do not use non-stable (non-GA) releases in production!

MariaDB 11.2 is the current development series of MariaDB. It is an evolution of MariaDB 11.1 with several entirely new features.

MariaDB 11.2 is a ***Release Candidate (RC)*** release.

For an overview of MariaDB 11.1 see the What is MariaDB 11.2? page.

Thanks, and enjoy MariaDB!

Notable Items

General

- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 11.2 for Ubuntu 18.04 LTS "Bionic" and Ubuntu 22.10 "Kinetic"
- In this release repositories for Debian 12 "Bookworm" have been added.
- mariadb-dump --force doesn't ignore error as it should (MDEV-31092)
- 280 Bytes lost in mysys/array.c, mysys/hash.c, sql/sp.cc, sql/sp.cc, sql/item_create.cc, sql/item_create.cc, sql/sql_yacc.yy:10748 when using oracle sql_mode (MDEV-26186)
- SQL/PL package body does not appear in I_S.ROUTINES.ROUTINE_DEFINITION (MDEV-30662)
- Unexpected result when combining DISTINCT, subselect and LIMIT (MDEV-28285)
- ROW variables do not get assigned from subselects (MDEV-31250)
- Crash after setting global session_track_system_variables to an invalid value (MDEV-25237)
- ODKU of non-versioning column inserts history row (MDEV-23100)
- UPDATE not working properly on transaction precise system versioned table (MDEV-25644)
- Assertion `const_item_cache == true' failed in Item_func::fix_fields (MDEV-31319)
- ANALYZE doesn't work with pushed derived tables (MDEV-29284)
- get_partition_set is never executed in ha_partition::multi_range_key_create_key due to bitwise & with 0 constant (MDEV-24712)
- Client can crash the server with a mysql_list_fields("view") call (MDEV-30159)
- I_S.parameters not immediately changed updated after procedure change (MDEV-31064)
- bzero wipes more bytes than necessary in set_global_from_ddl_log_entry (MDEV-31521)
- Assertion `0' failed in Type_handler_row::field_type upon TO_CHAR with wrong argument (MDEV-29152)
- mariadb-upgrade fails due to old_mode="" , with "Cannot load from mysql.proc. The table is probably corrupted" (MDEV-28915)

Character Sets, Data Types

- UBSAN: null pointer passed as argument 1, which is declared to never be null in my_strnncoll_binary on SELECT ... COUNT or GROUP_CONCAT (MDEV-28384)
- Possibly wrong result or Assertion `0' failed in Item_func_round::native_op (MDEV-23838)
- Assertion `(length % 4) == 0' failed in my_lengthsp_utf32 on SELECT (MDEV-29019)
- UBSAN: negation of -X cannot be represented in type 'long long int' ; cast to an unsigned type to negate this value to itself in Item_func_mul::int_op and Item_func_round::int_op (MDEV-30932)
- Assorted assertion failures in json_find_path with certain collations (MDEV-23187)

InnoDB

- Crashing on I/O error is unhelpful (MDEV-27593)
- SIGSEGV in log_sort_flush_list() in InnoDB crash recovery (MDEV-31354)
- InnoDB tables are being flagged as corrupted on an I/O bound server (MDEV-31767)

- Duplicate entry allowed into a UNIQUE column (MDEV-31120)
- Server Status Innodb_row_lock_time% is reported in seconds (MDEV-29311)
- innochecksum dies with Floating point exception (MDEV-31641)
- Add InnoDB engine information to the slow query log (MDEV-31558)
- Deadlock with 3 concurrent DELETEs by unique key (MDEV-10962)
- innodb protection against dual processes accessing data insufficient (MDEV-31568)
- ER_DUP_KEY in mysql.innodb_table_stats upon RENAME on sequence (MDEV-31607)
- Assertion `!strcmp(index->table->name.m_name, "SYS_FOREIGN") || !strcmp(index->table->name.m_name, "SYS_FOREIGN_COLS")' failed in btr_node_ptr_max_size (MDEV-19216)
- InnoDB: Failing assertion: page_type == i_s_page_type[page_type].type_value (MDEV-31386)
- btr_estimate_n_rows_in_range() accesses unfixed, unlatched page (MDEV-30648)
- MODIFY COLUMN can break FK constraints, and lead to unrestorable dumps (MDEV-31086)
- Recovery or backup failure after innodb_undo_log_truncate=ON (MDEV-31487)
- Assertion 'n & PENDING' failed in fil_space_t::set_needs_flush() (MDEV-31442)
- fil_node_open_file() releases fil_system.mutex allowing other thread to open its file node (MDEV-31256)
- ASAN errors in dict_v_col_t::detach upon adding key to virtual column (MDEV-31416)
- Purge trying to access freed secondary index page (MDEV-31264)
- Freed data pages are not always being scrubbed (MDEV-31253)
- InnoDB recovery hangs after reporting corruption (MDEV-31353)
- !cursor->index->is_committed() in row0ins.cc after update to 10.4.13 from 10.3.21 (MDEV-22739)
- innodb_undo_log_truncate=ON fails to wait for purge of enough transaction history (MDEV-31355)
- SET GLOBAL innodb_undo_log_truncate=ON does not free space when no undo logs exist (MDEV-31382)
- innodb_read_ahead_threshold (linear read-ahead) does not work (MDEV-29967)
- InnoDB recovery and mariadb-backup --prepare fail to report detailed progress (MDEV-29911)
- fil_ibd_create() may hijack the file handle of an old file (MDEV-31347)
- innodb_undo_log_truncate=ON recovery results in a corrupted undo log (MDEV-31373)
- Server freeze due to innodb_change_buffering and innodb_file_per_table=0 (MDEV-31088)
- Change buffer entries are left behind when freeing a page, causing secondary index corruption when the page is later reused (MDEV-31385)
- Foreign Key Constraint actions don't affect Virtual Column (MDEV-18114)

Aria

- Various crashes upon INSERT/UPDATE after changing Aria settings (MDEV-28054)
- Various crashes/asserts/corruptions when Aria encryption is enabled/used, but the encryption plugin is not loaded (MDEV-26258)

Spider

- SIGSEGV in spider_db_open_item_field and SIGSEGV in spider_db_print_item_type, on SELECT (MDEV-29447)
- Spider variables that double as table params overriding mechanism is buggy (MDEV-31524)

Optimizer

- ANALYZE FORMAT=JSON now includes InnoDB engine statistics for each table (MDEV-31577)
- Assertion `last_key_entry >= end_pos' failed in virtual bool JOIN_CACHE_HASHED::put_record() (MDEV-31348)
- Problem with open ranges on prefix blobs keys (MDEV-31800)
- Equal on two RANK window functions create wrong result (MDEV-20010)
- Recursive CTE execution is interrupted without errors or warnings (MDEV-31214)
- Assertion `s->table->opt_range_condition_rows <= s->found_records' failed in apply_selectivity_for_table (MDEV-31449)
- Inconsistency between MRR and SQL layer costs can cause poor query plan (MDEV-31479)
- MAX_SEL_ARG memory exhaustion is not visible in the optimizer trace (MDEV-30964)
- SHOW TABLES not working properly with lower_case_table_names=2 (MDEV-30765)
- Segfault on select query using index for group-by and filesort (MDEV-30143)
- Server crash in store_length, assertion failure in Type_handler_string_result::sort_length (MDEV-31743)

Replication

- Parallel Slave SQL Thread Can Update Seconds_Behind_Master with Active Workers (MDEV-30619)

- ALTER SEQUENCE ends up in optimistic parallel slave binlog out-of-order (MDEV-31503)
- STOP SLAVE takes very long time on a busy system (MDEV-13915)
- On slave XA COMMIT/XA ROLLBACK fail to return an error in read-only mode (MDEV-30978)
- Calling a function from a different database in a slave side trigger crashes (MDEV-29894)
- `rpl.rpl_manual_change_index_file` occasionally fails in BB with Result length mismatch (MDEV-30214)

Galera

- Node never returns from Donor/Desynced to Synced when `wsrep_mode = BF_ABORT_MARIABACKUP` (MDEV-31737)
- Node has been dropped from the cluster on Startup / Shutdown with async replica (MDEV-31413)
- KILL QUERY maintains nodes data consistency but breaks GTID sequence (MDEV-31075)
- Assertion failure `!lock.was_chosen_as_deadlock_victim in trx0trx.h:1065` (MDEV-30963)
- MariaDB stuck on starting commit state (waiting on commit order critical section) (MDEV-29293)
- Assertion `state() == s_aborting || state() == s_must_replay` failed in int wsrep::transaction::after_rollback() (MDEV-30013)
- Assertion `!wsrep_has_changes(thd) || (thd->lex->sql_command == SQLCOM_CREATE_TABLE && !thd->is_current_stmt_binlog_format_row()) || thd->wsrep_cs().transaction().state() == wsrep::transaction::s_aborted` failed (MDEV-30388)
- Server crashes when wsrep_sst_donor and wsrep_cluster_address set to NULL (MDEV-28433)
- Create temporary sequence can cause inconsistency (MDEV-31335)
- Galera 4 unable to query cluster state if not primary component (MDEV-21479)

Changelog

For a complete list of changes made in MariaDB 11.2.1, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 11.2.1, see the MariaDB Foundation release announcement.

Do not use non-stable (non-GA) releases in production!

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.0.7.3 MariaDB 11.2.0 Release Notes

The most recent release of MariaDB 11.2 is:
MariaDB 11.2.2 Stable (GA) Download Now
Alternate download from mariadb.org

Download 11.2.0 | Release Notes | Changelog | Overview of 11.2

Release date: 20 Jun 2023

Do not use *alpha* releases in production!

MariaDB 11.2 is a short-term development series of MariaDB, and will be maintained for one year after its Generally Available release. It is an evolution of MariaDB 11.1 with several entirely new features.

MariaDB 11.2.0 is a single preview release. Features are to be considered preview, and none are guaranteed to make it into MariaDB 11.2.

The preview is available as a container **quay.io/mariadb-foundation/mariadb-devel:11.2-preview**.

For an overview of MariaDB 11.2 see the What is MariaDB 11.2? page.

Thanks, and enjoy MariaDB!

InnoDB

- The InnoDB system tablespace is now shrunk by reclaiming unused space at startup (MDEV-14795)

JSON

- JSON_TABLE now allows retrieval of the key when iterating on JSON objects (MDEV-30145)
- New functions JSON_OBJECT_FILTER_KEYS, JSON_OBJECT_TO_ARRAY and JSON_ARRAY_INTERSECT to check for JSON intersection (MDEV-26182)

Miscellaneous

- All binlog* variables are now visible as system variables, specifically binlog_do_db, binlog_ignore_db, binlog_row_event_max_size (MDEV-30188)
- ALTER TABLE IMPORT enhancement (MDEV-26137)
- Temporary tables are now displayed in the Information Schema TABLES Table, SHOW TABLES and SHOW TABLE STATUS (MDEV-12459)
- Stored programs: validation of stored program statements (MDEV-5816)
- Remove the deprecated old_alter_table variable (MDEV-30905)
- Extend AES_ENCRYPT() and AES_DECRYPT() to support an initialization vector and algorithm (MDEV-9069)

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.0.8 MariaDB Server 11.1

Changes and Improvements in MariaDB 11.1
Current Version: 11.1.3 | Status: Stable (GA) | Release Date: 13 Nov 2023

Release Notes - MariaDB 11.1 Series
MariaDB 11.1 series release notes.

Changelogs - MariaDB 11.1 Series
MariaDB 11.1 changelogs

7.0.0.9 Changes and Improvements in MariaDB 11.1

The most recent release of MariaDB 11.1 is:
MariaDB 11.1.3 Stable (GA) Download Now
Alternate download from mariadb.org

Contents

MariaDB 11.1 is a current short-term release series, maintained until August 2024.

New Features & Improvements

JSON

- JSON_SCHEMA_VALID function for validating a JSON schema (MDEV-27128)

Optimizer

- Semi-join optimization for single-table UPDATE/DELETE statements. Update and delete statements that use subqueries can now use all subquery optimization strategies that MariaDB offers, so now if you use subqueries in UPDATE or DELETE, these statements will likely be much faster (MDEV-7487)
- Queries with the DATE or YEAR functions comparing against a constant can now make use of indexes, so these will be noticeably quicker in certain instances. For example `SELECT * FROM t2 WHERE YEAR(a) = 2019` or `SELECT * FROM t2 WHERE DATE(a) <= '2017-01-01'` (MDEV-8320)

Transactions

- The transaction_isolation option is now a system variable, and the tx_isolation system variable is deprecated (MDEV-21921)

InnoDB

- Remove innodb_defragment and related parameters (MDEV-30545)

Mariabackup

- Rename Mariabackup's xtrabackup_* files to mariadb_backup_* (MDEV-18931)

Variables

- For a list of all new variables, see System Variables Added in MariaDB 11.1.

List of All MariaDB 11.1 Releases

Date	Release	Status	Release Notes	Changelog
13 Nov 2023	**MariaDB 11.1.3**	Stable (GA)	Release Notes	Changelog
21 Aug 2023	**MariaDB 11.1.2**	Stable (GA)	Release Notes	Changelog
6 Jun 2023	**MariaDB 11.1.1**	RC	Release Notes	Changelog
27 Mar 2023	**MariaDB 11.1.0**	Alpha	Release Notes	

7.0.0.10 Release Notes - MariaDB 11.1 Series

MariaDB 11.1.3 Release Notes

Status: Stable (GA) | Release Date: 13 Nov 2023

MariaDB 11.1.2 Release Notes

Status: Stable (GA) | Release Date: 21 Aug 2023

MariaDB 11.1.1 Release Notes

Status: Release Candidate (RC) | Release Date: 6 Jun 2023

MariaDB 11.1.0 Release Notes

Status: Alpha | Release Date: 27 Mar 2023

7.0.0.10.1 MariaDB 11.1.3 Release Notes

Download | Release Notes | Changelog | Overview of 11.1

Alternate download from mariadb.org

Release date: 13 Nov 2023

MariaDB 11.1 is a current short-term stable series of MariaDB, maintained until July 2024. It is an evolution of MariaDB 11.1 with several entirely new features.

MariaDB 11.1.3 is a ***Stable (GA)*** release.

For an overview of MariaDB 11.1 see the What is MariaDB 11.1? page.

Thanks, and enjoy MariaDB!

Notable Items

InnoDB

- DROP INDEX followed by CREATE INDEX may corrupt data (MDEV-32132)
- ROW_FORMAT=COMPRESSED table: InnoDB: 2048 bytes should have been read. Only 0 bytes read. (MDEV-31875)
- Server aborts during alter operation when table doesn't have foreign index (MDEV-32527)
- row_merge_fts_doc_tokenize() handles FTS plugin parser inconsistently (MDEV-32578)
- InnoDB: tried to purge non-delete-marked record of an index on a virtual column prefix (MDEV-30024)
- lock_row_lock_current_waits counter in information_schema.innodb_metrics may become negative (MDEV-30658)
- SET GLOBAL innodb_max_purge_lag_wait=… hangs if innodb_read_only=ON (MDEV-31813)
- Auto-increment no longer works for explicit FTS_DOC_ID (MDEV-32017)
- Assertion `pos < table->n_def' failed in dict_table_get_nth_col (MDEV-32337)
- innochecksum man pages seem to be inconsistent with the binary (10.2.25) (MDEV-20583)
- innodb_compression_algorithm=0 (none) increments Innodb_num_pages_page_compression_error (MDEV-30825)
- wrong table name in innodb's "row too big" errors (MDEV-32128)
- Optimize is_file_on_ssd() to speedup opening tablespaces on Windows (MDEV-32228)
- Race condition between page write completion and log checkpoint (MDEV-32511)
- After crash recovery, Checksum mismatch + Failing assertion: !i || prev_id + 1 == space_id, (MDEV-31851)
- Deadlock due to log_free_check(), involving trx_purge_truncate_rseg_history() and trx_undo_assign_low() (MDEV-32049)
- Write-ahead logging is broken for freed pages (MDEV-32552)
- X-lock on supremum for prepared transaction for RR (MDEV-30165)
- Assertion failures in log_sort_flush_list upon crash recovery (MDEV-32029)
- Race condition between page write completion and log checkpoint (MDEV-32511)
- Assertion `purge_sys.tail.trx_no <= purge_sys.rseg->last_trx_no()' (MDEV-30100)
- Assertion `index->is_btree() || index->is_ibuf()' failed in btr_search_guess_on_hash (MDEV-30802)
- InnoDB hang in buf_flush_wait_LRU_batch_end() (MDEV-32134)
- InnoDB may hang when running out of buffer pool (MDEV-32588)
- Corrupt index(es) on busy table when using FOREIGN KEY (MDEV-30531)
- InnoDB Recovery doesn't display encryption message when no encryption configuration passed (MDEV-31098)
- Wrong information about innodb_checksum_algorithm in the information_schema.SYSTEM_VARIABLES (MDEV-31473)

- InnoDB may fail to recover after being killed in fil_delete_tablespace() (MDEV-31826)
- Create separate tpool thread for async aio (MDEV-31095)
- UNDO logs still growing for write-intensive workloads (MDEV-32050)
  - Increase the default for innodb_purge_batch_size from 300 to 1000 (MDEV-32050)
  - Deprecate innodb_purge_rseg_truncate_frequency.
  - The motivation for introducing this in MySQL seems to have been to avoid stalls due to freeing undo log pages or truncating undo log tablespaces. In MariaDB, innodb_undo_log_truncate=ON should be a much lighter operation because it will not involve any log checkpoint. (MDEV-32050)
- Slow full index scan in 10.6 vs 10.5 for the (slow) I/O-bound case (MDEV-30986)
- LOAD DATA into InnoDB w/partitions: huge performance loss, affected 10.6+ (MDEV-31835)
- Disable read-ahead for temporary tablespace (MDEV-32145)

Optimizer

- Crash when HAVING in a correlated subquery references columns in the outer query (MDEV-29731)
- Server crashes at TABLE::add_tmp_key (MDEV-32320)
- Server crashes inside filesort at my_decimal::to_binary (MDEV-32324)
- Assertion `bitmap_is_set(&m_part_info->read_partitions, m_part_spec.start_part)' failed in ha_partition::handle_ordered_index_scan (MDEV-24283)
- Crash when searching for the best split of derived table (MDEV-32064)
- Test case from opt_tvc.test fails with statement memory protection (MDEV-32225)
- Significant slowdown for query with many outer joins (MDEV-32351)
- test_if_skip_sort_order() should catch the join types JT_EQ_REF, JT_CONST and JT_SYSTEM and skip sort order for these (MDEV-32475)
- jointable materialization subquery optimization ignoring errors, then failing ASSERT. (MDEV-31983)
- Server crashes in JOIN::cleanup after erroneous query with view (MDEV-32164)
- Prepared statement return wrong result (missing row) (MDEV-9938)
- Assertion `range->rows >= s->found_records' failed in best_access_path (MDEV-32682)

Replication

- rpl.rpl_parallel_temptable failure due to incorrect commit optimization of temptables (MDEV-10356)
- Lock wait timeout with INSERT-SELECT, autoinc, and statement-based replication (MDEV-31482)
- strings/ctype-ucs2.c:2336: my_vsnprintf_utf32: Assertion `(n % 4) == 0' failed in my_vsnprintf_utf32 on INSERT (MDEV-32249)
- Assertion fails in MDL_context::acquire_lock upon parallel replication of CREATE SEQUENCE (MDEV-31792)
- SHOW SLAVE STATUS Last_SQL_Errno Race Condition on Errored Slave Restart (MDEV-31177)
- seconds_behind_master is inaccurate for Delayed replication (MDEV-32265)
- detailize the semisync replication magic number error (MDEV-32365)
- Parallel replication deadlock victim preference code errorneously removed (MDEV-31655)
- Parallel replication lags because innobase_kill_query() may fail to interrupt a lock wait (MDEV-32096)

Galera

- Assertion `state() == s_executing || state() == s_prepared || state() == s_committing || state() == s_must_abort || state() == s_replaying' failed. (MDEV-24912)
- Assertion `state() == s_executing || state() == s_preparing || state() == s_prepared || state() == s_must_abort || state() == s_aborting || state() == s_cert_failed || state() == s_must_replay' failed (MDEV-31285)
- wsrep_sst_mariabackup not working on FreeBSD (MDEV-31467)
- Galera library 26.4.16 fails with every server version (MDEV-32024)
- Galera node remains paused after interleaving FTWRLs (MDEV-32282)
- Failed to insert streaming client (MDEV-32051)
- When set at runtime, wsrep_sst_method accepts any value (MDEV-31470)
- galera needs packaging script changes to successfully build (MDEV-32642)
- replication breaks when using optimistic replication and replica is a galera node (MDEV-31833)
- McAfee database vulnerability scan caused MariaDB crash with signal 6 (system abort) (MDEV-27004)
- Assertion `mode_ == m_local || transaction_.is_streaming()' failed in int wsrep::client_state::bf_abort(wsrep::seqno) (MDEV-30217)

Data Definition

- MariaDB crash on calling function (MDEV-23902)
- ASAN errors in grn_obj_unlink / ha_mroonga::clear_indexes upon index operations (MDEV-31970)
- vcol circular references lead to stack overflow (MDEV-31112)
- OPTIMIZE TABLE crash (MDEV-28122)

- Server crashes in Alter_info::add_stat_drop_index upon CREATE TABLE (MDEV-32449)

Scripts and Clients

- mariadb-binlog -T/--table (mysqlbinlog) option (MDEV-25369)
- mariadb-admin (mysqladmin) wrong error with simple_password_check (MDEV-22418)
- mariadb-install-db shows warning on missing directory $pamtooldir/auth_pam_tool_dir (MDEV-32142)
- main.mysql_client_test, main.mysql_client_test_comp failed on ASAN build with error: 5888, status: 23, errno: 2 (MDEV-19369)
- mariadb-install-db (mysql_install_db) doesn't properly grant proxy privileges to all default root user accounts (MDEV-21194)

Tests

- main.events_stress or events.events_stress fails with view-protocol (MDEV-31455)
- main.delete_use_source fails (hangs) with view-protocol (MDEV-31457)
- main.sum_distinct-big and main.merge-big fail with timeout with view-protocol (MDEV-31465)
- main.secure_file_priv_win fails with 2nd execution PS protocol (MDEV-32023)
- Windows : mtr output on is messed up with large MTR_PARALLEL (MDEV-32387)
- main.mysql_client_test_comp failed in buildbot, error on exec (MDEV-16641)
- main.order_by_pack_big fails with view-protocol (MDEV-31460)
- mysql_install_db_win.test fails on second execution (MDEV-32232)

MariaBackup

- MariaBackup full backup failed with InnoDB: Failing assertion: success in storage/innobase/fil/fil0fil.cc line 657 (MDEV-18200)
- mbstream breaks page compression on XFS (MDEV-25734)

Character Sets, Data Types, Collations

- Prefix keys for CHAR work differently for MyISAM vs InnoDB (MDEV-30048)
- Inconsistent results of DISTINCT with NOPAD (MDEV-30050)
- Assertion `(length % 4) == 0' failed in my_lengthsp_utf32 on INSERT (MDEV-28835)
- Compressed varchar values lost on joins when sorting on columns from joined table(s) (MDEV-31724)
- UBSAN shift exponent X is too large for 64-bit type 'long long int' in sql/field.cc (MDEV-32226)
- Wrong bit encoding using COALESCE (MDEV-32244)

Spider

- Spider UBSAN runtime error: applying non-zero offset x to null pointer in st_spider_param_string_parse::restore_delims (MDEV-31117)
- Segfault when setting spider_delete_all_rows to 0 and delete all rows of a spider table, ASAN heap-use-after-free in spider_db_delete_all_rows (MDEV-31996)
- ASAN errors in spider_fields::free_conn_holder or spider_create_group_by_handler (MDEV-28998)
- ASAN: heap-buffer-overflow & stack-buffer-overflow in `spider_db_mbase_row::append_to_str` | SIGSEGV's in `memmove_avx_unaligned_erms` from memcpy in `Binary_string::q_append`, in `Static_binary_string::q_append` and `my_strntoull10rnd_8bit` | Unknown error 12801 (MDEV-29502)

General

- binlog_do_db option breaks importing sql dumps (MDEV-29989)
- Crashes in MDL_key::mdl_key_init with lower-case-table-names=2 (MDEV-32025)
- getting error 'Illegal parameter data types row and bigint for operation '+' ' when using ITERATE in a FOR..DO (MDEV-32275)
- Assertion `arena_for_set_stmt== 0' failed in LEX::set_arena_for_set_stmt upon SET STATEMENT (MDEV-17711)
- main.mysqlcheck fails on ARM with ASAN use-after-poison in my_mb_wc_filename (MDEV-26494)
- main.delayed fails with wrong error code or timeout when executed after main.deadlock_ftwrl (MDEV-27523)
- Assertion failed: !pfs->m_idle || (state == PSI_SOCKET_STATE_ACTIVE) (MDEV-28561)
- MyISAM wrong server status flags (MDEV-28820)
- Server crashes in check_sequence_fields upon CREATE TABLE .. SEQUENCE=1 AS SELECT .. (MDEV-29771)
- slow log Rows_examined out of range (MDEV-30820)
- " `rpm --setugids` " breaks PAM authentication (MDEV-30904)
- incorrect examined rows in case of stored function usage (MDEV-31742)

- Compilation failing on MacOS (unknown warning option -Wno-unused-but-set-variable) (MDEV-31890)
- Server crash upon inserting into Mroonga table with compressed column (MDEV-31966)
- hash unique corrupts index on virtual blobs (MDEV-32012)
- insert into an empty table fails with hash unique (MDEV-32015)
- Valgrind/MSAN warnings in dynamic_column_update_move_left (MDEV-32140)
- Memory leak showed in MDEV-6146 test suite (MDEV-32223)
- Test from subselect.test fails with statement memory protection (MDEV-32245)
- Memory leak when executing PS for query with IN subquery (MDEV-32369)
- Allow the setting of Auto_increment on FK referenced columns (MDEV-32018)
- mariadb-upgrade fails with sql_safe_updates = on (MDEV-29914)
- Assertion `!(thd->server_status & (1U | 8192U))' failed in MDL_context::release_transactional_locks (MDEV-32541)
- Information schema leaks table names and structure to unauthorized users (MDEV-32500)
- Missing CHACHA20-POLY1305 support in WolfSSL (MDEV-31653)
- incorrect error about cyclic reference about JSON type virtual column (MDEV-32586)
- Disable TLS v1.0 and 1.1 for MariaDB (MDEV-31369)
- Better indication of refusing to start because of ProtectHome (MDEV-25177)
- Database upgrade fails: slow_log table (MDEV-27757)
- myrocks_hotbackup.1 and test suite files installed when engine is disabled (MDEV-29993)
- client_ed25519.dll isn't inluded for HeidiSQL. (MDEV-31315)
- Assertion `!m_null_value' failed in int FixedBinTypeBundle<FbtImpl>::cmp_item_fbt::compare or in cmp_item_inet6::compare (MDEV-27207)
- type_test.type_test_double fails with 'NUMERIC_SCALE NULL' (MDEV-22243)
- LeakSanitizer errors in get_quick_select or Assertion `status_var.local_memory_used == 0 || !debug_assert_on_not_freed_memory' failed (MDEV-32476)
- Update signal hander user info more compassion and correct url (MDEV-32535)
- maria-install-db fails on MacOS (MDEV-31871)
- MSAN / Valgrind errors in Item_func_like::get_mm_leaf upon query from partitioned table (MDEV-32388)
- MSAN / Valgrind errors in Item_func_like::get_mm_leaf with temporal field (MDEV-32531)
- ASAN errors in base_list_iterator::next / setup_table_map upon 2nd execution of PS (MDEV-32656)
- safe_mutex: Found wrong usage of mutex 'LOCK_thd_data' and 'wait_mutex' (MDEV-32728)

Docker Official Images

- Invert single and double quotes for sql command definitions in healthcheck.sh due to failure under sql_mode=ANSI_QUOTES - contribution by Dominik Häckel
- healthcheck.sh --no-defaults behaviour was corrected - reported by Dominik Häckel
- Added /docker-entrypoint-init.d for tar{,compression} from mariadb-backup - instructions
- Refactor `docker_mariadb_init` in the entrypoint for extending the MariaDB image
- CIS failure due to world-writable directory /var/run/mysqld, added sticky bit - reported by @ollie1
- Add PROXY privileges for root@MARIADB_ROOT_HOST - reported by Matthieu Gusmini
- healthcheck.sh added --galera_online test, to match what the mariadb-operator does.

Variables

- Added the note_verbosity system variable to manage notes when an index cannot be used .

Security

- Fixes for the following security vulnerabilities :
  - CVE-2023-22084

Changelog

For a complete list of changes made in MariaDB 11.1.3, with links to detailed information on each push, see the changelog .

Contributors

For a full list of contributors to MariaDB 11.1.3, see the MariaDB Foundation release announcement .

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will

be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.0.10.2 MariaDB 11.1.2 Release Notes

The most recent release of MariaDB 11.1 is:
MariaDB 11.1.3 Stable (GA) Download Now
Alternate download from mariadb.org

Download 11.1.2 | Release Notes | Changelog | Overview of 11.1

Release date: 21 Aug 2023

MariaDB 11.1 is a current short-term stable series of MariaDB, maintained until August 2024. It is an evolution of MariaDB 11.0 with several entirely new features.

MariaDB 11.1.2 is a ***Stable (GA)*** release.

For an overview of MariaDB 11.1 see the What is MariaDB 11.1? page.

Thanks, and enjoy MariaDB!

Notable Items

Localization

- Create Swahili localization (MDEV-31530)

Upgrading from MySQL

- MariaDB now detects and converts previously incompatible MySQL partition schemes (MDEV-29253)

General

- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 11.1 for Ubuntu 18.04 LTS "Bionic" and Ubuntu 22.10 "Kinetic"
- In this release repositories for Debian 12 "Bookworm" have been added.
- mariadb-dump --force doesn't ignore error as it should (MDEV-31092)
- 280 Bytes lost in mysys/array.c, mysys/hash.c, sql/sp.cc, sql/sp.cc, sql/item_create.cc, sql/item_create.cc, sql/sql_yacc.yy:10748 when using oracle sql_mode (MDEV-26186)
- SQL/PL package body does not appear in `I_S.ROUTINES.ROUTINE_DEFINITION` (MDEV-30662)
- Unexpected result when combining DISTINCT, subselect and LIMIT (MDEV-28285)
- ROW variables do not get assigned from subselects (MDEV-31250)
- Crash after setting global session_track_system_variables to an invalid value (MDEV-25237)
- ODKU of non-versioning column inserts history row (MDEV-23100)
- UPDATE not working properly on transaction precise system versioned table (MDEV-25644)
- Assertion `` `const_item_cache == true' `` failed in `Item_func::fix_fields` (MDEV-31319)
- ANALYZE doesn't work with pushed derived tables (MDEV-29284)
- `get_partition_set` is never executed in `ha_partition::multi_range_key_create_key` due to bitwise & with 0 constant (MDEV-24712)
- Client can crash the server with a `mysql_list_fields("view")` call (MDEV-30159)
- I_S.parameters not immediately changed updated after procedure change (MDEV-31064)
- bzero wipes more bytes than necessary in `set_global_from_ddl_log_entry` (MDEV-31521)
- Assertion `` `0' `` failed in `Type_handler_row::field_type` upon `TO_CHAR` with wrong argument (MDEV-29152)
- mariadb-upgrade fails due to `old_mode=""` , with "Cannot load from mysql.proc. The table is probably corrupted" (MDEV-28915)

Character Sets, Data Types

- UBSAN: null pointer passed as argument 1, which is declared to never be null in `my_strnncoll_binary` on `SELECT ... COUNT` or `GROUP_CONCAT` (MDEV-28384)
- Possibly wrong result or Assertion `0' failed in `Item_func_round::native_op` (MDEV-23838)
- Assertion `` `(length % 4) == 0' `` failed in `my_lengthsp_utf32` on `SELECT` (MDEV-29019)
- UBSAN: negation of -X cannot be represented in type `'long long int'` ; cast to an unsigned type to negate this value to itself in `Item_func_mul::int_op` and `Item_func_round::int_op` (MDEV-30932)
- Assorted assertion failures in json_find_path with certain collations (MDEV-23187)

InnoDB

- Crashing on I/O error is unhelpful (MDEV-27593)
- SIGSEGV in `log_sort_flush_list()` in InnoDB crash recovery (MDEV-31354)
- InnoDB tables are being flagged as corrupted on an I/O bound server (MDEV-31767)
- Duplicate entry allowed into a UNIQUE column (MDEV-31120)
- Server Status Innodb_row_lock_time% is reported in seconds (MDEV-29311)
- innochecksum dies with Floating point exception (MDEV-31641)
- Add InnoDB engine information to the slow query log (MDEV-31558)
- Deadlock with 3 concurrent DELETEs by unique key (MDEV-10962)
- innodb protection against dual processes accessing data insufficient (MDEV-31568)
- ER_DUP_KEY in mysql.innodb_table_stats upon RENAME on sequence (MDEV-31607)
- Assertion `` `!strcmp(index->table->name.m_name, "SYS_FOREIGN") || !strcmp(index->table->name.m_name, "SYS_FOREIGN_COLS")' `` failed in `btr_node_ptr_max_size` (MDEV-19216)
- InnoDB: Failing assertion: `page_type == i_s_page_type[page_type].type_value` (MDEV-31386)
- `btr_estimate_n_rows_in_range()` accesses unfixed, unlatched page (MDEV-30648)
- `MODIFY COLUMN` can break FK constraints, and lead to unrestorable dumps (MDEV-31086)
- Recovery or backup failure after innodb_undo_log_truncate=ON (MDEV-31487)
- Assertion `'n & PENDING'` failed in `fil_space_t::set_needs_flush()` (MDEV-31442)
- `fil_node_open_file()` releases `fil_system.mutex` allowing other thread to open its file node (MDEV-31256)
- ASAN errors in `dict_v_col_t::detach` upon adding key to virtual column (MDEV-31416)
- Purge trying to access freed secondary index page (MDEV-31264)
- Freed data pages are not always being scrubbed (MDEV-31253)
- InnoDB recovery hangs after reporting corruption (MDEV-31353)
- `!cursor->index->is_committed()` in row0ins.cc after update to 10.4.13 from 10.3.21 (MDEV-22739)
- innodb_undo_log_truncate=ON fails to wait for purge of enough transaction history (MDEV-31355)
- SET GLOBAL innodb_undo_log_truncate=ON does not free space when no undo logs exist (MDEV-31382)
- innodb_read_ahead_threshold (linear read-ahead) does not work (MDEV-29967)
- InnoDB recovery and mariadb-backup --prepare fail to report detailed progress (MDEV-29911)
- `fil_ibd_create()` may hijack the file handle of an old file (MDEV-31347)
- innodb_undo_log_truncate=ON recovery results in a corrupted undo log (MDEV-31373)
- Server freeze due to innodb_change_buffering and innodb_file_per_table=0 (MDEV-31088)
- Change buffer entries are left behind when freeing a page, causing secondary index corruption when the page is later reused (MDEV-31385)
- Foreign Key Constraint actions don't affect Virtual Column (MDEV-18114)

Aria

- Various crashes upon INSERT/UPDATE after changing Aria settings (MDEV-28054)
- Various crashes/asserts/corruptions when Aria encryption is enabled/used, but the encryption plugin is not loaded (MDEV-26258)

Spider

- SIGSEGV in `spider_db_open_item_field` and SIGSEGV in `spider_db_print_item_type` , on SELECT (MDEV-29447)
- Spider variables that double as table params overriding mechanism is buggy (MDEV-31524)

Optimizer

- ANALYZE FORMAT=JSON now includes InnoDB engine statistics for each table (MDEV-31577)
- Assertion `` `last_key_entry >= end_pos' `` failed in virtual bool `JOIN_CACHE_HASHED::put_record()` (MDEV-31348)

- Problem with open ranges on prefix blobs keys (MDEV-31800)
- Equal on two RANK window functions create wrong result (MDEV-20010)
- Recursive CTE execution is interrupted without errors or warnings (MDEV-31214)
- Assertion `s->table->opt_range_condition_rows <= s->found_records' failed in apply_selectivity_for_table (MDEV-31449)
- Inconsistency between MRR and SQL layer costs can cause poor query plan (MDEV-31479)
- MAX_SEL_ARG memory exhaustion is not visible in the optimizer trace (MDEV-30964)
- SHOW TABLES not working properly with lower_case_table_names=2 (MDEV-30765)
- Segfault on select query using index for group-by and filesort (MDEV-30143)
- Server crash in store_length, assertion failure in Type_handler_string_result::sort_length (MDEV-31743)

Replication

- Parallel Slave SQL Thread Can Update Seconds_Behind_Master with Active Workers (MDEV-30619)
- ALTER SEQUENCE ends up in optimistic parallel slave binlog out-of-order (MDEV-31503)
- STOP SLAVE takes very long time on a busy system (MDEV-13915)
- On slave XA COMMIT/XA ROLLBACK fail to return an error in read-only mode (MDEV-30978)
- Calling a function from a different database in a slave side trigger crashes (MDEV-29894)
- rpl.rpl_manual_change_index_file occasionally fails in BB with Result length mismatch (MDEV-30214)

Galera

- Node never returns from Donor/Desynced to Synced when wsrep_mode = BF_ABORT_MARIABACKUP (MDEV-31737)
- Node has been dropped from the cluster on Startup / Shutdown with async replica (MDEV-31413)
- KILL QUERY maintains nodes data consistency but breaks GTID sequence (MDEV-31075)
- Assertion failure !lock.was_chosen_as_deadlock_victim in trx0trx.h:1065 (MDEV-30963)
- MariaDB stuck on starting commit state (waiting on commit order critical section) (MDEV-29293)
- Assertion state() == s_aborting || state() == s_must_replay failed in int wsrep::transaction::after_rollback() (MDEV-30013)
- Assertion !wsrep_has_changes(thd) || (thd->lex->sql_command == SQLCOM_CREATE_TABLE && !thd->is_current_stmt_binlog_format_row()) || thd->wsrep_cs().transaction().state() == wsrep::transaction::s_aborted failed (MDEV-30388)
- Server crashes when wsrep_sst_donor and wsrep_cluster_address set to NULL (MDEV-28433)
- Create temporary sequence can cause inconsistency (MDEV-31335)
- Galera 4 unable to query cluster state if not primary component (MDEV-21479)

Changelog

For a complete list of changes made in MariaDB 11.1.2, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 11.1.2, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.0.10.3 MariaDB 11.1.1 Release Notes

The most recent release of MariaDB 11.1 is:
MariaDB 11.1.3 Stable (GA) Download Now

Alternate download from mariadb.org

Download 11.1.1 | Release Notes | Changelog | Overview of 11.1

Release date: 6 Jun 2023

Do not use non-stable (non-GA) releases in production!

MariaDB 11.1 is the current short-term development series of MariaDB, which will be maintained for one year after the Stable (GA) release. It is an evolution of MariaDB 11.0 with several entirely new features.

MariaDB 11.1.1 is a ***Release Candidate (RC)*** release.

For an overview of MariaDB 11.1 see the What is MariaDB 11.1? page.

Thanks, and enjoy MariaDB!

Notable Items

- The transaction_isolation option is now a system variable, and the tx_isolation system variable is deprecated (MDEV-21921)

InnoDB

- Server crashes in st_join_table::choose_best_splitting (MDEV-31403)
- Crash with condition pushable into derived and containing outer reference (MDEV-31240)
- InnoDB does not free UNDO after the fix of MDEV-30671 (MDEV-31234)
- InnoDB hang fixes (MDEV-31158, MDEV-31343, MDEV-31350)
- Innodb_buffer_pool_read_requests is not updated correctly (MDEV-31309)
- InnoDB monitor trx_rseg_history_len was accidentally disabled by default (MDEV-31308)
- Revert "MDEV-30473: Do not allow GET_LOCK() / RELEASE_LOCK() in cluster"

Optimizer

- Crash with condition pushable into derived and containing outer reference (MDEV-31403 MDEV-31240)
- Crash with EXPLAIN EXTENDED for multi-table update of system table (MDEV-31224)

Changelog

For a complete list of changes made in MariaDB 11.1.1, with links to detailed information on each push, see the changelog.

Do not use non-stable (non-GA) releases in production!

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.0.10.4 MariaDB 11.1.0 Release Notes

The most recent release of MariaDB 11.1 is:
MariaDB 11.1.3 Stable (GA) Download Now

Alternate download from mariadb.org

Download | Release Notes | Changelog | Overview of 11.1

Release date: 27 Mar 2023

Do not use *alpha* releases in production!

MariaDB 11.1 is a current development series of MariaDB, and will be maintained for one year after its Generally Available release. It is an evolution of MariaDB 11.0 with several entirely new features.

MariaDB 11.1.0 is a single preview release. Features are to be considered preview, and none are guaranteed to make it into MariaDB 11.1.

For an overview of MariaDB 11.1 see the What is MariaDB 11.1? page.

Thanks, and enjoy MariaDB!

InnoDB

- Remove innodb_defragment and related parameters (MDEV-30545)

Optimizer

- Semi-join optimization for single-table UPDATE/DELETE statements. Update and delete statements that use subqueries can now use all subquery optimization strategies that MariaDB offers, so now if you use subqueries in UPDATE or DELETE, these statements will likely be much faster (MDEV-7487)
- Queries with the DATE or YEAR functions comparing against a constant can now make use of indexes, so these will be noticeably quicker in certain instances. For example `SELECT * FROM t2 WHERE YEAR(a) = 2019` or `SELECT * FROM t2 WHERE DATE(a) <= '2017-01-01'` (MDEV-8320)

General

- ALTER ONLINE TABLE - not released in the final MariaDB 11.1 (MDEV-16329)

JSON

- JSON_SCHEMA_VALID function for validating a JSON schema (MDEV-27128)

Mariabackup

- Rename Mariabackup's xtrabackup_* files to mariadb_backup_* (MDEV-18931)

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.0.11 MariaDB Server 11.0

Changes and Improvements in MariaDB 11.0

Current Version: 11.0.4 | Status: Stable (GA) | Release Date: 13 Nov 2023

Release Notes - MariaDB 11.0 Series
MariaDB 10.11 series release notes.

Changelogs - MariaDB 11.0 Series
MariaDB 11.0 changelogs

7.0.0.12 Changes and Improvements in MariaDB 11.0

The most recent release of MariaDB 11.0 is:
MariaDB 11.0.4 Stable (GA) Download Now
Alternate download from mariadb.org

Contents

MariaDB 11.0 is a current short-term release series, maintained until June 2024.

Upgrading

- See Upgrading Between Major MariaDB Versions and Upgrading from MariaDB 10.11 to MariaDB 11.0.

New Features & Improvements

Functions

- Given a time in picoseconds, the new function FORMAT_PICO_TIME returns a human-readable time value and unit indicator (MDEV-19629)

Optimizer

- Major improvements to the Optimizer. See The Optimizer Cost Model from MariaDB 11.0.

InnoDB

- The InnoDB Change Buffer has been removed (MDEV-29694)

Variables

- For a list of all new variables, see System Variables Added in MariaDB 11.0 and Status Variables Added in MariaDB 11.0.
- The default value for innodb_undo_tablespaces has been changed from 0 to 3 (MDEV-29986)
- The following variables have been deprecated:
  - innodb_defragment
  - innodb_defragment_n_pages
  - innodb_defragment_stats_accuracy
  - innodb_defragment_fill_factor_n_recs
  - innodb_defragment_fill_factor
  - innodb_defragment_frequency
  - innodb_file_per_table
  - innodb_flush_method

- The following deprecated variables have been removed:
  - innodb_change_buffer_max_size
  - innodb_change_buffering

List of All MariaDB 11.0 Releases

Date	Release	Status	Release Notes	Changelog
13 Nov 2023	**MariaDB 11.0.4**	Stable (GA)	Release Notes	Changelog
14 Aug 2023	**MariaDB 11.0.3**	Stable (GA)	Release Notes	Changelog
6 Jun 2023	**MariaDB 11.0.2**	Stable (GA)	Release Notes	Changelog
22 Feb 2023	**MariaDB 11.0.1**	RC (Release Candidate)	Release Notes	Changelog
27 Dec 2022	**MariaDB 11.0.0**	Alpha	Release Notes	

7.0.0.13 Release Notes - MariaDB 11.0 Series

MariaDB 11.0.4 Release Notes

Status: Stable (GA) | Release Date: 13 Nov 2023

MariaDB 11.0.3 Release Notes

Status: Stable (GA) | Release Date: 14 Aug 2023

MariaDB 11.0.2 Release Notes

Status: Stable (GA) | Release Date: 6 Jun 2023

MariaDB 11.0.1 Release Notes

Status: Release Candidate (RC) | Release Date: 22 Feb 2023

MariaDB 11.0.0 Release Notes

Status: Alpha | Release Date: 27 Dec 2022

7.0.0.13.1 MariaDB 11.0.4 Release Notes

Download | Release Notes | Changelog | Overview of 11.0

Alternate download from mariadb.org

Release date: 13 Nov 2023

MariaDB 11.0 is a current short-term stable series of MariaDB and will be maintained until June 2024. It is an evolution of MariaDB 10.11 with several entirely new features.

MariaDB 11.0.4 is a ***Stable (GA)*** release.

For an overview of MariaDB 11.0 see the What is MariaDB 11.0? page.

Thanks, and enjoy MariaDB!

Notable Items

InnoDB

- DROP INDEX followed by CREATE INDEX may corrupt data (MDEV-32132)
- ROW_FORMAT=COMPRESSED table: InnoDB: 2048 bytes should have been read. Only 0 bytes read. (MDEV-31875)
- Server aborts during alter operation when table doesn't have foreign index (MDEV-32527)
- row_merge_fts_doc_tokenize() handles FTS plugin parser inconsistently (MDEV-32578)

- InnoDB: tried to purge non-delete-marked record of an index on a virtual column prefix (MDEV-30024)
- lock_row_lock_current_waits counter in information_schema.innodb_metrics may become negative (MDEV-30658)
- SET GLOBAL innodb_max_purge_lag_wait=… hangs if innodb_read_only=ON (MDEV-31813)
- Auto-increment no longer works for explicit FTS_DOC_ID (MDEV-32017)
- Assertion `pos < table->n_def' failed in dict_table_get_nth_col (MDEV-32337)
- innochecksum man pages seem to be inconsistent with the binary (10.2.25) (MDEV-20583)
- innodb_compression_algorithm=0 (none) increments Innodb_num_pages_page_compression_error (MDEV-30825)
- wrong table name in innodb's "row too big" errors (MDEV-32128)
- Optimize is_file_on_ssd() to speedup opening tablespaces on Windows (MDEV-32228)
- Race condition between page write completion and log checkpoint (MDEV-32511)
- After crash recovery, Checksum mismatch + Failing assertion: !i || prev_id + 1 == space_id, (MDEV-31851)
- Deadlock due to log_free_check(), involving trx_purge_truncate_rseg_history() and trx_undo_assign_low() (MDEV-32049)
- Write-ahead logging is broken for freed pages (MDEV-32552)
- X-lock on supremum for prepared transaction for RR (MDEV-30165)
- Assertion failures in log_sort_flush_list upon crash recovery (MDEV-32029)
- Race condition between page write completion and log checkpoint (MDEV-32511)
- Assertion `purge_sys.tail.trx_no <= purge_sys.rseg->last_trx_no()' (MDEV-30100)
- Assertion `index->is_btree() || index->is_ibuf()' failed in btr_search_guess_on_hash (MDEV-30802)
- InnoDB hang in buf_flush_wait_LRU_batch_end() (MDEV-32134)
- InnoDB may hang when running out of buffer pool (MDEV-32588)
- Corrupt index(es) on busy table when using FOREIGN KEY (MDEV-30531)
- InnoDB Recovery doesn't display encryption message when no encryption configuration passed (MDEV-31098)
- Wrong information about innodb_checksum_algorithm in the information_schema.SYSTEM_VARIABLES (MDEV-31473)
- InnoDB may fail to recover after being killed in fil_delete_tablespace() (MDEV-31826)
- Create separate tpool thread for async aio (MDEV-31095)
- UNDO logs still growing for write-intensive workloads (MDEV-32050)
    - Increase the default for innodb_purge_batch_size from 300 to 1000 (MDEV-32050)
    - Deprecate innodb_purge_rseg_truncate_frequency.
    - The motivation for introducing this in MySQL seems to have been to avoid stalls due to freeing undo log pages or truncating undo log tablespaces. In MariaDB, innodb_undo_log_truncate=ON should be a much lighter operation because it will not involve any log checkpoint. (MDEV-32050)
- Slow full index scan in 10.6 vs 10.5 for the (slow) I/O-bound case (MDEV-30986)
- LOAD DATA into InnoDB w/partitions: huge performance loss, affected 10.6+ (MDEV-31835)
- Disable read-ahead for temporary tablespace (MDEV-32145)

Optimizer

- Crash when HAVING in a correlated subquery references columns in the outer query (MDEV-29731)
- Server crashes at TABLE::add_tmp_key (MDEV-32320)
- Server crashes inside filesort at my_decimal::to_binary (MDEV-32324)
- Assertion `bitmap_is_set(&m_part_info->read_partitions, m_part_spec.start_part)' failed in ha_partition::handle_ordered_index_scan (MDEV-24283)
- Crash when searching for the best split of derived table (MDEV-32064)
- Test case from opt_tvc.test fails with statement memory protection (MDEV-32225)
- Significant slowdown for query with many outer joins (MDEV-32351)
- test_if_skip_sort_order() should catch the join types JT_EQ_REF, JT_CONST and JT_SYSTEM and skip sort order for these (MDEV-32475)
- jointable materialization subquery optimization ignoring errors, then failing ASSERT. (MDEV-31983)
- Server crashes in JOIN::cleanup after erroneous query with view (MDEV-32164)
- Prepared statement return wrong result (missing row) (MDEV-9938)
- Assertion `range->rows >= s->found_records' failed in best_access_path (MDEV-32682)

Replication

- rpl.rpl_parallel_temptable failure due to incorrect commit optimization of temptables (MDEV-10356)
- Lock wait timeout with INSERT-SELECT, autoinc, and statement-based replication (MDEV-31482)
- strings/ctype-ucs2.c:2336: my_vsnprintf_utf32: Assertion `(n % 4) == 0' failed in my_vsnprintf_utf32 on INSERT (MDEV-32249)
- Assertion fails in MDL_context::acquire_lock upon parallel replication of CREATE SEQUENCE (MDEV-31792)
- SHOW SLAVE STATUS Last_SQL_Errno Race Condition on Errored Slave Restart (MDEV-31177)
- seconds_behind_master is inaccurate for Delayed replication (MDEV-32265)
- detailize the semisync replication magic number error (MDEV-32365)
- Parallel replication deadlock victim preference code errorneously removed (MDEV-31655)

- Parallel replication lags because innobase_kill_query() may fail to interrupt a lock wait (MDEV-32096)

Galera

- Assertion `state() == s_executing || state() == s_prepared || state() == s_committing || state() == s_must_abort || state() == s_replaying' failed. (MDEV-24912)
- Assertion `state() == s_executing || state() == s_preparing || state() == s_prepared || state() == s_must_abort || state() == s_aborting || state() == s_cert_failed || state() == s_must_replay' failed (MDEV-31285)
- wsrep_sst_mariabackup not working on FreeBSD (MDEV-31467)
- Galera library 26.4.16 fails with every server version (MDEV-32024)
- Galera node remains paused after interleaving FTWRLs (MDEV-32282)
- Failed to insert streaming client (MDEV-32051)
- When set at runtime, wsrep_sst_method accepts any value (MDEV-31470)
- galera needs packaging script changes to successfully build (MDEV-32642)
- replication breaks when using optimistic replication and replica is a galera node (MDEV-31833)
- McAfee database vulnerability scan caused MariaDB crash with signal 6 (system abort) (MDEV-27004)
- Assertion `mode_ == m_local || transaction_.is_streaming()' failed in int wsrep::client_state::bf_abort(wsrep::seqno) (MDEV-30217)

Data Definition

- MariaDB crash on calling function (MDEV-23902)
- ASAN errors in grn_obj_unlink / ha_mroonga::clear_indexes upon index operations (MDEV-31970)
- vcol circular references lead to stack overflow (MDEV-31112)
- OPTIMIZE TABLE crash (MDEV-28122)
- Server crashes in Alter_info::add_stat_drop_index upon CREATE TABLE (MDEV-32449)

Scripts and Clients

- mariadb-binlog -T/--table (mysqlbinlog) option (MDEV-25369)
- mariadb-admin (mysqladmin) wrong error with simple_password_check (MDEV-22418)
- mariadb-install-db shows warning on missing directory $pamtooldir/auth_pam_tool_dir (MDEV-32142)
- main.mysql_client_test, main.mysql_client_test_comp failed on ASAN build with error: 5888, status: 23, errno: 2 (MDEV-19369)
- mariadb-install-db (mysql_install_db) doesn't properly grant proxy privileges to all default root user accounts (MDEV-21194)

Tests

- main.events_stress or events.events_stress fails with view-protocol (MDEV-31455)
- main.delete_use_source fails (hangs) with view-protocol (MDEV-31457)
- main.sum_distinct-big and main.merge-big fail with timeout with view-protocol (MDEV-31465)
- main.secure_file_priv_win fails with 2nd execution PS protocol (MDEV-32023)
- Windows : mtr output on is messed up with large MTR_PARALLEL (MDEV-32387)
- main.mysql_client_test_comp failed in buildbot, error on exec (MDEV-16641)
- main.order_by_pack_big fails with view-protocol (MDEV-31460)
- mysql_install_db_win.test fails on second execution (MDEV-32232)

MariaBackup

- MariaBackup full backup failed with InnoDB: Failing assertion: success in storage/innobase/fil/fil0fil.cc line 657 (MDEV-18200)
- mbstream breaks page compression on XFS (MDEV-25734)

Character Sets, Data Types, Collations

- Prefix keys for CHAR work differently for MyISAM vs InnoDB (MDEV-30048)
- Inconsistent results of DISTINCT with NOPAD (MDEV-30050)
- Assertion `(length % 4) == 0' failed in my_lengthsp_utf32 on INSERT (MDEV-28835)
- Compressed varchar values lost on joins when sorting on columns from joined table(s) (MDEV-31724)
- UBSAN shift exponent X is too large for 64-bit type 'long long int' in sql/field.cc (MDEV-32226)
- Wrong bit encoding using COALESCE (MDEV-32244)

Spider

- Spider UBSAN runtime error: applying non-zero offset x to null pointer in st_spider_param_string_parse::restore_delims (MDEV-31117)
- Segfault when setting spider_delete_all_rows to 0 and delete all rows of a spider table, ASAN heap-use-after-free in spider_db_delete_all_rows (MDEV-31996)
- ASAN errors in spider_fields::free_conn_holder or spider_create_group_by_handler (MDEV-28998)
- ASAN: heap-buffer-overflow & stack-buffer-overflow in `spider_db_mbase_row::append_to_str` | SIGSEGV's in `memmove_avx_unaligned_erms` from memcpy in `Binary_string::q_append`, in `Static_binary_string::q_append` and `my_strntoull10rnd_8bit` | Unknown error 12801 (MDEV-29502)

General

- binlog_do_db option breaks importing sql dumps (MDEV-29989)
- Crashes in MDL_key::mdl_key_init with lower-case-table-names=2 (MDEV-32025)
- getting error 'Illegal parameter data types row and bigint for operation '+' ' when using ITERATE in a FOR..DO (MDEV-32275)
- Assertion `arena_for_set_stmt== 0' failed in LEX::set_arena_for_set_stmt upon SET STATEMENT (MDEV-17711)
- main.mysqlcheck fails on ARM with ASAN use-after-poison in my_mb_wc_filename (MDEV-26494)
- main.delayed fails with wrong error code or timeout when executed after main.deadlock_ftwrl (MDEV-27523)
- Assertion failed: !pfs->m_idle || (state == PSI_SOCKET_STATE_ACTIVE) (MDEV-28561)
- MyISAM wrong server status flags (MDEV-28820)
- Server crashes in check_sequence_fields upon CREATE TABLE .. SEQUENCE=1 AS SELECT .. (MDEV-29771)
- slow log Rows_examined out of range (MDEV-30820)
- " `rpm --setugids` " breaks PAM authentication (MDEV-30904)
- incorrect examined rows in case of stored function usage (MDEV-31742)
- Compilation failing on MacOS (unknown warning option -Wno-unused-but-set-variable) (MDEV-31890)
- Server crash upon inserting into Mroonga table with compressed column (MDEV-31966)
- hash unique corrupts index on virtual blobs (MDEV-32012)
- insert into an empty table fails with hash unique (MDEV-32015)
- Valgrind/MSAN warnings in dynamic_column_update_move_left (MDEV-32140)
- Memory leak showed in MDEV-6146 test suite (MDEV-32223)
- Test from subselect.test fails with statement memory protection (MDEV-32245)
- Memory leak when executing PS for query with IN subquery (MDEV-32369)
- Allow the setting of Auto_increment on FK referenced columns (MDEV-32018)
- mariadb-upgrade fails with sql_safe_updates = on (MDEV-29914)
- Assertion `!(thd->server_status & (1U | 8192U))' failed in MDL_context::release_transactional_locks (MDEV-32541)
- Information schema leaks table names and structure to unauthorized users (MDEV-32500)
- Missing CHACHA20-POLY1305 support in WolfSSL (MDEV-31653)
- incorrect error about cyclic reference about JSON type virtual column (MDEV-32586)
- Disable TLS v1.0 and 1.1 for MariaDB (MDEV-31369)
- Better indication of refusing to start because of ProtectHome (MDEV-25177)
- Database upgrade fails: slow_log table (MDEV-27757)
- myrocks_hotbackup.1 and test suite files installed when engine is disabled (MDEV-29993)
- client_ed25519.dll isn't inluded for HeidiSQL. (MDEV-31315)
- Assertion `!m_null_value' failed in int FixedBinTypeBundle<FbtImpl>::cmp_item_fbt::compare or in cmp_item_inet6::compare (MDEV-27207)
- type_test.type_test_double fails with 'NUMERIC_SCALE NULL' (MDEV-22243)
- LeakSanitizer errors in get_quick_select or Assertion `status_var.local_memory_used == 0 || !debug_assert_on_not_freed_memory' failed (MDEV-32476)
- Update signal hander user info more compassion and correct url (MDEV-32535)
- maria-install-db fails on MacOS (MDEV-31871)
- MSAN / Valgrind errors in Item_func_like::get_mm_leaf upon query from partitioned table (MDEV-32388)
- MSAN / Valgrind errors in Item_func_like::get_mm_leaf with temporal field (MDEV-32531)
- ASAN errors in base_list_iterator::next / setup_table_map upon 2nd execution of PS (MDEV-32656)
- safe_mutex: Found wrong usage of mutex 'LOCK_thd_data' and 'wait_mutex' (MDEV-32728)

Docker Official Images

- Invert single and double quotes for sql command definitions in healthcheck.sh due to failure under sql_mode=ANSI_QUOTES - contribution by Dominik Häckel
- healthcheck.sh --no-defaults behaviour was corrected - reported by Dominik Häckel
- Added /docker-entrypoint-init.d for tar{,compression} from mariadb-backup - instructions
- Refactor `docker_mariadb_init` in the entrypoint for extending the MariaDB image
- CIS failure due to world-writable directory /var/run/mysqld, added sticky bit - reported by @ollie1
- Add PROXY privileges for root@MARIADB_ROOT_HOST - reported by Matthieu Gusmini
- healthcheck.sh added --galera_online test, to match what the mariadb-operator does.

Variables

- Added the note_verbosity system variable to manage notes when an index cannot be used.

Security

- Fixes for the following security vulnerabilities:
  - CVE-2023-22084

Changelog

For a complete list of changes made in MariaDB 11.0.4, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 11.0.4, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.0.13.2 MariaDB 11.0.3 Release Notes

The most recent release of MariaDB 11.0 is:
MariaDB 11.0.4 Stable (GA) Download Now
Alternate download from mariadb.org

Download 11.0.3 | Release Notes | Changelog | Overview of 11.0

Release date: 14 Aug 2023

MariaDB 11.0 is a current short-term stable series of MariaDB and will be maintained until June 2024. It is an evolution of MariaDB 10.11 with several entirely new features.

MariaDB 11.0.3 is a ***Stable (GA)*** release.

For an overview of MariaDB 11.0 see the What is MariaDB 11.0? page.

Thanks, and enjoy MariaDB!

Notable Items

Upgrading from MySQL

- MariaDB now detects and converts previously incompatible MySQL partition schemes (MDEV-29253)

General

- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 11.0 for Ubuntu 18.04 LTS "Bionic" and Ubuntu 22.10 "Kinetic"
- In this release repositories for Debian 12 "Bookworm" have been added.
- mariadb-dump --force doesn't ignore error as it should (MDEV-31092)

- 280 Bytes lost in mysys/array.c, mysys/hash.c, sql/sp.cc, sql/sp.cc, sql/item_create.cc, sql/item_create.cc, sql/sql_yacc.yy:10748 when using oracle sql_mode (MDEV-26186)
- SQL/PL package body does not appear in `I_S.ROUTINES.ROUTINE_DEFINITION` (MDEV-30662)
- Unexpected result when combining DISTINCT, subselect and LIMIT (MDEV-28285)
- ROW variables do not get assigned from subselects (MDEV-31250)
- Crash after setting global session_track_system_variables to an invalid value (MDEV-25237)
- ODKU of non-versioning column inserts history row (MDEV-23100)
- UPDATE not working properly on transaction precise system versioned table (MDEV-25644)
- Assertion `` `const_item_cache == true' `` failed in `Item_func::fix_fields` (MDEV-31319)
- ANALYZE doesn't work with pushed derived tables (MDEV-29284)
- `get_partition_set` is never executed in `ha_partition::multi_range_key_create_key` due to bitwise & with 0 constant (MDEV-24712)
- Client can crash the server with a `mysql_list_fields("view")` call (MDEV-30159)
- `I_S.parameters` not immediatly changed updated after procedure change (MDEV-31064)
- bzero wipes more bytes than necessary in `set_global_from_ddl_log_entry` (MDEV-31521)
- Assertion `` `0' `` failed in `Type_handler_row::field_type` upon `TO_CHAR` with wrong argument (MDEV-29152)
- mysql_upgrade fails due to `old_mode=""` , with "Cannot load from mysql.proc. The table is probably corrupted" (MDEV-28915)

Character Sets, Data Types

- UBSAN: null pointer passed as argument 1, which is declared to never be null in `my_strnncoll_binary` on `SELECT ... COUNT` or `GROUP_CONCAT` (MDEV-28384)
- Possibly wrong result or Assertion `0' failed in `Item_func_round::native_op` (MDEV-23838)
- Assertion `` `(length % 4) == 0' `` failed in `my_lengthsp_utf32` on `SELECT` (MDEV-29019)
- UBSAN: negation of -X cannot be represented in type `'long long int'` ; cast to an unsigned type to negate this value to itself in `Item_func_mul::int_op` and `Item_func_round::int_op` (MDEV-30932)
- Assorted assertion failures in json_find_path with certain collations (MDEV-23187)

InnoDB

- Crashing on I/O error is unhelpful (MDEV-27593)
- SIGSEGV in `log_sort_flush_list()` in InnoDB crash recovery (MDEV-31354)
- InnoDB tables are being flagged as corrupted on an I/O bound server (MDEV-31767)
- Duplicate entry allowed into a UNIQUE column (MDEV-31120)
- Server Status Innodb_row_lock_time% is reported in seconds (MDEV-29311)
- innochecksum dies with Floating point exception (MDEV-31641)
- Add InnoDB engine information to the slow query log (MDEV-31558)
- Deadlock with 3 concurrent DELETEs by unique key (MDEV-10962)
- innodb protection against dual processes accessing data insufficient (MDEV-31568)
- ER_DUP_KEY in `mysql.innodb_table_stats` upon RENAME on sequence (MDEV-31607)
- Assertion `` `!strcmp(index->table->name.m_name, "SYS_FOREIGN") || !strcmp(index->table->name.m_name, "SYS_FOREIGN_COLS")' `` failed in `btr_node_ptr_max_size` (MDEV-19216)
- InnoDB: Failing assertion: `page_type == i_s_page_type[page_type].type_value` (MDEV-31386)
- `btr_estimate_n_rows_in_range()` accesses unfixed, unlatched page (MDEV-30648)
- `MODIFY COLUMN` can break FK constraints, and lead to unrestorable dumps (MDEV-31086)
- Recovery or backup failure after innodb_undo_log_truncate=ON (MDEV-31487)
- Assertion `'n & PENDING'` failed in `fil_space_t::set_needs_flush()` (MDEV-31442)
- `fil_node_open_file()` releases `fil_system.mutex` allowing other thread to open its file node (MDEV-31256)
- ASAN errors in `dict_v_col_t::detach` upon adding key to virtual column (MDEV-31416)
- Purge trying to access freed secondary index page (MDEV-31264)
- Freed data pages are not always being scrubbed (MDEV-31253)
- InnoDB recovery hangs after reporting corruption (MDEV-31353)
- `!cursor->index->is_committed()` in row0ins.cc after update to 10.4.13 from 10.3.21 (MDEV-22739)
- innodb_undo_log_truncate=ON fails to wait for purge of enough transaction history (MDEV-31355)
- SET GLOBAL innodb_undo_log_truncate=ON does not free space when no undo logs exist (MDEV-31382)
- innodb_read_ahead_threshold (linear read-ahead) does not work (MDEV-29967)
- InnoDB recovery and `mariadb-backup --prepare` fail to report detailed progress (MDEV-29911)
- `fil_ibd_create()` may hijack the file handle of an old file (MDEV-31347)
- innodb_undo_log_truncate=ON recovery results in a corrupted undo log (MDEV-31373)
- Server freeze due to innodb_change_buffering and innodb_file_per_table=0 (MDEV-31088)
- Change buffer entries are left behind when freeing a page, causing secondary index corruption when the page is later reused (MDEV-31385)
- Foreign Key Constraint actions don't affect Virtual Column (MDEV-18114)

Aria

- Various crashes upon INSERT/UPDATE after changing Aria settings (MDEV-28054)
- Various crashes/asserts/corruptions when Aria encryption is enabled/used, but the encryption plugin is not loaded (MDEV-26258)

Spider

- SIGSEGV in `spider_db_open_item_field` and SIGSEGV in `spider_db_print_item_type`, on SELECT (MDEV-29447)
- Spider variables that double as table params overriding mechanism is buggy (MDEV-31524)

Optimizer

- ANALYZE FORMAT=JSON now includes InnoDB engine statistics for each table (MDEV-31577)
- Assertion `` `last_key_entry >= end_pos' `` failed in virtual bool `JOIN_CACHE_HASHED::put_record()` (MDEV-31348)
- Problem with open ranges on prefix blobs keys (MDEV-31800)
- Equal on two RANK window functions create wrong result (MDEV-20010)
- Recursive CTE execution is interrupted without errors or warnings (MDEV-31214)
- Assertion `` `s->table->opt_range_condition_rows <= s->found_records' `` failed in `apply_selectivity_for_table` (MDEV-31449)
- Inconsistency between MRR and SQL layer costs can cause poor query plan (MDEV-31479)
- `MAX_SEL_ARG` memory exhaustion is not visible in the optimizer trace (MDEV-30964)
- SHOW TABLES not working properly with `lower_case_table_names=2` (MDEV-30765)
- Segfault on select query using index for group-by and filesort (MDEV-30143)
- Server crash in `store_length`, assertion failure in `Type_handler_string_result::sort_length` (MDEV-31743)

Replication

- Parallel Slave SQL Thread Can Update Seconds_Behind_Master with Active Workers (MDEV-30619)
- ALTER SEQUENCE ends up in optimistic parallel slave binlog out-of-order (MDEV-31503)
- STOP SLAVE takes very long time on a busy system (MDEV-13915)
- On slave XA COMMIT/XA ROLLBACK fail to return an error in read-only mode (MDEV-30978)
- Calling a function from a different database in a slave side trigger crashes (MDEV-29894)
- `rpl.rpl_manual_change_index_file` occasionally fails in BB with Result length mismatch (MDEV-30214)

Galera

- Node never returns from Donor/Desynced to Synced when `wsrep_mode = BF_ABORT_MARIABACKUP` (MDEV-31737)
- Node has been dropped from the cluster on Startup / Shutdown with async replica (MDEV-31413)
- KILL QUERY maintains nodes data consistency but breaks GTID sequence (MDEV-31075)
- Assertion failure `!lock.was_chosen_as_deadlock_victim in trx0trx.h:1065` (MDEV-30963)
- MariaDB stuck on starting commit state (waiting on commit order critical section) (MDEV-29293)
- Assertion `state() == s_aborting || state() == s_must_replay` failed in int wsrep::transaction::after_rollback() (MDEV-30013)
- Assertion `!wsrep_has_changes(thd) || (thd->lex->sql_command == SQLCOM_CREATE_TABLE && !thd->is_current_stmt_binlog_format_row()) || thd->wsrep_cs().transaction().state() == wsrep::transaction::s_aborted` failed (MDEV-30388)
- Server crashes when wsrep_sst_donor and wsrep_cluster_address set to NULL (MDEV-28433)
- Create temporary sequence can cause inconsistency (MDEV-31335)
- Galera 4 unable to query cluster state if not primary component (MDEV-21479)

Changelog

For a complete list of changes made in MariaDB 11.0.3, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 11.0.3, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.0.13.3 MariaDB 11.0.2 Release Notes

The most recent release of MariaDB 11.0 is:
MariaDB 11.0.4 Stable (GA) Download Now
Alternate download from mariadb.org

Download 11.0.2 | Release Notes | Changelog | Overview of 11.0

Release date: 6 Jun 2023

MariaDB 11.0 is a current short-term stable series of MariaDB and will be maintained until June 2024. It is an evolution of MariaDB 10.11 with several entirely new features.

MariaDB 11.0.2 is a ***Stable (GA)*** release.

For an overview of MariaDB 11.0 see the What is MariaDB 11.0? page.

Thanks, and enjoy MariaDB!

Notable Items

Functions

- Given a time in picoseconds, the new function FORMAT_PICO_TIME returns a human-readable time value and unit indicator (MDEV-19629)

InnoDB

- InnoDB does not free UNDO after the fix of MDEV-30671 (MDEV-31234)
- InnoDB hang fixes (MDEV-31158, MDEV-31343, MDEV-31350)
- Innodb_buffer_pool_read_requests is not updated correctly (MDEV-31309)
- InnoDB monitor trx_rseg_history_len was accidentally disabled by default (MDEV-31308)
- Revert "MDEV-30473: Do not allow GET_LOCK() / RELEASE_LOCK() in cluster"

Optimizer

- Split Materialized optimization is improved to re-fill the materialized table only if necessary. The fewer number of table refills is taken into account when choosing query plan, too (MDEV-26301)
- New optimizer_switch option, hash_join_cardinality, is added. It is ON by default. When set to ON, the optimizer will produce tighter bounds for hash join output cardinality. (MDEV-30812)
- Crash with condition pushable into derived and containing outer reference (MDEV-31403 MDEV-31240)
- Crash with EXPLAIN EXTENDED for multi-table update of system table (MDEV-31224)
- ANALYZE FORMAT=JSON now prints more information about Block Nested Loop joins: `block-nl-join` element now has `r_loops`, `r_effective_rows` and `r_other_time_ms` fields (MDEV-30806, MDEV-30830, MDEV-30972).

Variables

- New status variable: max_used_connections_time (MDEV-30543)

Changelog

For a complete list of changes made in MariaDB 11.0.2, with links to detailed information on each push, see the changelog.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.0.13.4 MariaDB 11.0.1 Release Notes

Download | Release Notes | Changelog | Overview of 11.0

Release date: 22 Feb 2023

Do not use non-stable (non-GA) releases in production!

MariaDB 11.0 is the current development series of MariaDB. It is an evolution of MariaDB 10.11 with several entirely new features.

For an overview of MariaDB 11.0 see the What is MariaDB 11.0? page.

Thanks, and enjoy MariaDB!

Notable Items

InnoDB

- Remove the global sequence DICT_HDR_ROW_ID for DB_ROW_ID (MDEV-19506)
- Remove the InnoDB change buffer (MDEV-29694)
- Deprecate innodb_file_per_table (MDEV-29983)
- Set innodb_undo_tablespaces=3 by default (MDEV-29986)
- Deprecate innodb_flush_method (MDEV-30136)
- Deprecate innodb_defragment and related parameters (MDEV-30544)
- InnoDB read-ahead may cause page writes (MDEV-26790)
- Read-ahead unnecessarily allocates and frees pages when a page is in the buffer pool (MDEV-30216)
- Full-text index corruption with system versioning (MDEV-25004)
- innodb_undo_log_truncate=ON recovery and backup fixes (MDEV-29999, MDEV-30179, MDEV-30438)
- Upgrade after a crash is not supported (MDEV-24412)
- Remove InnoDB buffer pool load throttling (MDEV-25417)
- InnoDB shutdown hangs when the change buffer is corrupted (MDEV-30009)
- `innodb_fast_shutdown=0` fails to report change buffer merge progress (MDEV-29984)
- `mariadb-backup --backup --incremental --throttle=...` hangs (MDEV-29896)
- Crash after recovery, with InnoDB: Tried to read (MDEV-30132)
- Trying to write ... bytes at ... outside the bounds (MDEV-30069)
- TRUNCATE breaks FOREIGN KEY locking (MDEV-29504, MDEV-29849)
- `INFORMATION_SCHEMA.INNODB_TABLESPACES_ENCRYPTION.NAME` is NULL for undo tablespaces (MDEV-30119)
- Fixed hangs and error handling in B-tree operations (MDEV-29603, MDEV-30400)
- InnoDB bulk insert fixes (MDEV-30047, MDEV-30321)
- InnoDB fails to start on `innodb_undo_tablespaces` mismatch (MDEV-30158)
- `mariabackup.skip_innodb` crashes when `innodb_undo_tablespaces > 0` (MDEV-30122)

Galera

- Fixes for cluster wide write conflict resolving (MDEV-29684)

Replication

- Parallel slave applying in binlog order is corrected for admin class of commands including ANALYZE (MDEV-30323)
- `Seconds_Behind_Master` is showed now more precisely at the slave applier start, including in the delayed mode (MDEV-29639)
- mariadb-binlog --verbose is made to show the type of compressed columns (MDEV-25277)
- Deadlock is resolved on replica involving `BACKUP STAGE BLOCK_COMMIT` and a committing user XA (MDEV-30423)

Docker Official Images

- 11.0, unlike previous version, no longer includes mysql named compatible executable symlinks inside the container.

Packaging

- mysql compatible symlinks are no longer installed in core package, but are delegated to a -compat package (MDEV-30203)
- Use of mysql named executables (except for Windows) will result in a deprecation warning (MDEV-29582)

General

- Infinite sequence of recursive calls when processing embedded CTE (MDEV-30248)
- Crash with a query containing nested WINDOW clauses (MDEV-30052)
- Json Range only affects first row of the result set (MDEV-30304)
- In this release repositories for Fedora 37 and Ubuntu 22.10 Kinetic have been added.

Changelog

For a complete list of changes made in MariaDB 11.0.1, with links to detailed information on each push, see the changelog.

Do not use non-stable (non-GA) releases in production!

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.0.13.5 MariaDB 11.0.0 Release Notes

The most recent release of MariaDB 11.0 is:
MariaDB 11.0.4 Stable (GA) Download Now
Alternate download from mariadb.org

Download | Release Notes | Changelog | Overview of 11.0

Release date: 27 Dec 2022

Do not use *alpha* releases in production!

MariaDB 11.0 is a current development series of MariaDB, and will be maintained for five years. It is an evolution of MariaDB 10.11 with several entirely new features.

MariaDB 11.0.0 is a single preview release. Features are to be considered preview, and none are guaranteed to make it into MariaDB 11.0.

For an overview of MariaDB 11.0 see the What is MariaDB 11.0? page.

Thanks, and enjoy MariaDB!

New optimizer cost model

This is the main change that practically defines this release. Read about new optimizer cost model on its own page.

Galera

- MDEV-22570 Implement wsrep_provider_options as plugin
- MDEV-29281 Add details about node eviction status to the JSON file with Galera node status

Removing/Deprecating old code

- MDEV-29694 Remove the InnoDB change buffer
- MDEV-30136 Deprecate innodb_flush_method
- MDEV-29983 Deprecate innodb_file_per_table
- MDEV-30128 remove support for 5.1- replication events
- MDEV-29582 deprecate mysql* names
- MDEV-29227 deprecate explicit_defaults_for_timestamp=0
- MDEV-28910 remove the 5.5.5- version hack
- MDEV-29668 SUPER no longer allows actions that have fine-grained dedicated privileges

Other changes

- MDEV-30203 Move mysql compatibility symlinks to different package
- MDEV-30153 ad hoc client versions are confusing
- MDEV-29986 Set innodb_undo_tablespaces=3 by default

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.1 MariaDB Server 10.11

Changes and Improvements in MariaDB 10.11
Current Version: 10.11.6 | Status: Stable (GA) | Release Date: 13 Nov 2023

Release Notes - MariaDB 10.11 Series
MariaDB 10.11 series release notes.

Changelogs - MariaDB 10.11 Series
MariaDB 10.11 changelogs

7.0.1.1 Changes and Improvements in MariaDB

10.11

The most recent release of MariaDB 10.11 is:
MariaDB 10.11.6 Stable (GA) Download Now
Alternate download from mariadb.org

Contents

MariaDB 10.11 is the current long-term maintenance release series, maintained until February 2028.

Upgrading

- See Upgrading Between Major MariaDB Versions and Upgrading from MariaDB 10.6 to MariaDB 10.11.

New Features & Improvements

This list includes features from the short-term releases MariaDB 10.7, MariaDB 10.8, MariaDB 10.9 and MariaDB 10.10.

Authentication

- GRANT to PUBLIC (MDEV-5215) (blog post)
- Separate SUPER and READ ONLY ADMIN privileges (MDEV-29596)
- bind_address now accepts a comma-separated list of addresses to bind to (MDEV-24377)
- password_reuse_check plugin is a new password validation plugin that prevents the new password from being the same as the one being used during the configurable retention period. (MDEV-9245, MariaDB 10.7)

Optimizer

- Make ANALYZE FORMAT=JSON show time spent in the query optimizer (MDEV-28926)
- Improve optimization of joins with many tables, including eq_ref tables (MDEV-28852, MariaDB 10.10)
- Table elimination does not work across derived tables (MDEV-26278, MariaDB 10.10)
- Histograms in the statistics tables are more precise and stored as JSON, not binary (MDEV-21130, MDEV-26519, blog post, MariaDB 10.8).
- Improve simple multibyte collation performance on the ASCII range (MDEV-26572, MariaDB 10.7).

Descending Indexes

- Individual columns in the index can now be explicitly sorted in the ascending or descending order. This can be useful for optimizing certain ORDER BY cases (MDEV-13756, MDEV-26938, MDEV-26939, MDEV-26996, MariaDB 10.8).

Replication

- Change defaults for CHANGE MASTER TO so that GTID-based replication is used by default if master supports it (MDEV-19801, MariaDB 10.10)
- Added global.slave_max_statement_time system variable for SQL thread to limit maximum execution time per query replicated (MDEV-27161, MariaDB 10.10)
- Deprecate MASTER_USE_GTID=Current_Pos to favor new MASTER_DEMOTE_TO_SLAVE option (MDEV-20122, MariaDB 10.10)
- Implement the --do-domain-ids, --ignore-domain-ids, and --ignore-server-ids options for mariadb-binlog (MDEV-20119, MariaDB 10.9)
- Semisync-slave server recovery is extended to work on new server_id server (MDEV-27342, MariaDB 10.9)
- mariadb-binlog --stop-never --raw now flushes the result file to disk after each processed event so the file can be listed with the actual bytes (MDEV-14608, MariaDB 10.9)
- Normally, ALTER TABLE gets fully executed on the primary first and only then it is replicated and starts executing on replicas. With this feature `ALTER TABLE` gets replicated and starts executing on replicas when it *starts* executing on the primary, not when it *finishes*. This way the replication lag caused by a heavy `ALTER TABLE` can be completely eliminated (MDEV-11675, MariaDB 10.8).
- Multi-source replication supports MySQL-style CHANNEL syntax (MDEV-26307, MariaDB 10.7)

mysqlbinlog GTID support

- mariadb-binlog (or `mysqlbinlog` as it was called back when the task was created) now supports both filtering events by GTID ranges through `--start-position` and `--stop-position`, and validating a binary log's ordering of GTIDs through `--gtid-strict-mode` (MDEV-4989, MariaDB 10.8).

Galera

- Implement a method to add IPs to allowlist for Galera Cluster node addresses that can make SST/IST requests (MDEV-27246, MariaDB 10.10)
- JSON file interface to wsrep node state / SST progress logging (MDEV-26971, MariaDB 10.9)

JSON

- JSON_OVERLAPS function (MDEV-27677, MariaDB 10.9)
- Implement range notation for JSONPath (MDEV-27911, MariaDB 10.9)
- Support JSONPath negative index (MDEV-22224, MariaDB 10.9)
- JSON_EQUALS function to check for equality between JSON objects (MDEV-23143, MariaDB 10.7).
- JSON_NORMALIZE function, which recursively sorts keys and removes spaces (MDEV-16375, MariaDB 10.7)

UUID

- New UUID data type (MDEV-4958, MariaDB 10.7)

SHOW ANALYZE FORMAT=JSON

- Extend SHOW EXPLAIN to support SHOW ANALYZE [FORMAT=JSON] (MDEV-27021, MariaDB 10.9)
- Add EXPLAIN FOR CONNECTION syntax support to SHOW EXPLAIN (MDEV-10000, MariaDB 10.9)

Information Schema

- Performance Issues reading the Information Schema Parameters table (MDEV-29104)
- Full table scan in the Information Schema Parameters and Information Schema Routines tables (MDEV-20609)

System versioning

- System versioning setting, system_versioning_insert_history, to allow history modification (MDEV-16546)
- mariadb-dump: dump and restore historical data (MDEV-16029)
- Add option to dump system versioned table as of specified timestamp (MDEV-16355, MariaDB 10.7).

InnoDB

- InnoDB performance improvements (MDEV-27557, MDEV-28185, MDEV-27767, MDEV-28313, MDEV-28137, MDEV-28465, MDEV-26789, MariaDB 10.9)
- In bulk insert, pre-sort and build indexes one page at a time (MDEV-24621, MariaDB 10.7)

InnoDB Redo Log Improvements

- autosize innodb_buffer_pool_chunk_size (MDEV-25342, MariaDB 10.8).
- Improve the redo log for concurrency (MDEV-14425, MariaDB 10.8).
- Remove FIL_PAGE_FILE_FLUSH_LSN (MDEV-27199, MariaDB 10.8).

UCA14 Collation

- Add UCA-14.0.0 collations (MDEV-27009, MariaDB 10.10)
- Improve contraction performance in UCA collations (MDEV-27265, MariaDB 10.10)
- Improve UCA collation performance for utf8mb3 and utf8mb4 (MDEV-27266, MariaDB 10.10)

Windows

- Passwordless login for mariadb root user, for OS admin users (MDEV-26715)
- On newer versions of Windows (Windows 10 1903 or later), the `mariadb` client defaults to the utf8mb4 character set. Several problems with Unicode input and output in client were fixed. Command line utilities now accept all Unicode characters in user names, database names, file names etc (in the past, characters were restricted to the current ANSI codepage) (MariaDB 10.8).

Spider Storage Engine

- This was mostly internal refactoring work. As a result one can now declare Spider connections using the `REMOTE_SERVER`, `REMOTE_DATABASE`, and `REMOTE_TABLE` attributes and not abuse the `COMMENT` field for that. This works both for the whole table and per partition (MDEV-5271, MDEV-27106, MariaDB 10.8).

Convert Partitions

- ALTER TABLE ... CONVERT PARTITION .. TO TABLE (MDEV-22166, MariaDB 10.7), and
- ALTER TABLE ... CONVERT TABLE ... TO PARTITION ... (MDEV-22165) as an easy way to convert tables to partitions and back in one command, instead of a sequence of CREATE/EXCHANGE/DROP (MariaDB 10.7)

Miscellaneous

- Add RANDOM_BYTES function (MDEV-25704, MariaDB 10.10)
- The INET4 data type (MDEV-23287, MariaDB 10.10)
- Re-design the upper level of handling UPDATE and DELETE statements (MDEV-28883, MariaDB 10.10)
- Deprecate the DES_ENCRYPT/DECRYPT functions (MDEV-27104, MariaDB 10.10)
- Hashicorp Key Management Plugin for implementing encryption using keys stored in the Hashicorp Vault KMS (MDEV-19281, MariaDB 10.9)
- Stored procedures already have support for the IN, OUT and INOUT parameter qualifiers. Added as well for stored functions and (IN only) cursors (MDEV-10654). This was a contribution by ManoharKB (MariaDB 10.8).
- Add an optional argument to the CRC32() function, as well as the CRC32C() function, which uses the Castagnoli polynomial. (MDEV-27208). **Note:** The order of the 2-ary arguments was swapped after the preview release: `crc32('MariaDB')=crc32(crc32('Maria'),'DB')` (MariaDB 10.8)
- my_print_defaults now handles `--default-*` options in exactly the same way as other MariaDB tools (MDEV-26238, MariaDB 10.8).
- UCA collations are now notably faster (MDEV-27266, MDEV-27265, MariaDB 10.8).
- NATURAL_SORT_KEY function (MDEV-4742, MariaDB 10.7).
- Five **provider plugins** (bzip2, lzma, lz4, lzo, snappy) provide compression capabilities to the server and storage engines (MDEV-12933, blog post, MariaDB 10.7).
- SFORMAT function for arbitrary text formatting (MDEV-25015, MariaDB 10.7)
- GET DIAGNOSTICS supports a new condition property name `ROW_NUMBER`. In multi-row inserts it allows one to retrieve a number of a row that has caused the error (MDEV-10075, MDEV-26611, MariaDB 10.7)

Variables

- For a list of all new variables, see System Variables Added in MariaDB 10.11.

- Rename slow queries variables (MDEV-7567)
  - log_slow_min_examined_row_limit (min_examined_row_limit)
  - log_slow_query (slow_query_log)
  - log_slow_query_file (slow_query_log_file). This was named log_slow_query_file_name in the MariaDB 10.11.0 preview release.
  - log_slow_query_time (long_query_time)
- replicate_rewrite_db is now a system variable, no longer just an option (MDEV-15530)
- Change default of explicit_defaults_for_timestamp to ON (MDEV-28632, MariaDB 10.10)
- --ssl option set as default for mariadb CLI (MDEV-27105, MariaDB 10.10)
- Merge (and deprecate) old to old_mode sql variable (MDEV-24920, MariaDB 10.9)
- Deprecate the keep_files_on_create variable (MDEV-23570, MariaDB 10.8).
- Deprecate wsrep_replicate_myisam (MDEV-24947, MariaDB 10.7)
- Deprecate wsrep_strict_ddl (MDEV-24843, MariaDB 10.7)

InnoDB Variables

- innodb_write_io_threads and innodb_read_io_threads are now dynamic, and their values can be changed without restarting the server (MDEV-11026)
- Removed innodb_version (MDEV-28554, MariaDB 10.10)
- Deprecated innodb_prefix_index_cluster_optimization (MariaDB 10.10)
- Deprecated innodb_change_buffering (MariaDB 10.9)
- innodb_disallow_writes removed (MDEV-25975, MariaDB 10.9)
- innodb_log_file_size is now dynamic (MDEV-27812, MariaDB 10.9)

Spider Variables

The following deprecated variables have been removed (MariaDB 10.10):

- spider_udf_ct_bulk_insert_interval
- spider_udf_ct_bulk_insert_rows
- spider_udf_ds_bulk_insert_rows
- spider_udf_ds_table_loop_mode
- spider_udf_ds_use_real_table
- spider_use_handle
- spider_udf_table_mon_mutex_count
- spider_use_handler

Security Vulnerabilities Fixed in MariaDB 10.11

> For a complete list of security vulnerabilities (CVEs) fixed across all versions of MariaDB, see the Security Vulnerabilities Fixed in MariaDB page.

- CVE-2023-22084: MariaDB 10.11.6
- CVE-2022-47015: MariaDB 10.11.3

List of All MariaDB 10.11 Releases

Date	Release	Status	Release Notes	Changelog
13 Nov 2023	**MariaDB 10.11.6**	Stable (GA)	Release Notes	Changelog
14 Aug 2023	**MariaDB 10.11.5**	Stable (GA)	Release Notes	Changelog
7 Jun 2023	**MariaDB 10.11.4**	Stable (GA)	Release Notes	Changelog
10 May 2023	**MariaDB 10.11.3**	Stable (GA)	Release Notes	Changelog
16 Feb 2023	**MariaDB 10.11.2**	Stable (GA)	Release Notes	Changelog
17 Nov 2022	**MariaDB 10.11.1**	RC	Release Notes	Changelog
26 Sep 2022	**MariaDB 10.11.0**	Alpha	Release Notes	

7.0.1.2 Release Notes - MariaDB 10.11 Series

MariaDB 10.11.6 Release Notes

Status: Stable (GA) | Release Date: 13 Nov 2023

MariaDB 10.11.5 Release Notes

Status: Stable (GA) | Release Date: 14 Aug 2023

MariaDB 10.11.4 Release Notes

Status: Stable (GA) | Release Date: 7 Jun 2023

MariaDB 10.11.3 Release Notes

Status: Stable (GA) | Release Date: 10 May 2023

MariaDB 10.11.2 Release Notes

Status: Stable (GA) | Release Date: 16 Feb 2023

MariaDB 10.11.1 Release Notes

Status: Release Candidate (RC) | Release Date: 17 Nov 2022

MariaDB 10.11.0 Release Notes

Status: Alpha | Release Date: 26 Sep 2022

7.0.1.2.1 MariaDB 10.11.6 Release Notes

Download | Release Notes | Changelog | Overview of 10.11

Alternate download from mariadb.org

Release date: 13 Nov 2023

MariaDB 10.11 is the current stable long term series of MariaDB, maintained until February 2028. It is an evolution of MariaDB 10.10 with several entirely new features.

MariaDB 10.11.6 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.11 see the What is MariaDB 10.11? page.

Thanks, and enjoy MariaDB!

Notable Items

InnoDB

- DROP INDEX followed by CREATE INDEX may corrupt data (MDEV-32132)
- ROW_FORMAT=COMPRESSED table: InnoDB: 2048 bytes should have been read. Only 0 bytes read. (MDEV-31875)
- Server aborts during alter operation when table doesn't have foreign index (MDEV-32527)
- row_merge_fts_doc_tokenize() handles FTS plugin parser inconsistently (MDEV-32578)
- InnoDB: tried to purge non-delete-marked record of an index on a virtual column prefix (MDEV-30024)
- lock_row_lock_current_waits counter in information_schema.innodb_metrics may become negative (MDEV-30658)
- SET GLOBAL innodb_max_purge_lag_wait=… hangs if innodb_read_only=ON (MDEV-31813)
- Auto-increment no longer works for explicit FTS_DOC_ID (MDEV-32017)
- Assertion `pos < table->n_def' failed in dict_table_get_nth_col (MDEV-32337)
- innochecksum man pages seem to be inconsistent with the binary (10.2.25) (MDEV-20583)
- innodb_compression_algorithm=0 (none) increments Innodb_num_pages_page_compression_error (MDEV-30825)
- wrong table name in innodb's "row too big" errors (MDEV-32128)
- Optimize is_file_on_ssd() to speedup opening tablespaces on Windows (MDEV-32228)
- Race condition between page write completion and log checkpoint (MDEV-32511)
- After crash recovery, Checksum mismatch + Failing assertion: !i || prev_id + 1 == space_id, (MDEV-31851)
- Deadlock due to log_free_check(), involving trx_purge_truncate_rseg_history() and trx_undo_assign_low() (MDEV-32049)

- Write-ahead logging is broken for freed pages (MDEV-32552)
- X-lock on supremum for prepared transaction for RR (MDEV-30165)
- Assertion failures in log_sort_flush_list upon crash recovery (MDEV-32029)
- Race condition between page write completion and log checkpoint (MDEV-32511)
- Assertion `purge_sys.tail.trx_no <= purge_sys.rseg->last_trx_no()' (MDEV-30100)
- Assertion `index->is_btree() || index->is_ibuf()' failed in btr_search_guess_on_hash (MDEV-30802)
- InnoDB hang in buf_flush_wait_LRU_batch_end() (MDEV-32134)
- InnoDB may hang when running out of buffer pool (MDEV-32588)
- Corrupt index(es) on busy table when using FOREIGN KEY (MDEV-30531)
- InnoDB Recovery doesn't display encryption message when no encryption configuration passed (MDEV-31098)
- Wrong information about innodb_checksum_algorithm in the information_schema.SYSTEM_VARIABLES (MDEV-31473)
- InnoDB may fail to recover after being killed in fil_delete_tablespace() (MDEV-31826)
- Create separate tpool thread for async aio (MDEV-31095)
- UNDO logs still growing for write-intensive workloads (MDEV-32050)
    - Increase the default for innodb_purge_batch_size from 300 to 1000 (MDEV-32050)
    - Deprecate innodb_purge_rseg_truncate_frequency.
    - The motivation for introducing this in MySQL seems to have been to avoid stalls due to freeing undo log pages or truncating undo log tablespaces. In MariaDB, innodb_undo_log_truncate=ON should be a much lighter operation because it will not involve any log checkpoint. (MDEV-32050)
- Slow full index scan in 10.6 vs 10.5 for the (slow) I/O-bound case (MDEV-30986)
- LOAD DATA into InnoDB w/partitions: huge performance loss, affected 10.6+ (MDEV-31835)
- Disable read-ahead for temporary tablespace (MDEV-32145)

Optimizer

- Crash when HAVING in a correlated subquery references columns in the outer query (MDEV-29731)
- Server crashes at TABLE::add_tmp_key (MDEV-32320)
- Server crashes inside filesort at my_decimal::to_binary (MDEV-32324)
- Assertion `bitmap_is_set(&m_part_info->read_partitions, m_part_spec.start_part)' failed in ha_partition::handle_ordered_index_scan (MDEV-24283)
- Crash when searching for the best split of derived table (MDEV-32064)
- Test case from opt_tvc.test fails with statement memory protection (MDEV-32225)
- Significant slowdown for query with many outer joins (MDEV-32351)
- test_if_skip_sort_order() should catch the join types JT_EQ_REF, JT_CONST and JT_SYSTEM and skip sort order for these (MDEV-32475)
- jointable materialization subquery optimization ignoring errors, then failing ASSERT. (MDEV-31983)
- Server crashes in JOIN::cleanup after erroneous query with view (MDEV-32164)
- Prepared statement return wrong result (missing row) (MDEV-9938)
- Assertion `range->rows >= s->found_records' failed in best_access_path (MDEV-32682)

Replication

- rpl.rpl_parallel_temptable failure due to incorrect commit optimization of temptables (MDEV-10356)
- Lock wait timeout with INSERT-SELECT, autoinc, and statement-based replication (MDEV-31482)
- strings/ctype-ucs2.c:2336: my_vsnprintf_utf32: Assertion `(n % 4) == 0' failed in my_vsnprintf_utf32 on INSERT (MDEV-32249)
- Assertion fails in MDL_context::acquire_lock upon parallel replication of CREATE SEQUENCE (MDEV-31792)
- SHOW SLAVE STATUS Last_SQL_Errno Race Condition on Errored Slave Restart (MDEV-31177)
- seconds_behind_master is inaccurate for Delayed replication (MDEV-32265)
- detailize the semisync replication magic number error (MDEV-32365)
- Parallel replication deadlock victim preference code errorneously removed (MDEV-31655)
- Parallel replication lags because innobase_kill_query() may fail to interrupt a lock wait (MDEV-32096)

Galera

- Assertion `state() == s_executing || state() == s_prepared || state() == s_committing || state() == s_must_abort || state() == s_replaying' failed. (MDEV-24912)
- Assertion `state() == s_executing || state() == s_preparing || state() == s_prepared || state() == s_must_abort || state() == s_aborting || state() == s_cert_failed || state() == s_must_replay' failed (MDEV-31285)
- wsrep_sst_mariabackup not working on FreeBSD (MDEV-31467)
- Galera library 26.4.16 fails with every server version (MDEV-32024)
- Galera node remains paused after interleaving FTWRLs (MDEV-32282)
- Failed to insert streaming client (MDEV-32051)
- When set at runtime, wsrep_sst_method accepts any value (MDEV-31470)
- galera needs packaging script changes to successfully build (MDEV-32642)

- replication breaks when using optimistic replication and replica is a galera node (MDEV-31833)
- McAfee database vulnerability scan caused MariaDB crash with signal 6 (system abort) (MDEV-27004)
- Assertion `mode_ == m_local || transaction_.is_streaming()' failed in int wsrep::client_state::bf_abort(wsrep::seqno) (MDEV-30217)

Data Definition

- MariaDB crash on calling function (MDEV-23902)
- ASAN errors in grn_obj_unlink / ha_mroonga::clear_indexes upon index operations (MDEV-31970)
- vcol circular references lead to stack overflow (MDEV-31112)
- OPTIMIZE TABLE crash (MDEV-28122)
- Server crashes in Alter_info::add_stat_drop_index upon CREATE TABLE (MDEV-32449)

Scripts and Clients

- mariadb-binlog -T/--table (mysqlbinlog) option (MDEV-25369)
- mariadb-admin (mysqladmin) wrong error with simple_password_check (MDEV-22418)
- mariadb-install-db shows warning on missing directory $pamtooldir/auth_pam_tool_dir (MDEV-32142)
- main.mysql_client_test, main.mysql_client_test_comp failed on ASAN build with error: 5888, status: 23, errno: 2 (MDEV-19369)
- mariadb-install-db (mysql_install_db) doesn't properly grant proxy privileges to all default root user accounts (MDEV-21194)

Tests

- main.events_stress or events.events_stress fails with view-protocol (MDEV-31455)
- main.delete_use_source fails (hangs) with view-protocol (MDEV-31457)
- main.sum_distinct-big and main.merge-big fail with timeout with view-protocol (MDEV-31465)
- main.secure_file_priv_win fails with 2nd execution PS protocol (MDEV-32023)
- Windows : mtr output on is messed up with large MTR_PARALLEL (MDEV-32387)
- main.mysql_client_test_comp failed in buildbot, error on exec (MDEV-16641)
- main.order_by_pack_big fails with view-protocol (MDEV-31460)
- mysql_install_db_win.test fails on second execution (MDEV-32232)

MariaBackup

- MariaBackup full backup failed with InnoDB: Failing assertion: success in storage/innobase/fil/fil0fil.cc line 657 (MDEV-18200)
- mbstream breaks page compression on XFS (MDEV-25734)

Character Sets, Data Types, Collations

- Prefix keys for CHAR work differently for MyISAM vs InnoDB (MDEV-30048)
- Inconsistent results of DISTINCT with NOPAD (MDEV-30050)
- Assertion `(length % 4) == 0' failed in my_lengthsp_utf32 on INSERT (MDEV-28835)
- Compressed varchar values lost on joins when sorting on columns from joined table(s) (MDEV-31724)
- UBSAN shift exponent X is too large for 64-bit type 'long long int' in sql/field.cc (MDEV-32226)
- Wrong bit encoding using COALESCE (MDEV-32244)

Spider

- Spider UBSAN runtime error: applying non-zero offset x to null pointer in st_spider_param_string_parse::restore_delims (MDEV-31117)
- Segfault when setting spider_delete_all_rows to 0 and delete all rows of a spider table, ASAN heap-use-after-free in spider_db_delete_all_rows (MDEV-31996)
- ASAN errors in spider_fields::free_conn_holder or spider_create_group_by_handler (MDEV-28998)
- ASAN: heap-buffer-overflow & stack-buffer-overflow in `spider_db_mbase_row::append_to_str` | SIGSEGV's in `memmove_avx_unaligned_erms` from memcpy in `Binary_string::q_append`, in `Static_binary_string::q_append` and `my_strntoull10rnd_8bit` | Unknown error 12801 (MDEV-29502)

General

- binlog_do_db option breaks importing sql dumps (MDEV-29989)
- Crashes in MDL_key::mdl_key_init with lower-case-table-names=2 (MDEV-32025)

- getting error 'Illegal parameter data types row and bigint for operation '+' ' when using ITERATE in a FOR..DO (MDEV-32275)
- Assertion `arena_for_set_stmt== 0' failed in LEX::set_arena_for_set_stmt upon SET STATEMENT (MDEV-17711)
- main.mysqlcheck fails on ARM with ASAN use-after-poison in my_mb_wc_filename (MDEV-26494)
- main.delayed fails with wrong error code or timeout when executed after main.deadlock_ftwrl (MDEV-27523)
- Assertion failed: !pfs->m_idle || (state == PSI_SOCKET_STATE_ACTIVE) (MDEV-28561)
- MyISAM wrong server status flags (MDEV-28820)
- Server crashes in check_sequence_fields upon CREATE TABLE .. SEQUENCE=1 AS SELECT .. (MDEV-29771)
- slow log Rows_examined out of range (MDEV-30820)
- " `rpm --setugids` " breaks PAM authentication (MDEV-30904)
- incorrect examined rows in case of stored function usage (MDEV-31742)
- Compilation failing on MacOS (unknown warning option -Wno-unused-but-set-variable) (MDEV-31890)
- Server crash upon inserting into Mroonga table with compressed column (MDEV-31966)
- hash unique corrupts index on virtual blobs (MDEV-32012)
- insert into an empty table fails with hash unique (MDEV-32015)
- Valgrind/MSAN warnings in dynamic_column_update_move_left (MDEV-32140)
- Memory leak showed in MDEV-6146 test suite (MDEV-32223)
- Test from subselect.test fails with statement memory protection (MDEV-32245)
- Memory leak when executing PS for query with IN subquery (MDEV-32369)
- Allow the setting of Auto_increment on FK referenced columns (MDEV-32018)
- mariadb-upgrade fails with sql_safe_updates = on (MDEV-29914)
- Assertion `!(thd->server_status & (1U | 8192U))' failed in MDL_context::release_transactional_locks (MDEV-32541)
- Information schema leaks table names and structure to unauthorized users (MDEV-32500)
- Missing CHACHA20-POLY1305 support in WolfSSL (MDEV-31653)
- incorrect error about cyclic reference about JSON type virtual column (MDEV-32586)
- Disable TLS v1.0 and 1.1 for MariaDB (MDEV-31369)
- Better indication of refusing to start because of ProtectHome (MDEV-25177)
- Database upgrade fails: slow_log table (MDEV-27757)
- myrocks_hotbackup.1 and test suite files installed when engine is disabled (MDEV-29993)
- client_ed25519.dll isn't inluded for HeidiSQL. (MDEV-31315)
- Assertion `!m_null_value' failed in int FixedBinTypeBundle<FbtImpl>::cmp_item_fbt::compare or in cmp_item_inet6::compare (MDEV-27207)
- type_test.type_test_double fails with 'NUMERIC_SCALE NULL' (MDEV-22243)
- LeakSanitizer errors in get_quick_select or Assertion `status_var.local_memory_used == 0 || !debug_assert_on_not_freed_memory' failed (MDEV-32476)
- Update signal hander user info more compassion and correct url (MDEV-32535)
- maria-install-db fails on MacOS (MDEV-31871)
- MSAN / Valgrind errors in Item_func_like::get_mm_leaf upon query from partitioned table (MDEV-32388)
- MSAN / Valgrind errors in Item_func_like::get_mm_leaf with temporal field (MDEV-32531)
- ASAN errors in base_list_iterator::next / setup_table_map upon 2nd execution of PS (MDEV-32656)
- safe_mutex: Found wrong usage of mutex 'LOCK_thd_data' and 'wait_mutex' (MDEV-32728)

SQL Error Log Plugin

- Added the sql_error_log_warnings system variable, which permits warnings to be logged in addition to errors.

Docker Official Images

- Invert single and double quotes for sql command definitions in healthcheck.sh due to failure under sql_mode=ANSI_QUOTES - contribution by Dominik Häckel
- healthcheck.sh --no-defaults behaviour was corrected - reported by Dominik Häckel
- Added /docker-entrypoint-init.d for tar{,compression} from mariadb-backup - instructions
- Refactor `docker_mariadb_init` in the entrypoint for extending the MariaDB image
- CIS failure due to world-writable directory /var/run/mysqld, added sticky bit - reported by @ollie1
- Add PROXY privileges for root@MARIADB_ROOT_HOST - reported by Matthieu Gusmini
- healthcheck.sh added --galera_online test, to match what the mariadb-operator does.

Warnings and Notes

- Added the note_verbosity system variable to manage notes when an index cannot be used.

Security

- Fixes for the following security vulnerabilities:
  - CVE-2023-22084

Changelog

For a complete list of changes made in MariaDB 10.11.6, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.11.6, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.1.2.2 MariaDB 10.11.5 Release Notes

The most recent release of MariaDB 10.11 is:
MariaDB 10.11.6 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.11.5 | Release Notes | Changelog | Overview of 10.11

Release date: 14 Aug 2023

MariaDB 10.11 is the current stable long term series of MariaDB, maintained until February 2028. It is an evolution of MariaDB 10.10 with several entirely new features.

MariaDB 10.11.5 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.11 see the What is MariaDB 10.11? page.

Thanks, and enjoy MariaDB!

Notable Items

Upgrading from MySQL

- MariaDB now detects and converts previously incompatible MySQL partition schemes (MDEV-29253)

General

- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.11 for Ubuntu 18.04 LTS "Bionic" and Ubuntu 22.10 "Kinetic"
- In this release repositories for Debian 12 "Bookworm" have been added.
- mariadb-dump --force doesn't ignore error as it should (MDEV-31092)
- 280 Bytes lost in mysys/array.c, mysys/hash.c, sql/sp.cc, sql/sp.cc, sql/item_create.cc, sql/item_create.cc, sql/sql_yacc.yy:10748 when using oracle sql_mode (MDEV-26186)
- SQL/PL package body does not appear in `I_S.ROUTINES.ROUTINE_DEFINITION` (MDEV-30662)
- Unexpected result when combining DISTINCT, subselect and LIMIT (MDEV-28285)
- ROW variables do not get assigned from subselects (MDEV-31250)
- Crash after setting global session_track_system_variables to an invalid value (MDEV-25237)
- ODKU of non-versioning column inserts history row (MDEV-23100)
- UPDATE not working properly on transaction precise system versioned table (MDEV-25644)
- Assertion `` `const_item_cache == true' `` failed in `Item_func::fix_fields` (MDEV-31319)
- ANALYZE doesn't work with pushed derived tables (MDEV-29284)

- `get_partition_set` is never executed in `ha_partition::multi_range_key_create_key` due to bitwise & with 0 constant (MDEV-24712)
- Client can crash the server with a `mysql_list_fields("view")` call (MDEV-30159)
- `I_S.parameters` not immediatly changed updated after procedure change (MDEV-31064)
- bzero wipes more bytes than necessary in `set_global_from_ddl_log_entry` (MDEV-31521)
- Assertion `` `0' `` failed in `Type_handler_row::field_type` upon `TO_CHAR` with wrong argument (MDEV-29152)
- mysql_upgrade fails due to `old_mode=""` , with "Cannot load from mysql.proc. The table is probably corrupted" (MDEV-28915)

Character Sets, Data Types

- UUIDs version >= 6 are now stored without byte-swapping, UUIDs with version >=8 and variant=0 are now considered invalid, old tables are supported, old (always byte swapped) and new (swapped for version < 6) UUIDs can be compared and converted transparently (MDEV-29959)
- UBSAN: null pointer passed as argument 1, which is declared to never be null in `my_strnncoll_binary` on `SELECT ... COUNT` or `GROUP_CONCAT` (MDEV-28384)
- Possibly wrong result or Assertion `0' failed in `Item_func_round::native_op` (MDEV-23838)
- Assertion `` `(length % 4) == 0' `` failed in `my_lengthsp_utf32` on `SELECT` (MDEV-29019)
- UBSAN: negation of -X cannot be represented in type `'long long int'` ; cast to an unsigned type to negate this value to itself in `Item_func_mul::int_op` and `Item_func_round::int_op` (MDEV-30932)
- Assorted assertion failures in json_find_path with certain collations (MDEV-23187)

InnoDB

- Crashing on I/O error is unhelpful (MDEV-27593)
- SIGSEGV in `log_sort_flush_list()` in InnoDB crash recovery (MDEV-31354)
- InnoDB tables are being flagged as corrupted on an I/O bound server (MDEV-31767)
- Duplicate entry allowed into a UNIQUE column (MDEV-31120)
- Server Status Innodb_row_lock_time% is reported in seconds (MDEV-29311)
- innochecksum dies with Floating point exception (MDEV-31641)
- Add InnoDB engine information to the slow query log (MDEV-31558)
- Deadlock with 3 concurrent DELETEs by unique key (MDEV-10962)
- innodb protection against dual processes accessing data insufficient (MDEV-31568)
- ER_DUP_KEY in `mysql.innodb_table_stats` upon RENAME on sequence (MDEV-31607)
- Assertion `` `!strcmp(index->table->name.m_name, "SYS_FOREIGN") || !strcmp(index->table->name.m_name, "SYS_FOREIGN_COLS")' `` failed in `btr_node_ptr_max_size` (MDEV-19216)
- InnoDB: Failing assertion: `page_type == i_s_page_type[page_type].type_value` (MDEV-31386)
- `btr_estimate_n_rows_in_range()` accesses unfixed, unlatched page (MDEV-30648)
- `MODIFY COLUMN` can break FK constraints, and lead to unrestorable dumps (MDEV-31086)
- Recovery or backup failure after innodb_undo_log_truncate=ON (MDEV-31487)
- Assertion `'n & PENDING'` failed in `fil_space_t::set_needs_flush()` (MDEV-31442)
- `fil_node_open_file()` releases `fil_system.mutex` allowing other thread to open its file node (MDEV-31256)
- ASAN errors in `dict_v_col_t::detach` upon adding key to virtual column (MDEV-31416)
- Purge trying to access freed secondary index page (MDEV-31264)
- Freed data pages are not always being scrubbed (MDEV-31253)
- InnoDB recovery hangs after reporting corruption (MDEV-31353)
- `!cursor->index->is_committed()` in row0ins.cc after update to 10.4.13 from 10.3.21 (MDEV-22739)
- innodb_undo_log_truncate=ON fails to wait for purge of enough transaction history (MDEV-31355)
- SET GLOBAL innodb_undo_log_truncate=ON does not free space when no undo logs exist (MDEV-31382)
- innodb_read_ahead_threshold (linear read-ahead) does not work (MDEV-29967)
- InnoDB recovery and `mariadb-backup --prepare` fail to report detailed progress (MDEV-29911)
- `fil_ibd_create()` may hijack the file handle of an old file (MDEV-31347)
- innodb_undo_log_truncate=ON recovery results in a corrupted undo log (MDEV-31373)
- Server freeze due to innodb_change_buffering and innodb_file_per_table=0 (MDEV-31088)
- Change buffer entries are left behind when freeing a page, causing secondary index corruption when the page is later reused (MDEV-31385)
- Foreign Key Constraint actions don't affect Virtual Column (MDEV-18114)

Aria

- Various crashes upon INSERT/UPDATE after changing Aria settings (MDEV-28054)
- Various crashes/asserts/corruptions when Aria encryption is enabled/used, but the encryption plugin is not loaded (MDEV-26258)

Spider

- SIGSEGV in `spider_db_open_item_field` and SIGSEGV in `spider_db_print_item_type`, on SELECT (MDEV-29447)
- Spider variables that double as table params overriding mechanism is buggy (MDEV-31524)

Optimizer

- ANALYZE FORMAT=JSON now includes InnoDB engine statistics for each table (MDEV-31577)
- Assertion `` `last_key_entry >= end_pos' `` failed in virtual bool `JOIN_CACHE_HASHED::put_record()` (MDEV-31348)
- Problem with open ranges on prefix blobs keys (MDEV-31800)
- Equal on two RANK window functions create wrong result (MDEV-20010)
- Recursive CTE execution is interrupted without errors or warnings (MDEV-31214)
- Assertion `` `s->table->opt_range_condition_rows <= s->found_records' `` failed in `apply_selectivity_for_table` (MDEV-31449)
- Inconsistency between MRR and SQL layer costs can cause poor query plan (MDEV-31479)
- `MAX_SEL_ARG` memory exhaustion is not visible in the optimizer trace (MDEV-30964)
- SHOW TABLES not working properly with `lower_case_table_names=2` (MDEV-30765)
- Segfault on select query using index for group-by and filesort (MDEV-30143)
- Server crash in `store_length`, assertion failure in `Type_handler_string_result::sort_length` (MDEV-31743)

Replication

- Parallel Slave SQL Thread Can Update Seconds_Behind_Master with Active Workers (MDEV-30619)
- ALTER SEQUENCE ends up in optimistic parallel slave binlog out-of-order (MDEV-31503)
- STOP SLAVE takes very long time on a busy system (MDEV-13915)
- On slave XA COMMIT/XA ROLLBACK fail to return an error in read-only mode (MDEV-30978)
- Calling a function from a different database in a slave side trigger crashes (MDEV-29894)
- `rpl.rpl_manual_change_index_file` occasionally fails in BB with Result length mismatch (MDEV-30214)

Galera

- Node never returns from Donor/Desynced to Synced when `wsrep_mode = BF_ABORT_MARIABACKUP` (MDEV-31737)
- Node has been dropped from the cluster on Startup / Shutdown with async replica (MDEV-31413)
- KILL QUERY maintains nodes data consistency but breaks GTID sequence (MDEV-31075)
- Assertion failure `!lock.was_chosen_as_deadlock_victim in trx0trx.h:1065` (MDEV-30963)
- MariaDB stuck on starting commit state (waiting on commit order critical section) (MDEV-29293)
- Assertion `state() == s_aborting || state() == s_must_replay` failed in int wsrep::transaction::after_rollback() (MDEV-30013)
- Assertion `!wsrep_has_changes(thd) || (thd->lex->sql_command == SQLCOM_CREATE_TABLE && !thd->is_current_stmt_binlog_format_row()) || thd->wsrep_cs().transaction().state() == wsrep::transaction::s_aborted` failed (MDEV-30388)
- Server crashes when wsrep_sst_donor and wsrep_cluster_address set to NULL (MDEV-28433)
- Create temporary sequence can cause inconsistency (MDEV-31335)
- Galera 4 unable to query cluster state if not primary component (MDEV-21479)

Changelog

For a complete list of changes made in MariaDB 10.11.5, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.11.5, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.1.2.3 MariaDB 10.11.4 Release Notes

The most recent release of MariaDB 10.11 is:
MariaDB 10.11.6 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.11.4 | Release Notes | Changelog | Overview of 10.11

Release date: 7 Jun 2023

MariaDB 10.11 is the current long term maintenance development series of MariaDB, maintained until February 2028. It is an evolution of MariaDB 10.10 with several entirely new features.

For an overview of MariaDB 10.11 see the What is MariaDB 10.11? page.

Thanks, and enjoy MariaDB!

Notable Items

InnoDB

- Server crashes in st_join_table::choose_best_splitting (MDEV-31403)
- Crash with condition pushable into derived and containing outer reference (MDEV-31240)
- InnoDB does not free UNDO after the fix of MDEV-30671 (MDEV-31234)
- InnoDB hang fixes (MDEV-31158, MDEV-31343, MDEV-31350)
- Innodb_buffer_pool_read_requests is not updated correctly (MDEV-31309)
- InnoDB monitor trx_rseg_history_len was accidentally disabled by default (MDEV-31308)
- Revert "MDEV-30473: Do not allow GET_LOCK() / RELEASE_LOCK() in cluster"

Optimizer

- Crash with condition pushable into derived and containing outer reference (MDEV-31403 MDEV-31240)
- Crash with EXPLAIN EXTENDED for multi-table update of system table (MDEV-31224)

Changelog

For a complete list of changes made in MariaDB 10.11.4, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.11.4, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.1.2.4 MariaDB 10.11.3 Release Notes

The most recent release of MariaDB 10.11 is:
MariaDB 10.11.6 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.11.3 | Release Notes | Changelog | Overview of 10.11

Release date: 10 May 2023

MariaDB 10.11 is the current long term maintenance development series of MariaDB, maintained until February 2028. It is an evolution of MariaDB 10.10 with several entirely new features.

For an overview of MariaDB 10.11 see the What is MariaDB 10.11? page.

Thanks, and enjoy MariaDB!

Notable Items

InnoDB

- Crash on ROLLBACK in a ROW_FORMAT=COMPRESSED table (MDEV-30882)
- UNIQUE USING HASH accepts duplicate entries for tricky collations (MDEV-30034)
- rec_get_offsets() is not optimal (MDEV-30567)
- Performance regression in fil_space_t::try_to_close() introduced in MDEV-23855 (MDEV-30775)
- InnoDB recovery hangs when buffer pool ran out of memory (MDEV-30551)
- InnoDB undo log truncation fails to wait for purge of history (MDEV-30671)
- MariaDB crash due to DB_FAIL reported for a corrupted page (MDEV-30397)
- Deadlock between INSERT and InnoDB non-persistent statistics update (MDEV-30638)
- InnoDB hang on B-tree split or merge (MDEV-29835)
- Performance regression in locking reads from secondary indexes (MDEV-30357)
- Improve adaptive flushing (MDEV-26055)
- Make page flushing even faster (MDEV-26827)
- Purge misses a chance to free not-yet-reused undo pages (MDEV-29593)
- InnoDB temporary tablespace: reclaiming of free space does not work (MDEV-26782)
- Deadlock between CHECK TABLE and bulk insert (MDEV-30798)
- UPPER() returns an empty string for U+0251 in uca1400 collations for utf8 (MDEV-30661)
- System-wide max transaction id corrupted after changing the undo tablespaces (MDEV-30311)
- Fix miscount of doublewrites by Innodb_data_written (MDEV-31124)

Backup

- mariadb-backup doesn't utilise innodb-undo-log-directory (if specified as a relative path) during copy-back operation (MDEV-28187)
- mariabackup issues error messages during InnoDB tablespaces export on partial backup preparing (MDEV-29050)
- mariadb-backup does not copy Aria logs if aria_log_dir_path is used (MDEV-30968)
- Race condition between buffer pool flush and log file deletion in mariadb-backup --prepare (MDEV-30860)

Replication

- Fixed a deadlock on parallel slave involving full image Write event on the sequence engine (MDEV-29621)
- Fixed an attempted out-of-order binlogging error on slave involving ALTER on the sequence engine (MDEV-31077)
- Corrected non-versioned master to versioned slave replication on no-unique attribute table (MDEV-30430)
- Mended encrypted binlog master to error out to gtid-mode slave when master could not decrypt a binlog file (MDEV-28798)
- Refined optimistic parallel slave to error-exit without any hang (MDEV-30780)
- Ensured SHOW-SLAVE-STATUS is processed on the parallel slave having a necessary mutex always intialized (MDEV-30620)
- Fixed the slave applier to report a correct error when gtid_slave_pos insert fails for some (engine) reasons (MDEV-31038)
- Made parallel slave reports in performance schema consistent with that of show-slave-status (MDEV-26071)

Optimizer

- Split Materialized optimization is improved to re-fill the materialized table only if necessary. The fewer number of table refills is taken into account when choosing query plan, too (MDEV-26301)
- New optimizer_switch option, hash_join_cardinality, is added. It is off by default. When set to ON, the optimizer will produce tighter bounds for hash join output cardinality. (MDEV-30812)
- Queries using `SELECT DISTINCT some_expression(aggregate_function())` could produce wrong query result. (MDEV-20057)
- ANALYZE FORMAT=JSON now prints more information about Block Nested Loop joins: `block-nl-join` element now has `r_loops`, `r_effective_rows` and `r_other_time_ms` fields (MDEV-30806, MDEV-30972).
- A GROUP BY query with `MIN(primary_key)` in select list and `primary_key<>const` in the WHERE could produce wrong result when executed with "Using index for group-by" strategy (MDEV-30605)
- EXPLAIN could erroneously report that Rowid Filter optimization is used for partitioned tables. Partitioned tables do not support it. (MDEV-30596)
- A bug in selectivity computations for SINGLE/DOUBLE_PREC_HB histograms could cause wrong estimates to be produced. This could cause the optimizer to pick sub-optimal query plans (MDEV-31067).

Docker Official Image

- Add replication setup to containers contributed by Md Sahil (MDEV-29762)
- Added LTS tags for easier identification of LTS releases:
  - lts-jammy
  - lts

General

- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.11 for Fedora 36.
- In this release repositories for Fedora 38 and Ubuntu 23.04 Lunar have been added.

Security

- Fixes for the following security vulnerabilities:
  - CVE-2022-47015

Changelog

For a complete list of changes made in MariaDB 10.11.3, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.11.3, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.1.2.5 MariaDB 10.11.2 Release Notes

Download | Release Notes | Changelog | Overview of 10.11

Release date: 16 Feb 2023

MariaDB 10.11 is a long term maintenance release series of MariaDB, maintained until February 2028. It is an evolution of MariaDB 10.10 with several entirely new features.

MariaDB 10.11.2 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.11 see the What is MariaDB 10.11? page.

Thanks, and enjoy MariaDB!

Notable Items

- As mentioned in the 10.11.1 release notes, our Yum/DNF/Zypper repositories for Red Hat Enterprise Linux, CentOS, Fedora, openSUSE, and SUSE are changing **with this release** to being signed with a new GPG key with SHA2 digest algorithms instead of SHA1. See this blog post and the GPG page for more details.

InnoDB

- Full-text index corruption with system versioning (MDEV-25004)
- innodb_undo_log_truncate=ON recovery and backup fixes (MDEV-29999, MDEV-30179, MDEV-30438)
- Upgrade after a crash is not supported (MDEV-24412)
- Remove InnoDB buffer pool load throttling (MDEV-25417)
- InnoDB shutdown hangs when the change buffer is corrupted (MDEV-30009)
- `innodb_fast_shutdown=0` fails to report change buffer merge progress (MDEV-29984)
- `mariadb-backup --backup --incremental --throttle=...` hangs (MDEV-29896)
- Crash after recovery, with InnoDB: Tried to read (MDEV-30132)
- Trying to write ... bytes at ... outside the bounds (MDEV-30069)
- TRUNCATE breaks FOREIGN KEY locking (MDEV-29504, MDEV-29849)
- `INFORMATION_SCHEMA.INNODB_TABLESPACES_ENCRYPTION.NAME` is NULL for undo tablespaces (MDEV-30119)
- Fixed hangs and error handling in B-tree operations (MDEV-29603, MDEV-30400)
- InnoDB bulk insert fixes (MDEV-30047, MDEV-30321)
- InnoDB fails to start on `innodb_undo_tablespaces` mismatch (MDEV-30158)
- `mariabackup.skip_innodb` crashes when `innodb_undo_tablespaces > 0` (MDEV-30122)

Galera

- Fixes for cluster wide write conflict resolving (MDEV-29684)

Replication

- Parallel slave applying in binlog order is corrected for admin class of commands including ANALYZE (MDEV-30323)
- Seconds_Behind_Master is now shown now more precisely at the slave applier start, including in the delayed mode (MDEV-29639)
- mariadb-binlog --verbose is made to show the type of compressed columns (MDEV-25277)
- Deadlock is resolved on replica involving `BACKUP STAGE BLOCK_COMMIT` and a committing user XA (MDEV-30423)

General

- Infinite sequence of recursive calls when processing embedded CTE (MDEV-30248)
- Crash with a query containing nested WINDOW clauses (MDEV-30052)
- Major performance regression with 10.6.11 (MDEV-29988)
- Json Range only affects first row of the result set (MDEV-30304)
- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.11 for Fedora 35.
- In this release repositories for Fedora 37 and Ubuntu 22.10 Kinetic have been added.

Changelog

For a complete list of changes made in MariaDB 10.11.1, with links to detailed information on each push, see the changelog.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.1.2.6 MariaDB 10.11.1 Release Notes

The most recent release of MariaDB 10.11 is:
MariaDB 10.11.6 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.11.1 | Release Notes | Changelog | Overview of 10.11

Release date: 17 Nov 2022

Do not use non-stable (non-GA) releases in production!

MariaDB 10.11 is a current development series of MariaDB. It is an evolution of MariaDB 10.10 with several entirely new features.

For an overview of MariaDB 10.11 see the What is MariaDB 10.11? page.

Thanks, and enjoy MariaDB!

Notable Items

Authentication

- bind_address now accepts a comma-separated list of addresses to bind to (MDEV-24377)

InnoDB

- Allow innodb_undo_tablespaces to be changed after database creation (MDEV-19229)

Replication

- Formerly only a server option, replicate_rewrite_db is now a global system variable (MDEV-15530)

Repositories

- Beginning with the next release (Q1 2023), our Yum, DNF, and Zypper repositories for Red Hat, Fedora, and SUSE will be migrated to being signed with a new GPG key. The key we are migrating to is the same one we already use for our Debian and Ubuntu Repositories.
  - The short Key ID is: `0xC74CD1D8`
  - The long Key ID is: `0xF1656F24C74CD1D8`
  - The full fingerprint of the key is: `177F 4010 FE56 CA33 3630 0305 F165 6F24 C74C D1D8`
  - The key can be imported now in preparation for this change using the following command:

```
sudo rpm --import https://supplychain.mariadb.com/MariaDB-Server-GPG-KEY
```

Docker Official Image

The following changes have been made to the `docker.io/library/mariadb` container image.

- The number of gpg packages packages has been removed, leaving enough to `apt-get update`, but `dirmngr` that would fetch keys has been removed. (inspired by issue #469)
- The environment variable `LANG=C.UTF-8` has been added for those that exec into containers and copy paste UTF8 characters (fixes issue #468).

- Adds OCI labels to image (fixes issue 436 and users need for version)
- MariaDB config: skip-host-cache and skip-name-resolve moved to `/etc/mysql/mariadb.conf.d/05-skipcache.cnf`

Changelog

For a complete list of changes made in MariaDB 10.11.1, with links to detailed information on each push, see the changelog.

Do not use non-stable (non-GA) releases in production!

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.1.2.7 MariaDB 10.11.0 Release Notes

The most recent release of MariaDB 10.11 is:
MariaDB 10.11.6 Stable (GA) Download Now
Alternate download from mariadb.org

Download | Release Notes | Changelog | Overview of 10.11

Release date: 26 Sep 2022

Do not use *alpha* releases in production!

MariaDB 10.11 is a current development series of MariaDB. It is an evolution of MariaDB 10.10 with several entirely new features.

Unlike recent new releases, MariaDB 10.11.0 is a single preview release. Features are to be considered preview, and none are guaranteed to make it into MariaDB 10.11.

The preview is available as a container **quay.io/mariadb-foundation/mariadb-devel:10.11-preview**.

For an overview of MariaDB 10.11 see the What is MariaDB 10.11? page.

Thanks, and enjoy MariaDB!

Authentication

- Windows - passwordless login for mariadb root user, for OS admin users, using the gssapi authentication plugin (MDEV-26715)
- GRANT to PUBLIC (MDEV-5215) (blog post)
- Separate SUPER and READ ONLY ADMIN privileges (MDEV-29596)

Optimizer

- Semi-join optimization for single-table update/delete statements (MDEV-7487)
- Allow pushdown of queries involving UNIONs in outer select to foreign engines (MDEV-25080)
- Make ANALYZE FORMAT=JSON show time spent in the query optimizer (MDEV-28926)

Information Schema

- Performance Issues reading the Information Schema Parameters table (MDEV-29104)
- Full table scan in the Information Schema Parameters and Information Schema Routines tables (MDEV-20609)

System versioning

- System versioning setting, system_versioning_insert_history, to allow history modification (MDEV-16546)
- mariadb-dump: dump and restore historical data (MDEV-16029)

InnoDB

- innodb_write_io_threads and innodb_read_io_threads are now dynamic, and their values can be changed without restarting the server (MDEV-11026)

General

- Rename slow queries variables (MDEV-7567)
  - log_slow_min_examined_row_limit (min_examined_row_limit)
  - log_slow_query (slow_query_log)
  - log_slow_query_file_name (slow_query_log_file) This will be renamed to log_slow_query_file in the next MariaDB 10.11 release.
  - log_slow_query_time (long_query_time)
- replicate_rewrite_db is now a system variable, no longer just an option (MDEV-15530)

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.2 MariaDB Server 10.10

Changes and Improvements in MariaDB 10.10
Current Version: 10.10.7 | Status: Stable (GA) | Release Date: 13 Nov 2023

Release Notes - MariaDB 10.10 Series
MariaDB 10.10 series release notes.

Changelogs - MariaDB 10.10 Series
MariaDB 10.10 changelogs

7.0.2.1 Changes and Improvements in MariaDB 10.10

MariaDB 10.10 is no longer maintained. Please use a more recent release.

The most recent release of MariaDB 10.10 is:
MariaDB 10.10.7 Stable (GA) Download Now
Alternate download from mariadb.org

Contents

MariaDB 10.10 is a short-term release series, maintained until November 2023.

New Features & Improvements

Replication

- Change defaults for CHANGE MASTER TO so that GTID-based replication is used by default if master supports it (MDEV-19801)
- Added global.slave_max_statement_time system variable for SQL thread to limit maximum execution time per query replicated (MDEV-27161)
- Deprecate MASTER_USE_GTID=Current_Pos to favor new MASTER_DEMOTE_TO_SLAVE option (MDEV-20122)

Optimizer

- Improve optimization of joins with many tables, including eq_ref tables (MDEV-28852)
- Table elimination does not work across derived tables (MDEV-26278)

UCA14 Collation

- Add UCA-14.0.0 collations (MDEV-27009)
- Improve contraction performance in UCA collations (MDEV-27265)
- Improve UCA collation performance for utf8mb3 and utf8mb4 (MDEV-27266)

Galera

- Implement a method to add IPs to allowlist for Galera Cluster node addresses that can make SST/IST requests (MDEV-27246)

Miscellaneous

- Change default of explicit_defaults_for_timestamp to ON (MDEV-28632)
- --ssl option set as default for mariadb CLI (MDEV-27105)
- Add RANDOM_BYTES function (MDEV-25704)
- The INET4 data type (MDEV-23287)
- Re-design the upper level of handling UPDATE and DELETE statements (MDEV-28883)
- Deprecate the DES_ENCRYPT/DECRYPT functions (MDEV-27104)

Variables

- For a list of all new variables, see System Variables Added in MariaDB 10.10.

InnoDB

- Removed innodb_version (MDEV-28554)
- Deprecated innodb_prefix_index_cluster_optimization

Spider

The following deprecated variables have been removed:

- spider_udf_ct_bulk_insert_interval
- spider_udf_ct_bulk_insert_rows
- spider_udf_ds_bulk_insert_rows
- spider_udf_ds_table_loop_mode
- spider_udf_ds_use_real_table
- spider_use_handle
- spider_udf_table_mon_mutex_count
- spider_use_handler

List of All MariaDB 10.10 Releases

Date	Release	Status	Release Notes	Changelog
13 Nov 2023	**MariaDB 10.10.7**	Stable (GA)	Release Notes	Changelog
14 Aug 2023	**MariaDB 10.10.6**	Stable (GA)	Release Notes	Changelog
7 Jun 2023	**MariaDB 10.10.5**	Stable (GA)	Release Notes	Changelog
10 May 2023	**MariaDB 10.10.4**	Stable (GA)	Release Notes	Changelog
6 Feb 2023	**MariaDB 10.10.3**	Stable (GA)	Release Notes	Changelog
17 Nov 2022	**MariaDB 10.10.2**	Stable (GA)	Release Notes	Changelog
22 Aug 2022	**MariaDB 10.10.1**	RC	Release Notes	Changelog
23 Jun 2022	**MariaDB 10.10.0**	Alpha	Release Notes	

7.0.2.2 Release Notes - MariaDB 10.10 Series

MariaDB 10.10.7 Release Notes
Status: Stable (GA) | Release Date: 13 Nov 2023

MariaDB 10.10.6 Release Notes
Status: Stable (GA) | Release Date: 14 Aug 2023

MariaDB 10.10.5 Release Notes
Status: Stable (GA) | Release Date: 7 Jun 2023

MariaDB 10.10.4 Release Notes
Status: Stable (GA) | Release Date: 10 May 2023

MariaDB 10.10.3 Release Notes
Status: Stable (GA) | Release Date: 6 Feb 2023

MariaDB 10.10.2 Release Notes
Status: Stable (GA) | Release Date: 17 Nov 2022

MariaDB 10.10.1 Release Notes
Status: RC | Release Date: 22 Aug 2022

MariaDB 10.10.0 Release Notes
Status: Alpha | Release Date: 23 Jun 2022

7.0.2.2.1 MariaDB 10.10.7 Release Notes

Download | Release Notes | Changelog | Overview of 10.10

Alternate download from mariadb.org

Release date: 13 Nov 2023

MariaDB 10.10 is a previous short-term maintenance stable series of MariaDB, maintained until November 2023. It is an evolution of MariaDB 10.9 with several entirely new features.

MariaDB 10.10.7 is a ***Stable (GA)*** release.

MariaDB 10.10.7 is the last release of the MariaDB 10.10 release series.

For an overview of MariaDB 10.10 see the What is MariaDB 10.10? page.

Thanks, and enjoy MariaDB!

Notable Items

- As per the MariaDB Maintenance Policy, this will be the final release of MariaDB 10.10

InnoDB

- DROP INDEX followed by CREATE INDEX may corrupt data (MDEV-32132)
- ROW_FORMAT=COMPRESSED table: InnoDB: 2048 bytes should have been read. Only 0 bytes read. (MDEV-31875)
- Server aborts during alter operation when table doesn't have foreign index (MDEV-32527)
- row_merge_fts_doc_tokenize() handles FTS plugin parser inconsistently (MDEV-32578)
- InnoDB: tried to purge non-delete-marked record of an index on a virtual column prefix (MDEV-30024)
- lock_row_lock_current_waits counter in information_schema.innodb_metrics may become negative (MDEV-30658)
- SET GLOBAL innodb_max_purge_lag_wait=… hangs if innodb_read_only=ON (MDEV-31813)
- Auto-increment no longer works for explicit FTS_DOC_ID (MDEV-32017)
- Assertion `pos < table->n_def' failed in dict_table_get_nth_col (MDEV-32337)
- innochecksum man pages seem to be inconsistent with the binary (10.2.25) (MDEV-20583)
- innodb_compression_algorithm=0 (none) increments Innodb_num_pages_page_compression_error (MDEV-30825)
- wrong table name in innodb's "row too big" errors (MDEV-32128)
- Optimize is_file_on_ssd() to speedup opening tablespaces on Windows (MDEV-32228)
- Race condition between page write completion and log checkpoint (MDEV-32511)
- After crash recovery, Checksum mismatch + Failing assertion: !i || prev_id + 1 == space_id, (MDEV-31851)
- Deadlock due to log_free_check(), involving trx_purge_truncate_rseg_history() and trx_undo_assign_low() (MDEV-32049)
- Write-ahead logging is broken for freed pages (MDEV-32552)
- X-lock on supremum for prepared transaction for RR (MDEV-30165)
- Assertion failures in log_sort_flush_list upon crash recovery (MDEV-32029)
- Race condition between page write completion and log checkpoint (MDEV-32511)
- Assertion `purge_sys.tail.trx_no <= purge_sys.rseg->last_trx_no()' (MDEV-30100)
- Assertion `index->is_btree() || index->is_ibuf()' failed in btr_search_guess_on_hash (MDEV-30802)
- InnoDB hang in buf_flush_wait_LRU_batch_end() (MDEV-32134)
- InnoDB may hang when running out of buffer pool (MDEV-32588)
- Corrupt index(es) on busy table when using FOREIGN KEY (MDEV-30531)
- InnoDB Recovery doesn't display encryption message when no encryption configuration passed (MDEV-31098)
- Wrong information about innodb_checksum_algorithm in the information_schema.SYSTEM_VARIABLES (MDEV-31473)
- InnoDB may fail to recover after being killed in fil_delete_tablespace() (MDEV-31826)
- Create separate tpool thread for async aio (MDEV-31095)
- UNDO logs still growing for write-intensive workloads (MDEV-32050)
  - Increase the default for innodb_purge_batch_size from 300 to 1000 (MDEV-32050)
  - Deprecate innodb_purge_rseg_truncate_frequency.
  - The motivation for introducing this in MySQL seems to have been to avoid stalls due to freeing undo log pages or truncating undo log tablespaces. In MariaDB, innodb_undo_log_truncate=ON should be a much lighter operation because it will not involve any log checkpoint. (MDEV-32050)
- Slow full index scan in 10.6 vs 10.5 for the (slow) I/O-bound case (MDEV-30986)
- LOAD DATA into InnoDB w/partitions: huge performance loss, affected 10.6+ (MDEV-31835)
- Disable read-ahead for temporary tablespace (MDEV-32145)

Optimizer

- Crash when HAVING in a correlated subquery references columns in the outer query (MDEV-29731)

- Server crashes at TABLE::add_tmp_key (MDEV-32320)
- Server crashes inside filesort at my_decimal::to_binary (MDEV-32324)
- Assertion `bitmap_is_set(&m_part_info->read_partitions, m_part_spec.start_part)' failed in ha_partition::handle_ordered_index_scan (MDEV-24283)
- Crash when searching for the best split of derived table (MDEV-32064)
- Test case from opt_tvc.test fails with statement memory protection (MDEV-32225)
- Significant slowdown for query with many outer joins (MDEV-32351)
- test_if_skip_sort_order() should catch the join types JT_EQ_REF, JT_CONST and JT_SYSTEM and skip sort order for these (MDEV-32475)
- jointable materialization subquery optimization ignoring errors, then failing ASSERT. (MDEV-31983)
- Server crashes in JOIN::cleanup after erroneous query with view (MDEV-32164)
- Prepared statement return wrong result (missing row) (MDEV-9938)
- Assertion `range->rows >= s->found_records' failed in best_access_path (MDEV-32682)

Replication

- rpl.rpl_parallel_temptable failure due to incorrect commit optimization of temptables (MDEV-10356)
- Lock wait timeout with INSERT-SELECT, autoinc, and statement-based replication (MDEV-31482)
- strings/ctype-ucs2.c:2336: my_vsnprintf_utf32: Assertion `(n % 4) == 0' failed in my_vsnprintf_utf32 on INSERT (MDEV-32249)
- Assertion fails in MDL_context::acquire_lock upon parallel replication of CREATE SEQUENCE (MDEV-31792)
- SHOW SLAVE STATUS Last_SQL_Errno Race Condition on Errored Slave Restart (MDEV-31177)
- seconds_behind_master is inaccurate for Delayed replication (MDEV-32265)
- detailize the semisync replication magic number error (MDEV-32365)
- Parallel replication deadlock victim preference code errorneously removed (MDEV-31655)
- Parallel replication lags because innobase_kill_query() may fail to interrupt a lock wait (MDEV-32096)

Galera

- Assertion `state() == s_executing || state() == s_prepared || state() == s_committing || state() == s_must_abort || state() == s_replaying' failed. (MDEV-24912)
- Assertion `state() == s_executing || state() == s_preparing || state() == s_prepared || state() == s_must_abort || state() == s_aborting || state() == s_cert_failed || state() == s_must_replay' failed (MDEV-31285)
- wsrep_sst_mariabackup not working on FreeBSD (MDEV-31467)
- Galera library 26.4.16 fails with every server version (MDEV-32024)
- Galera node remains paused after interleaving FTWRLs (MDEV-32282)
- Failed to insert streaming client (MDEV-32051)
- When set at runtime, wsrep_sst_method accepts any value (MDEV-31470)
- galera needs packaging script changes to successfully build (MDEV-32642)
- replication breaks when using optimistic replication and replica is a galera node (MDEV-31833)
- McAfee database vulnerability scan caused MariaDB crash with signal 6 (system abort) (MDEV-27004)
- Assertion `mode_ == m_local || transaction_.is_streaming()' failed in int wsrep::client_state::bf_abort(wsrep::seqno) (MDEV-30217)

Data Definition

- MariaDB crash on calling function (MDEV-23902)
- ASAN errors in grn_obj_unlink / ha_mroonga::clear_indexes upon index operations (MDEV-31970)
- vcol circular references lead to stack overflow (MDEV-31112)
- OPTIMIZE TABLE crash (MDEV-28122)
- Server crashes in Alter_info::add_stat_drop_index upon CREATE TABLE (MDEV-32449)

Scripts and Clients

- mariadb-binlog -T/--table (mysqlbinlog) option (MDEV-25369)
- mariadb-admin (mysqladmin) wrong error with simple_password_check (MDEV-22418)
- mariadb-install-db shows warning on missing directory $pamtooldir/auth_pam_tool_dir (MDEV-32142)
- main.mysql_client_test, main.mysql_client_test_comp failed on ASAN build with error: 5888, status: 23, errno: 2 (MDEV-19369)
- mariadb-install-db (mysql_install_db) doesn't properly grant proxy privileges to all default root user accounts (MDEV-21194)

Tests

- main.events_stress or events.events_stress fails with view-protocol (MDEV-31455)

- main.delete_use_source fails (hangs) with view-protocol (MDEV-31457)
- main.sum_distinct-big and main.merge-big fail with timeout with view-protocol (MDEV-31465)
- main.secure_file_priv_win fails with 2nd execution PS protocol (MDEV-32023)
- Windows : mtr output on is messed up with large MTR_PARALLEL (MDEV-32387)
- main.mysql_client_test_comp failed in buildbot, error on exec (MDEV-16641)
- main.order_by_pack_big fails with view-protocol (MDEV-31460)
- mysql_install_db_win.test fails on second execution (MDEV-32232)

MariaBackup

- MariaBackup full backup failed with InnoDB: Failing assertion: success in storage/innobase/fil/fil0fil.cc line 657 (MDEV-18200)
- mbstream breaks page compression on XFS (MDEV-25734)

Character Sets, Data Types, Collations

- Prefix keys for CHAR work differently for MyISAM vs InnoDB (MDEV-30048)
- Inconsistent results of DISTINCT with NOPAD (MDEV-30050)
- Assertion `(length % 4) == 0' failed in my_lengthsp_utf32 on INSERT (MDEV-28835)
- Compressed varchar values lost on joins when sorting on columns from joined table(s) (MDEV-31724)
- UBSAN shift exponent X is too large for 64-bit type 'long long int' in sql/field.cc (MDEV-32226)
- Wrong bit encoding using COALESCE (MDEV-32244)

Spider

- Spider UBSAN runtime error: applying non-zero offset x to null pointer in st_spider_param_string_parse::restore_delims (MDEV-31117)
- Segfault when setting spider_delete_all_rows to 0 and delete all rows of a spider table, ASAN heap-use-after-free in spider_db_delete_all_rows (MDEV-31996)
- ASAN errors in spider_fields::free_conn_holder or spider_create_group_by_handler (MDEV-28998)
- ASAN: heap-buffer-overflow & stack-buffer-overflow in `spider_db_mbase_row::append_to_str` | SIGSEGV's in `memmove_avx_unaligned_erms` from memcpy in `Binary_string::q_append`, in `Static_binary_string::q_append` and `my_strntoull10rnd_8bit` | Unknown error 12801 (MDEV-29502)

General

- binlog_do_db option breaks importing sql dumps (MDEV-29989)
- Crashes in MDL_key::mdl_key_init with lower-case-table-names=2 (MDEV-32025)
- getting error 'Illegal parameter data types row and bigint for operation '+' ' when using ITERATE in a FOR..DO (MDEV-32275)
- Assertion `arena_for_set_stmt== 0' failed in LEX::set_arena_for_set_stmt upon SET STATEMENT (MDEV-17711)
- main.mysqlcheck fails on ARM with ASAN use-after-poison in my_mb_wc_filename (MDEV-26494)
- main.delayed fails with wrong error code or timeout when executed after main.deadlock_ftwrl (MDEV-27523)
- Assertion failed: !pfs->m_idle || (state == PSI_SOCKET_STATE_ACTIVE) (MDEV-28561)
- MyISAM wrong server status flags (MDEV-28820)
- Server crashes in check_sequence_fields upon CREATE TABLE .. SEQUENCE=1 AS SELECT .. (MDEV-29771)
- slow log Rows_examined out of range (MDEV-30820)
- " `rpm --setugids` " breaks PAM authentication (MDEV-30904)
- incorrect examined rows in case of stored function usage (MDEV-31742)
- Compilation failing on MacOS (unknown warning option -Wno-unused-but-set-variable) (MDEV-31890)
- Server crash upon inserting into Mroonga table with compressed column (MDEV-31966)
- hash unique corrupts index on virtual blobs (MDEV-32012)
- insert into an empty table fails with hash unique (MDEV-32015)
- Valgrind/MSAN warnings in dynamic_column_update_move_left (MDEV-32140)
- Memory leak showed in MDEV-6146 test suite (MDEV-32223)
- Test from subselect.test fails with statement memory protection (MDEV-32245)
- Memory leak when executing PS for query with IN subquery (MDEV-32369)
- Allow the setting of Auto_increment on FK referenced columns (MDEV-32018)
- mariadb-upgrade fails with sql_safe_updates = on (MDEV-29914)
- Assertion `!(thd->server_status & (1U | 8192U))' failed in MDL_context::release_transactional_locks (MDEV-32541)
- Information schema leaks table names and structure to unauthorized users (MDEV-32500)
- Missing CHACHA20-POLY1305 support in WolfSSL (MDEV-31653)
- incorrect error about cyclic reference about JSON type virtual column (MDEV-32586)
- Disable TLS v1.0 and 1.1 for MariaDB (MDEV-31369)
- Better indication of refusing to start because of ProtectHome (MDEV-25177)

- Database upgrade fails: slow_log table (MDEV-27757)
- myrocks_hotbackup.1 and test suite files installed when engine is disabled (MDEV-29993)
- client_ed25519.dll isn't inluded for HeidiSQL. (MDEV-31315)
- Assertion `!m_null_value' failed in int FixedBinTypeBundle<FbtImpl>::cmp_item_fbt::compare or in cmp_item_inet6::compare (MDEV-27207)
- type_test.type_test_double fails with 'NUMERIC_SCALE NULL' (MDEV-22243)
- LeakSanitizer errors in get_quick_select or Assertion `status_var.local_memory_used == 0 || !debug_assert_on_not_freed_memory' failed (MDEV-32476)
- Update signal hander user info more compassion and correct url (MDEV-32535)
- maria-install-db fails on MacOS (MDEV-31871)
- MSAN / Valgrind errors in Item_func_like::get_mm_leaf upon query from partitioned table (MDEV-32388)
- MSAN / Valgrind errors in Item_func_like::get_mm_leaf with temporal field (MDEV-32531)
- ASAN errors in base_list_iterator::next / setup_table_map upon 2nd execution of PS (MDEV-32656)
- safe_mutex: Found wrong usage of mutex 'LOCK_thd_data' and 'wait_mutex' (MDEV-32728)

Docker Official Images

- Invert single and double quotes for sql command definitions in healthcheck.sh due to failure under sql_mode=ANSI_QUOTES - contribution by Dominik Häckel
- healthcheck.sh --no-defaults behaviour was corrected - reported by Dominik Häckel
- Added /docker-entrypoint-init.d for tar{,compression} from mariadb-backup - instructions
- Refactor `docker_mariadb_init` in the entrypoint for extending the MariaDB image
- CIS failure due to world-writable directory /var/run/mysqld, added sticky bit - reported by @ollie1
- Add PROXY privileges for root@MARIADB_ROOT_HOST - reported by Matthieu Gusmini
- healthcheck.sh added --galera_online test, to match what the mariadb-operator does.

Variables

- Added the note_verbosity system variable to manage notes when an index cannot be used.

Security

- Fixes for the following security vulnerabilities:
  - CVE-2023-22084

Changelog

For a complete list of changes made in MariaDB 10.10.7, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.10.7, see the MariaDB Foundation release announcement.

7.0.2.2.2 MariaDB 10.10.6 Release Notes

Download 10.10.6 | Release Notes | Changelog | Overview of 10.10

Release date: 14 Aug 2023

MariaDB 10.10 is a previous short-term maintenance stable series of MariaDB, maintained until November 2023. It is an evolution of MariaDB 10.9 with several entirely new features.

MariaDB 10.10.6 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.10 see the What is MariaDB 10.10? page.

Thanks, and enjoy MariaDB!

Notable Items

Upgrading from MySQL

- MariaDB now detects and converts previously incompatible MySQL partition schemes (MDEV-29253)

General

- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.10 for Ubuntu 18.04 LTS "Bionic" and Ubuntu 22.10 "Kinetic"
- mariadb-dump --force doesn't ignore error as it should (MDEV-31092)
- 280 Bytes lost in mysys/array.c, mysys/hash.c, sql/sp.cc, sql/sp.cc, sql/item_create.cc, sql/item_create.cc, sql/sql_yacc.yy:10748 when using oracle sql_mode (MDEV-26186)
- SQL/PL package body does not appear in `I_S.ROUTINES.ROUTINE_DEFINITION` (MDEV-30662)
- Unexpected result when combining DISTINCT, subselect and LIMIT (MDEV-28285)
- ROW variables do not get assigned from subselects (MDEV-31250)
- Crash after setting global session_track_system_variables to an invalid value (MDEV-25237)
- ODKU of non-versioning column inserts history row (MDEV-23100)
- UPDATE not working properly on transaction precise system versioned table (MDEV-25644)
- Assertion `` `const_item_cache == true' `` failed in `Item_func::fix_fields` (MDEV-31319)
- ANALYZE doesn't work with pushed derived tables (MDEV-29284)
- `get_partition_set` is never executed in `ha_partition::multi_range_key_create_key` due to bitwise & with 0 constant (MDEV-24712)
- Client can crash the server with a `mysql_list_fields("view")` call (MDEV-30159)
- `I_S.parameters` not immediatly changed updated after procedure change (MDEV-31064)
- bzero wipes more bytes than necessary in `set_global_from_ddl_log_entry` (MDEV-31521)
- Assertion `` `0' `` failed in `Type_handler_row::field_type` upon `TO_CHAR` with wrong argument (MDEV-29152)
- mysql_upgrade fails due to `old_mode=""` , with "Cannot load from mysql.proc. The table is probably corrupted" (MDEV-28915)

Character Sets, Data Types

- UUIDs version >= 6 are now stored without byte-swapping, UUIDs with version >=8 and variant=0 are now considered invalid, old tables are supported, old (always byte swapped) and new (swapped for version < 6) UUIDs can be compared and converted transparently (MDEV-29959)
- UBSAN: null pointer passed as argument 1, which is declared to never be null in `my_strnncoll_binary` on `SELECT ... COUNT` or `GROUP_CONCAT` (MDEV-28384)
- Possibly wrong result or Assertion `0' failed in `Item_func_round::native_op` (MDEV-23838)
- Assertion `` `(length % 4) == 0' `` failed in `my_lengthsp_utf32` on `SELECT` (MDEV-29019)
- UBSAN: negation of -X cannot be represented in type `'long long int'` ; cast to an unsigned type to negate this value to itself in `Item_func_mul::int_op` and `Item_func_round::int_op` (MDEV-30932)
- Assorted assertion failures in json_find_path with certain collations (MDEV-23187)

InnoDB

- SIGSEGV in `log_sort_flush_list()` in InnoDB crash recovery (MDEV-31354)
- InnoDB tables are being flagged as corrupted on an I/O bound server (MDEV-31767)
- Duplicate entry allowed into a UNIQUE column (MDEV-31120)
- Server Status Innodb_row_lock_time% is reported in seconds (MDEV-29311)
- innochecksum dies with Floating point exception (MDEV-31641)

- Add InnoDB engine information to the slow query log (MDEV-31558)
- Deadlock with 3 concurrent DELETEs by unique key (MDEV-10962)
- innodb protection against dual processes accessing data insufficient (MDEV-31568)
- ER_DUP_KEY in `mysql.innodb_table_stats` upon RENAME on sequence (MDEV-31607)
- Assertion `` `!strcmp(index->table->name.m_name, "SYS_FOREIGN") || !strcmp(index->table->name.m_name, "SYS_FOREIGN_COLS")' `` failed in `btr_node_ptr_max_size` (MDEV-19216)
- InnoDB: Failing assertion: `page_type == i_s_page_type[page_type].type_value` (MDEV-31386)
- `btr_estimate_n_rows_in_range()` accesses unfixed, unlatched page (MDEV-30648)
- `MODIFY COLUMN` can break FK constraints, and lead to unrestorable dumps (MDEV-31086)
- Recovery or backup failure after innodb_undo_log_truncate=ON (MDEV-31487)
- Assertion `'n & PENDING'` failed in `fil_space_t::set_needs_flush()` (MDEV-31442)
- `fil_node_open_file()` releases `fil_system.mutex` allowing other thread to open its file node (MDEV-31256)
- ASAN errors in `dict_v_col_t::detach` upon adding key to virtual column (MDEV-31416)
- Purge trying to access freed secondary index page (MDEV-31264)
- Freed data pages are not always being scrubbed (MDEV-31253)
- InnoDB recovery hangs after reporting corruption (MDEV-31353)
- `!cursor->index->is_committed()` in row0ins.cc after update to 10.4.13 from 10.3.21 (MDEV-22739)
- innodb_undo_log_truncate=ON fails to wait for purge of enough transaction history (MDEV-31355)
- `SET GLOBAL innodb_undo_log_truncate=ON` does not free space when no undo logs exist (MDEV-31382)
- innodb_read_ahead_threshold (linear read-ahead) does not work (MDEV-29967)
- InnoDB recovery and `mariadb-backup --prepare` fail to report detailed progress (MDEV-29911)
- `fil_ibd_create()` may hijack the file handle of an old file (MDEV-31347)
- innodb_undo_log_truncate=ON recovery results in a corrupted undo log (MDEV-31373)
- Server freeze due to innodb_change_buffering and innodb_file_per_table=0 (MDEV-31088)
- Change buffer entries are left behind when freeing a page, causing secondary index corruption when the page is later reused (MDEV-31385)
- Foreign Key Constraint actions don't affect Virtual Column (MDEV-18114)

Aria

- Various crashes upon INSERT/UPDATE after changing Aria settings (MDEV-28054)
- Various crashes/asserts/corruptions when Aria encryption is enabled/used, but the encryption plugin is not loaded (MDEV-26258)

Spider

- SIGSEGV in `spider_db_open_item_field` and SIGSEGV in `spider_db_print_item_type`, on SELECT (MDEV-29447)
- Spider variables that double as table params overriding mechanism is buggy (MDEV-31524)

Optimizer

- ANALYZE FORMAT=JSON now includes InnoDB engine statistics for each table (MDEV-31577)
- Assertion `` `last_key_entry >= end_pos' `` failed in virtual bool `JOIN_CACHE_HASHED::put_record()` (MDEV-31348)
- Problem with open ranges on prefix blobs keys (MDEV-31800)
- Equal on two RANK window functions create wrong result (MDEV-20010)
- Recursive CTE execution is interrupted without errors or warnings (MDEV-31214)
- Assertion `` `s->table->opt_range_condition_rows <= s->found_records' `` failed in `apply_selectivity_for_table` (MDEV-31449)
- Inconsistency between MRR and SQL layer costs can cause poor query plan (MDEV-31479)
- `MAX_SEL_ARG` memory exhaustion is not visible in the optimizer trace (MDEV-30964)
- SHOW TABLES not working properly with `lower_case_table_names=2` (MDEV-30765)
- Segfault on select query using index for group-by and filesort (MDEV-30143)
- Server crash in `store_length`, assertion failure in `Type_handler_string_result::sort_length` (MDEV-31743)

Replication

- Parallel Slave SQL Thread Can Update Seconds_Behind_Master with Active Workers (MDEV-30619)
- ALTER SEQUENCE ends up in optimistic parallel slave binlog out-of-order (MDEV-31503)
- STOP SLAVE takes very long time on a busy system (MDEV-13915)
- On slave XA COMMIT/XA ROLLBACK fail to return an error in read-only mode (MDEV-30978)

- Calling a function from a different database in a slave side trigger crashes (MDEV-29894)
- `rpl.rpl_manual_change_index_file` occasionally fails in BB with Result length mismatch (MDEV-30214)

Galera

- Node never returns from Donor/Desynced to Synced when `wsrep_mode = BF_ABORT_MARIABACKUP` (MDEV-31737)
- Node has been dropped from the cluster on Startup / Shutdown with async replica (MDEV-31413)
- KILL QUERY maintains nodes data consistency but breaks GTID sequence (MDEV-31075)
- Assertion failure `!lock.was_chosen_as_deadlock_victim in trx0trx.h:1065` (MDEV-30963)
- MariaDB stuck on starting commit state (waiting on commit order critical section) (MDEV-29293)
- Assertion `state() == s_aborting || state() == s_must_replay` failed in int wsrep::transaction::after_rollback() (MDEV-30013)
- Assertion `!wsrep_has_changes(thd) || (thd->lex->sql_command == SQLCOM_CREATE_TABLE && !thd->is_current_stmt_binlog_format_row()) || thd->wsrep_cs().transaction().state() == wsrep::transaction::s_aborted` failed (MDEV-30388)
- Server crashes when wsrep_sst_donor and wsrep_cluster_address set to NULL (MDEV-28433)
- Create temporary sequence can cause inconsistency (MDEV-31335)
- Galera 4 unable to query cluster state if not primary component (MDEV-21479)

Changelog

For a complete list of changes made in MariaDB 10.10.6, with links to detailed information on each push, see the changelog .

Contributors

For a full list of contributors to MariaDB 10.10.6, see the MariaDB Foundation release announcement .

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.2.2.3 MariaDB 10.10.5 Release Notes

The most recent release of MariaDB 10.10 is:
MariaDB 10.10.7 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.10.5 | Release Notes | Changelog | Overview of 10.10

Release date: 7 Jun 2023

MariaDB 10.10 is a current short-term stable series of MariaDB, maintained until November 2023. It is an evolution of MariaDB 10.9 with several entirely new features.

MariaDB 10.10.5 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.10 see the What is MariaDB 10.10? page.

Thanks, and enjoy MariaDB!

Notable Items

InnoDB

- Server crashes in st_join_table::choose_best_splitting (MDEV-31403)
- Crash with condition pushable into derived and containing outer reference (MDEV-31240)
- InnoDB does not free UNDO after the fix of MDEV-30671 (MDEV-31234)
- InnoDB hang fixes (MDEV-31158, MDEV-31343, MDEV-31350)
- Innodb_buffer_pool_read_requests is not updated correctly (MDEV-31309)
- InnoDB monitor trx_rseg_history_len was accidentally disabled by default (MDEV-31308)
- Revert "MDEV-30473: Do not allow GET_LOCK() / RELEASE_LOCK() in cluster"

Optimizer

- Crash with condition pushable into derived and containing outer reference (MDEV-31403 MDEV-31240)
- Crash with EXPLAIN EXTENDED for multi-table update of system table (MDEV-31224)

Changelog

For a complete list of changes made in MariaDB 10.10.5, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.10.5, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.2.2.4 MariaDB 10.10.4 Release Notes

The most recent release of MariaDB 10.10 is:
MariaDB 10.10.7 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.10.4 | Release Notes | Changelog | Overview of 10.10

Release date: 10 May 2023

MariaDB 10.10 is a current short-term maintenance stable series of MariaDB, maintained until November 2023. It is an evolution of MariaDB 10.9 with several entirely new features.

MariaDB 10.10.4 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.10 see the What is MariaDB 10.10? page.

Thanks, and enjoy MariaDB!

Notable Items

InnoDB

- Crash on ROLLBACK in a ROW_FORMAT=COMPRESSED table (MDEV-30882)
- UNIQUE USING HASH accepts duplicate entries for tricky collations (MDEV-30034)

- rec_get_offsets() is not optimal (MDEV-30567)
- Performance regression in fil_space_t::try_to_close() introduced in MDEV-23855 (MDEV-30775)
- InnoDB recovery hangs when buffer pool ran out of memory (MDEV-30551)
- InnoDB undo log truncation fails to wait for purge of history (MDEV-30671)
- MariaDB crash due to DB_FAIL reported for a corrupted page (MDEV-30397)
- Deadlock between INSERT and InnoDB non-persistent statistics update (MDEV-30638)
- InnoDB hang on B-tree split or merge (MDEV-29835)
- Performance regression in locking reads from secondary indexes (MDEV-30357)
- Improve adaptive flushing (MDEV-26055)
- Make page flushing even faster (MDEV-26827)
- Purge misses a chance to free not-yet-reused undo pages (MDEV-29593)
- InnoDB temporary tablespace: reclaiming of free space does not work (MDEV-26782)
- Deadlock between CHECK TABLE and bulk insert (MDEV-30798)
- UPPER() returns an empty string for U+0251 in uca1400 collations for utf8 (MDEV-30661)
- Fix miscount of doublewrites by Innodb_data_written (MDEV-31124)

Backup

- mariadb-backup doesn't utilise innodb-undo-log-directory (if specified as a relative path) during copy-back operation (MDEV-28187)
- mariabackup issues error messages during InnoDB tablespaces export on partial backup preparing (MDEV-29050)
- mariadb-backup does not copy Aria logs if aria_log_dir_path is used (MDEV-30968)
- Race condition between buffer pool flush and log file deletion in mariadb-backup --prepare (MDEV-30860)

Replication

- Fixed a deadlock on parallel slave involving full image Write event on the sequence engine (MDEV-29621)
- Fixed an attempted out-of-order binlogging error on slave involving ALTER on the sequence engine (MDEV-31077)
- Corrected non-versioned master to versioned slave replication on no-unique attribute table (MDEV-30430)
- Mended encrypted binlog master to error out to gtid-mode slave when master could not decrypt a binlog file (MDEV-28798)
- Refined optimistic parallel slave to error-exit without any hang (MDEV-30780)
- Ensured SHOW-SLAVE-STATUS is processed on the parallel slave having a necessary mutex always intialized (MDEV-30620)
- Fixed the slave applier to report a correct error when gtid_slave_pos insert fails for some (engine) reasons (MDEV-31038)
- Made parallel slave reports in performance schema consistent with that of show-slave-status (MDEV-26071)

Optimizer

- Split Materialized optimization is improved to re-fill the materialized table only if necessary. The fewer number of table refills is taken into account when choosing query plan, too (MDEV-26301).
- New optimizer_switch option, hash_join_cardinality, is added. It is off by default. When set to ON, the optimizer will produce tighter bounds for hash join output cardinality. (MDEV-30812)
- Queries using `SELECT DISTINCT some_expression(aggregate_function())` could produce wrong query result. (MDEV-20057)
- ANALYZE FORMAT=JSON now prints more information about Block Nested Loop joins: `block-nl-join` element now has `r_loops`, `r_effective_rows` and `r_other_time_ms` fields (MDEV-30806, MDEV-30830, MDEV-30972).
- A GROUP BY query with `MIN(primary_key)` in select list and `primary_key<>const` in the WHERE could produce wrong result when executed with "Using index for group-by" strategy (MDEV-30605)
- EXPLAIN could erroneously report that Rowid Filter optimization is used for partitioned tables. Partitioned tables do not support it. (MDEV-30596)
- A bug in selectivity computations for SINGLE/DOUBLE_PREC_HB histograms could cause wrong estimates to be produced. This could cause the optimizer to pick sub-optimal query plans (MDEV-31067).

Docker Official Image

- Add replication setup to containers contributed by Md Sahil (MDEV-29762)

General

- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.10 for Fedora 36.
- In this release repositories for Fedora 38 and Ubuntu 23.04 Lunar have been added.

Security

- Fixes for the following security vulnerabilities:
  - CVE-2022-47015

Changelog

For a complete list of changes made in MariaDB 10.10.4, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.10.4, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.2.2.5 MariaDB 10.10.3 Release Notes

Download | Release Notes | Changelog | Overview of 10.10

Release date: 6 Feb 2023

MariaDB 10.10 is a current short-term maintenance stable series of MariaDB, maintained until November 2023. It is an evolution of MariaDB 10.9 with several entirely new features.

MariaDB 10.10.3 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.10 see the What is MariaDB 10.10? page.

Thanks, and enjoy MariaDB!

Notable Items

- As mentioned in the 10.10.2 release notes, our Yum/DNF/Zypper repositories for Red Hat Enterprise Linux, CentOS, Fedora, openSUSE, and SUSE are changing **with this release** to being signed with a new GPG key with SHA2 digest algorithms instead of SHA1. See this blog post and the GPG page for more details.

InnoDB

- Full-text index corruption with system versioning (MDEV-25004)
- innodb_undo_log_truncate=ON recovery and backup fixes (MDEV-29999, MDEV-30179, MDEV-30438)
- Upgrade after a crash is not supported (MDEV-24412)
- Remove InnoDB buffer pool load throttling (MDEV-25417)
- InnoDB shutdown hangs when the change buffer is corrupted (MDEV-30009)
- `innodb_fast_shutdown=0` fails to report change buffer merge progress (MDEV-29984)
- `mariadb-backup --backup --incremental --throttle=...` hangs (MDEV-29896)
- Crash after recovery, with InnoDB: Tried to read (MDEV-30132)
- Trying to write ... bytes at ... outside the bounds (MDEV-30069)
- TRUNCATE breaks FOREIGN KEY locking (MDEV-29504, MDEV-29849)
- `INFORMATION_SCHEMA.INNODB_TABLESPACES_ENCRYPTION.NAME` is NULL for undo tablespaces (MDEV-30119)
- Fixed hangs and error handling in B-tree operations (MDEV-29603, MDEV-30400)
- InnoDB bulk insert fixes (MDEV-30047, MDEV-30321)

Galera

- Fixes for cluster wide write conflict resolving (MDEV-29684)

Replication

- Parallel slave applying in binlog order is corrected for admin class of commands including ANALYZE (MDEV-30323)
- Seconds_Behind_Master is now shown now more precisely at the slave applier start, including in the delayed mode (MDEV-29639)
- `mysqlbinlog --verbose` is made to show the type of compressed columns (MDEV-25277)
- Deadlock is resolved on replica involving `BACKUP STAGE BLOCK_COMMIT` and a committing user XA (MDEV-30423)

JSON

- JSON_PRETTY added as an alias for JSON_DETAILED (MDEV-19160)

General

- Infinite sequence of recursive calls when processing embedded CTE (MDEV-30248)
- Crash with a query containing nested WINDOW clauses (MDEV-30052)
- Major performance regression with 10.6.11 (MDEV-29988)
- Json Range only affects first row of the result set (MDEV-30304)
- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.10 for Fedora 35.
- In this release repositories for Fedora 37 and Ubuntu 22.10 Kinetic have been added.

Changelog

For a complete list of changes made in MariaDB 10.10.3, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.10.3, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.2.2.6 MariaDB 10.10.2 Release Notes

The most recent release of MariaDB 10.10 is:
MariaDB 10.10.7 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.10.2 | Release Notes | Changelog | Overview of 10.10

Release date: 17 Nov 2022

MariaDB 10.10 is a current short-term series of MariaDB, maintained until November 2023. It is an evolution of MariaDB 10.9 with several entirely new features.

MariaDB 10.10.2 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.10 see the What is MariaDB 10.10? page.

Thanks, and enjoy MariaDB!

Notable Items

SSL

- The server no longer tolerates incorrectly configured SSL (MDEV-29811). If you have enabled SSL in `my.cnf` but have not configured it properly (for example, a certificate file is missing), MariaDB used to silently disable SSL, leaving you under impression that everything was fine and connections were secure. Since this release, MariaDB will fail to start if SSL is enabled, but cannot be switched on.

Backup

- Assertion on info.page_size failed in xb_delta_open_matching_space (MDEV-18589)
- Mariabackup locks database for minutes (MDEV-28772)

InnoDB

- Adaptive hash index MDEV-27700, MDEV-29384
- MVCC and locking (MDEV-29666, MDEV-27927, MDEV-28709, MDEV-29635)
- Virtual columns (MDEV-29299, MDEV-29753)
- InnoDB crash recovery fixes (MDEV-29559)
- Race condition between KILL and transaction commit (MDEV-29368)
- Implement CHECK TABLE…EXTENDED for InnoDB (MDEV-24402)
- InnoDB persistent statistics fail to update after bulk insert (MDEV-28327)
- InnoDB bulk insert bug fixes (MDEV-29570, MDEV-29761)
- InnoDB hangs on multiple concurrent requests of a cold ROW_FORMAT=COMPRESSED page (MDEV-27983)

Galera

- Galera updated to 26.4.13
- Galera server crashes after 10.3 > 10.4 upgrade (MDEV-29375)
- wsrep_incoming_addresses status variable prints 0 as port number if the port is not mentioned in wsrep_node_incoming_address system variable (MDEV-28868)

Replication

- Minor correction in unsafe warning message (MDEV-28827)
- False replication error-stop of REVOKE PRIVILEGES from a non-existing user on primary (MDEV-28530) in combination with a filtering replica is corrected
- SET DEFAULT ROLE replication is mended on a replica that filters system tables (MDEV-28294)
- XA COMMIT is not binlogged when the XA transaction has not updated any transaction engine (MDEV-25616)
- Concurrent CREATE TRIGGER statements made to binlog without any mixup (MDEV-25606)

Repositories

- Beginning with the next release (Q1 2023), our Yum, DNF, and Zypper repositories for Red Hat, Fedora, and SUSE will be migrated to being signed with a new GPG key. The key we are migrating to is the same one we already use for our Debian and Ubuntu Repositories.
  - The short Key ID is: `0xC74CD1D8`
  - The long Key ID is: `0xF1656F24C74CD1D8`
  - The full fingerprint of the key is: `177F 4010 FE56 CA33 3630 0305 F165 6F24 C74C D1D8`
  - The key can be imported now in preparation for this change using the following command:

```
sudo rpm --import https://supplychain.mariadb.com/MariaDB-Server-GPG-KEY
```

Docker Official Image

The following changes have been made to the `docker.io/library/mariadb` container image.

- The number of gpg packages packages has been removed, leaving enough to `apt-get update`, but `dirmngr` that would fetch keys has been removed. (inspired by issue #469)
- The environment variable `LANG=C.UTF-8` has been added for those that exec into containers and copy paste UTF8 characters (fixes issue #468).
- Adds OCI labels to image (fixes issue 436 and users need for version)
- MariaDB config: skip-host-cache and skip-name-resolve moved to `/etc/mysql/mariadb.conf.d/05-skipcache.cnf`

Changelog

For a complete list of changes made in MariaDB 10.10.2, with links to detailed information on each push, see the changelog.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.2.2.7 MariaDB 10.10.1 Release Notes

The most recent release of MariaDB 10.10 is:
MariaDB 10.10.7 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.10.1 | Release Notes | Changelog | Overview of 10.10

Release date: 22 Aug 2022

Do not use non-stable (non-GA) releases in production!

MariaDB 10.10 is a current short-term support development series of MariaDB. It is an evolution of MariaDB 10.9 with several entirely new features.

MariaDB 10.10.1 is a ***Release Candidate (RC)*** release.

For an overview of MariaDB 10.10 see the What is MariaDB 10.10? page.

Thanks, and enjoy MariaDB!

Notable Items

InnoDB

- InnoDB corruption due to lack of file locking (MDEV-28495)
- FULLTEXT search with apostrophe, and mandatory words (MDEV-20797)
- ALTER TABLE IMPORT TABLESPACE corrupts an encrypted table (MDEV-28779)
- ALTER TABLE wrong-result fix (MDEV-26294)
- Crash recovery fixes (MDEV-28668, MDEV-28731)
- DDL crash recovery fixes (MDEV-28752, MDEV-28802, MDEV-28864, MDEV-28870, MDEV-28923, MDEV-28977)
- Avoid crashes on corrupted data (MDEV-13542, MDEV-18519, MDEV-21098, MDEV-22388, MDEV-28457, MDEV-28950)
- Bulk load bug fixes (MDEV-28242, MDEV-28679)
- Performance fixes (MDEV-28708, MDEV-28766)

- Remove innodb_version (MDEV-28554)
- Deprecate and ignore the parameter innodb_prefix_index_cluster_optimization (MDEV-28540)
- Some InnoDB counters are duplicating generic SHOW STATUS (MDEV-28539)
- Useless output in SHOW ENGINE INNODB STATUS (MDEV-28542)

Replication

- ER_SLAVE_INCIDENT error is specified now on slave to be seen with SHOW-SLAVE-STATUS (MDEV-21087)
- INCIDENT_EVENT is no longer binlogged when a being logged transaction can be safely rolledback (MDEV-21443)
- sequences related row-format events are made to correspond to binlog_row_image (MDEV-28487)
- Possible reason of FLUSH BINARY LOGS hang is eliminated (MDEV-28948)
- Fix out-of-order gtid error in the circular semisync setup (MDEV-28609)
- Added global.slave_max_statement_time system variable for SQL thread to limit maximum execution time per query replicated (MDEV-27161)
- Deprecate MASTER_USE_GTID=Current_Pos to favor new MASTER_DEMOTE_TO_SLAVE option (MDEV-20122)
- MASTER_USE_GTID defaults of CHANGE MASTER TO and RESET SLAVE are changed to be compatible with GTID-based replication (MDEV-19801)

Galera

- Possible to write/update with read_only=ON and not a SUPER privilege (MDEV-28546)
- Node crashes with Transport endpoint is not connected mysqld got signal 6 (MDEV-25068)
- Galera4 not able to report proper wsrep_incoming_addresses (MDEV-20627)
- Galera should replicate nextval()-related changes in sequences with INCREMENT <> 0, at least NOCACHE ones with engine=InnoDB (MDEV-27862)
- Add support for OpenSSL 3.0 in Galera (MDEV-25949)
- Implement a method to add IPs to allowlist for Galera Cluster node addresses that can make SST/IST requests (MDEV-27246)

Optimizer

- Server crash in JOIN_CACHE::free or in copy_fields (MDEV-23809)
  - Queries that use DISTINCT and an always-constant function like COLLATION(aggegate_func(...)) could cause a server crash. Note that COLLATION() is a special function - its value is constant even if its argument is not costant.
- Crash when using ANY predicand with redundant subquery in GROUP BY clause (MDEV-29139)
  - A query with a subuquery in this form could cause a crash:

```
... ANY (SELECT ... GROUP BY (SELECT redundant_subselect_here)) ...
```

- MariaDB Server SEGV on INSERT .. SELECT (MDEV-26427)
  - Certain queries in form "INSERT ... SELECT with_aggregate_or_window_func" could cause a crash.
- restore_prev_nj_state() doesn't update cur_sj_inner_tables correctly (MDEV-28749)
  - Subquery semi-join optimization could miss LooseScan or FirstMatch strategies for certain queries.
- Optimizer uses all partitions after upgrade to 10.3 (MDEV-28246)
  - For multi-table UPDATE or DELETE queries, the optimizer failed to apply Partition Pruning optimization for the table that is updated or deleted from.
- Range optimizer regression for key IN (const, ...) (MDEV-25020)
  - The issue can be observed on MariaDB 10.5.9 and later versions which have the fix for MDEV-9750. That fix introduceds optimizer_max_sel_arg_weight.
  - If one sets optimizer_max_sel_arg_weight to a very high value or zero (which means "unlimited") and runs queries that produce heavy-weight graphs, they can observe a performance slowdown, e.g.:

```
table.keyXpartY [NOT] IN ( ... )
```

- Wrong result with table elimination combined with not_null_range_scan (MDEV-28858)
  - If one runs with optimizer_switch='not_null_range_scan=on' (which is not enabled by default), a query that does a join and has const tables could produce a wrong result.
- Assertion `tmp >= 0' failed in best_access_path (MDEV-28882)
  - If one uses histogram_type=JSON_HB, has collected a histogram of that type and runs a query that selects a very narrow range near histogram end, they can hit an assertion in the optimizer due to rounding errors in the histogram causing negative selectivity.

General

- Crash in JSON_EXTRACT (MDEV-29188)
- ALTER TABLE ALGORITHM=NOCOPY does not work after upgrade (MDEV-28727)
- Server crash upon CREATE VIEW with unknown column in ON condition (MDEV-29088)
- password_reuse_check plugin mixes username and password (MDEV-28838)
- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.10 for Debian 10 "Buster" for ppc64el

Changelog

For a complete list of changes made in MariaDB 10.10.1, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.10.1, see the MariaDB Foundation release announcement.

Do not use non-stable (non-GA) releases in production!

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.2.2.8 MariaDB 10.10.0 Release Notes

The most recent release of MariaDB 10.10 is:
MariaDB 10.10.7 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.10.0 | Release Notes | Changelog | Overview of 10.10

Release date: 23 Jun 2022

Do not use *alpha* releases in production!

MariaDB 10.10 is a current development series of MariaDB. It is an evolution of MariaDB 10.9 with several entirely new features.

MariaDB 10.10.0 is not a single release, but is instead a number of preview releases based on feature branches. Each should be considered ***Alpha***.

For an overview of MariaDB 10.10 see the What is MariaDB 10.10? page.

Thanks, and enjoy MariaDB!

List of packages

Remember, these features are in separate *preview packages*. The subsection header text corresponds to the preview package name.

Replication

- Change defaults for CHANGE MASTER TO so that GTID-based replication is used by default if master supports it (MDEV-19801)
- Deprecate MASTER_USE_GTID=Current_Pos to favor new MASTER_DEMOTE_TO_SLAVE option (MDEV-20122)

Available as container: quay.io/mariadb-foundation/mariadb-devel:10.10-gtid

Optimizer

- Improve optimization of joins with many tables, including eq_ref tables (MDEV-28852)
- Table elimination does not work across derived tables (MDEV-26278)

Available as container: quay.io/mariadb-foundation/mariadb-devel:10.10-optimizer

UCA14 Collation

- Add UCA-14.0.0 collations (MDEV-27009)
- Improve contraction performance in UCA collations (MDEV-27265)
- Improve UCA collation performance for utf8mb3 and utf8mb4 (MDEV-27266)

Available as container: quay.io/mariadb-foundation/mariadb-devel:10.10-uca14

DDL

- ALTER ONLINE TABLE (MDEV-16329) (not included in MariaDB 10.10.1)
- Atomic CREATE OR REPLACE TABLE (MDEV-25292) (not included in MariaDB 10.10.1)

Available as container: quay.io/mariadb-foundation/mariadb-devel:10.10-ddl

Galera

- Implement a method to add IPs to allowlist for Galera Cluster node addresses that can make SST/IST requests (MDEV-27246)

Miscellaneous

- Change default of explicit_defaults_for_timestamp to ON (MDEV-28632)
- --ssl option set as default for mariadb CLI (MDEV-27105)
- Add RANDOM_BYTES function (MDEV-25704)
- The INET4 data type (MDEV-23287)
- Re-design the upper level of handling UPDATE and DELETE statements (MDEV-28883)
- Deprecate the DES_ENCRYPT/DECRYPT functions (MDEV-27104)

Available as container: quay.io/mariadb-foundation/mariadb-devel:10.10-misc

---

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.3 MariaDB Server 10.9

Changes and Improvements in MariaDB 10.9
Current Version: 10.9.8 | Status: Stable (GA) | Release Date: 14 Aug 2023

Release Notes - MariaDB 10.9 Series
MariaDB 10.9 series release notes.

Changelogs - MariaDB 10.9 Series
MariaDB 10.9 changelogs

There are 1 related questions.

7.0.3.1 Changes and Improvements in MariaDB 10.9

MariaDB 10.9 is no longer maintained. Please use a more recent release.

The most recent release of MariaDB 10.9 is:
MariaDB 10.9.8 Stable (GA) Download Now

Contents

MariaDB 10.9 is a previous short-term release series. The first stable release was in August 2022, and it was maintained until August 2023.

New Features & Improvements

JSON

- JSON_OVERLAPS function (MDEV-27677)
- Implement range notation for JSONPath (MDEV-27911)
- Support JSONPath negative index (MDEV-22224)

InnoDB

- innodb_log_file_size is now dynamic (MDEV-27812)
- InnoDB performance improvements (MDEV-27557, MDEV-28185, MDEV-27767, MDEV-28313, MDEV-28137, MDEV-28465, MDEV-26789)
- innodb_disallow_writes removed (MDEV-25975)

Hashicorp Key Management Plugin

- Hashicorp Key Management Plugin for implementing encryption using keys stored in the Hashicorp Vault KMS (MDEV-19281)

Replication and Galera

- Implement the --do-domain-ids, --ignore-domain-ids, and --ignore-server-ids options for mariadb-binlog (MDEV-20119)
- Semisync-slave server recovery is extended to work on new server_id server (MDEV-27342)
- mariadb-binlog --stop-never --raw now flushes the result file to disk after each processed event so the file can be listed with the actual bytes (MDEV-14608)
- JSON file interface to wsrep node state / SST progress logging (MDEV-26971)

SHOW ANALYZE FORMAT=JSON

- Extend SHOW EXPLAIN to support SHOW ANALYZE [FORMAT=JSON] (MDEV-27021)
- Add EXPLAIN FOR CONNECTION syntax support to SHOW EXPLAIN (MDEV-10000)

Variables

- For a list of all new variables, see System Variables Added in MariaDB 10.9.
- Merge old to old_mode sql variable (MDEV-24920)

The following variables have been deprecated:

- innodb_change_buffering
- old (replaced by old_mode)

Security Vulnerabilities Fixed in MariaDB 10.9

For a complete list of security vulnerabilities (CVEs) fixed across all versions of MariaDB, see the Security Vulnerabilities Fixed in MariaDB page.

- CVE-2022-47015: MariaDB 10.9.6
- CVE-2022-38791: MariaDB 10.9.2
- CVE-2022-32091: MariaDB 10.9.2
- CVE-2022-32089: MariaDB 10.9.2
- CVE-2022-32084: MariaDB 10.9.2
- CVE-2022-32082: MariaDB 10.9.2
- CVE-2022-32081: MariaDB 10.9.2
- CVE-2018-25032: MariaDB 10.9.2

List of All MariaDB 10.9 Releases

Date	Release	Status	Release Notes	Changelog
14 Aug 2023	**MariaDB 10.9.8**	Stable (GA)	Release Notes	Changelog
7 Jun 2023	**MariaDB 10.9.7**	Stable (GA)	Release Notes	Changelog
10 May 2023	**MariaDB 10.9.6**	Stable (GA)	Release Notes	Changelog
6 Feb 2023	**MariaDB 10.9.5**	Stable (GA)	Release Notes	Changelog
7 Nov 2022	**MariaDB 10.9.4**	Stable (GA)	Release Notes	Changelog
19 Sep 2022	**MariaDB 10.9.3**	Stable (GA)	Release Notes	Changelog
22 Aug 2022	**MariaDB 10.9.2**	Stable (GA)	Release Notes	Changelog
20 May 2022	**MariaDB 10.9.1**	RC	Release Notes	Changelog
23 Mar 2022	**MariaDB 10.9.0**	Alpha	Release Notes	

7.0.3.2 Release Notes - MariaDB 10.9 Series

MariaDB 10.9 was a short-term maintenance stable series of MariaDB maintained until August 2023.

MariaDB 10.9.8 Release Notes
Status: Stable (GA) | Release Date: 14 Aug 2023

MariaDB 10.9.7 Release Notes
Status: Stable (GA) | Release Date: 7 Jun 2023

MariaDB 10.9.6 Release Notes
Status: Stable (GA) | Release Date: 10 May 2023

MariaDB 10.9.5 Release Notes
Status: Stable (GA) | Release Date: 6 Feb 2023

MariaDB 10.9.4 Release Notes
Status: Stable (GA) | Release Date: 7 Nov 2022

MariaDB 10.9.3 Release Notes
Status: Stable (GA) | Release Date: 19 Sep 2022

MariaDB 10.9.2 Release Notes
Status: Stable (GA) | Release Date: 22 Aug 2022

MariaDB 10.9.1 Release Notes
Status: RC | Release Date: 20 May 2022

MariaDB 10.9.0 Release Notes
Status: Alpha | Release Date: 23 Mar 2022

7.0.3.2.1 MariaDB 10.9.8 Release Notes

Download | Release Notes | Changelog | Overview of 10.9

Alternate download from mariadb.org

Release date: 14 Aug 2023

MariaDB 10.9 is a previous short-term stable series of MariaDB, maintained until August 2023. It is an evolution of MariaDB 10.8 with several entirely new features.

MariaDB 10.9.8 is a ***Stable (GA)*** release.

MariaDB 10.9.8 is the last release of the MariaDB 10.9 release series.

For an overview of MariaDB 10.9 see the What is MariaDB 10.9? page.

Thanks, and enjoy MariaDB!

Notable Items

- As per the MariaDB Maintenance Policy, this will be the final release of MariaDB 10.9

Upgrading from MySQL

- MariaDB now detects and converts previously incompatible MySQL partition schemes (MDEV-29253)

General

- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.9 for Ubuntu 18.04 LTS "Bionic"

and Ubuntu 22.10 "Kinetic"

- mariadb-dump --force doesn't ignore error as it should (MDEV-31092)
- 280 Bytes lost in mysys/array.c, mysys/hash.c, sql/sp.cc, sql/sp.cc, sql/item_create.cc, sql/item_create.cc, sql/sql_yacc.yy:10748 when using oracle sql_mode (MDEV-26186)
- SQL/PL package body does not appear in `I_S.ROUTINES.ROUTINE_DEFINITION` (MDEV-30662)
- Unexpected result when combining DISTINCT, subselect and LIMIT (MDEV-28285)
- ROW variables do not get assigned from subselects (MDEV-31250)
- Crash after setting global session_track_system_variables to an invalid value (MDEV-25237)
- ODKU of non-versioning column inserts history row (MDEV-23100)
- UPDATE not working properly on transaction precise system versioned table (MDEV-25644)
- Assertion `` `const_item_cache == true' `` failed in `Item_func::fix_fields` (MDEV-31319)
- ANALYZE doesn't work with pushed derived tables (MDEV-29284)
- `get_partition_set` is never executed in `ha_partition::multi_range_key_create_key` due to bitwise & with 0 constant (MDEV-24712)
- Client can crash the server with a `mysql_list_fields("view")` call (MDEV-30159)
- `I_S.parameters` not immediatly changed updated after procedure change (MDEV-31064)
- bzero wipes more bytes than necessary in `set_global_from_ddl_log_entry` (MDEV-31521)
- Assertion `` `0' `` failed in `Type_handler_row::field_type` upon `TO_CHAR` with wrong argument (MDEV-29152)
- mysql_upgrade fails due to `old_mode=""` , with "Cannot load from mysql.proc. The table is probably corrupted" (MDEV-28915)

Character Sets, Data Types

- UUIDs version >= 6 are now stored without byte-swapping, UUIDs with version >=8 and variant=0 are now considered invalid, old tables are supported, old (always byte swapped) and new (swapped for version < 6) UUIDs can be compared and converted transparently (MDEV-29959)
- UBSAN: null pointer passed as argument 1, which is declared to never be null in `my_strnncoll_binary` on `SELECT ... COUNT` or `GROUP_CONCAT` (MDEV-28384)
- Possibly wrong result or Assertion `0' failed in `Item_func_round::native_op` (MDEV-23838)
- Assertion `` `(length % 4) == 0' `` failed in `my_lengthsp_utf32` on `SELECT` (MDEV-29019)
- UBSAN: negation of -X cannot be represented in type `'long long int'` ; cast to an unsigned type to negate this value to itself in `Item_func_mul::int_op` and `Item_func_round::int_op` (MDEV-30932)
- Assorted assertion failures in json_find_path with certain collations (MDEV-23187)

InnoDB

- Crashing on I/O error is unhelpful (MDEV-27593)
- SIGSEGV in `log_sort_flush_list()` in InnoDB crash recovery (MDEV-31354)
- InnoDB tables are being flagged as corrupted on an I/O bound server (MDEV-31767)
- Duplicate entry allowed into a UNIQUE column (MDEV-31120)
- Server Status Innodb_row_lock_time% is reported in seconds (MDEV-29311)
- innochecksum dies with Floating point exception (MDEV-31641)
- Add InnoDB engine information to the slow query log (MDEV-31558)
- Deadlock with 3 concurrent DELETEs by unique key (MDEV-10962)
- innodb protection against dual processes accessing data insufficient (MDEV-31568)
- ER_DUP_KEY in `mysql.innodb_table_stats` upon RENAME on sequence (MDEV-31607)
- Assertion `` `!strcmp(index->table->name.m_name, "SYS_FOREIGN") || !strcmp(index->table->name.m_name, "SYS_FOREIGN_COLS")' `` failed in `btr_node_ptr_max_size` (MDEV-19216)
- InnoDB: Failing assertion: `page_type == i_s_page_type[page_type].type_value` (MDEV-31386)
- `btr_estimate_n_rows_in_range()` accesses unfixed, unlatched page (MDEV-30648)
- `MODIFY COLUMN` can break FK constraints, and lead to unrestorable dumps (MDEV-31086)
- Recovery or backup failure after innodb_undo_log_truncate=ON (MDEV-31487)
- Assertion `'n & PENDING'` failed in `fil_space_t::set_needs_flush()` (MDEV-31442)
- `fil_node_open_file()` releases `fil_system.mutex` allowing other thread to open its file node (MDEV-31256)
- ASAN errors in `dict_v_col_t::detach` upon adding key to virtual column (MDEV-31416)
- Purge trying to access freed secondary index page (MDEV-31264)
- Freed data pages are not always being scrubbed (MDEV-31253)
- InnoDB recovery hangs after reporting corruption (MDEV-31353)
- `!cursor->index->is_committed()` in row0ins.cc after update to 10.4.13 from 10.3.21 (MDEV-22739)
- innodb_undo_log_truncate=ON fails to wait for purge of enough transaction history (MDEV-31355)
- SET GLOBAL innodb_undo_log_truncate=ON does not free space when no undo logs exist (MDEV-31382)
- innodb_read_ahead_threshold (linear read-ahead) does not work (MDEV-29967)
- InnoDB recovery and `mariadb-backup --prepare` fail to report detailed progress (MDEV-29911)
- `fil_ibd_create()` may hijack the file handle of an old file (MDEV-31347)

- innodb_undo_log_truncate=ON recovery results in a corrupted undo log (MDEV-31373)
- Server freeze due to innodb_change_buffering and innodb_file_per_table=0 (MDEV-31088)
- Change buffer entries are left behind when freeing a page, causing secondary index corruption when the page is later reused (MDEV-31385)
- Foreign Key Constraint actions don't affect Virtual Column (MDEV-18114)

Aria

- Various crashes upon INSERT/UPDATE after changing Aria settings (MDEV-28054)
- Various crashes/asserts/corruptions when Aria encryption is enabled/used, but the encryption plugin is not loaded (MDEV-26258)

Spider

- SIGSEGV in `spider_db_open_item_field` and SIGSEGV in `spider_db_print_item_type`, on SELECT (MDEV-29447)
- Spider variables that double as table params overriding mechanism is buggy (MDEV-31524)

Optimizer

- ANALYZE FORMAT=JSON now includes InnoDB engine statistics for each table (MDEV-31577)
- Assertion `` `last_key_entry >= end_pos' `` failed in virtual bool `JOIN_CACHE_HASHED::put_record()` (MDEV-31348)
- Problem with open ranges on prefix blobs keys (MDEV-31800)
- Equal on two RANK window functions create wrong result (MDEV-20010)
- Recursive CTE execution is interrupted without errors or warnings (MDEV-31214)
- Assertion `` `s->table->opt_range_condition_rows <= s->found_records' `` failed in `apply_selectivity_for_table` (MDEV-31449)
- Inconsistency between MRR and SQL layer costs can cause poor query plan (MDEV-31479)
- `MAX_SEL_ARG` memory exhaustion is not visible in the optimizer trace (MDEV-30964)
- SHOW TABLES not working properly with `lower_case_table_names=2` (MDEV-30765)
- Segfault on select query using index for group-by and filesort (MDEV-30143)
- Server crash in `store_length`, assertion failure in `Type_handler_string_result::sort_length` (MDEV-31743)

Replication

- Parallel Slave SQL Thread Can Update Seconds_Behind_Master with Active Workers (MDEV-30619)
- ALTER SEQUENCE ends up in optimistic parallel slave binlog out-of-order (MDEV-31503)
- STOP SLAVE takes very long time on a busy system (MDEV-13915)
- On slave XA COMMIT/XA ROLLBACK fail to return an error in read-only mode (MDEV-30978)
- Calling a function from a different database in a slave side trigger crashes (MDEV-29894)
- `rpl.rpl_manual_change_index_file` occasionally fails in BB with Result length mismatch (MDEV-30214)

Galera

- Node never returns from Donor/Desynced to Synced when `wsrep_mode = BF_ABORT_MARIABACKUP` (MDEV-31737)
- Node has been dropped from the cluster on Startup / Shutdown with async replica (MDEV-31413)
- KILL QUERY maintains nodes data consistency but breaks GTID sequence (MDEV-31075)
- Assertion failure `!lock.was_chosen_as_deadlock_victim in trx0trx.h:1065` (MDEV-30963)
- MariaDB stuck on starting commit state (waiting on commit order critical section) (MDEV-29293)
- Assertion `state() == s_aborting || state() == s_must_replay` failed in int wsrep::transaction::after_rollback() (MDEV-30013)
- Assertion `!wsrep_has_changes(thd) || (thd->lex->sql_command == SQLCOM_CREATE_TABLE && !thd->is_current_stmt_binlog_format_row()) || thd->wsrep_cs().transaction().state() == wsrep::transaction::s_aborted` failed (MDEV-30388)
- Server crashes when wsrep_sst_donor and wsrep_cluster_address set to NULL (MDEV-28433)
- Create temporary sequence can cause inconsistency (MDEV-31335)
- Galera 4 unable to query cluster state if not primary component (MDEV-21479)

Changelog

For a complete list of changes made in MariaDB 10.9.8, with links to detailed information on each push, see the changelog

.

Contributors

For a full list of contributors to MariaDB 10.9.8, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.3.2.2 MariaDB 10.9.7 Release Notes

The most recent release of MariaDB 10.9 is:
MariaDB 10.9.8 Stable (GA) Download Now

Download 10.9.7 | Release Notes | Changelog | Overview of 10.9

Release date: 7 Jun 2023

MariaDB 10.9 is the current short-term maintenance stable series of MariaDB, maintained until August 2023. It is an evolution of MariaDB 10.8 with several entirely new features.

MariaDB 10.9.7 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.9 see the What is MariaDB 10.9? page.

Thanks, and enjoy MariaDB!

Notable Items

InnoDB

- Server crashes in st_join_table::choose_best_splitting (MDEV-31403)
- Crash with condition pushable into derived and containing outer reference (MDEV-31240)
- InnoDB does not free UNDO after the fix of MDEV-30671 (MDEV-31234)
- InnoDB hang fixes (MDEV-31158, MDEV-31343, MDEV-31350)
- Innodb_buffer_pool_read_requests is not updated correctly (MDEV-31309)
- InnoDB monitor trx_rseg_history_len was accidentally disabled by default (MDEV-31308)
- Revert "MDEV-30473: Do not allow GET_LOCK() / RELEASE_LOCK() in cluster"

Optimizer

- Crash with condition pushable into derived and containing outer reference (MDEV-31403 MDEV-31240)
- Crash with EXPLAIN EXTENDED for multi-table update of system table (MDEV-31224)

Changelog

For a complete list of changes made in MariaDB 10.9.7, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.9.7, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.3.2.3 MariaDB 10.9.6 Release Notes

The most recent release of MariaDB 10.9 is:
MariaDB 10.9.8 Stable (GA) Download Now

Download 10.9.6 | Release Notes | Changelog | Overview of 10.9

Release date: 10 May 2023

MariaDB 10.9 is the current short-term maintenance stable series of MariaDB, maintained until August 2023. It is an evolution of MariaDB 10.8 with several entirely new features.

MariaDB 10.9.6 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.9 see the What is MariaDB 10.9? page.

Thanks, and enjoy MariaDB!

Notable Items

InnoDB

- Crash on ROLLBACK in a ROW_FORMAT=COMPRESSED table (MDEV-30882)
- UNIQUE USING HASH accepts duplicate entries for tricky collations (MDEV-30034)
- rec_get_offsets() is not optimal (MDEV-30567)
- Performance regression in fil_space_t::try_to_close() introduced in MDEV-23855 (MDEV-30775)
- InnoDB recovery hangs when buffer pool ran out of memory (MDEV-30551)
- InnoDB undo log truncation fails to wait for purge of history (MDEV-30671)
- MariaDB crash due to DB_FAIL reported for a corrupted page (MDEV-30397)
- Deadlock between INSERT and InnoDB non-persistent statistics update (MDEV-30638)
- InnoDB hang on B-tree split or merge (MDEV-29835)
- Performance regression in locking reads from secondary indexes (MDEV-30357)
- Improve adaptive flushing (MDEV-26055)
- Make page flushing even faster (MDEV-26827)
- Purge misses a chance to free not-yet-reused undo pages (MDEV-29593)
- InnoDB temporary tablespace: reclaiming of free space does not work (MDEV-26782)
- Deadlock between CHECK TABLE and bulk insert (MDEV-30798)
- Fix miscount of doublewrites by Innodb_data_written (MDEV-31124)

Backup

- mariadb-backup doesn't utilise innodb-undo-log-directory (if specified as a relative path) during copy-back operation (MDEV-28187)
- mariabackup issues error messages during InnoDB tablespaces export on partial backup preparing (MDEV-29050)
- mariadb-backup does not copy Aria logs if aria_log_dir_path is used (MDEV-30968)
- Race condition between buffer pool flush and log file deletion in mariadb-backup --prepare (MDEV-30860)

Replication

- Fixed a deadlock on parallel slave involving full image Write event on the sequence engine (MDEV-29621)

- Fixed an attempted out-of-order binlogging error on slave involving ALTER on the sequence engine (MDEV-31077)
- Corrected non-versioned master to versioned slave replication on no-unique attribute table (MDEV-30430)
- Mended encrypted binlog master to error out to gtid-mode slave when master could not decrypt a binlog file (MDEV-28798)
- Refined optimistic parallel slave to error-exit without any hang (MDEV-30780)
- Ensured SHOW-SLAVE-STATUS is processed on the parallel slave having a necessary mutex always intialized (MDEV-30620)
- Fixed the slave applier to report a correct error when gtid_slave_pos insert fails for some (engine) reasons (MDEV-31038)
- Made parallel slave reports in performance schema consistent with that of show-slave-status (MDEV-26071)

Optimizer

- Split Materialized optimization is improved to re-fill the materialized table only if necessary. The fewer number of table refills is taken into account when choosing query plan, too (MDEV-26301).
- New optimizer_switch option, hash_join_cardinality , is added. It is off by default. When set to ON, the optimizer will produce tighter bounds for hash join output cardinality. (MDEV-30812)
- Queries using `SELECT DISTINCT some_expression(aggregate_function())` could produce wrong query result. (MDEV-20057)
- ANALYZE FORMAT=JSON now prints more information about Block Nested Loop joins: `block-nl-join` element now has `r_loops` , `r_effective_rows` and `r_other_time_ms` fields (MDEV-30806 , MDEV-30972).
- A GROUP BY query with `MIN(primary_key)` in select list and `primary_key<>const` in the WHERE could produce wrong result when executed with "Using index for group-by" strategy (MDEV-30605)
- EXPLAIN could erroneously report that Rowid Filter optimization is used for partitioned tables. Partitioned tables do not support it. (MDEV-30596)
- A bug in selectivity computations for SINGLE/DOUBLE_PREC_HB histograms could cause wrong estimates to be produced. This could cause the optimizer to pick sub-optimal query plans (MDEV-31067).

Docker Official Image

- Add replication setup to containers contributed by Md Sahil (MDEV-29762)

General

- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.9 for Fedora 36.
- In this release repositories for Fedora 38 and Ubuntu 23.04 Lunar have been added.

Security

- Fixes for the following security vulnerabilities :
  - CVE-2022-47015

Changelog

For a complete list of changes made in MariaDB 10.9.6, with links to detailed information on each push, see the changelog .

Contributors

For a full list of contributors to MariaDB 10.9.6, see the MariaDB Foundation release announcement .

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.3.2.4 MariaDB 10.9.5 Release Notes

Download | Release Notes | Changelog | Overview of 10.9

Release date: 6 Feb 2023

MariaDB 10.9 is the current short-term maintenance stable series of MariaDB, maintained until August 2023. It is an evolution of MariaDB 10.8 with several entirely new features.

MariaDB 10.9.5 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.9 see the What is MariaDB 10.9? page.

Thanks, and enjoy MariaDB!

Notable Items

- As mentioned in the 10.9.4 release notes, our Yum/DNF/Zypper repositories for Red Hat Enterprise Linux, CentOS, Fedora, openSUSE, and SUSE are changing **with this release** to being signed with a new GPG key with SHA2 digest algorithms instead of SHA1. See this blog post and the GPG page for more details.

InnoDB

- Full-text index corruption with system versioning (MDEV-25004)
- innodb_undo_log_truncate=ON recovery and backup fixes (MDEV-29999, MDEV-30179, MDEV-30438)
- Upgrade after a crash is not supported (MDEV-24412)
- Remove InnoDB buffer pool load throttling (MDEV-25417)
- InnoDB shutdown hangs when the change buffer is corrupted (MDEV-30009)
- `innodb_fast_shutdown=0` fails to report change buffer merge progress (MDEV-29984)
- `mariadb-backup --backup --incremental --throttle=...` hangs (MDEV-29896)
- Crash after recovery, with InnoDB: Tried to read (MDEV-30132)
- Trying to write ... bytes at ... outside the bounds (MDEV-30069)
- TRUNCATE breaks FOREIGN KEY locking (MDEV-29504, MDEV-29849)
- `INFORMATION_SCHEMA.INNODB_TABLESPACES_ENCRYPTION.NAME` is NULL for undo tablespaces (MDEV-30119)
- Fixed hangs and error handling in B-tree operations (MDEV-29603, MDEV-30400)
- InnoDB bulk insert fixes (MDEV-30047, MDEV-30321)

Galera

- Fixes for cluster wide write conflict resolving (MDEV-29684)

Replication

- Parallel slave applying in binlog order is corrected for admin class of commands including ANALYZE (MDEV-30323)
- Seconds_Behind_Master is now shown now more precisely at the slave applier start, including in the delayed mode (MDEV-29639)
- `mysqlbinlog --verbose` is made to show the type of compressed columns (MDEV-25277)
- Deadlock is resolved on replica involving `BACKUP STAGE BLOCK_COMMIT` and a committing user XA (MDEV-30423)

JSON

- JSON_PRETTY added as an alias for JSON_DETAILED (MDEV-19160)

General

- Infinite sequence of recursive calls when processing embedded CTE (MDEV-30248)
- Crash with a query containing nested WINDOW clauses (MDEV-30052)
- Major performance regression with 10.6.11 (MDEV-29988)
- Json Range only affects first row of the result set (MDEV-30304)
- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.9 for Fedora 35.
- In this release repositories for Fedora 37 and Ubuntu 22.10 Kinetic have been added.

Changelog

For a complete list of changes made in MariaDB 10.9.5, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.9.5, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.3.2.5 MariaDB 10.9.4 Release Notes

The most recent release of MariaDB 10.9 is:
MariaDB 10.9.8 Stable (GA) Download Now

Download 10.9.4 | Release Notes | Changelog | Overview of 10.9

Release date: 7 Nov 2022

MariaDB 10.9 is the current short-term maintenance stable series of MariaDB, maintained until August 2023. It is an evolution of MariaDB 10.8 with several entirely new features.

MariaDB 10.9.4 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.9 see the What is MariaDB 10.9? page.

Thanks, and enjoy MariaDB!

Notable Items

SSL

- The server no longer tolerates incorrectly configured SSL (MDEV-29811). If you have enabled SSL in `my.cnf` but have not configured it properly (for example, a certificate file is missing), MariaDB used to silently disable SSL, leaving you under impression that everything was fine and connections were secure. Since this release, MariaDB will fail to start if SSL is enabled, but cannot be switched on.

Backup

- Assertion on info.page_size failed in xb_delta_open_matching_space (MDEV-18589)
- Mariabackup locks database for minutes (MDEV-28772)

InnoDB

- Adaptive hash index MDEV-27700, MDEV-29384
- MVCC and locking (MDEV-29666, MDEV-27927, MDEV-28709, MDEV-29635)
- Virtual columns (MDEV-29299, MDEV-29753)
- InnoDB crash recovery fixes (MDEV-29559)
- Race condition between KILL and transaction commit (MDEV-29368)
- Implement CHECK TABLE…EXTENDED for InnoDB (MDEV-24402)

- InnoDB persistent statistics fail to update after bulk insert (MDEV-28327)
- InnoDB bulk insert bug fixes (MDEV-29570, MDEV-29761)

Galera

- Galera updated to 26.4.13
- Galera server crashes after 10.3 > 10.4 upgrade (MDEV-29375)
- wsrep_incoming_addresses status variable prints 0 as port number if the port is not mentioned in wsrep_node_incoming_address system variable (MDEV-28868)

Replication

- XA COMMIT is not binlogged when the XA transaction has not updated any transaction engine (MDEV-25616)
- Concurrent CREATE TRIGGER statements made to binlog without any mixup (MDEV-25606)

Repositories

- Beginning with the next release (Q1 2023), our Yum, DNF, and Zypper repositories for Red Hat, Fedora, and SUSE will be migrated to being signed with a new GPG key. The key we are migrating to is the same one we already use for our Debian and Ubuntu Repositories.
  - The short Key ID is: `0xC74CD1D8`
  - The long Key ID is: `0xF1656F24C74CD1D8`
  - The full fingerprint of the key is: `177F 4010 FE56 CA33 3630 0305 F165 6F24 C74C D1D8`
  - The key can be imported now in preparation for this change using the following command:

```
sudo rpm --import https://supplychain.mariadb.com/MariaDB-Server-GPG-KEY
```

Docker Official Image

The following changes have been made to the `docker.io/library/mariadb` container image.

- The number of gpg packages packages has been removed, leaving enough to `apt-get update`, but `dirmngr` that would fetch keys has been removed. (inspired by issue #469)
- The environment variable `LANG=C.UTF-8` has been added for those that exec into containers and copy paste UTF8 characters (fixes issue #468).
- Adds OCI labels to image (fixes issue 436 and users need for version)
- MariaDB config: skip-host-cache and skip-name-resolve moved to `/etc/mysql/mariadb.conf.d/05-skipcache.cnf`

Changelog

For a complete list of changes made in MariaDB 10.9.4, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.9.4, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.3.2.6 MariaDB 10.9.3 Release Notes

The most recent release of MariaDB 10.9 is:

MariaDB 10.9.8 Stable (GA) Download Now

Download 10.9.3 | Release Notes | Changelog | Overview of 10.9

Release date: 19 Sep 2022

MariaDB 10.9 is the current short-term maintenance stable series of MariaDB, maintained until August 2023. It is an evolution of MariaDB 10.8 with several entirely new features.

MariaDB 10.9.3 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.9 see the What is MariaDB 10.9? page.

Thanks, and enjoy MariaDB!

Notable Items

Issues Fixed

- Assertion mysql_mutex_assert_owner(&log_sys.flush_order_mutex) failed in mtr_t::commit() (MDEV-29383)
- Frequent "Data structure corruption" in InnoDB after OOM (MDEV-29374)
- Recovery or backup of instant ALTER TABLE is incorrect (MDEV-29438)
- InnoDB Temporary Tablespace (ibtmp1) is continuously growing (MDEV-28240)
- Full text index corruption if shutdown before changes are fully flushed (MDEV-29342)
- JSON_VALUE() does not parse NULL properties properly (MDEV-27151)
- InnoDB hangs on multiple concurrent requests of a cold ROW_FORMAT=COMPRESSED page (MDEV-27983)

Changelog

For a complete list of changes made in MariaDB 10.9.3, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.9.3, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.3.2.7 MariaDB 10.9.2 Release Notes

The most recent release of MariaDB 10.9 is:
MariaDB 10.9.8 Stable (GA) Download Now

Download 10.9.2 | Release Notes | Changelog | Overview of 10.9

Release date: 22 Aug 2022

MariaDB 10.9 is a current stable series of MariaDB, maintained until August 2023. It is an evolution of MariaDB 10.8 with several entirely new features.

MariaDB 10.9.2 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.9 see the What is MariaDB 10.9? page.

Thanks, and enjoy MariaDB!

Notable Items

InnoDB

- InnoDB corruption due to lack of file locking (MDEV-28495)
- FULLTEXT search with apostrophe, and mandatory words (MDEV-20797)
- ALTER TABLE IMPORT TABLESPACE corrupts an encrypted table (MDEV-28779)
- ALTER TABLE wrong-result fix (MDEV-26294)
- Crash recovery fixes (MDEV-28668 , MDEV-28731)
- DDL crash recovery fixes (MDEV-28752 , MDEV-28802 , MDEV-28864 , MDEV-28870 , MDEV-28923 , MDEV-28977)
- Avoid crashes on corrupted data (MDEV-13542 , MDEV-18519 , MDEV-21098 , MDEV-22388 , MDEV-28457 , MDEV-28950)
- Bulk load bug fixes (MDEV-28242 , MDEV-28679)
- Performance fixes (MDEV-28708 , MDEV-28766)

Replication

- explicit_defaults_for_timestamp is stored in binlog, so that CREATE TABLE on slave would always have the same effect as on master. (MDEV-29078)
- ER_SLAVE_INCIDENT error is specified now on slave to be seen with SHOW-SLAVE-STATUS (MDEV-21087)
- INCIDENT_EVENT is no longer binlogged when a being logged transaction can be safely rolledback (MDEV-21443)
- sequences related row-format events are made to correspond to binlog_row_image (MDEV-28487)
- Possible reason of FLUSH BINARY LOGS hang is eliminated (MDEV-28948)
- Fix out-of-order gtid error in the circular semisync setup (MDEV-28609)

Galera

- Possible to write/update with read_only=ON and not a SUPER privilege (MDEV-28546)
- Node crashes with Transport endpoint is not connected mysqld got signal 6 (MDEV-25068)
- Galera4 not able to report proper wsrep_incoming_addresses (MDEV-20627)
- Galera should replicate nextval()-related changes in sequences with INCREMENT <> 0, at least NOCACHE ones with engine=InnoDB (MDEV-27862)
- Add support for OpenSSL 3.0 in Galera (MDEV-25949)

Optimizer

- Server crash in JOIN_CACHE::free or in copy_fields (MDEV-23809)
  - Queries that use DISTINCT and an always-constant function like COLLATION(aggegate_func(...)) could cause a server crash. Note that COLLATION() is a special function - its value is constant even if its argument is not costant.
- Crash when using ANY predicand with redundant subquery in GROUP BY clause (MDEV-29139)
  - A query with a subuquery in this form could cause a crash:

```
... ANY (SELECT ... GROUP BY (SELECT redundant_subselect_here)) ...
```

- MariaDB Server SEGV on INSERT .. SELECT (MDEV-26427)
  - Certain queries in form "INSERT ... SELECT with_aggregate_or_window_func" could cause a crash.
- restore_prev_nj_state() doesn't update cur_sj_inner_tables correctly (MDEV-28749)
  - Subquery semi-join optimization could miss LooseScan or FirstMatch strategies for certain queries.
- Optimizer uses all partitions after upgrade to 10.3 (MDEV-28246)
  - For multi-table UPDATE or DELETE queries, the optimizer failed to apply Partition Pruning optimization for the table that is updated or deleted from.
- Range optimizer regression for key IN (const, ...) (MDEV-25020)
  - The issue can be observed on MariaDB 10.5.9 and later versions which have the fix for MDEV-9750 . That fix introduceds optimizer_max_sel_arg_weight.
  - If one sets optimizer_max_sel_arg_weight to a very high value or zero (which means "unlimited") and runs queries that produce heavy-weight graphs, they can observe a performance slowdown, e.g.:

```
table.keyXpartY [NOT] IN ( ... )
```

- Wrong result with table elimination combined with not_null_range_scan (MDEV-28858)
  - If one runs with optimizer_switch='not_null_range_scan=on' (which is not enabled by default), a query that does a join and has const tables could produce a wrong result.
- Assertion `tmp >= 0' failed in best_access_path (MDEV-28882)
  - If one uses histogram_type=JSON_HB, has collected a histogram of that type and runs a query that selects a very narrow range near histogram end, they can hit an assertion in the optimizer due to rounding errors in the histogram causing negative selectivity.

Spider

- The Spider version number now matches the server version (and the spider_version system variable removed) (MDEV-26282)
- spider_init_sql_alloc_size and spider_buffer_size have been deprecated (MDEV-27926, MDEV-28560)
- Spider's high-availability feature has been deprecated (MDEV-28479)

JSON

- JSON_TABLE: extract document fragment into JSON column (MDEV-25875)

CONNECT

- CONNECT Engine now supports INSERT IGNORE with Mysql Table type (MDEV-27766)

General

- explicit_defaults_for_timestamp now also has a session scope, not only global (MDEV-29225)
- New mariadb client option, `-enable-cleartext-plugin`. Option does not do anything, and is for MySQL-compatibility purposes only.
- Crash in JSON_EXTRACT (MDEV-29188)
- ALTER TABLE ALGORITHM=NOCOPY does not work after upgrade (MDEV-28727)
- Server crash upon CREATE VIEW with unknown column in ON condition (MDEV-29088)
- password_reuse_check plugin mixes username and password (MDEV-28838)
- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.9 for Debian 10 "Buster" for ppc64el

Security

- Fixes for the following security vulnerabilities:
  - CVE-2022-32082
  - CVE-2022-32089
  - CVE-2022-32081
  - CVE-2018-25032
  - CVE-2022-32091
  - CVE-2022-38791
  - CVE-2022-32084

Changelog

For a complete list of changes made in MariaDB 10.9.2, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.9.2, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.3.2.8 MariaDB 10.9.1 Release Notes

The most recent release of MariaDB 10.9 is:
MariaDB 10.9.8 Stable (GA) Download Now

Download 10.9.1 | Release Notes | Changelog | Overview of 10.9

Release date: 20 May 2022

Do not use non-stable (non-GA) releases in production!

MariaDB 10.9 is a current development series of MariaDB. It is an evolution of MariaDB 10.8 with several entirely new features.

MariaDB 10.9.1 is a ***Release Candidate (RC)*** release.

For an overview of MariaDB 10.9 see the What is MariaDB 10.9? page.

Thanks, and enjoy MariaDB!

Notable Items

InnoDB

- innodb_disallow_writes removed (MDEV-25975)
- InnoDB gap locking fixes (MDEV-20605, MDEV-28422)
- InnoDB performance improvements (MDEV-27557, MDEV-28185, MDEV-27767, MDEV-28313, MDEV-28137, MDEV-28465, MDEV-26789)
- Backup regression fixes (MDEV-27919)
- InnoDB portability: FreeBSD futexes (MDEV-26476), POWER and s390x transactional memory (MDEV-27956)
- ALTER TABLE: Fixed bogus duplicate key errors (MDEV-15250)
- DDL and crash recovery fixes (MDEV-27274, MDEV-27234, MDEV-27817)
- Requests to recalculate persistent statistics were sometimes lost (MDEV-27805)
- Deprecate the parameter innodb_change_buffering (MDEV-27735)
- Allow SET GLOBAL innodb_log_file_size (MDEV-27812)

Replication

- New options for mysqlbinlog --do-domain-ids, --ignore-domain-ids, and --ignore-server-ids are implemented (MDEV-20119)
- Semisync-slave server recovery is refined to correctly rollback prepared transaction (MDEV-28461)
- Circular semisync setup endless event circulation is handled (MDEV-27760)
- Semisync-slave server recovery is extended to work on new server_id server (MDEV-27342)
- Server initialization time gtid_slave_pos purge related reason of crashing in binlog background thread is removed (MDEV-26473)
- Shutdown of the semisync master can't produce inconsistent state anymore (MDEV-11853)
- Binlogs disappear after rsync IST (MDEV-28583)
- master crash is eliminated in compressed semisync replication protocol with packet counting amendment (MDEV-25580)
- OPTIMIZE on a sequence does not cause counterfactual ER_BINLOG_UNSAFE_STATEMENT anymore (MDEV-24617)
- Automatically generated Gtid_log_list_event is made to recognize within replication event group as a formal member (MDEV-28550)
- Replication unsafe INSERT .. ON DUPLICATE KEY UPDATE using two or more unique key values at a time with MIXED format binlogging is corrected (MDEV-28310)

- Replication unsafe INSERT .. ON DUPLICATE KEY UPDATE stops issuing unnecessary "Unsafe statement" with MIXED binlog format (MDEV-21810)
- Incomplete replication event groups are detected to error out by the slave IO thread (MDEV-27697)
- mysqlbinlog --stop-never --raw now flushes the result file to disk after each processed event so the file can be listed with the actual bytes (MDEV-14608)

Backup

- Incorrect binlogs after Galera SST using rsync and mariabackup (MDEV-27524)
- mariabackup does not detect multi-source replication slave (MDEV-21037)
- Useless warning "InnoDB: Allocated tablespace ID <id> for <tablename>, old maximum was 0" during backup stage (MDEV-27343)
- mariabackup prepare fails for incrementals if a new schema is created after full backup is taken (MDEV-28446)

Optimizer

- Query performance degradation in newer MariaDB versions when using many tables (MDEV-28073)
- A SEGV in Item_field::used_tables/update_depend_map_for_order... (MDEV-26402)
- ANALYZE FORMAT=JSON fields are incorrect for UNION ALL queries (MDEV-27699)
- Subquery in an UPDATE query uses full scan instead of range (MDEV-22377)
- Assertion `item1->type() == Item::FIELD_ITEM ... (MDEV-19398)
- Server crashes in Expression_cache_tracker::fetch_current_stats (MDEV-28268)
- MariaDB server crash at Item_subselect::init_expr_cache_tracker (MDEV-26164, MDEV-26047)
- Crash with union of my_decimal type in ORDER BY clause (MDEV-25994)
- SIGSEGV in st_join_table::cleanup (MDEV-24560)
- Assertion `!eliminated' failed in Item_subselect::exec (MDEV-28437)

Spider

- spider_crd_type and spider_crd_weight have been deprecated (MDEV-28010)

General

- Auto-create history partitions for system-versioned tables (MDEV-17554)
- mariadb-dump `--order-by-size` option (MDEV-28074)
- Server error messages are now available in Chinese (MDEV-28227)
- For RHEL/CentOS 7, non x86_64 architectures are no longer supported upstream and so our support will also be dropped with this release
- Packages for Ubuntu 22.04 LTS "Jammy" and Fedora 36 are now available in this release
- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.5 for Debian 9 "Stretch", Ubuntu 21.10 "Impish", and Fedora 34

Changelog

For a complete list of changes made in MariaDB 10.9.1, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.9.1, see the MariaDB Foundation release announcement.

Do not use non-stable (non-GA) releases in production!

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.3.2.9 MariaDB 10.9.0 Release Notes

The most recent release of MariaDB 10.9 is:
MariaDB 10.9.8 Stable (GA) Download Now

Download 10.9.0 | Release Notes | Changelog | Overview of 10.9

Release date: 23 March 2022

Do not use *alpha* releases in production!

MariaDB 10.9 is a current development series of MariaDB. It is an evolution of MariaDB 10.8 with several entirely new features.

MariaDB 10.9.0 is not a single release, but is instead a number of preview releases based on feature branches. Each should be considered ***Alpha***. Read more about feature preview releases here.

Thanks, and enjoy MariaDB!

List of packages

Remember, these features are in separate *preview packages*. The subsection header text corresponds to the preview package name.

JSON

- JSON_OVERLAPS function (MDEV-27677)
- Implement range notation for JSONPath (MDEV-27911)
- Support JSONPath negative index (MDEV-22224)

SHOW ANALYZE FORMAT=JSON

- Extend SHOW EXPLAIN to support SHOW ANALYZE [FORMAT=JSON] (MDEV-27021)
- Add EXPLAIN FOR CONNECTION syntax support to SHOW EXPLAIN (MDEV-10000)

Async redo log write

- Asynchronous redo log write (MDEV-26603) (not included in MariaDB 10.9.1)

Miscellaneous

- Implement the --do-domain-ids, --ignore-domain-ids, and --ignore-server-ids options for mariadb-binlog/mysqlbinlog (MDEV-20119)
- information_schema.tables.table_type now shows TEMPORARY for local temporary tables (MDEV-12459) (not included in MariaDB 10.9.1)
- Merge old to old_mode sql variable (MDEV-24920)
- Hashicorp Key Management Plugin for implementing encryption using keys stored in the Hashicorp Vault KMS (MDEV-19281)
- JSON file interface to wsrep node state / SST progress logging (MDEV-26971)
- Allow innodb_log_file_size to change without server restart (MDEV-27812)

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.4 MariaDB Server 10.8

Changes and Improvements in MariaDB 10.8

Current Version: 10.8.8 | Status: Stable (GA) | Release Date: 10 May 2023

Release Notes - MariaDB 10.8 Series

MariaDB 10.8 Series Release Notes

Changelogs - MariaDB 10.8 Series

MariaDB 10.8 changelogs

7.0.4.1 Changes and Improvements in MariaDB 10.8

MariaDB 10.8 is no longer maintained. Please use a more recent release.

The most recent release of MariaDB 10.8 is:
MariaDB 10.8.8 Stable (GA) Download Now

Contents

MariaDB 10.8 is a previous short-term maintenance series. The first stable release was in May 2022, and it was maintained for one year.

Upgrading

- See Upgrading Between Major MariaDB Versions and Upgrading from MariaDB 10.7 to MariaDB 10.8.

New Features & Improvements

Stored Procedures INOUT Parameters

- Stored procedures already have support for the IN, OUT and INOUT parameter qualifiers. Added as well for stored functions and (IN only) cursors (MDEV-10654). This was a contribution by ManoharKB.

Lag free ALTER TABLE in replication

- Normally, ALTER TABLE gets fully executed on the primary first and only then it is replicated and starts executing on replicas. With this feature `ALTER TABLE` gets replicated and starts executing on replicas when it *starts* executing on

the primary, not when it *finishes*. This way the replication lag caused by a heavy `ALTER TABLE` can be completely eliminated (MDEV-11675).

Descending indexes

- Individual columns in the index can now be explicitly sorted in the ascending or descending order. This can be useful for optimizing certain ORDER BY cases (MDEV-13756, MDEV-26938, MDEV-26939, MDEV-26996).

InnoDB redo log improvements

- autosize innodb_buffer_pool_chunk_size (MDEV-25342).
- Improve the redo log for concurrency (MDEV-14425).
- Remove FIL_PAGE_FILE_FLUSH_LSN (MDEV-27199).

JSON Histograms

- Histograms in the statistics tables are more precise and stored as JSON, not binary (MDEV-21130, MDEV-26519, blog post).

Spider Storage Engine Improvements

- This was mostly internal refactoring work. As a result one can now declare Spider connections using the `REMOTE_SERVER`, `REMOTE_DATABASE`, and `REMOTE_TABLE` attributes and not abuse the `COMMENT` field for that. This works both for the whole table and per partition (MDEV-5271, MDEV-27106).

Misc. features

- Add an optional argument to the CRC32() function, as well as the CRC32C() function, which uses the Castagnoli polynomial. (MDEV-27208). **Note:** The order of the 2-ary arguments was swapped after the preview release: `crc32('MariaDB')=crc32(crc32('Maria'),'DB')`
- Deprecate the keep_files_on_create variable (MDEV-23570).
- my_print_defaults now handles `--default-*` options in exactly the same way as other MariaDB tools (MDEV-26238).
- UCA collations are now notably faster (MDEV-27266, MDEV-27265).

mysqlbinlog GTID support

- mariadb-binlog (or `mysqlbinlog` as it was called back when the task was created) now supports both filtering events by GTID ranges through `--start-position` and `--stop-position`, and validating a binary log's ordering of GTIDs through `--gtid-strict-mode` (MDEV-4989).

Windows - Improved i18n support

- On newer versions of Windows (Windows 10 1903 or later), the `mariadb` client defaults to the utf8mb4 character set. Several problems with Unicode input and output in client were fixed. Command line utilities now accept all Unicode characters in user names, database names, file names etc (in the past, characters were restricted to the current ANSI codepage).

Variables

- For a list of all new variables, see System Variables Added in MariaDB 10.8.

Security Vulnerabilities Fixed in MariaDB 10.8

> For a complete list of security vulnerabilities (CVEs) fixed across all versions of MariaDB, see the Security Vulnerabilities Fixed in MariaDB page.

- CVE-2023-5157: MariaDB 10.8.4
- CVE-2022-47015: MariaDB 10.8.8
- CVE-2022-38791: MariaDB 10.8.4
- CVE-2022-32091: MariaDB 10.8.4

- CVE-2022-32089: MariaDB 10.8.4
- CVE-2022-32084: MariaDB 10.8.4
- CVE-2022-32082: MariaDB 10.8.4
- CVE-2022-32081: MariaDB 10.8.4
- CVE-2022-24052: MariaDB 10.8.1
- CVE-2022-24051: MariaDB 10.8.1
- CVE-2022-24050: MariaDB 10.8.1
- CVE-2022-24048: MariaDB 10.8.1
- CVE-2021-46659: MariaDB 10.8.1
- CVE-2018-25032: MariaDB 10.8.4

List of All MariaDB 10.8 Releases

Date	Release	Status	Release Notes	Changelog
10 May 2023	**MariaDB 10.8.8**	Stable (GA)	Release Notes	Changelog
6 Feb 2023	**MariaDB 10.8.7**	Stable (GA)	Release Notes	Changelog
7 Nov 2022	**MariaDB 10.8.6**	Stable (GA)	Release Notes	Changelog
19 Sep 2022	**MariaDB 10.8.5**	Stable (GA)	Release Notes	Changelog
15 Aug 2022	**MariaDB 10.8.4**	Stable (GA)	Release Notes	Changelog
20 May 2022	**MariaDB 10.8.3**	Stable (GA)	Release Notes	Changelog
12 Feb 2022	**MariaDB 10.8.2**	RC	Release Notes	Changelog
09 Feb 2022	**MariaDB 10.8.1**	RC	Release Notes	Changelog
21 Dec 2021	**MariaDB 10.8.0**	Alpha	Release Notes	

7.0.5 MariaDB Server 10.7

Changes and Improvements in MariaDB 10.7
Current Version: 10.7.8 | Status: Stable (GA) | Release Date: 6 Feb 2023

Release Notes - MariaDB 10.7 Series
MariaDB 10.7 Series Release Notes

Changelogs - MariaDB 10.7 Series
MariaDB 10.7 changelogs

7.0.5.1 Changes and Improvements in MariaDB 10.7

MariaDB 10.7 is no longer maintained. Please use a more recent release.

The most recent release of MariaDB 10.7 is:
MariaDB 10.7.8 Stable (GA) Download Now

Contents

MariaDB 10.7 is a previous short-term maintenance stable series. The first stable release was in February 2022, and it was maintained for one year.

Upgrading

- See Upgrading Between Major MariaDB Versions and Upgrading from MariaDB 10.6 to MariaDB 10.7.

New Features & Improvements

UUID

- New UUID data type (MDEV-4958)

JSON

- JSON_EQUALS function to check for equality between JSON objects (MDEV-23143).
- JSON_NORMALIZE function, which recursively sorts keys and removes spaces (MDEV-16375)

Natural Sort

- NATURAL_SORT_KEY function (MDEV-4742).

Optimization

- Improve simple multibyte collation performance on the ASCII range (MDEV-26572).

Provider Plugins

- Five **provider plugins** (bzip2, lzma, lz4, lzo, snappy) provide compression capabilities to the server and storage engines (MDEV-12933, blog post).

SFORMAT

- SFORMAT function for arbitrary text formatting (MDEV-25015)

mariadb-dump

- Add option to dump system versioned table as of specified timestamp (MDEV-16355).

Convert Partitions

- ALTER TABLE ... CONVERT PARTITION .. TO TABLE (MDEV-22166), and
- ALTER TABLE ... CONVERT TABLE ... TO PARTITION ... (MDEV-22165) as an easy way to convert tables to

partitions and back in one command, instead of a sequence of CREATE/EXCHANGE/DROP

Password Reuse

- password_reuse_check plugin is a new password validation plugin that prevents the new password from being the same as the one being used during the configurable retention period. (MDEV-9245)

Replication

- Multi-source replication supports MySQL-style CHANNEL syntax (MDEV-26307)

InnoDB Bulk Insert

- In bulk insert, pre-sort and build indexes one page at a time (MDEV-24621)

Diagnostics

- GET DIAGNOSTICS supports a new condition property name `ROW_NUMBER`. In multi-row inserts it allows one to retrieve a number of a row that has caused the error (MDEV-10075, MDEV-26611)

Variables

The following deprecated variables have been removed :

- wsrep_replicate_myisam (MDEV-24947)
- wsrep_strict_ddl (MDEV-24843)

Security Vulnerabilities Fixed in MariaDB 10.7

> For a complete list of security vulnerabilities (CVEs) fixed across all versions of MariaDB, see the Security Vulnerabilities Fixed in MariaDB page.

- CVE-2023-5157: MariaDB 10.7.5
- CVE-2022-38791: MariaDB 10.7.5
- CVE-2022-32091: MariaDB 10.7.5
- CVE-2022-32089: MariaDB 10.7.5
- CVE-2022-32088: MariaDB 10.7.4
- CVE-2022-32087: MariaDB 10.7.4
- CVE-2022-32086: MariaDB 10.7.4
- CVE-2022-32085: MariaDB 10.7.4
- CVE-2022-32084: MariaDB 10.7.5
- CVE-2022-32083: MariaDB 10.7.4
- CVE-2022-32082: MariaDB 10.7.5
- CVE-2022-32081: MariaDB 10.7.5
- CVE-2022-27458: MariaDB 10.7.4
- CVE-2022-27457: MariaDB 10.7.4
- CVE-2022-27456: MariaDB 10.7.4
- CVE-2022-27455: MariaDB 10.7.4
- CVE-2022-27452: MariaDB 10.7.4
- CVE-2022-27451: MariaDB 10.7.4
- CVE-2022-27449: MariaDB 10.7.4
- CVE-2022-27448: MariaDB 10.7.4
- CVE-2022-27447: MariaDB 10.7.4
- CVE-2022-27446: MariaDB 10.7.4
- CVE-2022-27445: MariaDB 10.7.4
- CVE-2022-27444: MariaDB 10.7.4
- CVE-2022-27387: MariaDB 10.7.4
- CVE-2022-27386: MariaDB 10.7.4
- CVE-2022-27384: MariaDB 10.7.4
- CVE-2022-27383: MariaDB 10.7.4
- CVE-2022-27382: MariaDB 10.7.4
- CVE-2022-27381: MariaDB 10.7.4
- CVE-2022-27380: MariaDB 10.7.4

- CVE-2022-27379: MariaDB 10.7.4
- CVE-2022-27378: MariaDB 10.7.4
- CVE-2022-27377: MariaDB 10.7.4
- CVE-2022-27376: MariaDB 10.7.4
- CVE-2022-24052: MariaDB 10.7.2
- CVE-2022-24051: MariaDB 10.7.2
- CVE-2022-24050: MariaDB 10.7.2
- CVE-2022-24048: MariaDB 10.7.2
- CVE-2022-21595: MariaDB 10.7.2
- CVE-2022-0778: MariaDB 10.7.2
- CVE-2021-46669: MariaDB 10.7.4
- CVE-2021-46668: MariaDB 10.7.3
- CVE-2021-46665: MariaDB 10.7.3
- CVE-2021-46664: MariaDB 10.7.3
- CVE-2021-46663: MariaDB 10.7.3
- CVE-2021-46661: MariaDB 10.7.3
- CVE-2021-46659: MariaDB 10.7.2
- CVE-2018-25032: MariaDB 10.7.5

List of All MariaDB 10.7 Releases

Date	Release	Status	Release Notes	Changelog
6 Feb 2023	**MariaDB 10.7.8**	Stable (GA)	Release Notes	Changelog
7 Nov 2022	**MariaDB 10.7.7**	Stable (GA)	Release Notes	Changelog
19 Sep 2022	**MariaDB 10.7.6**	Stable (GA)	Release Notes	Changelog
15 Aug 2022	**MariaDB 10.7.5**	Stable (GA)	Release Notes	Changelog
20 May 2022	**MariaDB 10.7.4**	Stable (GA)	Release Notes	Changelog
12 Feb 2022	**MariaDB 10.7.3**	Stable (GA)	Release Notes	Changelog
9 Feb 2022	**MariaDB 10.7.2**	Stable (GA)	Release Notes	Changelog
8 Nov 2021	**MariaDB 10.7.1**	RC	Release Notes	Changelog
17 Sep 2021	**MariaDB 10.7.0**	Alpha	Release Notes	

7.0.6 MariaDB Server 10.6

Changes and Improvements in MariaDB 10.6

Current Version: 10.6.16 | Status: Stable (GA) | Release Date: 13 Nov 2023

Release Notes - MariaDB 10.6 Series

MariaDB 10.6 Series Release Notes

Changelogs - MariaDB 10.6 Series

MariaDB 10.6 changelogs

7.0.6.1 Changes and Improvements in MariaDB 10.6

The most recent release of MariaDB 10.6 is:

MariaDB 10.6.16 Stable (GA) Download Now

Alternate download from mariadb.org

Contents

MariaDB 10.6 is the current long-term maintenance stable version. The first stable release was in July 2021, and it will be maintained until July 2026.

Upgrading

- See Upgrading Between Major MariaDB Versions and Upgrading from MariaDB 10.5 to MariaDB 10.6.

New Features & Improvements

Atomic DDL

- CREATE TABLE, ALTER TABLE, RENAME TABLE, DROP TABLE, DROP DATABASE and related DDL statements are now atomic. Either the statement is fully completed, or everything is reverted to it's original state. Note that when deleting multiple tables with DROP TABLE, only each individual drop is atomic, not the full list of tables). (MDEV-23842).

SQL Syntax

- Implement SQL-standard SELECT ... OFFSET ... FETCH (MDEV-23908)
- Add SELECT ... SKIP LOCKED syntax (InnoDB only) (MDEV-13115)
- Indexes can be ignored (MDEV-7317, MDEV-25075)
- JSON_TABLE, used to extract JSON data based on a JSON path expression and to return it as a relational table (MDEV-17399)

Oracle Compatibility

- Anonymous subqueries in a FROM clause (no AS clause) are permitted in ORACLE mode (MDEV-19162)
- ADD_MONTHS() added (MDEV-20025)
- TO_CHAR() added (MDEV-20017)
- SYS_GUID() added (MDEV-24285)
- MINUS is mapped to EXCEPT in UNION (MDEV-20021)
- ROWNUM function returns the current number of accepted rows in the current context (MDEV-24089)

InnoDB

- Optimization to speed up inserts into an empty table (MDEV-515)
- We intended to deprecate and eventually remove the InnoDB's COMPRESSED row format. The first step was to make the tables read-only by default, but this plan was abandoned from MariaDB 10.6.6 (MDEV-23497) (MDEV-27736)
- Information Schema SYS_TABLESPACES now directly reflects the filesystem, and SYS_DATAFILES has been removed (MDEV-22343)
- Defer writes to the InnoDB temporary tablespace (MDEV-12227)
- The old MariaDB 5.5-compatible `innodb` checksum is no longer supported, only `crc32`. Removed the `*innodb` and `*none` options from innodb_checksum_algorithm, and the `--strict-check` / `-C` and `--write` / `-w` options from innochecksum (MDEV-25105)

Replication, Galera and Binlog

- Increase master_host limit to 255, user to 128 (MDEV-24312)
- The wsrep_mode system variable, for turning on WSREP features which are not part of default behavior (including the experimental Aria replication) (MDEV-20008, MDEV-20715, MDEV-24946)
- The delay between binary log purges can now be specified with much greater precision. The system variable binlog_expire_logs_seconds is introduced as a form of alias for expire_logs_days, which now accepts a precision of 1/1000000 days (MDEV-19371)
- Allow transition from unencrypted to TLS Galera cluster communication without cluster downtime (MDEV-22131)

Sys Schema

- Bundle sys-schema, a collection of views, functions and procedures to help administrators get insight into database usage. (MDEV-9077)

Performance Schema

- Merged replication instrumentation and tables (MDEV-16437, MDEV-20220)

Information Schema

- The views INFORMATION_SCHEMA.KEYWORDS and INFORMATION_SCHEMA.SQL_FUNCTIONS have been added to the information schema (MDEV-25129)

Storage Engines

- TokuDB has been removed (MDEV-19780)
- CassandraSE has been removed (MDEV-23024)

Character Sets

- The `utf8` character set (and related collations) is now by default an alias for `utf8mb3` rather than the other way around. It can be set to imply `utf8mb4` by changing the value of the old_mode system variable (MDEV-8334)

General

- Bundle sys schema (MDEV-9077)
- Do not resend unchanged resultset metadata for prepared statements (MDEV-19237)
- --bind-address=hostname now listens on both IPv6 and IPv4 addresses (MDEV-6536)
- Support systemd socket activation (MDEV-5536)
- For the GSSAPI plugin, support AD or local group name, and SIDs on Windows (MDEV-23959)
- Check for $MARIADB_HOME/my.cnf (MDEV-21365)

Variables

- For a list of all new variables, see System Variables Added in MariaDB 10.6 and Status Variables Added in MariaDB 10.6.
- max_recursive_iterations has been reduced to 1000 (MDEV-17239)

InnoDB Variables

The following deprecated variables have been removed (MDEV-23397):

- innodb_adaptive_max_sleep_delay
- innodb_background_scrub_data_check_interval
- innodb_background_scrub_data_compressed
- innodb_background_scrub_data_interval
- innodb_background_scrub_data_uncompressed
- innodb_buffer_pool_instances
- innodb_commit_concurrency
- innodb_concurrency_tickets
- innodb_file_format
- innodb_large_prefix
- innodb_lock_schedule_algorithm

- innodb_log_checksums
- innodb_log_compressed_pages
- innodb_log_files_in_group
- innodb_log_optimize_ddl
- innodb_page_cleaners
- innodb_replication_delay
- innodb_scrub_log
- innodb_scrub_log_speed
- innodb_sync_array_size
- innodb_thread_concurrency
- innodb_thread_sleep_delay
- innodb_undo_logs

Security Vulnerabilities Fixed in MariaDB 10.6

For a complete list of security vulnerabilities (CVEs) fixed across all versions of MariaDB, see the Security Vulnerabilities Fixed in MariaDB page.

- CVE-2023-5157: MariaDB 10.6.9
- CVE-2023-22084: MariaDB 10.6.16
- CVE-2022-47015: MariaDB 10.6.13
- CVE-2022-38791: MariaDB 10.6.9
- CVE-2022-32091: MariaDB 10.6.9
- CVE-2022-32089: MariaDB 10.6.9
- CVE-2022-32088: MariaDB 10.6.8
- CVE-2022-32087: MariaDB 10.6.8
- CVE-2022-32086: MariaDB 10.6.8
- CVE-2022-32085: MariaDB 10.6.8
- CVE-2022-32084: MariaDB 10.6.9
- CVE-2022-32083: MariaDB 10.6.8
- CVE-2022-32082: MariaDB 10.6.9
- CVE-2022-32081: MariaDB 10.6.9
- CVE-2022-31624: MariaDB 10.6.5
- CVE-2022-27458: MariaDB 10.6.8
- CVE-2022-27457: MariaDB 10.6.8
- CVE-2022-27456: MariaDB 10.6.8
- CVE-2022-27455: MariaDB 10.6.8
- CVE-2022-27452: MariaDB 10.6.8
- CVE-2022-27451: MariaDB 10.6.8
- CVE-2022-27449: MariaDB 10.6.8
- CVE-2022-27448: MariaDB 10.6.8
- CVE-2022-27447: MariaDB 10.6.8
- CVE-2022-27446: MariaDB 10.6.8
- CVE-2022-27445: MariaDB 10.6.8
- CVE-2022-27444: MariaDB 10.6.8
- CVE-2022-27387: MariaDB 10.6.8
- CVE-2022-27386: MariaDB 10.6.8
- CVE-2022-27385: MariaDB 10.6.5
- CVE-2022-27384: MariaDB 10.6.8
- CVE-2022-27383: MariaDB 10.6.8
- CVE-2022-27382: MariaDB 10.6.8
- CVE-2022-27381: MariaDB 10.6.8
- CVE-2022-27380: MariaDB 10.6.8
- CVE-2022-27379: MariaDB 10.6.8
- CVE-2022-27378: MariaDB 10.6.8
- CVE-2022-27377: MariaDB 10.6.8
- CVE-2022-27376: MariaDB 10.6.8
- CVE-2022-24052: MariaDB 10.6.6
- CVE-2022-24051: MariaDB 10.6.6
- CVE-2022-24050: MariaDB 10.6.6
- CVE-2022-24048: MariaDB 10.6.6
- CVE-2022-21595: MariaDB 10.6.6
- CVE-2022-0778: MariaDB 10.6.6
- CVE-2021-46669: MariaDB 10.6.8
- CVE-2021-46668: MariaDB 10.6.7

- CVE-2021-46667: MariaDB 10.6.5
- CVE-2021-46665: MariaDB 10.6.7
- CVE-2021-46664: MariaDB 10.6.7
- CVE-2021-46663: MariaDB 10.6.7
- CVE-2021-46662: MariaDB 10.6.5
- CVE-2021-46661: MariaDB 10.6.7
- CVE-2021-46659: MariaDB 10.6.6
- CVE-2021-46658: MariaDB 10.6.3
- CVE-2021-35604: MariaDB 10.6.3
- CVE-2021-2389: MariaDB 10.6.4 [2]
- CVE-2021-2372: MariaDB 10.6.4 [2]
- CVE-2018-25032: MariaDB 10.6.9

List of All MariaDB 10.6 Releases

Date	Release	Status	Release Notes	Changelog
13 Nov 2023	**MariaDB 10.6.16**	Stable (GA)	Release Notes	Changelog
14 Aug 2023	**MariaDB 10.6.15**	Stable (GA)	Release Notes	Changelog
7 Jun 2023	**MariaDB 10.6.14**	Stable (GA)	Release Notes	Changelog
10 May 2023	**MariaDB 10.6.13**	Stable (GA)	Release Notes	Changelog
6 Feb 2023	**MariaDB 10.6.12**	Stable (GA)	Release Notes	Changelog
7 Nov 2022	**MariaDB 10.6.11**	Stable (GA)	Release Notes	Changelog
19 Sep 2022	**MariaDB 10.6.10**	Stable (GA)	Release Notes	Changelog
15 Aug 2022	**MariaDB 10.6.9**	Stable (GA)	Release Notes	Changelog
20 May 2022	**MariaDB 10.6.8**	Stable (GA)	Release Notes	Changelog
12 Feb 2022	**MariaDB 10.6.7**	Stable (GA)	Release Notes	Changelog
9 Feb 2022	**MariaDB 10.6.6**	Stable (GA)	Release Notes	Changelog
8 Nov 2021	**MariaDB 10.6.5**	Stable (GA)	Release Notes	Changelog
6 Aug 2021	**MariaDB 10.6.4**	Stable (GA)	Release Notes	Changelog
6 Jul 2021	**MariaDB 10.6.3**	Stable (GA)	Release Notes	Changelog
18 Jun 2021	**MariaDB 10.6.2**	RC	Release Notes	Changelog
21 May 2021	**MariaDB 10.6.1**	Beta	Release Notes	Changelog
26 Apr 2021	**MariaDB 10.6.0**	Alpha	Release Notes	Changelog

7.0.6.2 Release Notes - MariaDB 10.6 Series

MariaDB 10.6.16 Release Notes
Status: Stable (GA) | Release Date: 13 Nov 2023

MariaDB 10.6.15 Release Notes
Status: Stable (GA) | Release Date: 14 Aug 2023

MariaDB 10.6.14 Release Notes
Status: Stable (GA) | Release Date: 7 Jun 2023

MariaDB 10.6.13 Release Notes
Status: Stable (GA) | Release Date: 10 May 2023

MariaDB 10.6.12 Release Notes
Status: Stable (GA) | Release Date: 6 Feb 2023

MariaDB 10.6.11 Release Notes
Status: Stable (GA) | Release Date: 7 Nov 2022

MariaDB 10.6.10 Release Notes
Status: Stable (GA) | Release Date: 19 Sep 2022

MariaDB 10.6.9 Release Notes
Status: Stable (GA) | Release Date: 15 Aug 2022

MariaDB 10.6.8 Release Notes
Status: Stable (GA) | Release Date: 20 May 2022

MariaDB 10.6.7 Release Notes
Status: Stable (GA) | Release Date: 12 Feb 2022

MariaDB 10.6.6 Release Notes
Status: Stable (GA) | Release Date: 9 Feb 2022

MariaDB 10.6.5 Release Notes
Status: Stable (GA) | Release Date: 8 Nov 2021

MariaDB 10.6.4 Release Notes
Status: Stable (GA) | Release Date: 6 Aug 2021

MariaDB 10.6.3 Release Notes
Status: Stable (GA) | Release Date: 6 Jul 2021

MariaDB 10.6.2 Release Notes
Status: RC | Release Date: 18 Jun 2021

MariaDB 10.6.1 Release Notes
Status: Beta | Release Date: 21 May 2021

MariaDB 10.6.0 Release Notes
Status: Alpha | Release Date: 26 Apr 2021

7.0.6.2.1 MariaDB 10.6.15 Release Notes

The most recent release of MariaDB 10.6 is:
MariaDB 10.6.16 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.6.15 | Release Notes | Changelog | Overview of 10.6

Release date: 14 Aug 2023

MariaDB 10.6 is a previous long-term series of MariaDB, maintained until July 2026. It is an evolution of MariaDB 10.5 with several entirely new features.

MariaDB 10.6.15 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.6 see the What is MariaDB 10.6? page.

Thanks, and enjoy MariaDB!

Notable Items

Upgrading from MySQL

- MariaDB now detects and converts previously incompatible MySQL partition schemes (MDEV-29253)

General

- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.6 for Ubuntu 18.04 LTS "Bionic" and Ubuntu 22.10 "Kinetic"
- mariadb-dump --force doesn't ignore error as it should (MDEV-31092)
- 280 Bytes lost in mysys/array.c, mysys/hash.c, sql/sp.cc, sql/sp.cc, sql/item_create.cc, sql/item_create.cc, sql/sql_yacc.yy:10748 when using oracle sql_mode (MDEV-26186)
- SQL/PL package body does not appear in `I_S.ROUTINES.ROUTINE_DEFINITION` (MDEV-30662)
- Unexpected result when combining DISTINCT, subselect and LIMIT (MDEV-28285)
- ROW variables do not get assigned from subselects (MDEV-31250)
- Crash after setting global session_track_system_variables to an invalid value (MDEV-25237)
- ODKU of non-versioning column inserts history row (MDEV-23100)
- UPDATE not working properly on transaction precise system versioned table (MDEV-25644)
- Assertion `` `const_item_cache == true' `` failed in `Item_func::fix_fields` (MDEV-31319)
- ANALYZE doesn't work with pushed derived tables (MDEV-29284)
- `get_partition_set` is never executed in `ha_partition::multi_range_key_create_key` due to bitwise & with 0 constant (MDEV-24712)
- Client can crash the server with a `mysql_list_fields("view")` call (MDEV-30159)
- `I_S.parameters` not immediatly changed updated after procedure change (MDEV-31064)
- bzero wipes more bytes than necessary in `set_global_from_ddl_log_entry` (MDEV-31521)
- Assertion `` `0' `` failed in `Type_handler_row::field_type` upon `TO_CHAR` with wrong argument (MDEV-29152)
- mysql_upgrade fails due to `old_mode=""` , with "Cannot load from mysql.proc. The table is probably corrupted" (MDEV-28915)

Character Sets, Data Types

- UBSAN: null pointer passed as argument 1, which is declared to never be null in `my_strnncoll_binary` on `SELECT ... COUNT` or `GROUP_CONCAT` (MDEV-28384)
- Possibly wrong result or Assertion `0' failed in `Item_func_round::native_op` (MDEV-23838)
- Assertion `` `(length % 4) == 0' `` failed in `my_lengthsp_utf32` on `SELECT` (MDEV-29019)
- UBSAN: negation of -X cannot be represented in type `'long long int'` ; cast to an unsigned type to negate this value to itself in `Item_func_mul::int_op` and `Item_func_round::int_op` (MDEV-30932)
- Assorted assertion failures in json_find_path with certain collations (MDEV-23187)

InnoDB

- Crashing on I/O error is unhelpful (MDEV-27593)
- SIGSEGV in `log_sort_flush_list()` in InnoDB crash recovery (MDEV-31354)
- InnoDB tables are being flagged as corrupted on an I/O bound server (MDEV-31767)
- Deadlock with 3 concurrent DELETEs by unique key (MDEV-10962)
- Server Status Innodb_row_lock_time% is reported in seconds (MDEV-29311)
- innochecksum dies with Floating point exception (MDEV-31641)
- Add InnoDB engine information to the slow query log (MDEV-31558)
- innodb protection against dual processes accessing data insufficient (MDEV-31568)
- ER_DUP_KEY in `mysql.innodb_table_stats` upon RENAME on sequence (MDEV-31607)
- Assertion `` `!strcmp(index->table->name.m_name, "SYS_FOREIGN") || !strcmp(index->table->name.m_name, "SYS_FOREIGN_COLS")' `` failed in `btr_node_ptr_max_size` (MDEV-19216)
- InnoDB: Failing assertion: `page_type == i_s_page_type[page_type].type_value` (MDEV-31386)
- `btr_estimate_n_rows_in_range()` accesses unfixed, unlatched page (MDEV-30648)
- `MODIFY COLUMN` can break FK constraints, and lead to unrestorable dumps (MDEV-31086)
- Recovery or backup failure after innodb_undo_log_truncate=ON (MDEV-31487)
- Assertion `'n & PENDING'` failed in `fil_space_t::set_needs_flush()` (MDEV-31442)
- `fil_node_open_file()` releases `fil_system.mutex` allowing other thread to open its file node (MDEV-31256)
- ASAN errors in `dict_v_col_t::detach` upon adding key to virtual column (MDEV-31416)
- Purge trying to access freed secondary index page (MDEV-31264)
- Freed data pages are not always being scrubbed (MDEV-31253)
- InnoDB recovery hangs after reporting corruption (MDEV-31353)
- `!cursor->index->is_committed()` in row0ins.cc after update to 10.4.13 from 10.3.21 (MDEV-22739)
- innodb_undo_log_truncate=ON fails to wait for purge of enough transaction history (MDEV-31355)
- SET GLOBAL innodb_undo_log_truncate=ON does not free space when no undo logs exist (MDEV-31382)
- innodb_read_ahead_threshold (linear read-ahead) does not work (MDEV-29967)
- InnoDB recovery and `mariadb-backup --prepare` fail to report detailed progress (MDEV-29911)
- `fil_ibd_create()` may hijack the file handle of an old file (MDEV-31347)
- innodb_undo_log_truncate=ON recovery results in a corrupted undo log (MDEV-31373)

- Server freeze due to innodb_change_buffering and innodb_file_per_table=0 (MDEV-31088)
- Change buffer entries are left behind when freeing a page, causing secondary index corruption when the page is later reused (MDEV-31385)
- Foreign Key Constraint actions don't affect Virtual Column (MDEV-18114)

Aria

- Various crashes upon INSERT/UPDATE after changing Aria settings (MDEV-28054)
- Various crashes/asserts/corruptions when Aria encryption is enabled/used, but the encryption plugin is not loaded (MDEV-26258)

Spider

- SIGSEGV in `spider_db_open_item_field` and SIGSEGV in `spider_db_print_item_type`, on SELECT (MDEV-29447)
- Spider variables that double as table params overriding mechanism is buggy (MDEV-31524)

Optimizer

- ANALYZE FORMAT=JSON now includes InnoDB engine statistics for each table (MDEV-31577)
- Assertion `` `last_key_entry >= end_pos' `` failed in virtual bool `JOIN_CACHE_HASHED::put_record()` (MDEV-31348)
- Problem with open ranges on prefix blobs keys (MDEV-31800)
- Equal on two RANK window functions create wrong result (MDEV-20010)
- Recursive CTE execution is interrupted without errors or warnings (MDEV-31214)
- Assertion `` `s->table->opt_range_condition_rows <= s->found_records' `` failed in `apply_selectivity_for_table` (MDEV-31449)
- Inconsistency between MRR and SQL layer costs can cause poor query plan (MDEV-31479)
- `MAX_SEL_ARG` memory exhaustion is not visible in the optimizer trace (MDEV-30964)
- SHOW TABLES not working properly with `lower_case_table_names=2` (MDEV-30765)
- Segfault on select query using index for group-by and filesort (MDEV-30143)
- Server crash in `store_length`, assertion failure in `Type_handler_string_result::sort_length` (MDEV-31743)

Replication

- Parallel Slave SQL Thread Can Update Seconds_Behind_Master with Active Workers (MDEV-30619)
- ALTER SEQUENCE ends up in optimistic parallel slave binlog out-of-order (MDEV-31503)
- STOP SLAVE takes very long time on a busy system (MDEV-13915)
- On slave XA COMMIT/XA ROLLBACK fail to return an error in read-only mode (MDEV-30978)
- Calling a function from a different database in a slave side trigger crashes (MDEV-29894)
- `rpl.rpl_manual_change_index_file` occasionally fails in BB with Result length mismatch (MDEV-30214)

Galera

- Node never returns from Donor/Desynced to Synced when `wsrep_mode = BF_ABORT_MARIABACKUP` (MDEV-31737)
- Node has been dropped from the cluster on Startup / Shutdown with async replica (MDEV-31413)
- KILL QUERY maintains nodes data consistency but breaks GTID sequence (MDEV-31075)
- Assertion failure `!lock.was_chosen_as_deadlock_victim in trx0trx.h:1065` (MDEV-30963)
- MariaDB stuck on starting commit state (waiting on commit order critical section) (MDEV-29293)
- Assertion `state() == s_aborting || state() == s_must_replay` failed in int wsrep::transaction::after_rollback() (MDEV-30013)
- Assertion `!wsrep_has_changes(thd) || (thd->lex->sql_command == SQLCOM_CREATE_TABLE && !thd->is_current_stmt_binlog_format_row()) || thd->wsrep_cs().transaction().state() == wsrep::transaction::s_aborted` failed (MDEV-30388)
- Server crashes when wsrep_sst_donor and wsrep_cluster_address set to NULL (MDEV-28433)
- Create temporary sequence can cause inconsistency (MDEV-31335)
- Galera 4 unable to query cluster state if not primary component (MDEV-21479)

Changelog

For a complete list of changes and bugfixes made in MariaDB 10.6.15, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.6.15, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.6.2.2 MariaDB 10.6.14 Release Notes

The most recent release of MariaDB 10.6 is:
MariaDB 10.6.16 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.6.14 | Release Notes | Changelog | Overview of 10.6

Release date: 7 Jun 2023

Warning: This version can cause InnoDB file corruption under high I/O-bound load. Stick to an earlier release, or upgrade to a more recent release, if possible. See MDEV-31767.

MariaDB 10.6 is the current long-term series of MariaDB, maintained until July 2026. It is an evolution of MariaDB 10.5 with several entirely new features.

MariaDB 10.6.14 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.6 see the What is MariaDB 10.6? page.

Thanks, and enjoy MariaDB!

Notable Items

InnoDB

- Server crashes in st_join_table::choose_best_splitting (MDEV-31403)
- Crash with condition pushable into derived and containing outer reference (MDEV-31240)
- InnoDB does not free UNDO after the fix of MDEV-30671 (MDEV-31234)
- InnoDB hang fixes (MDEV-31158, MDEV-31343, MDEV-31350)
- Innodb_buffer_pool_read_requests is not updated correctly (MDEV-31309)
- InnoDB monitor trx_rseg_history_len was accidentally disabled by default (MDEV-31308)
- Revert "MDEV-30473: Do not allow GET_LOCK() / RELEASE_LOCK() in cluster"

Optimizer

- Crash with condition pushable into derived and containing outer reference (MDEV-31403 MDEV-31240)
- Crash with EXPLAIN EXTENDED for multi-table update of system table (MDEV-31224)

Changelog

For a complete list of changes and bugfixes made in MariaDB 10.6.14, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.6.14, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.6.2.3 MariaDB 10.6.13 Release Notes

The most recent release of MariaDB 10.6 is:
MariaDB 10.6.16 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.6.13 | Release Notes | Changelog | Overview of 10.6

Release date: 10 May 2023

MariaDB 10.6 is the current long-term series of MariaDB, maintained until July 2026. It is an evolution of MariaDB 10.5 with several entirely new features.

MariaDB 10.6.13 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.6 see the What is MariaDB 10.6? page.

Thanks, and enjoy MariaDB!

Notable Items

InnoDB

- Crash on ROLLBACK in a ROW_FORMAT=COMPRESSED table (MDEV-30882)
- UNIQUE USING HASH accepts duplicate entries for tricky collations (MDEV-30034)
- rec_get_offsets() is not optimal (MDEV-30567)
- Performance regression in fil_space_t::try_to_close() introduced in MDEV-23855 (MDEV-30775)
- InnoDB recovery hangs when buffer pool ran out of memory (MDEV-30551)
- InnoDB undo log truncation fails to wait for purge of history (MDEV-30671)
- MariaDB crash due to DB_FAIL reported for a corrupted page (MDEV-30397)
- Deadlock between INSERT and InnoDB non-persistent statistics update (MDEV-30638)
- InnoDB hang on B-tree split or merge (MDEV-29835)
- Performance regression in locking reads from secondary indexes (MDEV-30357)
- Improve adaptive flushing (MDEV-26055)
- Make page flushing even faster (MDEV-26827)
- Purge misses a chance to free not-yet-reused undo pages (MDEV-29593)
- InnoDB temporary tablespace: reclaiming of free space does not work (MDEV-26782)
- Fix miscount of doublewrites by Innodb_data_written (MDEV-31124)

Backup

- mariadb-backup doesn't utilise innodb-undo-log-directory (if specified as a relative path) during copy-back operation (MDEV-28187)
- mariabackup issues error messages during InnoDB tablespaces export on partial backup preparing (MDEV-29050)
- mariadb-backup does not copy Aria logs if aria_log_dir_path is used (MDEV-30968)
- Race condition between buffer pool flush and log file deletion in mariadb-backup --prepare (MDEV-30860)

Replication

- Fixed a deadlock on parallel slave involving full image Write event on the sequence engine (MDEV-29621)
- Fixed an attempted out-of-order binlogging error on slave involving ALTER on the sequence engine (MDEV-31077)
- Corrected non-versioned master to versioned slave replication on no-unique attribute table (MDEV-30430)
- Mended encrypted binlog master to error out to gtid-mode slave when master could not decrypt a binlog file (MDEV-28798)
- Refined optimistic parallel slave to error-exit without any hang (MDEV-30780)
- Ensured SHOW-SLAVE-STATUS is processed on the parallel slave having a necessary mutex always intialized (MDEV-30620)
- Fixed the slave applier to report a correct error when gtid_slave_pos insert fails for some (engine) reasons (MDEV-31038)
- Made parallel slave reports in performance schema consistent with that of show-slave-status (MDEV-26071)

Optimizer

- Split Materialized optimization is improved to re-fill the materialized table only if necessary. The fewer number of table refills is taken into account when choosing query plan, too (MDEV-26301).
- New optimizer_switch option, hash_join_cardinality , is added. It is off by default. When set to ON, the optimizer will produce tighter bounds for hash join output cardinality. (MDEV-30812)
- Queries using `SELECT DISTINCT some_expression(aggregate_function())` could produce wrong query result. (MDEV-20057)
- ANALYZE FORMAT=JSON now prints more information about Block Nested Loop joins: `block-nl-join` element now has `r_loops` , `r_effective_rows` and `r_other_time_ms` fields (MDEV-30806 , MDEV-30830 , MDEV-30972).
- A GROUP BY query with `MIN(primary_key)` in select list and `primary_key<>const` in the WHERE could produce wrong result when executed with "Using index for group-by" strategy (MDEV-30605)
- EXPLAIN could erroneously report that Rowid Filter optimization is used for partitioned tables. Partitioned tables do not support it. (MDEV-30596)
- A bug in selectivity computations for SINGLE/DOUBLE_PREC_HB histograms could cause wrong estimates to be produced. This could cause the optimizer to pick sub-optimal query plans (MDEV-31067).

Docker Official Image

- Add replication setup to containers contributed by Md Sahil (MDEV-29762)

General

- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.6 for Fedora 36.
- In this release repositories for Fedora 38 and Ubuntu 23.04 Lunar have been added.

Security

- Fixes for the following security vulnerabilities :
  - CVE-2022-47015

Changelog

For a complete list of changes and bugfixes made in MariaDB 10.6.13, with links to detailed information on each push, see the changelog .

Contributors

For a full list of contributors to MariaDB 10.6.13, see the MariaDB Foundation release announcement .

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.6.2.4 MariaDB 10.6.12 Release Notes

Download | Release Notes | Changelog | Overview of 10.6

Release date: 6 Feb 2023

MariaDB 10.6 is the current long-term series of MariaDB, maintained until July 2026. It is an evolution of MariaDB 10.5 with several entirely new features.

MariaDB 10.6.12 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.6 see the What is MariaDB 10.6? page.

Thanks, and enjoy MariaDB!

Notable Items

- As mentioned in the 10.6.11 release notes, our Yum/DNF/Zypper repositories for Red Hat Enterprise Linux, CentOS, Fedora, openSUSE, and SUSE are changing **with this release** to being signed with a new GPG key with SHA2 digest algorithms instead of SHA1. See this blog post and the GPG page for more details.

InnoDB

- Full-text index corruption with system versioning (MDEV-25004)
- innodb_undo_log_truncate=ON recovery and backup fixes (MDEV-29999, MDEV-30179, MDEV-30438)
- Upgrade after a crash is not supported (MDEV-24412)
- Remove InnoDB buffer pool load throttling (MDEV-25417)
- InnoDB shutdown hangs when the change buffer is corrupted (MDEV-30009)
- `innodb_fast_shutdown=0` fails to report change buffer merge progress (MDEV-29984)
- `mariadb-backup --backup --incremental --throttle=...` hangs (MDEV-29896)
- Crash after recovery, with InnoDB: Tried to read (MDEV-30132)
- Trying to write ... bytes at ... outside the bounds (MDEV-30069)
- TRUNCATE breaks FOREIGN KEY locking (MDEV-29504, MDEV-29849)
- `INFORMATION_SCHEMA.INNODB_TABLESPACES_ENCRYPTION.NAME` is NULL for undo tablespaces (MDEV-30119)
- Fixed hangs and error handling in B-tree operations (MDEV-29603, MDEV-30400)

Galera

- Fixes for cluster wide write conflict resolving (MDEV-29684)

Replication

- Parallel slave applying in binlog order is corrected for admin class of commands including ANALYZE (MDEV-30323)
- Seconds_Behind_Master is now shown now more precisely at the slave applier start, including in the delayed mode (MDEV-29639)
- `mysqlbinlog --verbose` is made to show the type of compressed columns (MDEV-25277)
- Deadlock is resolved on replica involving `BACKUP STAGE BLOCK_COMMIT` and a committing user XA (MDEV-30423)

JSON

- JSON_PRETTY added as an alias for JSON_DETAILED (MDEV-19160)

General

- Infinite sequence of recursive calls when processing embedded CTE (MDEV-30248)
- Crash with a query containing nested WINDOW clauses (MDEV-30052)
- Major performance regression with 10.6.11 (MDEV-29988)
- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.6 for Fedora 35.

- In this release repositories for Fedora 37 and Ubuntu 22.10 Kinetic have been added.

Changelog

For a complete list of changes and bugfixes made in MariaDB 10.6.12, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.6.12, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.6.2.5 MariaDB 10.6.11 Release Notes

The most recent release of MariaDB 10.6 is:
MariaDB 10.6.16 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.6.11 | Release Notes | Changelog | Overview of 10.6

Release date: 7 Nov 2022

MariaDB 10.6 is the current long-term maintenance stable series of MariaDB, maintained until July 2026. It is an evolution of MariaDB 10.5 with several entirely new features.

MariaDB 10.6.11 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.6 see the What is MariaDB 10.6? page.

Thanks, and enjoy MariaDB!

Notable Items

SSL

- The server no longer tolerates incorrectly configured SSL (MDEV-29811). If you have enabled SSL in `my.cnf` but have not configured it properly (for example, a certificate file is missing), MariaDB used to silently disable SSL, leaving you under impression that everything was fine and connections were secure. Since this release, MariaDB will fail to start if SSL is enabled, but cannot be switched on.

Backup

- Assertion on info.page_size failed in xb_delta_open_matching_space (MDEV-18589)

InnoDB

- Adaptive hash index MDEV-27700, MDEV-29384
- MVCC and locking (MDEV-29666, MDEV-27927, MDEV-28709, MDEV-29635)
- Virtual columns (MDEV-29299, MDEV-29753)
- InnoDB crash recovery fixes (MDEV-29559)

- Race condition between KILL and transaction commit (MDEV-29368)
- Implement CHECK TABLE…EXTENDED for InnoDB (MDEV-24402)
- InnoDB persistent statistics fail to update after bulk insert (MDEV-28327)

Galera

- Galera updated to 26.4.13
- Galera server crashes after 10.3 > 10.4 upgrade (MDEV-29375)
- wsrep_incoming_addresses status variable prints 0 as port number if the port is not mentioned in wsrep_node_incoming_address system variable (MDEV-28868)

Replication

- XA COMMIT is not binlogged when the XA transaction has not updated any transaction engine (MDEV-25616)
- Concurrent CREATE TRIGGER statements made to binlog without any mixup (MDEV-25606)

Repositories

- Beginning with the next release (Q1 2023), our Yum, DNF, and Zypper repositories for Red Hat, Fedora, and SUSE will be migrated to being signed with a new GPG key. The key we are migrating to is the same one we already use for our Debian and Ubuntu Repositories.
  - The short Key ID is: `0xC74CD1D8`
  - The long Key ID is: `0xF1656F24C74CD1D8`
  - The full fingerprint of the key is: `177F 4010 FE56 CA33 3630 0305 F165 6F24 C74C D1D8`
  - The key can be imported now in preparation for this change using the following command:

```
sudo rpm --import https://supplychain.mariadb.com/MariaDB-Server-GPG-KEY
```

Docker Official Image

The following changes have been made to the `docker.io/library/mariadb` container image.

- The number of gpg packages packages has been removed, leaving enough to `apt-get update`, but `dirmngr` that would fetch keys has been removed. (inspired by issue #469)
- The environment variable `LANG=C.UTF-8` has been added for those that exec into containers and copy paste UTF8 characters (fixes issue #468).
- Adds OCI labels to image (fixes issue 436 and users need for version)
- MariaDB config: skip-host-cache and skip-name-resolve moved to `/etc/mysql/mariadb.conf.d/05-skipcache.cnf`

Changelog

For a complete list of changes and bugfixes made in MariaDB 10.6.11, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.6.11, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.6.2.6 MariaDB 10.6.10 Release Notes

The most recent release of MariaDB 10.6 is:
MariaDB 10.6.16 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.6.10 | Release Notes | Changelog | Overview of 10.6

Release date: 19 Sep 2022

MariaDB 10.6 is the current long-term maintenance stable series of MariaDB, maintained until July 2026. It is an evolution of MariaDB 10.5 with several entirely new features.

MariaDB 10.6.10 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.6 see the What is MariaDB 10.6? page.

Thanks, and enjoy MariaDB!

Notable Items

Issues Fixed

- Assertion mysql_mutex_assert_owner(&log_sys.flush_order_mutex) failed in mtr_t::commit() (MDEV-29383)
- Frequent "Data structure corruption" in InnoDB after OOM (MDEV-29374)
- Recovery or backup of instant ALTER TABLE is incorrect (MDEV-29438)
- InnoDB Temporary Tablespace (ibtmp1) is continuously growing (MDEV-28240)
- Full text index corruption if shutdown before changes are fully flushed (MDEV-29342)
- JSON_VALUE() does not parse NULL properties properly (MDEV-27151)
- InnoDB hangs on multiple concurrent requests of a cold ROW_FORMAT=COMPRESSED page (MDEV-27983)

Changelog

For a complete list of changes and bugfixes made in MariaDB 10.6.10, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.6.10, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.6.2.7 MariaDB 10.6.9 Release Notes

The most recent release of MariaDB 10.6 is:
MariaDB 10.6.16 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.6.9 | Release Notes | Changelog | Overview of 10.6

Release date: 15 Aug 2022

MariaDB 10.6 is the current long-term maintenance stable series of MariaDB. It is an evolution of MariaDB 10.5 with several

entirely new features.

MariaDB 10.6.8 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.6 see the What is MariaDB 10.6? page.

Thanks, and enjoy MariaDB!

Regressions

Unfortunately, some bugs have been found post release, so this is updated to help avoid these issues if possible. All are fixed in MariaDB 10.6.10.

- MDEV-29374 — a crash shortly after a page merge (can be triggered by an UPDATE or DELETE or, say, rollback of an INSERT) can cause data corruption
- MDEV-29342 — if InnoDB table had a fulltext index and only one new row was inserted between the last sync (performed asynchronously by a dedicated thread) and server shutdown, the fulltext index wasn't properly updated and became out of sync with the data. Before 10.6.9 this happened silently, in 10.6.9 an assertion crashed the server after detecting the inconsistency.

Notable Items

InnoDB

- InnoDB corruption due to lack of file locking (MDEV-28495)
- FULLTEXT search with apostrophe, and mandatory words (MDEV-20797)
- ALTER TABLE IMPORT TABLESPACE corrupts an encrypted table (MDEV-28779)
- ALTER TABLE wrong-result fix (MDEV-26294)
- Crash recovery fixes (MDEV-28668, MDEV-28731)
- DDL crash recovery fixes (MDEV-28752, MDEV-28802, MDEV-28864, MDEV-28870, MDEV-28923, MDEV-28977)
- Avoid crashes on corrupted data (MDEV-13542, MDEV-18519, MDEV-21098, MDEV-22388, MDEV-28457)

Replication

- explicit_defaults_for_timestamp is stored in binlog, so that CREATE TABLE on slave would always have the same effect as on master. (MDEV-29078)
- ER_SLAVE_INCIDENT error is specified now on slave to be seen with SHOW-SLAVE-STATUS (MDEV-21087)
- INCIDENT_EVENT is no longer binlogged when a being logged transaction can be safely rolledback (MDEV-21443)
- sequences related row-format events are made to correspond to binlog_row_image (MDEV-28487)
- Possible reason of FLUSH BINARY LOGS hang is eliminated (MDEV-28948)
- Fix out-of-order gtid error in the circular semisync setup (MDEV-28609)

Galera

- Possible to write/update with read_only=ON and not a SUPER privilege (MDEV-28546)
- Node crashes with Transport endpoint is not connected mysqld got signal 6 (MDEV-25068)
- Galera4 not able to report proper wsrep_incoming_addresses (MDEV-20627)
- Galera should replicate nextval()-related changes in sequences with INCREMENT <> 0, at least NOCACHE ones with engine=InnoDB (MDEV-27862)
- Add support for OpenSSL 3.0 in Galera (MDEV-25949)

Optimizer

- Server crash in JOIN_CACHE::free or in copy_fields (MDEV-23809)
  - Queries that use DISTINCT and an always-constant function like COLLATION(aggegate_func(...)) could cause a server crash. Note that COLLATION() is a special function - its value is constant even if its argument is not costant.
- Crash when using ANY predicand with redundant subquery in GROUP BY clause (MDEV-29139)
  - A query with a subuquery in this form could cause a crash:

```
... ANY (SELECT ... GROUP BY (SELECT redundant_subselect_here)) ...
```

- MariaDB Server SEGV on INSERT .. SELECT (MDEV-26427)
  - Certain queries in form "INSERT ... SELECT with_aggregate_or_window_func" could cause a crash.
- restore_prev_nj_state() doesn't update cur_sj_inner_tables correctly (MDEV-28749)
  - Subquery semi-join optimization could miss LooseScan or FirstMatch strategies for certain queries.
- Optimizer uses all partitions after upgrade to 10.3 (MDEV-28246)
  - For multi-table UPDATE or DELETE queries, the optimizer failed to apply Partition Pruning optimization for the table that is updated or deleted from.
- Range optimizer regression for key IN (const, ...) (MDEV-25020)
  - The issue can be observed on MariaDB 10.5.9 and later versions which have the fix for MDEV-9750. That fix introduceds optimizer_max_sel_arg_weight.
  - If one sets optimizer_max_sel_arg_weight to a very high value or zero (which means "unlimited") and runs queries that produce heavy-weight graphs, they can observe a performance slowdown, e.g.:

    ```
    table.keyXpartY [NOT] IN ( ... )
    ```

- Wrong result with table elimination combined with not_null_range_scan (MDEV-28858)
  - If one runs with optimizer_switch='not_null_range_scan=on' (which is not enabled by default), a query that does a join and has const tables could produce a wrong result.

OpenSSL

- Backport OpenSSL-3.0 compatibility to 10.6 branch (MDEV-28133)

JSON

- JSON_TABLE: extract document fragment into JSON column (MDEV-25875)

CONNECT

- CONNECT Engine now supports INSERT IGNORE with Mysql Table type (MDEV-27766)

mariadb Client

- New mariadb client option, `-enable-cleartext-plugin`. Option does not do anything, and is for MySQL-compatibility purposes only.

General

- explicit_defaults_for_timestamp now also has a session scope, not only global (MDEV-29225)
- Crash in JSON_EXTRACT (MDEV-29188)
- ALTER TABLE ALGORITHM=NOCOPY does not work after upgrade (MDEV-28727)
- Server crash upon CREATE VIEW with unknown column in ON condition (MDEV-29088)
- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.6 for Debian 10 "Buster" for ppc64el
- Repositories for Ubuntu 22.04 and RHEL/Rocky 9 have been added in this release

Security

- Fixes for the following security vulnerabilities:
  - CVE-2023-5157
  - CVE-2022-32082
  - CVE-2022-32089
  - CVE-2022-32081
  - CVE-2018-25032
  - CVE-2022-32091
  - CVE-2022-32084
  - CVE-2022-38791

Changelog

For a complete list of changes and bugfixes made in MariaDB 10.6.9, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.6.9, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.6.2.8 MariaDB 10.6.8 Release Notes

The most recent release of MariaDB 10.6 is:
MariaDB 10.6.16 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.6.8 | Release Notes | Changelog | Overview of 10.6

Release date: 20 May 2022

MariaDB 10.6 is the current long-term maintenance stable series of MariaDB. It is an evolution of MariaDB 10.5 with several entirely new features.

MariaDB 10.6.8 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.6 see the What is MariaDB 10.6? page.

Thanks, and enjoy MariaDB!

Note that MariaDB 10.6.8 is not yet available for Ubuntu 22.04 due to MDEV-28133. MariaDB 10.6.9 will be available.

Notable Items

InnoDB

- innodb_disallow_writes removed (MDEV-25975)
- InnoDB gap locking fixes (MDEV-20605, MDEV-28422)
- InnoDB performance improvements (MDEV-27557, MDEV-28185, MDEV-27767, MDEV-28313, MDEV-28137, MDEV-28465, MDEV-26789)
- Backup regression fixes (MDEV-27919)
- InnoDB portability: FreeBSD futexes (MDEV-26476), POWER and s390x transactional memory (MDEV-27956)
- ALTER TABLE: Fixed bogus duplicate key errors (MDEV-15250)
- DDL and crash recovery fixes (MDEV-27274, MDEV-27234, MDEV-27817)
- Requests to recalculate persistent statistics were sometimes lost (MDEV-27805)

Replication

- Semisync-slave server recovery is refined to correctly rollback prepared transaction (MDEV-28461)
- Circular semisync setup endless event circulation is handled (MDEV-27760)
- Semisync-slave server recovery is extended to work on new server_id server (MDEV-27342)
- Server initialization time gtid_slave_pos purge related reason of crashing in binlog background thread is removed

(MDEV-26473)
- Shutdown of the semisync master can't produce inconsistent state anymore (MDEV-11853)
- Binlogs disappear after rsync IST (MDEV-28583)
- master crash is eliminated in compressed semisync replication protocol with packet counting amendment (MDEV-25580)
- OPTIMIZE on a sequence does not cause counterfactual ER_BINLOG_UNSAFE_STATEMENT anymore (MDEV-24617)
- Automatically generated Gtid_log_list_event is made to recognize within replication event group as a formal member (MDEV-28550)
- Replication unsafe INSERT .. ON DUPLICATE KEY UPDATE using two or more unique key values at a time with MIXED format binlogging is corrected (MDEV-28310)
- Replication unsafe INSERT .. ON DUPLICATE KEY UPDATE stops issuing unnecessary "Unsafe statement" with MIXED binlog format (MDEV-21810)
- Incomplete replication event groups are detected to error out by the slave IO thread (MDEV-27697)
- mysqlbinlog --stop-never --raw now flushes the result file to disk after each processed event so the file can be listed with the actual bytes (MDEV-14608)

Backup

- Incorrect binlogs after Galera SST using rsync and mariabackup (MDEV-27524)
- mariabackup does not detect multi-source replication slave (MDEV-21037)
- Useless warning "InnoDB: Allocated tablespace ID <id> for <tablename>, old maximum was 0" during backup stage (MDEV-27343)
- mariabackup prepare fails for incrementals if a new schema is created after full backup is taken (MDEV-28446))

Optimizer

- Query performance degradation in newer MariaDB versions when using many tables (MDEV-28073)
- A SEGV in Item_field::used_tables/update_depend_map_for_order... (MDEV-26402)
- ANALYZE FORMAT=JSON fields are incorrect for UNION ALL queries (MDEV-27699)
- Subquery in an UPDATE query uses full scan instead of range (MDEV-22377)
- Assertion `item1->type() == Item::FIELD_ITEM ... (MDEV-19398)
- Server crashes in Expression_cache_tracker::fetch_current_stats (MDEV-28268)
- MariaDB server crash at Item_subselect::init_expr_cache_tracker (MDEV-26164, MDEV-26047)
- Crash with union of my_decimal type in ORDER BY clause (MDEV-25994)
- SIGSEGV in st_join_table::cleanup (MDEV-24560)
- Assertion `!eliminated' failed in Item_subselect::exec (MDEV-28437)

General

- Server error messages are now available in Chinese (MDEV-28227)
- For RHEL/CentOS 7, non x86_64 architectures are no longer supported upstream and so our support will also be dropped with this release
- Packages for Ubuntu 22.04 LTS "Jammy" and Fedora 36 are not yet available pending the resolution of MDEV-28133: *Backport OpenSSL-3.0 compatibility to 10.6 branch*
- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.6 for Debian 9 "Stretch", Ubuntu 21.10 "Impish", and Fedora 34

Security

- Fixes for the following security vulnerabilities:
  - CVE-2021-46669
  - CVE-2022-27376
  - CVE-2022-27377
  - CVE-2022-27378
  - CVE-2022-27379
  - CVE-2022-27380
  - CVE-2022-27381
  - CVE-2022-27382
  - CVE-2022-27383
  - CVE-2022-27384
  - CVE-2022-27386
  - CVE-2022-27387
  - CVE-2022-27444
  - CVE-2022-27445

- CVE-2022-27446
- CVE-2022-27447
- CVE-2022-27448
- CVE-2022-27449
- CVE-2022-27451
- CVE-2022-27452
- CVE-2022-27455
- CVE-2022-27456
- CVE-2022-27457
- CVE-2022-27458
- CVE-2022-32087
- CVE-2022-32086
- CVE-2022-32085
- CVE-2022-32083
- CVE-2022-32088

Changelog

For a complete list of changes and bugfixes made in MariaDB 10.6.8, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.6.8, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.6.2.9 MariaDB 10.6.7 Release Notes

The most recent release of MariaDB 10.6 is:
MariaDB 10.6.16 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.6.7 | Release Notes | Changelog | Overview of 10.6

Release date: 12 Feb 2022

MariaDB 10.6 is the current long-term maintenance stable series of MariaDB. It is an evolution of MariaDB 10.5 with several entirely new features.

MariaDB 10.6.7 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.6 see the What is MariaDB 10.6? page.

Thanks, and enjoy MariaDB!

- This release fixes a blocking problem with the MariaDB 10.6.6 release when manually running mariadb-upgrade. (MDEV-27789)
- See MariaDB 10.6.6 for other changes since the previous release.

InnoDB

- Set innodb_change_buffering=none by default (MDEV-27734)

Security

- Fixes for the following security vulnerabilities:
  - CVE-2021-46665
  - CVE-2021-46664
  - CVE-2021-46661
  - CVE-2021-46668
  - CVE-2021-46663

Changelog

For a complete list of changes and bugfixes made in MariaDB 10.6.7, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.6.7, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.6.2.10 MariaDB 10.6.6 Release Notes

The most recent release of MariaDB 10.6 is:
MariaDB 10.6.16 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.6.6 | Release Notes | Changelog | Overview of 10.6

Release date: 9 Feb 2022

This release is no longer available for download after a problem was noticed when manually running mariadb-upgrade. See MDEV-27789 for more details.

Please use a later release.

MariaDB 10.6 is the current long-term maintenance stable series of MariaDB. It is an evolution of MariaDB 10.5 with several entirely new features.

MariaDB 10.6.6 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.6 see the What is MariaDB 10.6? page.

Thanks, and enjoy MariaDB!

Notable Items

InnoDB

- --skip-symbolic-links does not disallow `.isl` file creation (MDEV-26870)
- Indexed CHAR columns are broken with NO_PAD collations (MDEV-25440)
- insert-intention lock conflicts with waiting ORDINARY lock (MDEV-27025)

- Crash recovery improvements (MDEV-26784, MDEV-27022, MDEV-27183, MDEV-27610)
- mariabackup skips valid .ibd file (MDEV-26326)
- Allow seamless upgrade despite ROW_FORMAT=COMPRESSED (MDEV-27736)

Galera

- Galera updated to 26.4.11
- Galera SST scripts should use ssl_capath (not ssl_ca) for CA directory (MDEV-27181)
- Alter Sequence do not replicate to another nodes with in Galera Cluster (MDEV-19353)
- Galera crash - Assertion. Possible parallel writeset problem (MDEV-26803)
- CREATE TABLE with FOREIGN KEY constraint fails to apply in parallel (MDEV-27276)
- Galera cluster node consider old server_id value even after modification of server_id `[wsrep_gtid_mode=ON]` (MDEV-26223)

Replication

- Seconds behind master corrected from artificial spikes at relay-log rotation (MDEV-16091)
- Statement rollback in binlog when transaction creates or drop temporary table is set right (MDEV-26833)
- CREATE-or-REPLACE SEQUENCE is made to binlog with the DDL flag to stabilize its parallel execution on slave (MDEV-27365)

Packaging & Misc

- prohibition running two upgrades in parallel (MDEV-27068, MDEV-27107, MDEV-27279)
- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.6 for Ubuntu 21.04 Hirsute, CentOS 8, and Fedora 33
- mariadb_repo_setup script updated to version 2022-02-08, with the following fixes and enhancements:
  - Default location of the script has been moved to: https://r.mariadb.com/downloads/mariadb_repo_setup (old location is deprecated, but still works)
  - The GPG keyring file, used with Debian and Ubuntu repositories, has moved to: https://supplychain.mariadb.com/mariadb-keyring-2019.gpg and the checksum for the file can be found at: https://supplychain.mariadb.com/mariadb-keyring-2019.gpg.sha256
  - Support for RHEL and SLES aarch64 repositories added
  - New function added to verify that the MariaDB Server version, if specified on the command line, follows the correct naming and that a corresponding repository actually exists.
  - Fixed repository pinning for Ubuntu and Debian repositories
  - MariaDB Server 10.7 is now the default server version

Docker Library

- Faster initialization by disabling binary logging during initialization (MDEV-27074)
- mysql_upgrade can be run if needed using the environment variable MARIADB_AUTO_UPGRADE=1 (MDEV-25670)
- A healthcheck script /usr/local/bin/healthcheck.sh is installed in the container with various checking options (MDEV-25434)
- mysql@localhost user is created with the environment variable MARIADB_MYSQL_LOCALHOST_USER=1 and additional grants (beyond USAGE) with MARIADB_MYSQL_LOCALHOST_GRANTS={global grant list} (MDEV-27732)
- skip innodb buffer pool loads/dumps on temporary startup/shutdown for faster startup/initialization, and accurate "healthcheck.sh --innodb_buffer_pool_loaded"
- change group ownership on datadir/socket dir (issue #401)
- log note about note on Securing system users, mysql_secure_installation not required (reddit suggestion)
- speed up Docker Library initialization of timezones (MDEV-27608, MDEV-23326)
- MariaDB names of executable programs and scripts used instead of historical mysql ones

Security

- Fixes for the following security vulnerabilities:
  - CVE-2022-24052
  - CVE-2022-24051
  - CVE-2022-24050
  - CVE-2022-24048
  - CVE-2021-46659
  - CVE-2022-0778
  - CVE-2022-21595

Changelog

For a complete list of changes and bugfixes made in MariaDB 10.6.6, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.6.6, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.6.2.11 MariaDB 10.6.5 Release Notes

The most recent release of MariaDB 10.6 is:
MariaDB 10.6.16 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.6.5 | Release Notes | Changelog | Overview of 10.6

Release date: 8 Nov 2021

MariaDB 10.6 is the current stable series of MariaDB. It is an evolution of MariaDB 10.5 with several entirely new features.

MariaDB 10.6.5 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.6 see the What is MariaDB 10.6? page.

Thanks, and enjoy MariaDB!

Notable Items

InnoDB

- Linux after kernel version 5.10 has a io-uring regression causing a write to storage to be lost, or not acknowledged. As such innodb_use_native_aio will default to 0 (off) until 5.16. If innodb_use_native_aio is enabled in your configuration, a warning will be logged, however it will continue with the io-uring enabled, potentially resulting in a hang, or an assertion later. The long term support kernel 5.14.14 we haven't observed failures, and 5.15.0-rc7 failures have been observed, though less frequently. If you have innodb_use_native_aio explicitly enabled, and are using watch out for a lack of InnoDB updates followed by a 10 minute timeout. See MDEV-26674 for details.
- `ALTER TABLE…IMPORT TABLESPACE` fixes (MDEV-18543, MDEV-20931, MDEV-26131, MDEV-26621)
- `innodb_undo_log_truncate` fixes (MDEV-26445, MDEV-26450, MDEV-26672, MDEV-26864)
- Page I/O performance fixes (MDEV-25215, MDEV-26547, MDEV-26626, MDEV-26819)
- Replication timeouts with `XA PREPARE` (MDEV-26682)
- Improved DDL and data dictionary (MDEV-25919)
- Performance fixes (MDEV-26356, MDEV-26467, MDEV-26826)

Replication

- Memory hogging on slave by ROW event applier is eliminated (MDEV-26712)
- `mysql --binary-mode` now properly handles `\\0` in data (MDEV-25444)
- Fixes race condition between `SHOW BINARY LOGS` and `RESET MASTER` (MDEV-20215)
- Missed statement rollback in case transaction drops or create temporary table is corrected (MDEV-26833)

Audit Plugin

- The `QUERY_DDL` server_audit_events setting now logs `CREATE/DROP [PROCEDURE / FUNCTION / USER]` statements. See MariaDB Audit Plugin - Log Settings. (MDEV-23457)

Packaging & Misc

- Session tracking flag in OK_PACKET (MDEV-26868)
- Some views force server (and mysqldump) to generate invalid SQL for their definitions (MDEV-26299)

Security

- Fixes for the following security vulnerabilities:
  - CVE-2021-46667
  - CVE-2021-46662
  - CVE-2022-27385
  - CVE-2022-27385
  - CVE-2022-31624

Changelog

For a complete list of changes and bugfixes made in MariaDB 10.6.5, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.6.5, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.6.2.12 MariaDB 10.6.4 Release Notes

The most recent release of MariaDB 10.6 is:
MariaDB 10.6.16 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.6.4 | Release Notes | Changelog | Overview of 10.6

Release date: 6 Aug 2021

Warning: This version can cause InnoDB file corruption on FreeBSD and on AIX. Stick to an earlier release, or upgrade to a more recent release, if you are running either of these environments. See MDEV-26537.

MariaDB 10.6 is the current stable series of MariaDB. It is an evolution of MariaDB 10.5 with several entirely new features.

MariaDB 10.6.4 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.6 see the What is MariaDB 10.6? page.

Thanks, and enjoy MariaDB!

Notable Items

InnoDB

- InnoDB no longer acquires advisory file locks by default (MDEV-24393)
- Encryption: Automatically disable key rotation checks for file_key_management plugin (MDEV-14180)
- Some fixes from MySQL 5.7.35 (MDEV-26205)
- Fixed scrubbing on AIX (MDEV-26110)
- buf_pool.flush_list corrupted by buffer pool resizing or ROW_FORMAT=COMPRESSED (MDEV-26200)

Optimizer

- A query that uses ORDER BY .. LIMIT clause and "Range checked for each record optimization" could produce incorrect results under some circumstances (MDEV-25858)
- Queries that have more than 32 equality conditions comparing columns of different tables ("tableX.colX=tableY.colY) could cause a stack overrun in the query optimizer (MDEV-17783 , MDEV-23937)
- "Condition pushdown into derived table" optimization cannot be applied if the expression being pushed refers to a derived table column which is computed from expression that has a stored function call, @session variable reference, or other similar construct. The fix for MDEV-25969 makes it so that only the problematic part of the condition is not pushed. The rest of the condition is now pushed. (MDEV-25969)
- A query with window function on the left side of the subquery could cause a crash. (MDEV-25630)
- Fixed the issue fixed in MySQL Bug #76803: DML or locking SELECT statements that use outer joins could produce this warning in the error log: `[ERROR] InnoDB: Unlock row could not find a 3 mode lock on the record.` (MDEV-26106)

Packaging & Misc

- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.6 for Ubuntu 20.10 Groovy
- Debian 11 Bullseye repositories added
- Galera updated to 26.4.9
- Linux on IBM Z (s390x) architecture added with releases on Ubuntu-20.04 Focal

Security

- Fixes for the following security vulnerabilities :
  - CVE-2021-2372
  - CVE-2021-2389

Changelog

For a complete list of changes and bugfixes made in MariaDB 10.6.4, with links to detailed information on each push, see the changelog .

Contributors

For a full list of contributors to MariaDB 10.6.4, see the MariaDB Foundation release announcement .

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.6.2.13 MariaDB 10.6.3 Release Notes

The most recent release of MariaDB 10.6 is:
MariaDB 10.6.16 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.6.3 | Release Notes | Changelog | Overview of 10.6

Release date: 6 Jul 2021

MariaDB 10.6 is the current stable series of MariaDB. It is an evolution of MariaDB 10.5 with several entirely new features.

MariaDB 10.6.3 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.6 see the What is MariaDB 10.6? page.

Thanks, and enjoy MariaDB!

Notable Changes

InnoDB

- Maximum value of innodb_lock_wait_timeout is now 100000000, which means infinite timeout (MDEV-26067)
- Write performance improvements: MDEV-25954, MDEV-25948, MDEV-25113, MDEV-26004, MDEV-25801
- Atomic DDL rewrite (MDEV-25506)
- Thinly provisioned SSD support for page_compressed tables (MDEV-26029)

Replication

- Fix binlog background thread hang at shutdown (MDEV-26031)

General

- The views INFORMATION_SCHEMA.KEYWORDS and INFORMATION_SCHEMA.SQL_FUNCTIONS have been added to the information schema (MDEV-25129)
- Assertion `thd->locked_tables_mode == LTM_NONE' failed in Locked_tables_list::init_locked_tables (MDEV-25837)

Security

- Fixes for the following security vulnerabilities:
  - CVE-2021-35604
  - CVE-2021-46658

Changelog

For a complete list of changes and bugfixes made in MariaDB 10.6.3, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.6.3, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.6.2.14 MariaDB 10.6.2 Release Notes

The most recent release of MariaDB 10.6 is:
MariaDB 10.6.16 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.6.2 | Release Notes | Changelog | Overview of 10.6

Release date: 18 Jun 2021

MariaDB 10.6 is the current development series of MariaDB. It is an evolution of MariaDB 10.5 with several entirely new features.

MariaDB 10.6.2 is a ***Release Candidate (RC)*** release.

Do not use non-stable (non-GA) releases in production!

For an overview of MariaDB 10.6 see the What is MariaDB 10.6? page.

Register now for our MariaDB Community Server 10.6 webinar to be held 2021-06-29 and be one of the first to see the biggest features coming in MariaDB Community Server 10.6, with an exclusive opportunity to have your questions answered by MariaDB engineering and product leads.

Thanks, and enjoy MariaDB!

Notable Changes

InnoDB

- When innodb_adaptive_hash_index=OFF (the default), the following counters (which reflect btr_cur_n_non_sea) will no longer be updated (MDEV-25882):
  - adaptive_hash_index in INFORMATION_SCHEMA.INNODB_METRICS
  - Innodb_adaptive_hash_non_hash_searches in INFORMATION_SCHEMA.GLOBAL_STATUS

Replication

- Semisync replica recovery is introduced. rpl-semi-sync-slave-enabled = ON server executes a special recovery branch to guarantee its consistency with a primary server (MDEV-21117)

General

- Error messages now use "MariaDB" instead of "MySQL" (MDEV-22189)
- Implement `FLUSH TABLES tbl_name [, tbl_name] ... WITH READ LOCK` for views (MDEV-15888)
- All statements can be prepared, except `PREPARE`, `EXECUTE`, and `DEALLOCATE / DROP PREPARE` (MDEV-16708)

Changelog

For a complete list of changes and bugfixes made in MariaDB 10.6.2, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.6.2, see the MariaDB Foundation release announcement.

Do not use non-stable (non-GA) releases in production!

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.6.2.15 MariaDB 10.6.1 Release Notes

The most recent release of MariaDB 10.6 is:
MariaDB 10.6.16 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.6.1 | Release Notes | Changelog | Overview of 10.6

Release date: 21 May 2021

MariaDB 10.6 is the current development series of MariaDB. It is an evolution of MariaDB 10.5 with several entirely new features.

MariaDB 10.6.1 is a ***Beta*** release.

Do not use *beta* releases in production!

For an overview of MariaDB 10.6 see the What is MariaDB 10.6? page.

Register now for our MariaDB Community Server 10.6 webinar to be held 2021-06-29 and be one of the first to see the biggest features coming in MariaDB Community Server 10.6, with an exclusive opportunity to have your questions answered by MariaDB engineering and product leads.

Thanks, and enjoy MariaDB!

Notable Changes

Notable changes of this release include:

Atomic DDL

- CREATE TABLE, ALTER TABLE, RENAME TABLE, DROP TABLE, DROP DATABASE and related DDL statements are now atomic. Either the statement is fully completed, or everything is reverted to it's original state. Note that when deleting multiple tables with DROP TABLE, only each individual drop is atomic, not the full list of tables). (MDEV-23842).

Replication, Galera and Binlog

- The delay between binary log purges can now be specified with much greater precision. The system variable binlog_expire_logs_seconds is introduced as a form of alias for expire_logs_days, which now accepts a precision of 1/1000000 days (MDEV-19371)
- Allow transition from unencrypted to TLS Galera cluster communication without cluster downtime (MDEV-22131)
- DDL information logged on all Galera cluster nodes if wsrep_debug is set to SERVER and wsrep_OSU_method is 'TOI' (MDEV-9609)

- For the mysqlbinlog / mariadb-binlog `--base64-output` option, removed the deprecated `always` option, and changed the default to `auto` (MDEV-25222)

Oracle Compatibility

- ADD_MONTHS() added (MDEV-20025)
- TO_CHAR() added (MDEV-20017)
- SYS_GUID() added (MDEV-24285)
- MINUS is mapped to EXCEPT in UNION (MDEV-20021)
- ROWNUM function returns the current number of accepted rows in the current context (MDEV-24089)

Character Sets

- The `utf8` character set (and related collations) is now by default an alias for `utf8mb3` rather than the other way around. It can be set to imply `utf8mb4` by changing the value of the old_mode system variable (MDEV-8334)

Clients

- For clients such as mysql / mariadb, the connection property specified via the command line (e.g. --port=3306) will now force its type (MDEV-14974)

Changelog

For a complete list of changes made in MariaDB 10.6.1, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.6.1, see the MariaDB Foundation release announcement.

Do not use *beta* releases in production!

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.6.2.16 MariaDB 10.6.0 Release Notes

The most recent release of MariaDB 10.6 is:
MariaDB 10.6.16 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.6.0 | Release Notes | Changelog | Overview of 10.6

Release date: 26 Apr 2021

MariaDB 10.6 is the current development series of MariaDB. It is an evolution of MariaDB 10.5 with several entirely new features.

MariaDB 10.6.0 is an ***Alpha*** release.

Do not use *alpha* releases in production!

For an overview of MariaDB 10.6 see the What is MariaDB 10.6? page.

Thanks, and enjoy MariaDB!

Notable Changes

This is the first alpha release in the MariaDB 10.6 series.

Notable changes of this release include:

SQL Syntax

- Indexes can be ignored (MDEV-7317)
- Implement SQL-standard SELECT ... OFFSET ... FETCH (MDEV-23908)
- Add SELECT ... SKIP LOCKED syntax (InnoDB only) (MDEV-13115)
- JSON_TABLE, used to extract JSON data based on a JSON path expression and to return it as a relational table (MDEV-17399)

Oracle Compatibility

- Anonymous subqueries in a FROM clause (no AS clause) are permitted in ORACLE mode

Storage Engines

- TokuDB has been removed (MDEV-19780)
- CassandraSE has been removed (MDEV-23024)

InnoDB

- Optimization to speed up inserts into an empty table (MDEV-515)
- Make InnoDB's COMPRESSED row format read-only by default (MDEV-23497)
- Information Schema SYS_TABLESPACES now directly reflects the filesystem, and SYS_DATAFILES has been removed (MDEV-22343)
- `innodb_flush_method=O_DIRECT` is enabled by default (MDEV-24854), and `liburing` replaces `libaio` on recent Linux kernels (MDEV-24883).
- The InnoDB transaction deadlock reporter was improved (MDEV-24738).
- The old MariaDB 5.5-compatible `innodb` checksum is no longer supported, only `crc32` and `full_crc32`. Removed the `*innodb` and `*none` options from innodb_checksum_algorithm, and the `--strict-check` / `-C` and `--write` / `-w` options from innochecksum (MDEV-25105)

Replication, Galera and Binlog

- Increase master_host limit to 255, user to 128 (MDEV-24312)
- The wsrep_mode system variable, for turning on WSREP features which are not part of default behavior (including the experimental Aria replication) (MDEV-20008, MDEV-20715, MDEV-24946)

Sys Schema

- Bundle sys-schema, a collection of views, functions and procedures to help administrators get insight into database usage. (MDEV-9077)

Performance Schema

- Merged replication instrumentation and tables (MDEV-16437, MDEV-20220)

General

- Do not resend unchanged resultset metadata for prepared statements (MDEV-19237)
- --bind-address=hostname now listens on both IPv6 and IPv4 addresses (MDEV-6536)
- Support systemd socket activation (MDEV-5536)
- For the GSSAPI plugin, support AD or local group name, and SIDs on Windows (MDEV-23959)
- Check for $MARIADB_HOME/my.cnf (MDEV-21365)

- max_recursive_iterations has been reduced to 1000 (MDEV-17239)
- Setting system variables to negative values will no longer set them to the maximum value (MDEV-22219)

Galera

- wsrep_mode variable for turning on WSREP features which are not part of default behavior.
- wsrep_strict_ddl has been deprecated. Use wsrep_mode=STRICT_REPLICATION instead.

InnoDB Variables

The following deprecated variables have been removed (MDEV-23397):

- innodb_adaptive_max_sleep_delay
- innodb_background_scrub_data_check_interval
- innodb_background_scrub_data_compressed
- innodb_background_scrub_data_interval
- innodb_background_scrub_data_uncompressed
- innodb_buffer_pool_instances
- innodb_commit_concurrency
- innodb_concurrency_tickets
- innodb_file_format
- innodb_large_prefix
- innodb_log_checksums
- innodb_log_compressed_pages
- innodb_log_files_in_group
- innodb_log_optimize_ddl
- innodb_page_cleaners
- innodb_replication_delay
- innodb_scrub_log
- innodb_scrub_log_speed
- innodb_thread_concurrency
- innodb_thread_sleep_delay
- innodb_undo_logs

Changelog

For a complete list of changes made in MariaDB 10.6.0, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.6.0, see the MariaDB Foundation release announcement.

Do not use *alpha* releases in production!

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.7 MariaDB Server 10.5

Changes and Improvements in MariaDB 10.5

Current Version: 10.5.23 | Status: Stable (GA) | Release Date: 13 Nov 2023

Release Notes - MariaDB 10.5 Series
MariaDB 10.5 Series Release Notes

Changelogs - MariaDB 10.5 Series
MariaDB 10.5 changelogs

There are 2 related questions.

7.0.7.1 Changes and Improvements in MariaDB 10.5

The most recent release of MariaDB 10.5 is:
MariaDB 10.5.23 Stable (GA) Download Now
Alternate download from mariadb.org

Contents

MariaDB 10.5 is a previous major stable version The first stable release was in June 2020, and it will be maintained until June 2025.

Upgrading

- See Upgrading Between Major MariaDB Versions and Upgrading from MariaDB 10.4 to MariaDB 10.5.

Implemented Features

ColumnStore

- This release of MariaDB Server includes the MariaDB ColumnStore storage engine. *Note, that plugins have independent maturity levels and MariaDB ColumnStore in 10.5.4 has **Beta** maturity.*

Binaries Named mariadb (mysql Symlinked)

- All binaries previously beginning with `mysql` now begin with `mariadb`, with symlinks for the corresponding `mysql`

command. (MDEV-21303)
- When starting the MariaDB server via the systemd service it will be started using the `mariadbd` binary name, so this will now show up in the system process list instead of `mysqld`
- Same for the `mariadbd-safe` wrapper script. Even when called via the `mysqld_safe` symlink, it will start the actual server process as `mariadbd`, not `mysqld` now. This also affects startup via system service init scripts on platforms that don't yet have switched to SystemD

INET 6 Data Type

- New INET6 data type for storing IPv6 addresses (MDEV-274)

Amazon S3

- The S3 storage engine allows one to archive MariaDB tables in Amazon S3, or any third-party public or private cloud that implements S3 API (MDEV-22606)
- Both S3 tables and partitioned S3 tables are discoverable. This means that if you create a partitioned S3 table, both the partitioned table and its partitions can be directly used by another server that has access to the S3 storage. (MDEV-22088)

Privileges Made More Granular

- Split SUPER privilege to smaller privileges (MDEV-21743). New privileges were added so that more fine grained tuning of what each user can do can be applied:
  - BINLOG ADMIN
  - BINLOG REPLAY
  - CONNECTION ADMIN
  - FEDERATED ADMIN
  - READ_ONLY ADMIN
  - REPLICATION MASTER ADMIN
  - REPLICATION SLAVE ADMIN
  - SET USER
- The REPLICATION CLIENT privilege was renamed to BINLOG MONITOR. The old syntax is understood for compatibility (MDEV-21743).
- The SHOW MASTER STATUS statement was renamed to SHOW BINLOG STATUS (MDEV-21743). The old syntax is understood for compatibility.
- A number of statements changed the privileges that they require. The old privileges were historically inappropriately chosen in the upstream. 10.5.2 fixes this problem. Note, these changes are incompatible to previous versions. A number of GRANT commands might be needed after upgrade.
  - SHOW BINLOG EVENTS now requires the BINLOG MONITOR privilege (required REPLICATION SLAVE prior to 10.5.2).
  - SHOW SLAVE HOSTS now requires the REPLICATION MASTER ADMIN privilege (required REPLICATION SLAVE prior to 10.5.2).
  - SHOW SLAVE STATUS now requires the REPLICATION SLAVE ADMIN or the SUPER privilege (required REPLICATION CLIENT or SUPER prior to 10.5.2).
  - SHOW RELAYLOG EVENTS now requires the REPLICATION SLAVE ADMIN privilege (required REPLICATION SLAVE prior to 10.5.2).
- In order to help the server understand which version a privilege record was written by, the mysql.global_priv.priv field contains a new JSON field, `version_id` (MDEV-21704)
- SHOW PRIVILEGES now correctly lists the `Delete history` privilege, rather than displaying it as `Delete versioning rows`. (MDEV-20382)

InnoDB: Performance Improvements etc.

- Extend SHOW STATUS LIKE 'Innodb_%' (MDEV-18582)
- Clean up INFORMATION_SCHEMA.INNODB_ tables (MDEV-19940)
- Doublewrite buffer is unnecessarily used for newly (re)initialized pages (MDEV-19738)
- Defer change buffer merge until pages are requested (MDEV-19514)
- Remove dummy tablespace for the redo log (MDEV-18115)
- Optimize access to InnoDB page header fields (MDEV-21133)
- Remove multiple InnoDB buffer pool instances (MDEV-15058)
  - Columns that indicated the buffer pool instance from the Information Schema innodb_buffer_page, innodb_buffer_page_lru, innodb_buffer_pool_stats, innodb_cmpmem and innodb_cmpmem_reset tables now return a dummy value of `0`.
- Remove buf_page_t::newest_modification (MDEV-21132)
- Replace recv_sys_t::addr_hash with a std::map (MDEV-19586)

- Obsolete internal parser for FK in InnoDB (MDEV-20480)
- InnoDB thread pool for background tasks (MDEV-16264)
- An upgrade will only be possible after a clean shutdown. mariabackup --prepare will not work with backups taken before version 10.5.2.
- Efficient InnoDB redo log record format (MDEV-12353)
- Improve InnoDB redo log group commit performance (MDEV-21534)
- Do not acquire InnoDB record locks when covering table locks exist (MDEV-14479)
- Issue a message on changing deprecated innodb_log_files_in_group (MDEV-21990)
- Optimize append only files for NVDIMM (MDEV-17084)
- Avoid writing freed InnoDB pages (MDEV-15528)

InnoDB New Defaults for Variables

- innodb_adaptive_hash_index now defaults to `OFF` (MDEV-20487)
- innodb_checksum_algorithm now defaults to `full_crc32` (MDEV-19534)

InnoDB Removed or Deprecated Variables

- innodb_buffer_pool_instances
- innodb_checksums (MDEV-19534)
- innodb_locks_unsafe_for_binlog (MDEV-19544)
- innodb_log_checksums (MDEV-19543)
- innodb_log_files_in_group (MDEV-14425 & MDEV-20907)
- innodb_log_optimize_ddl (MDEV-19747)
- innodb_rollback_segments (MDEV-19570)
- innodb_scrub_log and innodb_scrub_log_speed (MDEV-21870)
- Remove INFORMATION_SCHEMA.INNODB_TABLESPACES_SCRUBBING table and deprecate and ignore:
- innodb-background-scrub-data-uncompressed
- innodb-background-scrub-data-compressed
- innodb-background-scrub-data-interval
- innodb-background-scrub-data-check-interval (MDEV-15528)
- innodb_stats_sample_pages (MDEV-19551)
- innodb_undo_logs (MDEV-19570)
- innodb_thread_concurrency
- innodb_commit_concurrency
- innodb_replication_delay
- innodb_concurrency_tickets
- innodb_thread_sleep_delay
- innodb_adaptive_max_sleep_delay (MDEV-23379)

Performance Schema Updates to Match MySQL 5.7 Instrumentation and Tables

- Memory (MDEV-16431)
- Meta data locking (MDL) (MDEV-16432)
- Prepared statements (ps) (MDEV-16433)
- [show] status instrumentation and tables (MDEV-16438)
- Stored procedures (MDEV-16434)
- Sxlocks (MDEV-16436)
- Transactions (MDEV-16435)
- User variables (MDEV-16439)

Galera: Full GTID Support

- Add full GTID support to Galera cluster (commit). With this feature all nodes in a cluster will have the same GTID for replicated events originating from the cluster. Also added a new variable, `wsrep_gtid_seq_no` , to manually update the WSREP GTID sequence number in the cluster (similar to how the `gtid_seq_no` variable is used for non-WSREP transactions).
- Add new mode to wsrep_OSU_method in which Galera checks storage engine of the affected table (MDEV-20051)
- Galera: Replicate MariaDB GTID to other nodes in the cluster (MDEV-20720)

Binary Log and Replication: More Metadata

- slave_parallel_mode now defaults to `optimistic` (MDEV-18648).
- Make REPLICA a synonym for SLAVE in SQL statements (MDEV-20601)

- `ENFORCE` option for slave_run_triggers_for_rbr (MDEV-21833)
- Extended binlog metadata (MDEV-20477) to include new fields. This was done to solve replication issues when the Master and Slave table had different definitions for a column which could lead to data loss (MDEV-19708). It also enables us to do better replication with pluggable data types in the future.
  - The new metadata fields are:
    - Signedness of Numeric Columns
    - Character Set of Character Columns and Binary Columns
    - Column Name
    - String Value of SET Columns
    - String Value of ENUM Columns
    - Primary Key
    - Character Set of SET Columns and ENUM Columns
    - Geometry Type
  - Also added a new global variable, binlog_row_metadata to control the amount of metadata logged. Possible values are:
    - `FULL` - all metadata is logged
    - `MINIMAL` - only metadata required by a worker is logged
    - `NO_LOG` - No metadata is logged (default)
- Binary log DDL entries can now be marked that they should be ignored if the target table doesn't exist (implicit `IF EXISTS`).
- mariadb-binlog output is extended to show all replication flags. Example of output: `SET @@session.foreign_key_checks=1, @@session.sql_auto_is_null=0, @@session.unique_checks=1, @@session.autocommit=1, @@session.check_constraint_checks=1, @@session.sql_if_exists=0/*!*/`.
- SHOW BINLOG EVENTS and SHOW RELAYLOG EVENTS are extended to show replication flags.

Syntax

- INSERT ... RETURNING (MDEV-10014) - returns SELECT of inserted rows (analogous to DELETE ... RETURNING)
- REPLACE ... RETURNING (MDEV-10014)
- EXCEPT ALL and INTERSECT ALL (MDEV-18844)
- Application period tables: WITHOUT OVERLAPS (MDEV-16978)
- Setup default partitions for system versioning (MDEV-19903)
- Database comments in CREATE DATABASE and ALTER DATABASE statements (MDEV-307)
- ALTER TABLE ... RENAME INDEX / KEY (MDEV-7318)
- ALTER TABLE ... RENAME COLUMN (MDEV-16290)
- ALTER TABLE and RENAME TABLE now support `IF EXISTS`.
- Add VISIBLE attribute for indexes in CREATE TABLE (MDEV-22199)
- Recursive CTE cycle detection using CYCLE clause (MDEV-20632)
- RELEASE_ALL_LOCKS hold by GET_LOCK() (MDEV-10569)
- Fix REFERENCES constraint in column definition (MDEV-20729)

JSON

- Added JSON_ARRAYAGG. This returns a JSON array containing an element for each value in a given set of JSON or SQL values. It acts on a column or an expression that evaluates to a single value.
- Added JSON_OBJECTAGG. This returns a JSON object containing key-value pairs. It takes two expressions that evaluate to a single value, or two column names, as arguments, the first used as a key, and the second as a value.

Thread Pool

- Information Schema tables (THREAD_POOL_GROUPS, THREAD_POOL_QUEUES, THREAD_POOL_STATS and THREAD_POOL_WAITS) for internals of generic thread_pool (MDEV-19313).

Performance Improvements

- Speed up binary row logging code
- Range optimizer speedups. Removed double calls to records_in_range() for some cases.
- Costs for using MEMORY tables updated to be more accurate
- Fixed that 'ref' access is preferred over 'range' for the same index.
- Improve connect speed (up to 25%). (MDEV-19515)

Query Optimizer

- Improve Protocol performance for numeric data by avoiding unnecessary character string conversions (MDEV-23162

, MDEV-23478)
- ANALYZE for statements is improved, now it also shows the time spent checking the WHERE clause and doing other auxiliary operations (MDEV-20854)
- Inferred IS NOT NULL predicates can be used by the range optimizer (MDEV-15777)
- Allow packed sort keys and values of non-sorted fields in the sort buffer (MDEV-21263 & MDEV-21580)
    - Makes filesort temporary files much smaller when VARCHAR, CHAR or BLOBs are used!

General

- The Information Schema SYSTEM_VARIABLES Table has a new column showing from which config file a variable derives its value (MDEV-12684)
- Switch Perl DBI scripts from DBD::mysql to DBD::MariaDB driver (MDEV-19755)
- The Aria max key length is now 2000 bytes, compared to 1000 bytes in MyISAM.
- DROP TABLE now reliably deletes table remnants inside a storage engine even if the .frm file is missing (MDEV-11412)
- Accelerated crc32() function for AMD64, ARMv8, POWER 8 (MDEV-22669)
- Binary tarball size has been reduced (MDEV-21943)

PCRE (Perl Compatible Regular Expressions)

- Migrate to PCRE2 (MDEV-14024), a newer version of the pcre library.

Variables

- For a list of all new variables, see System Variables Added in MariaDB 10.5 and Status Variables Added in MariaDB 10.5.
- The Information Schema SYSTEM_VARIABLES Table has a new column showing from which config file a variable derives its value (MDEV-12684).
- Port show_old_temporals from MySQL 5.6 (MDEV-19906). If set, old temporal data types (created with a pre-10.0 version of MariaDB) are displayed with a /* mariadb-5.3 */ comment.
- Numerous deprecated variables removed (MDEV-18650)
    - multi_range_count
    - thread_concurrency
    - timed_mutexes

Security Vulnerabilities Fixed in MariaDB 10.5

> For a complete list of security vulnerabilities (CVEs) fixed across all versions of MariaDB, see the Security Vulnerabilities Fixed in MariaDB page.

- CVE-2023-5157: MariaDB 10.5.17
- CVE-2023-22084: MariaDB 10.5.23
- CVE-2022-47015: MariaDB 10.5.20
- CVE-2022-38791: MariaDB 10.5.17
- CVE-2022-32091: MariaDB 10.5.17
- CVE-2022-32089: MariaDB 10.5.17
- CVE-2022-32088: MariaDB 10.5.16
- CVE-2022-32087: MariaDB 10.5.16
- CVE-2022-32086: MariaDB 10.5.16
- CVE-2022-32085: MariaDB 10.5.16
- CVE-2022-32084: MariaDB 10.5.17
- CVE-2022-32083: MariaDB 10.5.16
- CVE-2022-32082: MariaDB 10.5.17
- CVE-2022-32081: MariaDB 10.5.17
- CVE-2022-31624: MariaDB 10.5.13
- CVE-2022-27458: MariaDB 10.5.16
- CVE-2022-27457: MariaDB 10.5.16
- CVE-2022-27456: MariaDB 10.5.16
- CVE-2022-27455: MariaDB 10.5.16
- CVE-2022-27452: MariaDB 10.5.16
- CVE-2022-27451: MariaDB 10.5.16
- CVE-2022-27449: MariaDB 10.5.16
- CVE-2022-27448: MariaDB 10.5.16
- CVE-2022-27447: MariaDB 10.5.16

- CVE-2022-27446: MariaDB 10.5.16
- CVE-2022-27445: MariaDB 10.5.16
- CVE-2022-27444: MariaDB 10.5.16
- CVE-2022-27387: MariaDB 10.5.16
- CVE-2022-27386: MariaDB 10.5.16
- CVE-2022-27385: MariaDB 10.5.13
- CVE-2022-27384: MariaDB 10.5.16
- CVE-2022-27383: MariaDB 10.5.16
- CVE-2022-27382: MariaDB 10.5.16
- CVE-2022-27381: MariaDB 10.5.16
- CVE-2022-27380: MariaDB 10.5.16
- CVE-2022-27379: MariaDB 10.5.16
- CVE-2022-27378: MariaDB 10.5.16
- CVE-2022-27377: MariaDB 10.5.16
- CVE-2022-27376: MariaDB 10.5.16
- CVE-2022-24052: MariaDB 10.5.14
- CVE-2022-24051: MariaDB 10.5.14
- CVE-2022-24050: MariaDB 10.5.14
- CVE-2022-24048: MariaDB 10.5.14
- CVE-2022-21595: MariaDB 10.5.14
- CVE-2022-21451: MariaDB 10.5.10
- CVE-2022-21427: MariaDB 10.5.7
- CVE-2022-0778: MariaDB 10.5.14
- CVE-2021-46669: MariaDB 10.5.16
- CVE-2021-46668: MariaDB 10.5.15
- CVE-2021-46667: MariaDB 10.5.13
- CVE-2021-46666: MariaDB 10.5.11
- CVE-2021-46665: MariaDB 10.5.15
- CVE-2021-46664: MariaDB 10.5.15
- CVE-2021-46663: MariaDB 10.5.15
- CVE-2021-46662: MariaDB 10.5.13
- CVE-2021-46661: MariaDB 10.5.15
- CVE-2021-46659: MariaDB 10.5.14
- CVE-2021-46658: MariaDB 10.5.12
- CVE-2021-46657: MariaDB 10.5.11
- CVE-2021-35604: MariaDB 10.5.13
- CVE-2021-27928: MariaDB 10.5.9
- CVE-2021-2389: MariaDB 10.5.12
- CVE-2021-2372: MariaDB 10.5.12
- CVE-2021-2194: MariaDB 10.5.7
- CVE-2021-2166: MariaDB 10.5.10
- CVE-2021-2154: MariaDB 10.5.10
- CVE-2021-2022: MariaDB 10.5.5
- CVE-2020-28912: MariaDB 10.5.7
- CVE-2020-15180: MariaDB 10.5.6
- CVE-2020-14812: MariaDB 10.5.7
- CVE-2020-14789: MariaDB 10.5.7
- CVE-2020-14776: MariaDB 10.5.7
- CVE-2020-14765: MariaDB 10.5.7
- CVE-2018-25032: MariaDB 10.5.17

Resources

- 10.5 and beyond (video presentation by Sergei Golubchik)

List of All MariaDB 10.5 Releases

Date	Release	Status	Release Notes	Changelog
13 Nov 2023	**MariaDB 10.5.23**	Stable (GA)	Release Notes	Changelog
14 Aug 2023	**MariaDB 10.5.22**	Stable (GA)	Release Notes	Changelog
7 Jun 2023	**MariaDB 10.5.21**	Stable (GA)	Release Notes	Changelog
10 May 2023	**MariaDB 10.5.20**	Stable (GA)	Release Notes	Changelog

6 Feb 2023	**MariaDB 10.5.19**	Stable (GA)	Release Notes	Changelog
7 Nov 2022	**MariaDB 10.5.18**	Stable (GA)	Release Notes	Changelog
15 Aug 2022	**MariaDB 10.5.17**	Stable (GA)	Release Notes	Changelog
20 May 2022	**MariaDB 10.5.16**	Stable (GA)	Release Notes	Changelog
12 Feb 2022	**MariaDB 10.5.15**	Stable (GA)	Release Notes	Changelog
9 Feb 2022	**MariaDB 10.5.14**	Stable (GA)	Release Notes	Changelog
8 Nov 2021	**MariaDB 10.5.13**	Stable (GA)	Release Notes	Changelog
6 Aug 2021	**MariaDB 10.5.12**	Stable (GA)	Release Notes	Changelog
23 Jun 2021	**MariaDB 10.5.11**	Stable (GA)	Release Notes	Changelog
7 May 2021	**MariaDB 10.5.10**	Stable (GA)	Release Notes	Changelog
22 Feb 2021	**MariaDB 10.5.9**	Stable (GA)	Release Notes	Changelog
11 Nov 2020	**MariaDB 10.5.8**	Stable (GA)	Release Notes	Changelog
3 Nov 2020	**MariaDB 10.5.7**	Stable (GA)	Release Notes	Changelog
7 Oct 2020	**MariaDB 10.5.6**	Stable (GA)	Release Notes	Changelog
10 Aug 2020	**MariaDB 10.5.5**	Stable (GA)	Release Notes	Changelog
24 Jun 2020	**MariaDB 10.5.4**	Stable (GA)	Release Notes	Changelog
12 May 2020	**MariaDB 10.5.3**	RC	Release Notes	Changelog
26 Mar 2020	**MariaDB 10.5.2**	Beta	Release Notes	Changelog
14 Feb 2020	**MariaDB 10.5.1**	Beta	Release Notes	Changelog
3 Dec 2019	**MariaDB 10.5.0**	Alpha	Release Notes	Changelog

7.0.7.2 Release Notes - MariaDB 10.5 Series

MariaDB 10.5.23 Release Notes
Status: Stable (GA) | Release Date: 13 Nov 2023

MariaDB 10.5.22 Release Notes
Status: Stable (GA) | Release Date: 14 Aug 2023

MariaDB 10.5.21 Release Notes
Status: Stable (GA) | Release Date: 7 Jun 2023

MariaDB 10.5.20 Release Notes
Status: Stable (GA) | Release Date: 10 May 2023

MariaDB 10.5.19 Release Notes
Status: Stable (GA) | Release Date: 6 Feb 2023

MariaDB 10.5.18 Release Notes
Status: Stable (GA) | Release Date: 7 Nov 2022

MariaDB 10.5.17 Release Notes
Status: Stable (GA) | Release Date: 15 Aug 2022

MariaDB 10.5.16 Release Notes
Status: Stable (GA) | Release Date: 20 May 2022

MariaDB 10.5.15 Release Notes
Status: Stable (GA) | Release Date: 12 Feb 2022

MariaDB 10.5.14 Release Notes
Status: Stable (GA) | Release Date: 9 Feb 2022

MariaDB 10.5.13 Release Notes
Status: Stable (GA) | Release Date: 8 Nov 2021

MariaDB 10.5.12 Release Notes
Status: Stable (GA) | Release Date: 6 Aug 2021

MariaDB 10.5.11 Release Notes
Status: Stable (GA) | Release Date: 23 Jun 2021

MariaDB 10.5.10 Release Notes
Status: Stable (GA) | Release Date: 7 May 2021

MariaDB 10.5.9 Release Notes
Status: Stable (GA) | Release Date: 22 Feb 2021

MariaDB 10.5.8 Release Notes
Status: Stable (GA) | Release Date: 11 Nov 2020

MariaDB 10.5.7 Release Notes
Status: Stable (GA) | Release Date: 3 Nov 2020

MariaDB 10.5.6 Release Notes
Status: Stable (GA) | Release Date: 7 Oct 2020

MariaDB 10.5.5 Release Notes
Status: Stable (GA) | Release Date: 10 Aug 2020

MariaDB 10.5.4 Release Notes
Status: Stable (GA) | Release Date: 24 Jun 2020

MariaDB 10.5.3 Release Notes
Status: RC | Release Date: 12 May 2020

MariaDB 10.5.2 Release Notes
Status: Beta | Release Date: 26 Mar 2020

MariaDB 10.5.1 Release Notes
Status: Beta | Release Date: 14 Feb 2020

MariaDB 10.5.0 Release Notes
Status: Alpha | Release Date: 3 Dec 2019

7.0.7.2.1 MariaDB 10.5.23 Release Notes

Download | Release Notes | Changelog | Overview of 10.5

Alternate download from mariadb.org

Release date: 13 Nov 2023

MariaDB 10.5 is a previous *stable* series of MariaDB, maintained until June 2025. It is an evolution of MariaDB 10.4 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.5.23 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.5 see the What is MariaDB 10.5? page.

Thanks, and enjoy MariaDB!

Notable Items

InnoDB

- DROP INDEX followed by CREATE INDEX may corrupt data (MDEV-32132)
- ROW_FORMAT=COMPRESSED table: InnoDB: 2048 bytes should have been read. Only 0 bytes read. (MDEV-31875)
- Server aborts during alter operation when table doesn't have foreign index (MDEV-32527)
- row_merge_fts_doc_tokenize() handles FTS plugin parser inconsistently (MDEV-32578)
- InnoDB: tried to purge non-delete-marked record of an index on a virtual column prefix (MDEV-30024)
- lock_row_lock_current_waits counter in information_schema.innodb_metrics may become negative (MDEV-30658)
- SET GLOBAL innodb_max_purge_lag_wait=… hangs if innodb_read_only=ON (MDEV-31813)
- Auto-increment no longer works for explicit FTS_DOC_ID (MDEV-32017)
- Assertion `pos < table->n_def' failed in dict_table_get_nth_col (MDEV-32337)
- innochecksum man pages seem to be inconsistent with the binary (10.2.25) (MDEV-20583)
- innodb_compression_algorithm=0 (none) increments Innodb_num_pages_page_compression_error (MDEV-30825)
- wrong table name in innodb's "row too big" errors (MDEV-32128)
- Optimize is_file_on_ssd() to speedup opening tablespaces on Windows (MDEV-32228)
- Race condition between page write completion and log checkpoint (MDEV-32511)
- After crash recovery, Checksum mismatch + Failing assertion: !i || prev_id + 1 == space_id, (MDEV-31851)
- Deadlock due to log_free_check(), involving trx_purge_truncate_rseg_history() and trx_undo_assign_low() (MDEV-32049)
- Write-ahead logging is broken for freed pages (MDEV-32552)
- X-lock on supremum for prepared transaction for RR (MDEV-30165)

Optimizer

- Crash when HAVING in a correlated subquery references columns in the outer query (MDEV-29731)
- Server crashes at TABLE::add_tmp_key (MDEV-32320)
- Server crashes inside filesort at my_decimal::to_binary (MDEV-32324)
- Assertion `bitmap_is_set(&m_part_info->read_partitions, m_part_spec.start_part)' failed in ha_partition::handle_ordered_index_scan (MDEV-24283)
- Crash when searching for the best split of derived table (MDEV-32064)
- Test case from opt_tvc.test fails with statement memory protection (MDEV-32225)
- Significant slowdown for query with many outer joins (MDEV-32351)
- test_if_skip_sort_order() should catch the join types JT_EQ_REF, JT_CONST and JT_SYSTEM and skip sort order for these (MDEV-32475)

Replication

- rpl.rpl_parallel_temptable failure due to incorrect commit optimization of temptables (MDEV-10356)
- Lock wait timeout with INSERT-SELECT, autoinc, and statement-based replication (MDEV-31482)
- strings/ctype-ucs2.c:2336: my_vsnprintf_utf32: Assertion `(n % 4) == 0' failed in my_vsnprintf_utf32 on INSERT (MDEV-32249)
- Assertion fails in MDL_context::acquire_lock upon parallel replication of CREATE SEQUENCE (MDEV-31792)
- SHOW SLAVE STATUS Last_SQL_Errno Race Condition on Errored Slave Restart (MDEV-31177)
- seconds_behind_master is inaccurate for Delayed replication (MDEV-32265)
- detailize the semisync replication magic number error (MDEV-32365)
- Parallel replication deadlock victim preference code errorneously removed (MDEV-31655)

Galera

- Assertion `state() == s_executing || state() == s_prepared || state() == s_committing || state() == s_must_abort || state() == s_replaying' failed. (MDEV-24912)
- Assertion `state() == s_executing || state() == s_preparing || state() == s_prepared || state() == s_must_abort || state() == s_aborting || state() == s_cert_failed || state() == s_must_replay' failed (MDEV-31285)
- wsrep_sst_mariabackup not working on FreeBSD (MDEV-31467)
- Galera library 26.4.16 fails with every server version (MDEV-32024)
- Galera node remains paused after interleaving FTWRLs (MDEV-32282)
- Failed to insert streaming client (MDEV-32051)
- When set at runtime, wsrep_sst_method accepts any value (MDEV-31470)
- galera needs packaging script changes to successfully build (MDEV-32642)
- replication breaks when using optimistic replication and replica is a galera node (MDEV-31833)
- McAfee database vulnerability scan caused MariaDB crash with signal 6 (system abort) (MDEV-27004)

Data Definition

- MariaDB crash on calling function (MDEV-23902)

- ASAN errors in grn_obj_unlink / ha_mroonga::clear_indexes upon index operations (MDEV-31970)
- vcol circular references lead to stack overflow (MDEV-31112)

Scripts and Clients

- mariadb-binlog -T/--table (mysqlbinlog) option (MDEV-25369)
- mariadb-admin (mysqladmin) wrong error with simple_password_check (MDEV-22418)
- mariadb-install-db shows warning on missing directory $pamtooldir/auth_pam_tool_dir (MDEV-32142)
- main.mysql_client_test, main.mysql_client_test_comp failed on ASAN build with error: 5888, status: 23, errno: 2 (MDEV-19369)
- mariadb-install-db (mysql_install_db) doesn't properly grant proxy privileges to all default root user accounts (MDEV-21194)

Tests

- main.events_stress or events.events_stress fails with view-protocol (MDEV-31455)
- main.delete_use_source fails (hangs) with view-protocol (MDEV-31457)
- main.sum_distinct-big and main.merge-big fail with timeout with view-protocol (MDEV-31465)
- main.secure_file_priv_win fails with 2nd execution PS protocol (MDEV-32023)
- Windows : mtr output on is messed up with large MTR_PARALLEL (MDEV-32387)
- main.mysql_client_test_comp failed in buildbot, error on exec (MDEV-16641)
- main.order_by_pack_big fails with view-protocol (MDEV-31460)

MariaBackup

- MariaBackup full backup failed with InnoDB: Failing assertion: success in storage/innobase/fil/fil0fil.cc line 657 (MDEV-18200)
- mbstream breaks page compression on XFS (MDEV-25734)

Character Sets, Data Types, Collations

- Prefix keys for CHAR work differently for MyISAM vs InnoDB (MDEV-30048)
- Inconsistent results of DISTINCT with NOPAD (MDEV-30050)
- Assertion `(length % 4) == 0' failed in my_lengthsp_utf32 on INSERT (MDEV-28835)
- Compressed varchar values lost on joins when sorting on columns from joined table(s) (MDEV-31724)
- UBSAN shift exponent X is too large for 64-bit type 'long long int' in sql/field.cc (MDEV-32226)
- Wrong bit encoding using COALESCE (MDEV-32244)

Spider

- Spider UBSAN runtime error: applying non-zero offset x to null pointer in st_spider_param_string_parse::restore_delims (MDEV-31117)
- Segfault when setting spider_delete_all_rows to 0 and delete all rows of a spider table, ASAN heap-use-after-free in spider_db_delete_all_rows (MDEV-31996)
- ASAN errors in spider_fields::free_conn_holder or spider_create_group_by_handler (MDEV-28998)
- ASAN: heap-buffer-overflow & stack-buffer-overflow in `spider_db_mbase_row::append_to_str` | SIGSEGV's in `memmove_avx_unaligned_erms` from memcpy in `Binary_string::q_append`, in `Static_binary_string::q_append` and `my_strntoull10rnd_8bit` | Unknown error 12801 (MDEV-29502)

General

- binlog_do_db option breaks importing sql dumps (MDEV-29989)
- Crashes in MDL_key::mdl_key_init with lower-case-table-names=2 (MDEV-32025)
- getting error 'Illegal parameter data types row and bigint for operation '+' ' when using ITERATE in a FOR..DO (MDEV-32275)
- Assertion `arena_for_set_stmt== 0' failed in LEX::set_arena_for_set_stmt upon SET STATEMENT (MDEV-17711)
- main.mysqlcheck fails on ARM with ASAN use-after-poison in my_mb_wc_filename (MDEV-26494)
- main.delayed fails with wrong error code or timeout when executed after main.deadlock_ftwrl (MDEV-27523)
- Assertion failed: !pfs->m_idle || (state == PSI_SOCKET_STATE_ACTIVE) (MDEV-28561)
- MyISAM wrong server status flags (MDEV-28820)
- Server crashes in check_sequence_fields upon CREATE TABLE .. SEQUENCE=1 AS SELECT .. (MDEV-29771)
- slow log Rows_examined out of range (MDEV-30820)
- " `rpm --setugids` " breaks PAM authentication (MDEV-30904)
- incorrect examined rows in case of stored function usage (MDEV-31742)

- Compilation failing on MacOS (unknown warning option -Wno-unused-but-set-variable) (MDEV-31890)
- Server crash upon inserting into Mroonga table with compressed column (MDEV-31966)
- hash unique corrupts index on virtual blobs (MDEV-32012)
- insert into an empty table fails with hash unique (MDEV-32015)
- Valgrind/MSAN warnings in dynamic_column_update_move_left (MDEV-32140)
- Memory leak showed in MDEV-6146 test suite (MDEV-32223)
- Test from subselect.test fails with statement memory protection (MDEV-32245)
- Memory leak when executing PS for query with IN subquery (MDEV-32369)
- Allow the setting of Auto_increment on FK referenced columns (MDEV-32018)
- mariadb-upgrade fails with sql_safe_updates = on (MDEV-29914)
- Assertion `!(thd->server_status & (1U | 8192U))' failed in MDL_context::release_transactional_locks (MDEV-32541)
- Information schema leaks table names and structure to unauthorized users (MDEV-32500)
- Missing CHACHA20-POLY1305 support in WolfSSL (MDEV-31653)
- incorrect error about cyclic reference about JSON type virtual column (MDEV-32586)
- Disable TLS v1.0 and 1.1 for MariaDB (MDEV-31369)
- Better indication of refusing to start because of ProtectHome (MDEV-25177)
- Database upgrade fails: slow_log table (MDEV-27757)
- myrocks_hotbackup.1 and test suite files installed when engine is disabled (MDEV-29993)
- client_ed25519.dll isn't inluded for HeidiSQL. (MDEV-31315)
- Assertion `!m_null_value' failed in int FixedBinTypeBundle<FbtImpl>::cmp_item_fbt::compare or in cmp_item_inet6::compare (MDEV-27207)
- type_test.type_test_double fails with 'NUMERIC_SCALE NULL' (MDEV-22243)
- LeakSanitizer errors in get_quick_select or Assertion `status_var.local_memory_used == 0 || !debug_assert_on_not_freed_memory' failed (MDEV-32476)
- Update signal hander user info more compassion and correct url (MDEV-32535)

Docker Official Images

- Invert single and double quotes for sql command definitions in healthcheck.sh due to failure under sql_mode=ANSI_QUOTES - contribution by Dominik Häckel
- healthcheck.sh --no-defaults behaviour was corrected - reported by Dominik Häckel
- Added /docker-entrypoint-init.d for tar{,compression} from mariadb-backup - instructions
- Refactor `docker_mariadb_init` in the entrypoint for extending the MariaDB image
- CIS failure due to world-writable directory /var/run/mysqld, added sticky bit - reported by @ollie1
- Add PROXY privileges for root@MARIADB_ROOT_HOST - reported by Matthieu Gusmini
- healthcheck.sh added --galera_online test, to match what the mariadb-operator does.

Security

- Fixes for the following security vulnerabilities :
  - CVE-2023-22084

Changelog

For a complete list of changes made in MariaDB 10.5.23, with links to detailed information on each push, see the changelog .

Contributors

For a full list of contributors to MariaDB 10.5.23, see the MariaDB Foundation release announcement .

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.7.2.2 MariaDB 10.5.22 Release Notes

Download | Release Notes | Changelog | Overview of 10.5

Alternate download from mariadb.org

Release date: 14 Aug 2023

MariaDB 10.5 is a previous *stable* series of MariaDB, maintained until June 2025. It is an evolution of MariaDB 10.4 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.5.22 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.5 see the What is MariaDB 10.5? page.

Thanks, and enjoy MariaDB!

Notable Items

General

- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.5 for Ubuntu 18.04 LTS "Bionic"
- mariadb-dump --force doesn't ignore error as it should (MDEV-31092)
- 280 Bytes lost in mysys/array.c, mysys/hash.c, sql/sp.cc, sql/sp.cc, sql/item_create.cc, sql/item_create.cc, sql/sql_yacc.yy:10748 when using oracle sql_mode (MDEV-26186)
- SQL/PL package body does not appear in `I_S.ROUTINES.ROUTINE_DEFINITION` (MDEV-30662)
- Unexpected result when combining DISTINCT, subselect and LIMIT (MDEV-28285)
- ROW variables do not get assigned from subselects (MDEV-31250)
- Crash after setting global session_track_system_variables to an invalid value (MDEV-25237)
- ODKU of non-versioning column inserts history row (MDEV-23100)
- UPDATE not working properly on transaction precise system versioned table (MDEV-25644)
- Assertion `` `const_item_cache == true' `` failed in `Item_func::fix_fields` (MDEV-31319)
- ANALYZE doesn't work with pushed derived tables (MDEV-29284)
- `get_partition_set` is never executed in `ha_partition::multi_range_key_create_key` due to bitwise & with 0 constant (MDEV-24712)
- Client can crash the server with a `mysql_list_fields("view")` call (MDEV-30159)
- `I_S.parameters` not immediatly changed updated after procedure change (MDEV-31064)

Character Sets, Data Types

- UBSAN: null pointer passed as argument 1, which is declared to never be null in `my_strnncoll_binary` on `SELECT ... COUNT` or `GROUP_CONCAT` (MDEV-28384)
- Possibly wrong result or Assertion `0' failed in `Item_func_round::native_op` (MDEV-23838)
- Assertion `` `(length % 4) == 0' `` failed in `my_lengthsp_utf32` on `SELECT` (MDEV-29019)
- UBSAN: negation of -X cannot be represented in type `'long long int'` ; cast to an unsigned type to negate this value to itself in `Item_func_mul::int_op` and `Item_func_round::int_op` (MDEV-30932)
- Assorted assertion failures in json_find_path with certain collations (MDEV-23187)

InnoDB

- innochecksum dies with Floating point exception (MDEV-31641)
- Deadlock with 3 concurrent DELETEs by unique key (MDEV-10962)
- innodb protection against dual processes accessing data insufficient (MDEV-31568)
- Assertion `` `!strcmp(index->table->name.m_name, "SYS_FOREIGN") || !strcmp(index->table->name.m_name, "SYS_FOREIGN_COLS")' `` failed in `btr_node_ptr_max_size` (MDEV-19216)
- `MODIFY COLUMN` can break FK constraints, and lead to unrestorable dumps (MDEV-31086)
- Recovery or backup failure after innodb_undo_log_truncate=ON (MDEV-31487)
- Assertion `'n & PENDING'` failed in `fil_space_t::set_needs_flush()` (MDEV-31442)
- `fil_node_open_file()` releases `fil_system.mutex` allowing other thread to open its file node (MDEV-31256)
- Freed data pages are not always being scrubbed (MDEV-31253)
- innodb_undo_log_truncate=ON fails to wait for purge of enough transaction history (MDEV-31355)
- SET GLOBAL innodb_undo_log_truncate=ON does not free space when no undo logs exist (MDEV-31382)
- innodb_read_ahead_threshold (linear read-ahead) does not work (MDEV-29967)
- `fil_ibd_create()` may hijack the file handle of an old file (MDEV-31347)
- innodb_undo_log_truncate=ON recovery results in a corrupted undo log (MDEV-31373)

- Foreign Key Constraint actions don't affect Virtual Column (MDEV-18114)

Aria

- Various crashes upon INSERT/UPDATE after changing Aria settings (MDEV-28054)
- Various crashes/asserts/corruptions when Aria encryption is enabled/used, but the encryption plugin is not loaded (MDEV-26258)

Spider

- SIGSEGV in `spider_db_open_item_field` and SIGSEGV in `spider_db_print_item_type`, on SELECT (MDEV-29447)
- Spider variables that double as table params overriding mechanism is buggy (MDEV-31524)

Optimizer

- Assertion `` `last_key_entry >= end_pos' `` failed in virtual bool `JOIN_CACHE_HASHED::put_record()` (MDEV-31348)
- Problem with open ranges on prefix blobs keys (MDEV-31800)
- Equal on two RANK window functions create wrong result (MDEV-20010)
- Recursive CTE execution is interrupted without errors or warnings (MDEV-31214)
- Assertion `` `s->table->opt_range_condition_rows <= s->found_records' `` failed in `apply_selectivity_for_table` (MDEV-31449)
- Inconsistency between MRR and SQL layer costs can cause poor query plan (MDEV-31479)
- `MAX_SEL_ARG` memory exhaustion is not visible in the optimizer trace (MDEV-30964)
- SHOW TABLES not working properly with `lower_case_table_names=2` (MDEV-30765)
- Segfault on select query using index for group-by and filesort (MDEV-30143)
- Server crash in `store_length`, assertion failure in `Type_handler_string_result::sort_length` (MDEV-31743)

Replication

- Parallel Slave SQL Thread Can Update Seconds_Behind_Master with Active Workers (MDEV-30619)
- ALTER SEQUENCE ends up in optimistic parallel slave binlog out-of-order (MDEV-31503)
- STOP SLAVE takes very long time on a busy system (MDEV-13915)
- On slave XA COMMIT/XA ROLLBACK fail to return an error in read-only mode (MDEV-30978)
- `rpl.rpl_manual_change_index_file` occasionally fails in BB with Result length mismatch (MDEV-30214)

Galera

- Node has been dropped from the cluster on Startup / Shutdown with async replica (MDEV-31413)
- KILL QUERY maintains nodes data consistency but breaks GTID sequence (MDEV-31075)
- MariaDB stuck on starting commit state (waiting on commit order critical section) (MDEV-29293)
- Assertion `state() == s_aborting || state() == s_must_replay` failed in int wsrep::transaction::after_rollback() (MDEV-30013)
- Assertion `!wsrep_has_changes(thd) || (thd->lex->sql_command == SQLCOM_CREATE_TABLE && !thd->is_current_stmt_binlog_format_row()) || thd->wsrep_cs().transaction().state() == wsrep::transaction::s_aborted` failed (MDEV-30388)
- Server crashes when wsrep_sst_donor and wsrep_cluster_address set to NULL (MDEV-28433)
- Create temporary sequence can cause inconsistency (MDEV-31335)
- Galera 4 unable to query cluster state if not primary component (MDEV-21479)

Changelog

For a complete list of changes made in MariaDB 10.5.22, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.5.22, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.7.2.3 MariaDB 10.5.21 Release Notes

The most recent release of MariaDB 10.5 is:
MariaDB 10.5.23 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.5.21 | Release Notes | Changelog | Overview of 10.5

Release date: 7 Jun 2023

MariaDB 10.5 is a previous *stable* series of MariaDB, maintained until June 2025. It is an evolution of MariaDB 10.4 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.5.21 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.5 see the What is MariaDB 10.5? page.

Thanks, and enjoy MariaDB!

Notable Items

InnoDB

- Server crashes in st_join_table::choose_best_splitting (MDEV-31403)
- Crash with condition pushable into derived and containing outer reference (MDEV-31240)
- InnoDB does not free UNDO after the fix of MDEV-30671 (MDEV-31234)
- Revert "MDEV-30473: Do not allow GET_LOCK() / RELEASE_LOCK() in cluster"

Optimizer

- Crash with condition pushable into derived and containing outer reference (MDEV-31403 MDEV-31240)
- Crash with EXPLAIN EXTENDED for multi-table update of system table (MDEV-31224)

Changelog

For a complete list of changes made in MariaDB 10.5.21, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.5.21, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.7.2.4 MariaDB 10.5.20 Release Notes

The most recent release of MariaDB 10.5 is:
MariaDB 10.5.23 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.5.20 | Release Notes | Changelog | Overview of 10.5

Release date: 10 May 2023

MariaDB 10.5 is a previous *stable* series of MariaDB, maintained until June 2025. It is an evolution of MariaDB 10.4 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.5.20 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.5 see the What is MariaDB 10.5? page.

Thanks, and enjoy MariaDB!

Notable Items

InnoDB

- Crash on ROLLBACK in a ROW_FORMAT=COMPRESSED table (MDEV-30882)
- UNIQUE USING HASH accepts duplicate entries for tricky collations (MDEV-30034)
- rec_get_offsets() is not optimal (MDEV-30567)
- Performance regression in fil_space_t::try_to_close() introduced in MDEV-23855 (MDEV-30775)
- InnoDB recovery hangs when buffer pool ran out of memory (MDEV-30551)
- InnoDB undo log truncation fails to wait for purge of history (MDEV-30671)
- Fix miscount of doublewrites by Innodb_data_written (MDEV-31124)

Backup

- mariadb-backup doesn't utilise innodb-undo-log-directory (if specified as a relative path) during copy-back operation (MDEV-28187)
- mariabackup issues error messages during InnoDB tablespaces export on partial backup preparing (MDEV-29050)
- mariadb-backup does not copy Aria logs if aria_log_dir_path is used (MDEV-30968)
- Race condition between buffer pool flush and log file deletion in mariadb-backup --prepare (MDEV-30860)

Replication

- Fixed a deadlock on parallel slave involving full image Write event on the sequence engine (MDEV-29621)
- Fixed an attempted out-of-order binlogging error on slave involving ALTER on the sequence engine (MDEV-31077)
- Corrected non-versioned master to versioned slave replication on no-unique attribute table (MDEV-30430)
- Mended encrypted binlog master to error out to gtid-mode slave when master could not decrypt a binlog file (MDEV-28798)
- Refined optimistic parallel slave to error-exit without any hang (MDEV-30780)
- Ensured SHOW-SLAVE-STATUS is processed on the parallel slave having a necessary mutex always intialized (MDEV-30620)
- Fixed the slave applier to report a correct error when gtid_slave_pos insert fails for some (engine) reasons (MDEV-31038)

Optimizer

- Split Materialized optimization is improved to re-fill the materialized table only if necessary. The fewer number of table refills is taken into account when choosing query plan, too (MDEV-26301).
- Queries using `SELECT DISTINCT some_expression(aggregate_function())` could produce wrong query result. (MDEV-20057)
- A GROUP BY query with `MIN(primary_key)` in select list and `primary_key<>const` in the WHERE could

produce wrong result when executed with "Using index for group-by" strategy (MDEV-30605)
- EXPLAIN could erroneously report that Rowid Filter optimization is used for partitioned tables. Partitioned tables do not support it. (MDEV-30596)
- A bug in selectivity computations for SINGLE/DOUBLE_PREC_HB histograms could cause wrong estimates to be produced. This could cause the optimizer to pick sub-optimal query plans (MDEV-31067).

Docker Official Images

- Add replication setup to containers contributed by Md Sahil (MDEV-29762)

Security

- Fixes for the following security vulnerabilities:
  - CVE-2022-47015

Changelog

For a complete list of changes made in MariaDB 10.5.20, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.5.20, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.7.2.5 MariaDB 10.5.19 Release Notes

Download | Release Notes | Changelog | Overview of 10.5

Release date: 6 Feb 2023

MariaDB 10.5 is a previous *stable* series of MariaDB, maintained until June 2025. It is an evolution of MariaDB 10.4 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.5.19 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.5 see the What is MariaDB 10.5? page.

Thanks, and enjoy MariaDB!

Notable Items

- As mentioned in the 10.5.18 release notes, our Yum/DNF/Zypper repositories for Red Hat Enterprise Linux, CentOS, Fedora, openSUSE, and SUSE are changing **with this release** to being signed with a new GPG key with SHA2 digest algorithms instead of SHA1. See this blog post and the GPG page for more details.

InnoDB

- Full-text index corruption with system versioning (MDEV-25004)
- innodb_undo_log_truncate=ON recovery and backup fixes (MDEV-29999, MDEV-30179, MDEV-30438)
- Upgrade after a crash is not supported (MDEV-24412)
- Remove InnoDB buffer pool load throttling (MDEV-25417)

- InnoDB shutdown hangs when the change buffer is corrupted (MDEV-30009)
- `innodb_fast_shutdown=0` fails to report change buffer merge progress (MDEV-29984)

Galera

- Fixes for cluster wide write conflict resolving (MDEV-29684)

Replication

- Parallel slave applying in binlog order is corrected for admin class of commands including ANALYZE (MDEV-30323)
- Seconds_Behind_Master is now shown now more precisely at the slave applier start, including in the delayed mode (MDEV-29639)
- `mysqlbinlog --verbose` is made to show the type of compressed columns (MDEV-25277)
- Deadlock is resolved on replica involving `BACKUP STAGE BLOCK_COMMIT` and a committing user XA (MDEV-30423)

JSON

- JSON_PRETTY added as an alias for JSON_DETAILED (MDEV-19160)

General

- Infinite sequence of recursive calls when processing embedded CTE (MDEV-30248)
- Crash with a query containing nested WINDOW clauses (MDEV-30052)
- Major performance regression with 10.6.11 (MDEV-29988)
- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.5 for Fedora 35.
- In this release repositories for Fedora 37 have been added.

Changelog

For a complete list of changes made in MariaDB 10.5.19, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.5.19, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.7.2.6 MariaDB 10.5.18 Release Notes

The most recent release of MariaDB 10.5 is:
MariaDB 10.5.23 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.5.18 | Release Notes | Changelog | Overview of 10.5

Release date: 7 Nov 2022

MariaDB 10.5 is a previous *stable* series of MariaDB, maintained until June 2025. It is an evolution of MariaDB 10.4 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.5.18 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.5 see the What is MariaDB 10.5? page.

Thanks, and enjoy MariaDB!

Notable Items

SSL

- The server no longer tolerates incorrectly configured SSL (MDEV-29811). If you have enabled SSL in `my.cnf` but have not configured it properly (for example, a certificate file is missing), MariaDB used to silently disable SSL, leaving you under impression that everything was fine and connections were secure. Since this release, MariaDB will fail to start if SSL is enabled, but cannot be switched on.

Backup

- mariabackup --compress hangs (MDEV-29043)
- Assertion on info.page_size failed in xb_delta_open_matching_space (MDEV-18589)

InnoDB

- InnoDB unnecessarily extends data files (MDEV-13013)
- Adaptive hash index MDEV-27700, MDEV-29384
- MVCC and locking MDEV-29666, MDEV-27927
- Virtual columns MDEV-29299, MDEV-29753
- InnoDB crash recovery fixes (MDEV-29438, MDEV-29475)
- InnoDB crash recovery fixes (MDEV-29559)
- MVCC and locking (MDEV-28709)
- Race condition between KILL and transaction commit (MDEV-29368)

Galera

- Galera updated to 26.4.13
- Galera server crashes after 10.3 > 10.4 upgrade (MDEV-29375)
- wsrep_incoming_addresses status variable prints 0 as port number if the port is not mentioned in wsrep_node_incoming_address system variable (MDEV-28868)

JSON

- JSON_VALUE() does not parse NULL properties properly (MDEV-27151)

Replication

- minor correction in unsafe warning message (MDEV-28827)
- False replication error-stop of REVOKE PRIVILEGES from a non-existing user on primary (MDEV-28530) in combination with a filtering replica is corrected
- SET DEFAULT ROLE replication is mended on a replica that filters system tables (MDEV-28294)
- XA COMMIT is not binlogged when the XA transaction has not updated any transaction engine (MDEV-25616)
- Concurrent CREATE TRIGGER statements made to binlog without any mixup (MDEV-25606)

Repositories

- Beginning with the next release (Q1 2023), our Yum, DNF, and Zypper repositories for Red Hat, Fedora, and SUSE will be migrated to being signed with a new GPG key. The key we are migrating to is the same one we already use for our Debian and Ubuntu Repositories.
  - The short Key ID is: `0xC74CD1D8`
  - The long Key ID is: `0xF1656F24C74CD1D8`
  - The full fingerprint of the key is: `177F 4010 FE56 CA33 3630 0305 F165 6F24 C74C D1D8`
  - The key can be imported now in preparation for this change using the following command:

```
sudo rpm --import https://supplychain.mariadb.com/MariaDB-Server-GPG-KEY
```

Docker Official Image

The following changes have been made to the `docker.io/library/mariadb` container image.

- The number of gpg packages packages has been removed, leaving enough to `apt-get update`, but `dirmngr` that would fetch keys has been removed. (inspired by issue #469)
- The environment variable `LANG=C.UTF-8` has been added for those that exec into containers and copy paste UTF8 characters (fixes issue #468).
- Adds OCI labels to image (fixes issue 436 and users need for version)
- MariaDB config: skip-host-cache and skip-name-resolve moved to `/etc/mysql/mariadb.conf.d/05-skipcache.cnf`

Changelog

For a complete list of changes made in MariaDB 10.5.18, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.5.18, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.7.2.7 MariaDB 10.5.17 Release Notes

The most recent release of MariaDB 10.5 is:
MariaDB 10.5.23 Stable (GA) Download Now

Alternate download from mariadb.org

Download 10.5.17 | Release Notes | Changelog | Overview of 10.5

Release date: 15 Aug 2022

MariaDB 10.5 is a previous *stable* series of MariaDB. It is an evolution of MariaDB 10.4 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.5.17 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.5 see the What is MariaDB 10.5? page.

Thanks, and enjoy MariaDB!

Notable Items

InnoDB

- InnoDB corruption due to lack of file locking (MDEV-28495)
- FULLTEXT search with apostrophe, and mandatory words (MDEV-20797)
- ALTER TABLE IMPORT TABLESPACE corrupts an encrypted table (MDEV-28779)

- ALTER TABLE wrong-result fix (MDEV-26294)
- Crash recovery fixes (MDEV-28668, MDEV-28731)

Replication

- explicit_defaults_for_timestamp is stored in binlog, so that `CREATE TABLE` on slave would always have the same effect as on master. (MDEV-29078)
- ER_SLAVE_INCIDENT error is specified now on slave to be seen with SHOW-SLAVE-STATUS (MDEV-21087)
- INCIDENT_EVENT is no longer binlogged when a being logged transaction can be safely rolledback (MDEV-21443)
- sequences related row-format events are made to correspond to binlog_row_image (MDEV-28487)
- Possible reason of FLUSH BINARY LOGS hang is eliminated (MDEV-28948)

Galera

- Possible to write/update with read_only=ON and not a SUPER privilege (MDEV-28546)
- Node crashes with Transport endpoint is not connected mysqld got signal 6 (MDEV-25068)
- Galera4 not able to report proper wsrep_incoming_addresses (MDEV-20627)
- Galera should replicate nextval()-related changes in sequences with INCREMENT <> 0, at least NOCACHE ones with engine=InnoDB (MDEV-27862)

Optimizer

- Server crash in JOIN_CACHE::free or in copy_fields (MDEV-23809)
  - Queries that use DISTINCT and an always-constant function like COLLATION(aggegate_func(...)) could cause a server crash. Note that COLLATION() is a special function - its value is constant even if its argument is not costant.
- Crash when using ANY predicand with redundant subquery in GROUP BY clause (MDEV-29139)
  - A query with a subuquery in this form could cause a crash:

```
... ANY (SELECT ... GROUP BY (SELECT redundant_subselect_here)) ...
```

- MariaDB Server SEGV on INSERT .. SELECT (MDEV-26427)
  - Certain queries in form "INSERT ... SELECT with_aggregate_or_window_func" could cause a crash.
- restore_prev_nj_state() doesn't update cur_sj_inner_tables correctly (MDEV-28749)
  - Subquery semi-join optimization could miss LooseScan or FirstMatch strategies for certain queries.
- Optimizer uses all partitions after upgrade to 10.3 (MDEV-28246)
  - For multi-table UPDATE or DELETE queries, the optimizer failed to apply Partition Pruning optimization for the table that is updated or deleted from.
- Range optimizer regression for key IN (const, ...) (MDEV-25020)
  - The issue can be observed on MariaDB 10.5.9 and later versions which have the fix for MDEV-9750. That fix introduceds optimizer_max_sel_arg_weight.
  - If one sets optimizer_max_sel_arg_weight to a very high value or zero (which means "unlimited") and runs queries that produce heavy-weight graphs, they can observe a performance slowdown, e.g.:

```
table.keyXpartY [NOT] IN ( ... )
```

- Wrong result with table elimination combined with not_null_range_scan (MDEV-28858)
  - If one runs with optimizer_switch='not_null_range_scan=on' (which is not enabled by default), a query that does a join and has const tables could produce a wrong result.

CONNECT

- CONNECT Engine now supports INSERT IGNORE with Mysql Table type (MDEV-27766)

mariadb Client

- New mariadb client option, `-enable-cleartext-plugin` . Option does not do anything, and is for MySQL-compatibility purposes only.

General

- explicit_defaults_for_timestamp now also has a session scope, not only global (MDEV-29225)
- MariaDB can be built with OpenSSL 3.0

- HELP was updated to include the latest content
- Crash in JSON_EXTRACT (MDEV-29188)
- ALTER TABLE ALGORITHM=NOCOPY does not work after upgrade (MDEV-28727)
- Server crash upon CREATE VIEW with unknown column in ON condition (MDEV-29088)
- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.5 for Debian 10 "Buster" for ppc64el

Security

- Fixes for the following security vulnerabilities:
  - CVE-2023-5157
  - CVE-2022-32082
  - CVE-2022-32089
  - CVE-2022-32081
  - CVE-2018-25032
  - CVE-2022-32091
  - CVE-2022-32084
  - CVE-2022-38791

Changelog

For a complete list of changes made in MariaDB 10.5.17, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.5.17, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.7.2.8 MariaDB 10.5.16 Release Notes

The most recent release of MariaDB 10.5 is:
MariaDB 10.5.23 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.5.16 | Release Notes | Changelog | Overview of 10.5

Release date: 20 May 2022

MariaDB 10.5 is a previous *stable* series of MariaDB. It is an evolution of MariaDB 10.4 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.5.16 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.5 see the What is MariaDB 10.5? page.

Thanks, and enjoy MariaDB!

Notable Items

InnoDB

- innodb_disallow_writes removed (MDEV-25975)
- InnoDB gap locking fixes (MDEV-20605, MDEV-28422)
- InnoDB performance improvements (MDEV-27557, MDEV-28185)

Replication

- Server initialization time gtid_slave_pos purge related reason of crashing in binlog background thread is removed (MDEV-26473)
- Shutdown of the semisync master can't produce inconsistent state anymore (MDEV-11853)
- Binlogs disappear after rsync IST (MDEV-28583)
- master crash is eliminated in compressed semisync replication protocol with packet counting amendment (MDEV-25580)
- OPTIMIZE on a sequence does not cause counterfactual ER_BINLOG_UNSAFE_STATEMENT anymore (MDEV-24617)
- Automatically generated Gtid_log_list_event is made to recognize within replication event group as a formal member (MDEV-28550)
- Replication unsafe INSERT .. ON DUPLICATE KEY UPDATE using two or more unique key values at a time with MIXED format binlogging is corrected (MDEV-28310)
- Replication unsafe INSERT .. ON DUPLICATE KEY UPDATE stops issuing unnecessary "Unsafe statement" with MIXED binlog format (MDEV-21810)
- Incomplete replication event groups are detected to error out by the slave IO thread (MDEV-27697)
- mysqlbinlog --stop-never --raw now flushes the result file to disk after each processed event so the file can be listed with the actual bytes (MDEV-14608)

Backup

- Incorrect binlogs after Galera SST using rsync and mariabackup (MDEV-27524)
- mariabackup does not detect multi-source replication slave (MDEV-21037)
- Useless warning "InnoDB: Allocated tablespace ID <id> for <tablename>, old maximum was 0" during backup stage (MDEV-27343)
- mariabackup prepare fails for incrementals if a new schema is created after full backup is taken (MDEV-28446)

Optimizer

- A SEGV in Item_field::used_tables/update_depend_map_for_order... (MDEV-26402)
- ANALYZE FORMAT=JSON fields are incorrect for UNION ALL queries (MDEV-27699)
- Subquery in an UPDATE query uses full scan instead of range (MDEV-22377)
- Assertion `item1->type() == Item::FIELD_ITEM ... (MDEV-19398)
- Server crashes in Expression_cache_tracker::fetch_current_stats (MDEV-28268)
- MariaDB server crash at Item_subselect::init_expr_cache_tracker (MDEV-26164, MDEV-26047)
- Crash with union of my_decimal type in ORDER BY clause (MDEV-25994)
- SIGSEGV in st_join_table::cleanup (MDEV-24560)
- Assertion `!eliminated' failed in Item_subselect::exec (MDEV-28437)

General

- Server error messages are now available in Chinese (MDEV-28227)
- For RHEL/CentOS 7, non x86_64 architectures are no longer supported upstream and so our support will also be dropped with this release
- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.5 for Debian 9 "Stretch", Ubuntu 21.10 "Impish", and Fedora 34

Security

- Fixes for the following security vulnerabilities:
  - CVE-2021-46669
  - CVE-2022-27376
  - CVE-2022-27377
  - CVE-2022-27378
  - CVE-2022-27379
  - CVE-2022-27380
  - CVE-2022-27381
  - CVE-2022-27382

- CVE-2022-27383
- CVE-2022-27384
- CVE-2022-27386
- CVE-2022-27387
- CVE-2022-27444
- CVE-2022-27445
- CVE-2022-27446
- CVE-2022-27447
- CVE-2022-27448
- CVE-2022-27449
- CVE-2022-27451
- CVE-2022-27452
- CVE-2022-27455
- CVE-2022-27456
- CVE-2022-27457
- CVE-2022-27458
- CVE-2022-32087
- CVE-2022-32086
- CVE-2022-32085
- CVE-2022-32083
- CVE-2022-32088

Changelog

For a complete list of changes made in MariaDB 10.5.16, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.5.16, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.7.2.9 MariaDB 10.5.15 Release Notes

The most recent release of MariaDB 10.5 is:
MariaDB 10.5.23 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.5.15 | Release Notes | Changelog | Overview of 10.5

Release date: 12 Feb 2022

MariaDB 10.5 is a previous *stable* series of MariaDB. It is an evolution of MariaDB 10.4 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.5.15 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.5 see the What is MariaDB 10.5? page.

Thanks, and enjoy MariaDB!

Notable Items

- This release fixes a blocking problem with the MariaDB 10.5.14 release when manually running mariadb-upgrade. (MDEV-27789)
- See MariaDB 10.5.14 for other changes since the previous release.

InnoDB

- Set innodb_change_buffering=none by default (MDEV-27734)

Security

- Fixes for the following security vulnerabilities:
  - CVE-2021-46665
  - CVE-2021-46664
  - CVE-2021-46661
  - CVE-2021-46668
  - CVE-2021-46663

Changelog

For a complete list of changes made in MariaDB 10.5.15, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.5.15, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.7.2.10 MariaDB 10.5.14 Release Notes

The most recent release of MariaDB 10.5 is:
MariaDB 10.5.23 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.5.14 | Release Notes | Changelog | Overview of 10.5

Release date: 9 Feb 2022

This release is no longer available for download after a problem was noticed when manually running mariadb-upgrade. See MDEV-27789 for more details.

Please use a later release.

MariaDB 10.5 is a previous *stable* series of MariaDB. It is an evolution of MariaDB 10.4 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.5.14 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.5 see the What is MariaDB 10.5? page.

Thanks, and enjoy MariaDB!

Notable Items

InnoDB

- --skip-symbolic-links does not disallow `.isl` file creation (MDEV-26870)
- Indexed CHAR columns are broken with NO_PAD collations (MDEV-25440)
- insert-intention lock conflicts with waiting ORDINARY lock (MDEV-27025)
- Crash recovery improvements (MDEV-26784, MDEV-27022, MDEV-27183, MDEV-27610)

Galera

- Galera updated to 26.4.11
- Galera SST scripts should use ssl_capath (not ssl_ca) for CA directory (MDEV-27181)
- Alter Sequence do not replicate to another nodes with in Galera Cluster (MDEV-19353)
- Galera crash - Assertion. Possible parallel writeset problem (MDEV-26803)
- CREATE TABLE with FOREIGN KEY constraint fails to apply in parallel (MDEV-27276)
- Galera cluster node consider old server_id value even after modification of server_id `[wsrep_gtid_mode=ON]` (MDEV-26223)

Replication

- Seconds behind master corrected from artificial spikes at relay-log rotation (MDEV-16091)
- Statement rollback in binlog when transaction creates or drop temporary table is set right (MDEV-26833)
- CREATE-or-REPLACE SEQUENCE is made to binlog with the DDL flag to stabilize its parallel execution on slave (MDEV-27365)

Packaging & Misc

- prohibition running two upgrades in parallel (MDEV-27068, MDEV-27107, MDEV-27279)
- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.5 for Ubuntu 21.04 Hirsute, CentOS 8, and Fedora 33
- mariadb_repo_setup script updated to version 2022-02-08, with the following fixes and enhancements:
  - Default location of the script has been moved to: https://r.mariadb.com/downloads/mariadb_repo_setup (old location is deprecated, but still works)
  - The GPG keyring file, used with Debian and Ubuntu repositories, has moved to: https://supplychain.mariadb.com/mariadb-keyring-2019.gpg and the checksum for the file can be found at: https://supplychain.mariadb.com/mariadb-keyring-2019.gpg.sha256
  - Support for RHEL and SLES aarch64 repositories added
  - New function added to verify that the MariaDB Server version, if specified on the command line, follows the correct naming and that a corresponding repository actually exists.
  - Fixed repository pinning for Ubuntu and Debian repositories
  - MariaDB Server 10.7 is now the default server version

Docker Library

- Faster initialization by disabling binary logging during initialization (MDEV-27074)
- mysql_upgrade can be run if needed using the environment variable MARIADB_AUTO_UPGRADE=1 (MDEV-25670)
- A healthcheck script /usr/local/bin/healthcheck.sh is installed in the container with various checking options (MDEV-25434)
- mysql@localhost user is created with the environment variable MARIADB_MYSQL_LOCALHOST_USER=1 and additional grants (beyond USAGE) with MARIADB_MYSQL_LOCALHOST_GRANTS={global grant list} (MDEV-27732)
- skip innodb buffer pool loads/dumps on temporary startup/shutdown for faster startup/initialization, and accurate "healthcheck.sh --innodb_buffer_pool_loaded"
- change group ownership on datadir/socket dir (issue #401)
- log note about note on Securing system users, mysql_secure_installation not required (reddit suggestion)
- speed up Docker Library initialization of timezones (MDEV-27608, MDEV-23326)

Security

- Fixes for the following security vulnerabilities:

- CVE-2022-24052
- CVE-2022-24051
- CVE-2022-24050
- CVE-2022-24048
- CVE-2021-46659
- CVE-2022-0778
- CVE-2022-21595

Changelog

For a complete list of changes made in MariaDB 10.5.14, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.5.14, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.7.2.11 MariaDB 10.5.13 Release Notes

The most recent release of MariaDB 10.5 is:
MariaDB 10.5.23 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.5.13 | Release Notes | Changelog | Overview of 10.5

Release date: 8 Nov 2021

MariaDB 10.5 is a previous *stable* series of MariaDB. It is an evolution of MariaDB 10.4 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.5.13 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.5 see the What is MariaDB 10.5? page.

Thanks, and enjoy MariaDB!

Notable Items

Galera

- Fix for `WSREP: invalid state ROLLED_BACK (FATAL)` (MDEV-25114)

InnoDB

- `ALTER TABLE…IMPORT TABLESPACE` fixes (MDEV-18543, MDEV-20931, MDEV-26131, MDEV-26621)
- `innodb_undo_log_truncate` fixes (MDEV-26445, MDEV-26450, MDEV-26672, MDEV-26864)
- Page I/O performance fixes (MDEV-25215, MDEV-26547, MDEV-26626, MDEV-26819)
- Replication timeouts with `XA PREPARE` (MDEV-26682)

Replication

- Memory hogging on slave by ROW event applier is eliminated (MDEV-26712)
- `mysql --binary-mode` now properly handles `\\0` in data (MDEV-25444)
- Fixes race condition between `SHOW BINARY LOGS` and `RESET MASTER` (MDEV-20215)
- Missed statement rollback in case transaction drops or create temporary table is corrected (MDEV-26833)

Audit Plugin

- The `QUERY_DDL` server_audit_events setting now logs `CREATE/DROP [PROCEDURE / FUNCTION / USER]` statements. See MariaDB Audit Plugin - Log Settings. (MDEV-23457)

Packaging & Misc

- Session tracking flag in OK_PACKET (MDEV-26868)
- Some views force server (and mysqldump) to generate invalid SQL for their definitions (MDEV-26299)

Security

- Fixes for the following security vulnerabilities:
  - CVE-2021-35604
  - CVE-2021-46667
  - CVE-2021-46662
  - CVE-2022-27385
  - CVE-2022-27385
  - CVE-2022-31624

Changelog

For a complete list of changes made in MariaDB 10.5.13, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.5.13, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.7.2.12 MariaDB 10.5.12 Release Notes

The most recent release of MariaDB 10.5 is:
MariaDB 10.5.23 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.5.12 | Release Notes | Changelog | Overview of 10.5

Release date: 6 Aug 2021

Warning: This version can cause InnoDB file corruption on FreeBSD and on AIX. If you are using AIX, please, stick to an earlier release, or upgrade to a more recent release. If you are using FreeBSD, upgrade to the bugfix release (the version ends with _1) of the mariadb-server from the Ports Collection. See MDEV-26537.

MariaDB 10.5 is a previous *stable* series of MariaDB. It is an evolution of MariaDB 10.4 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.5.12 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.5 see the What is MariaDB 10.5? page.

Thanks, and enjoy MariaDB!

Notable Items

InnoDB

- InnoDB no longer acquires advisory file locks by default (MDEV-24393)
- Encryption: Automatically disable key rotation checks for file_key_management plugin (MDEV-14180)
- Some fixes from MySQL 5.7.35 (MDEV-26205)
- Fixed scrubbing on AIX (MDEV-26110)
- Improved page flushing performance (MDEV-25954, MDEV-25948, MDEV-25801, MDEV-25113, MDEV-26004)

Optimizer

- A query that uses ORDER BY .. LIMIT clause and "Range checked for each record optimization" could produce incorrect results under some circumstances (MDEV-25858)
- Queries that have more than 32 equality conditions comparing columns of different tables ("tableX.colX=tableY.colY) could cause a stack overrun in the query optimizer (MDEV-17783, MDEV-23937)
- "Condition pushdown into derived table" optimization cannot be applied if the expression being pushed refers to a derived table column which is computed from expression that has a stored function call, @session variable reference, or other similar construct. The fix for MDEV-25969 makes it so that only the problematic part of the condition is not pushed. The rest of the condition is now pushed. (MDEV-25969)
- A query with window function on the left side of the subquery could cause a crash. (MDEV-25630)
- Fixed the issue fixed in MySQL Bug #76803: DML or locking SELECT statements that use outer joins could produce this warning in the error log: `[ERROR] InnoDB: Unlock row could not find a 3 mode lock on the record.` (MDEV-26106)

Packaging & Misc

- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.5 for Ubuntu 20.10 Groovy
- Debian 11 Bullseye repositories added
- Galera updated to 26.4.9
- Linux on IBM Z (s390x) architecture added with releases on Ubuntu-20.04 Focal

Security

- Fixes for the following security vulnerabilities:
  - CVE-2021-2372
  - CVE-2021-2389
  - CVE-2021-46658

Changelog

For a complete list of changes made in MariaDB 10.5.12, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.5.12, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will

be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.7.2.13 MariaDB 10.5.11 Release Notes

The most recent release of MariaDB 10.5 is:
MariaDB 10.5.23 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.5.11 | Release Notes | Changelog | Overview of 10.5

Release date: 23 Jun 2021

MariaDB 10.5 is the current *stable* series of MariaDB. It is an evolution of MariaDB 10.4 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.5.11 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.5 see the What is MariaDB 10.5? page.

Thanks, and enjoy MariaDB!

Notable Items

This version of MariaDB is being released now to fix the following two regressions:

- Table alias from previous statement interferes later commands (MDEV-25672)
- Join using derived with aggregation returns incorrect results (MDEV-25714)

In addition to the above, this release also contains the following fixes:

InnoDB

- InnoDB spatial indexes miss large geometry fields after MDEV-25459 (MDEV-25758)
- Double free of transaction during truncate operation (MDEV-25663)
- Double free of table when inplace alter FTS add index fails (MDEV-25721)
- Potential hang in purge for virtual columns (MDEV-25664)
- Change buffer records are lost under heavy load (MDEV-25783)
- Not applying INSERT_REUSE_REDUNDANT (MDEV-25745)
- InnoDB recovery fails with `[ERROR] InnoDB: Not applying INSERT_REUSE_REDUNDANT due to corruption` (MDEV-25745)
- CHECK TABLE harvests InnoDB: Index 'abdcef' contains 10001 entries, should be 10000 (MDEV-25783)

Replication

- Do not replicate killed multi-table OPTIMIZE TABLE when the signal arrives before any table has been processed (MDEV-22530)
- Fix optimistic parallel applier to not deadlock on admin commands OPTIMIZE, REPAIR, and ANALYZE (MDEV-17515)
- Backport MDEV-20821 parallel slave server shutdown hang (MDEV-22370)
- Removed deprecated `--base64-output` to correct BINLOG clause in mysqlbinlog output (MDEV-25222)

Security

- Fixes for the following security vulnerabilities:
  - CVE-2021-46666
  - CVE-2021-46657

Changelog

For a complete list of changes made in MariaDB 10.5.11, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.5.11, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.7.2.14 MariaDB 10.5.10 Release Notes

The most recent release of MariaDB 10.5 is:
MariaDB 10.5.23 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.5.10 | Release Notes | Changelog | Overview of 10.5

Release date: 7 May 2021

MariaDB 10.5 is the current *stable* series of MariaDB. It is an evolution of MariaDB 10.4 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.5.10 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.5 see the What is MariaDB 10.5? page.

Thanks, and enjoy MariaDB!

Notable Changes

- ST_DISTANCE_SPHERE for calculating the spherical distance between two geometries (point or multipoint) on a sphere (MDEV-13467)
- Crash with invalid multi-table update of view in 2nd execution of SP (MDEV-24823)
- Incorrect name resolution for subqueries in ON expressions (MDEV-25362)
- Complex query in Store procedure corrupts results (MDEV-25182)
- `DELETE HISTORY` may delete current data on system-versioned table (MDEV-25468)
- Crashes with nested table value constructors (MDEV-22786)
- Server crashes in `thd_clear_errors()` (MDEV-23542)
- The statement set `password=password('')` executed in PS mode fails in case it is run by a user with expired password (MDEV-25197)

mariabackup

- RENAME TABLE causes "Ignoring data file" messages (MDEV-25568)

InnoDB

- Deprecated the `*innodb` and `*none` options in innodb_checksum_algorithm (MDEV-25106)
- MVCC read from index on CHAR or VARCHAR wrongly omits rows (MDEV-25459)
- Race conditions in persistent statistics (MDEV-10682, MDEV-18802, MDEV-25051)

- Sequence created by one connection remains invisible to another (MDEV-24545)
- innodb_flush_method=O_DIRECT fails on compressed tables (MDEV-25121)
- RESET MASTER hangs (MDEV-24302)
- InnoDB crash recovery fixes (MDEV-25031, MDEV-25110)

Replication

- Replication Heartbeat event was uncapable to cary 4GB+ offsets (MDEV-16146)
- FLUSH LOGS race against Binlog checkpoint event creation (MDEV-24526)
- slave_compressed_protocol did not work correctly with semi-sync (MDEV-24773)
- DROP TABLE should not cause "Query caused different errors on master and slave" on slave when it failed on master (MDEV-25530)
- Killing server during RESET MASTER may lose MyRocks transaction (MDEV-25305)

Galera

- Galera updated to 26.4.8
- SET PASSWORD command fail with wsrep api (MDEV-25258)
- Long BF log wait turns on InnoDB Monitor output without telling, never turns it off (MDEV-25319)
- Assertion `` `state_ == s_exec' `` failed in `wsrep::client_state::start_transaction` (MDEV-22227)
- Frequently Crashing Mariadb Cluster 10.4.18 (MDEV-24980)
- Signal 11 on `TABLE_LIST::placeholder()` (MDEV-24878)
- `ALTER TABLE` not replicated with Galera in MariaDB 10.5.9 (MDEV-24956)
- "Flush SSL" command doesn't reload wsrep cert (MDEV-22668)
- Avoid unnecessary rollbacks with SR (MDEV-25553)

Packaging & Misc

- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.5 for Ubuntu 16.04 Xenial and Fedora 32
- Ubuntu 21.04 Hirsute and Fedora 34 repositories added

Security

- Fixes for the following security vulnerabilities:
  - CVE-2021-2166
  - CVE-2021-2154
  - CVE-2022-21451

MongoDB protocol support files for the CONNECT engine are missing in this release.
If you want to use CONNECT engine with MongoDB, you need to download
Mongo2.jar or Mongo3.jar and put a path to this file into the `connect_class_path` in the `my.cnf`.

Changelog

For a complete list of changes made in MariaDB 10.5.10, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.5.10, see the MariaDB Foundation release announcement.

7.0.7.2.15 MariaDB 10.5.9 Release Notes

The most recent release of MariaDB 10.5 is:
MariaDB 10.5.23 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.5.9 | Release Notes | Changelog | Overview of 10.5

Release date: 22 Feb 2021

Last month long-time MariaDB VP of Engineering, Rasmus Johansson, passed due to complications from cancer. His loss has been felt keenly by the whole MariaDB team. Our thoughts are with his family during this difficult time and this release is dedicated to his memory.

MariaDB 10.5 is the current *stable* series of MariaDB. It is an evolution of MariaDB 10.4 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.5.9 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.5 see the What is MariaDB 10.5? page.

Thanks, and enjoy MariaDB!

Notable Changes

InnoDB

- MDEV-24188 - Hang in `buf_page_create()` after reusing a previously freed page
- MDEV-24275 - InnoDB persistent stats analyze forces full scan forcing lock crash
- MDEV-24449 - Corruption of system tablespace or last recovered page
- MDEV-24109 - InnoDB hangs with innodb_flush_sync=OFF
- MDEV-24537 - innodb_max_dirty_pages_pct_lwm=0 lost its special meaning
- Fixed bugs in the handling of freed pages - MDEV-24569, MDEV-24695, MDEV-24765, MDEV-24864
- MDEV-12227 - Defer writes to the InnoDB temporary tablespace

Galera

- Galera updated to 26.4.7
- MDEV-23328 - Server hang due to Galera lock conflict resolution
- MDEV-23851 - BF-BF Conflict issue because of UK GAP locks
- MDEV-20717 - Plugin system variables and activation options can break mysqld --wsrep_recover
- MDEV-24469 - Assertion `active() == false` failed with "XA START.."
- MDEV-23647 - Garbd can't initiate SST anymore in 10.5
- MDEV-25179 - wsrep_provider and wsrep_notify_cmd system variables are now read-only

Replication

- MDEV-8134 - relay-log is corrected to rotate past 999999
- MDEV-23033 - fixed slave applier for row-based events with FK constraints on virtual columns
- MDEV-4633 - Relay_Log_Space of Show-Slave-Status is made thread-safe
- MDEV-10272 - add master host/port info to slave thread exit messages
- MDEV-23846 - improves mysqlbinlog error message issuing
- MDEV-24087 - replication of S3 ALTER PARTITION corrected
- MDEV-23610 - New privilege REPLICA MONITOR (also accessible as SLAVE MONITOR)

ColumnStore

- MariaDB ColumnStore updated to 5.5.1
- MariaDB ColumnStore deb and rpm packages now have a version of `10.5.9-5.5.1` so one can see both the server

version (10.5.9) and the plugin version (5.5.1) without needing to check the Available Versions table in the ColumnStore docs
- The MariaDB ColumnStore plugin is no longer provided for 32-bit x86 (i386) builds

Misc

- MariaDB is fixed to build on the Apple M1 CPU
- MariaDB is fixed to build on AIX
- MDEV-24122 - anomalies in mysql.user tables on previously 5.7 MySQL versions corrected
- MDEV-24093 - Detect during mysql_upgrade if `type_mysql_json.so` is needed and load it
- Binary tarballs now use WolfSSL v4.6.0 and pcre2-10.36
- MDEV-23630 - mysqldump --system option
- Fixes for the following security vulnerabilities:
  - CVE-2021-27928

Changelog

For a complete list of changes made in MariaDB 10.5.9, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.5.9, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.7.2.16 MariaDB 10.5.8 Release Notes

The most recent release of MariaDB 10.5 is:
MariaDB 10.5.23 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.5.8 | Release Notes | Changelog | Overview of 10.5

Release date: 11 Nov 2020

MariaDB 10.5 is the current *stable* series of MariaDB. It is an evolution of MariaDB 10.4 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

For an overview of MariaDB 10.5 see the What is MariaDB 10.5? page.

Thanks, and enjoy MariaDB!

Notable Changes

- Out-of-cycle release to fix regressions in MariaDB 10.5.7
- Follow up to MDEV-19838 to alter protocol checks to support the following implementations (which add garbage to the end of some packets):
  - PHP PDO (all versions) (MDEV-24121)
  - mysqlnd (from PHP < 7.3) (MDEV-24121)
  - mysql-connector-python (all versions) (MDEV-24134)

    - and mysql-connector-java (all versions)
- Arbitrary InnoDB buffer pool and data file corruption (MDEV-24096)
- The query optimizer consumed a lot of memory when handling construct in form of `key_column [NOT] IN (large-list-of constants)` (MDEV-24117)

Changelog

For a complete list of changes made in MariaDB 10.5.8, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.5.8, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.7.2.17 MariaDB 10.5.7 Release Notes

The most recent release of MariaDB 10.5 is:
MariaDB 10.5.23 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.5.7 | Release Notes | Changelog | Overview of 10.5

Release date: 3 Nov 2020

MariaDB 10.5 is the current *stable* series of MariaDB. It is an evolution of MariaDB 10.4 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

For an overview of MariaDB 10.5 see the What is MariaDB 10.5? page.

Thanks, and enjoy MariaDB!

Notable Changes

This release introduced an InnoDB data corruption bug (MDEV-24096). If any InnoDB tables contain indexed virtual columns or unique indexes on BLOB or TEXT columns, any InnoDB tables or tablespaces may become irreparably corrupted.

- Improved write performance (MDEV-23399, MDEV-23855, MDEV-24037)
- MDEV-18323 It is now possible to upgrade from MySQL 5.7 Tables containing JSON, by loading the MYSQL_JSON datatype plugin first. See Making MariaDB understand MySQL JSON.
- Update S3 engine to maturity Gamma
- mariadbd --temp-pool option deprecated and defaulted to zero (MDEV-22278)
- BLACKHOLE Storage Engine maximum index size increased from 1000 to 3500 bytes (MDEV-24017)
- Calculating (auto rounding) issue (MDEV-23702)
- Temporary tables can no longer overwrite existing files. Instead an error is returned should a conflict occur (MDEV-23569)
- Binlog checksum verification at recovery time (MDEV-23832)
- Verbose print-out of Geometry types by mysqlbinlog (MDEV-22330)

- SHOW BINLOG EVENTS from <pos> validates <pos> when binlog checksummed (MDEV-21839)
- Freeing memory of replicate_do_table (MDEV-23534)
- Corrected verbose mysqlbinlog output for multi-record Rows-log-event (MDEV-16372)
- SET GLOBAL replicate_do_db = DEFAULT no longer causes crash (MDEV-20744)
- User killed queries that were running an index condition pushdown in InnoDB will now return an error (MDEV-23938)
- Wrong direxec param data caused crash; Numerous fixes about Mac builds (by Dmitri Shulga) (MDEV-19838)
- server_audit plugin now logs proxy users (MDEV-19443)
- Crash on `SELECT` on a table with indexed virtual columns (MDEV-18366)
- InnoDB updated to 5.7.32 (MDEV-23989)
- Bug fixes related to adaptive hash index (MDEV-23452, MDEV-23370)
- Fixed a bug in the recovery of encrypted tables (MDEV-23456)
- Fixed a race condition in MVCC reads (MDEV-22924)
- ALTER TABLE fixes (MDEV-22277, MDEV-22939, MDEV-23199, MDEV-23356, MDEV-23499, MDEV-23672, MDEV-23685, MDEV-23722)
- Diskspace not reused for BLOB in data file (MDEV-23072)
- InnoDB: Failing assertion: `!space->referenced()` (MDEV-23651)
- `SIGSEGV` in `maria_create()` because of double free (MDEV-23222)
- `CREATE TEMPORARY TABLE .. LIKE` (system versioned table) returns error if unique index is defined in the table (MDEV-23968)
- Error upon querying the view, that selecting from versioned table with partitions (MDEV-23779)
- `CREATE .. SELECT` wrong result on join versioned table (MDEV-23799)
- `SIGSEGV` in `check_fields` on `UPDATE` (MDEV-22805)
- Parser fix (MDEV-23094)
- Add CRC-32 code to `mysys` , giving notable speedup in checksum calculation on x64 (MDEV-19935)
- Faster CRC-32C checksum calculations (MDEV-23495, MDEV-22749)
- Fixes to potential corruption bugs (MDEV-23973, MDEV-24054)
- Fixed delayed replication with S3 storage engine slave (MDEV-23691)
- Deadlock between `BACKUP STAGE BLOCK_COMMIT` and parallel replication (MDEV-23586)
- `CREATE` fails after `DROP` without `.frm` (MDEV-23549)

S3 Storage Engine

- Update S3 engine to maturity Gamma
- Add the s3_use_http and s3_port system variables

Galera

- Galera wsrep library updated to 26.4.6
- Fixed assertion failure on before_commit (MDEV-22681)
- Fixed assertion after ROLLBACK AND CHAIN (MDEV-22055)
- Fixed replication of DROP TRIGGER (MDEV-23638)
- IPv6 SST handling improved (MDEV-21770, MDEV-23576, MDEV-23580, MDEV-23581, MDEV-23574)
- Fixed SIGSEGV in lock_rec_unlock (MDEV-23101)
- Fixed replication of timezone if only 1 timezone is loaded (MDEV-22626)
- Fixed replication of CREATE OR REPLACE TRIGGER (MDEV-21578)
- Fixed SST FLUSH TABLES WITH READ LOCK timeout (MDEV-22543)

Notes

- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.5 for CentOS/RHEL 6 and Fedora 31
- Packages for Ubuntu 20.10 "Groovy Gorilla" added
- Packages for Debian 10 "buster" arm64 and ppc64el added
- Packages for Debian 9 "stretch" arm64 added

- Fixes for the following security vulnerabilities:
  - CVE-2020-14812
  - CVE-2020-14765
  - CVE-2020-14776
  - CVE-2020-14789
  - CVE-2020-28912 (MDEV-24040)
  - CVE-2021-2194 (MDEV-18366)
  - CVE-2022-21427

Changelog

For a complete list of changes made in MariaDB 10.5.7, with links to detailed information on each push, see the changelog.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.7.2.18 MariaDB 10.5.6 Release Notes

The most recent release of MariaDB 10.5 is:
MariaDB 10.5.23 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.5.6 | Release Notes | Changelog | Overview of 10.5

Release date: 7 Oct 2020

MariaDB 10.5 is the current *stable* series of MariaDB. It is an evolution of MariaDB 10.4 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

For an overview of MariaDB 10.5 see the What is MariaDB 10.5? page.

Thanks, and enjoy MariaDB!

Notable Changes

- Fixes for the following security vulnerabilities:
  - CVE-2020-15180

Changelog

For a complete list of changes made in MariaDB 10.5.6, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.5.6, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.7.2.19 MariaDB 10.5.5 Release Notes

The most recent release of MariaDB 10.5 is:
MariaDB 10.5.23 Stable (GA) Download Now
Alternate download from mariadb.org

Download | Release Notes | Changelog | Overview of 10.5

Release date: 10 Aug 2020

MariaDB 10.5 is the current *stable* series of MariaDB. It is an evolution of MariaDB 10.4 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

For an overview of MariaDB 10.5 see the What is MariaDB 10.5? page.

Thanks, and enjoy MariaDB!

Notable Changes

Deprecated variables

- innodb_thread_concurrency
- innodb_commit_concurrency
- innodb_replication_delay
- innodb_concurrency_tickets
- innodb_thread_sleep_delay
- innodb_adaptive_max_sleep_delay (MDEV-23379)

InnoDB

- Fixed corruption in delete buffering (MDEV-22497)
- Fixed a deadlock in FLUSH TABLES…FOR EXPORT (MDEV-22890)
- InnoDB data file extension is not crash-safe (MDEV-23190)
- Minor fixes related to encryption and FULLTEXT INDEX
- Dropping the adaptive hash index may cause DDL to lock up InnoDB (MDEV-22456)
- innodb_log_optimize_ddl=OFF is not crash safe (MDEV-21347)
- Mariadb service won't shutdown when it's running and the OS datetime updated backwards (MDEV-17481)
- Doublewrite recovery can corrupt data pages (MDEV-11799)
- Fixed race conditions related to buffer pool resizing
- ALTER TABLE fixes (MDEV-22637, MDEV-23244, MDEV-22988, MDEV-23295, MDEV-22771, MDEV-22811, MDEV-22899)
- Slow InnoDB shutdown on large instance (MDEV-22778)
- Fixes to performance regressions introduced in MariaDB 10.5.4 (MDEV-23017, MDEV-23369, MDEV-23410)
- Performance improvements (MDEV-22110, MDEV-22930, MDEV-23379, MDEV-22778)
- Correctly implemented the scrubbing of freed pages (MDEV-8139)
- Crash recovery fixes (MDEV-21347, MDEV-23190, MDEV-11799)

Replication

- Make the binlog dump thread to log into errorlog a requested GTID position (MDEV-20428)
- Fix stop of the optimistic parallel slave at requested START-SLAVE-UNTIL position (MDEV-15152)
- Properly handle `RESET MASTER TO` value, when the value exceeds the max allowed `2147483647` (MDEV-22451)
- Correct 'relay-log.info' updating by concurrent parallel workers (MDEV-22806)
- Eliminate deadlock involving parallel workers, `STOP SLAVE` and `FLUSH TABLES WITH READ LOCK` (MDEV-23089)
- Correct master-slave automatic reconnection by slave to always pass through all steps of the initial connect. Specifically, do not skip master notification about slave binlog checksum awareness (MDEV-14203)
- Refine mysqlbinlog output to print out `START TRANSACTION` at `Gtid_log_event` processing which satisfies clients that submit the output with `sql_mode=oracle` (MDEV-23108)
- Replication aborts with `ER_SLAVE_CONVERSION_FAILED` upon `CREATE ... SELECT` in ORACLE mode (MDEV-19632)

Optimizer

- Improve Protocol performance for numeric data by avoiding unnecessary character string conversions (MDEV-23162)
- ALTER TABLE ... ANALYZE PARTITION ... with EITS reads and locks all rows ... (MDEV-21472)
- Print ranges in the optimizer trace created for non-indexed columns when optimizer_use_condition_selectivity >2 Now the optimizer trace shows the ranges constructed while getting estimates from EITS (MDEV-22665)
- `LATERAL DERIVED` is not clearly visible in EXPLAIN FORMAT=JSON, make `LATERAL DERIVED` tables visible in `EXPLAIN FORMAT=JSON` output (MDEV-17568)
- Crash on WITH RECURSIVE large query (MDEV-22748)
- Crash with Prepared Statement with a '?' parameter inside a re-used CTE (MDEV-22779)

Other

- div_precision_increment is now taken into account for all intermediate calculations. Previously results could be unpredictable. Note that this means results will have a lower precision in some cases - see div_precision_increment (MDEV-19232)
- mariadb_schema data type qualifier allowing MariaDB native date types in an SQL_MODE that has conflicting data type translations.
- MariaDB could crash after changing the query_cache size (MDEV-5924)
- Errors and SIGSEGV on CREATE TABLE w/ various charsets (MDEV-22111)
- Crash in `CREATE TABLE AS SELECT` when the precision of returning type = 0 (MDEV-22502)
- XA: Reject DDL operations between PREPARE and COMMIT (MDEV-22420)
- Stop `mariabackup --prepare` on errors during innodb redo log applying (MDEV-22354)
- Server crashes in `mysql_alter_table` upon adding a non-null date column under `NO_ZERO_DATE` with `ALGORITHM=INPLACE` (MDEV-18042)
- Can't uninstall plugin if the library file doesn't exist (MDEV-21258)
- Mariabackup parameter cleanup (MDEV-18215, MDEV-21298, MDEV-21301, MDEV-22894)
- Rounding functions return wrong datatype (MDEV-23366, MDEV-23367, MDEV-23368, MDEV-23350, MDEV-23351, MDEV-23337, MDEV-23323)
- Create `mariadb.sys` user on each update even is the user is not needed (MDEV-23102)
- INFORMATION_SCHEMA.INNODB_TABLESPACES_ENCRYPTION required SUPER instead PROCESS privilege (MDEV-23003)
- Reinforce DDL operation rejection after XA PREPARE (MDEV-22420)
- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.5 for Ubuntu 19.10 Eoan and Fedora 30
- Fixes for the following security vulnerabilities:
  - CVE-2021-2022

Changelog

For a complete list of changes made in MariaDB 10.5.5, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.5.5, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.7.2.20 MariaDB 10.5.4 Release Notes

Download | Release Notes | Changelog | Overview of 10.5

Release date: 24 Jun 2020

MariaDB 10.5 is the current *stable* series of MariaDB. It is an evolution of MariaDB 10.4 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.5.4 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.5 see the What is MariaDB 10.5? page.

Thanks, and enjoy MariaDB!

Notable Changes

This is the first Stable (GA) release in the MariaDB 10.5 series.

- This release of MariaDB Server includes the S3 storage engine. *Note, that plugins have independent maturity levels and S3 storage engine in 10.5.4 has **Alpha** maturity.*
- This release of MariaDB Server includes the MariaDB ColumnStore storage engine. *Note, that plugins have independent maturity levels and MariaDB ColumnStore in 10.5.4 has **Beta** maturity.*
- New Gamma version of the Spider Storage Engine, 3.3.15.
- DROP TABLE now reliably deletes table remnants inside a storage engine even if the .frm file is missing (MDEV-11412)
- Accelerated `crc32()` function for AMD64, ARMv8, POWER 8 (MDEV-22669)
- Lots of bug fixes, see the changelog.
- Galera wsrep library updated to 26.4.5

Variables

- Limit innodb_encryption_threads to 255 (MDEV-22258).
- Minimum value of max_sort_length raised to 8 (previously 4) so fixed size data types like DOUBLE and BIGINT are not truncated for lower values of max_sort_length (MDEV-22715).

InnoDB

- DROP TABLE improvements: MDEV-8069, MDEV-11412, MDEV-22456
- InnoDB Performance improvements: MDEV-15053, MDEV-22593, MDEV-22697, MDEV-22871, MDEV-22841

Changelog

For a complete list of changes made in MariaDB 10.5.4, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.5.4, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.7.2.21 MariaDB 10.5.3 Release Notes

The most recent release of MariaDB 10.5 is:
MariaDB 10.5.23 Stable (GA) Download Now
Alternate download from mariadb.org

Download | Release Notes | Changelog | Overview of 10.5

Release date: 12 May 2020

MariaDB 10.5 is the current *development* series of MariaDB. It is an evolution of MariaDB 10.4 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.5.3 is a ***Release Candidate*** (RC) release.

Do not use non-stable (non-GA) releases in production!

For an overview of MariaDB 10.5 see the What is MariaDB 10.5? page.

Thanks, and enjoy MariaDB!

Notable Changes

Notable changes of this release include:

Syntax

- Application period tables: WITHOUT OVERLAPS (MDEV-16978)
- Introduce a file format constraint to ALTER TABLE. See innodb_instant_alter_column_allowed (MDEV-20590)

Large Pages

- Modernise Linux Large Page support (multiplesizes) (MDEV-18851)

Storage Engines

- Partitioned S3 tables are discoverable. This means that if you create a partitioned S3 table, both the partitioned table and its partitions can be directly used by another server that has access to the S3 storage. (MDEV-22088)

Performance

- Optimizer flag rowid_filter leads to long query (MDEV-21794)
- WSREP_ON is unnecessarily expensive to evaluate (MDEV-22203)
- Misc wsrep performance optimization (MDEV-7962)

Security

- Added system user for user view which allows to remove root (MDEV-19650)
- WolfSSL updated
- ALTER USER doesn't remove excess authentication plugins from mysql.global_priv (MDEV-21928)
- mysql_upgrade creating empty global_priv table (MDEV-21244)

Aria

- Updated aria_pack to support transactional tables and added the `--datadir`, `--ignore-control-file` and `--require-control-file` options. More details here

ALTER TABLE

- Error on online ADD PRIMARY KEY after instant DROP/reorder (MDEV-21658)
- Assertion failure in file data0type.cc (MDEV-20726)
- Server aborts upon attempt to create foreign key on spatial field (MDEV-21792)

- DROP COLUMN, DROP INDEX is wrongly claimed to be ALGORITHM=INSTANT (MDEV-22465)
- Introduce a file format constraint (MDEV-20590)
- FORCE all partition to rebuild if any one of the partition does rebuild (MDEV-21832)
- InnoDB aborts while adding instant column for discarded tablespace (MDEV-22446)
- Misc ALTER TABLE assertion failure (MDEV-22358)

Optimizer

- Optimizer, Wrong query results with `optimizer_switch="split_materialized=on"` (MDEV-21614)
- SHOW GRANTS does not quote role names properly (MDEV-20076)
- Paritioning INSERT chooses wrong partition for RANGE partitioning by DECIMAL column (MDEV-21195)

Mariabackup

- Mariabackup does not honor ignore_db_dirs from server config (MDEV-19347)
- Mariabackup `--ftwrl-wait-timeout` never times out on explicit lock (MDEV-20230)

Crash Recovery

- Loop of Read redo log up to LSN (MDEV-21826)
- `buf_page_get_gen()` should apply buffered page initialized redo log during recovery (MDEV-21572)
- Running out of file descriptors and eventual crash (MDEV-18027)
- Efficient InnoDB redo log record format (MDEV-12353)
- Punch holes when pages are freed (MDEV-15528)

Other

- Use MariaDB in error messages instead of MySQL (MDEV-17812)
- FULLTEXT INDEX, Assertion `!table->fts->in_queue' failed in `fts_optimize_remove_table` (MDEV-21550)
- Wrong estimate of affected BLOB columns in update of PRIMARY KEY (MDEV-22384)
- Duplicate key value is silently truncated to 64 characters in `print_keydup_error` (MDEV-20604)
- Session tracking returns incorrectly long tracking data (MDEV-22504)
- Add `pam_user_map.so` file to binary tarball package (MDEV-21913)
- Misc fixes for Mac build (MENT-606)
- mysql_upgrade is made aware of the upstream slave tables to issue warnings when that takes place (MDEV-10047)
- InnoDB ALTER TABLE fixes (MDEV-21564, MDEV-19092, MDEV-21549)
- InnoDB FULLTEXT INDEX fixes (MDEV-21563)
- Corruption for `SET GLOBAL innodb_` string variables (MDEV-22393)
- Test suite, Add JUnit support to MTR to generate XML test result (MDEV-22176)
- mysqldump parameter, `--ignore-table-data`, added (MDEV-22037)
- Refactored `MYSQL_BIN_LOG::xid_count_per_binlog` to satisfy UBSAN enabled build (MDEV-20923)
- Unregister of slave threads disconnected before COM_BINLOG_DUMP (Bug#29915479)
- Server can fail while replicating conditional comments (Bug#28388217)
- Added the `xml-report` option to mysql-test-run (MDEV-22176)
- Packages and repositories for Ubuntu 20.04 "focal" added

Changelog

For a complete list of changes made in MariaDB 10.5.3, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.5.3, see the MariaDB Foundation release announcement.

Do not use non-stable (non-GA) releases in production!

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community

announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.7.2.22 MariaDB 10.5.2 Release Notes

The most recent release of MariaDB 10.5 is:
MariaDB 10.5.23 Stable (GA) Download Now
Alternate download from mariadb.org

Download | Release Notes | Changelog | Overview of 10.5

Release date: 26 Mar 2020

MariaDB 10.5 is the current *development* series of MariaDB. It is an evolution of MariaDB 10.4 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.5.2 is a ***Beta*** release.

Do not use *beta* releases in production!

For an overview of MariaDB 10.5 see the What is MariaDB 10.5? page.

Thanks, and enjoy MariaDB!

Notable Changes

Notable changes of this release include:

Syntax

- RELEASE_ALL_LOCKS (MDEV-10569)
- ALTER TABLE ... RENAME INDEX / KEY (MDEV-7318)
- ALTER TABLE ... RENAME COLUMN (MDEV-16290)
- Recursive CTE cycle detection using CYCLE clause (MDEV-20632)
- ALTER TABLE and RENAME TABLE now support `IF EXISTS`.

Privileges

- Split SUPER privilege to smaller privileges (MDEV-21743). New privileges were added so that more fine grained tuning of what each user can do can be applied:
  - BINLOG ADMIN
  - BINLOG REPLAY
  - CONNECTION ADMIN
  - FEDERATED ADMIN
  - READ_ONLY ADMIN
  - REPLICATION MASTER ADMIN
  - REPLICATION SLAVE ADMIN
  - SET USER
- The REPLICATION CLIENT privilege was renamed to BINLOG MONITOR. The old syntax is understood for compatibility (MDEV-21743).
- The SHOW MASTER STATUS statement was renamed to SHOW BINLOG STATUS (MDEV-21743). The old syntax is understood for compatibility.
- A number of statements changed the privileges that they require. The old privileges were historically inappropriately chosen in the upstream. 10.5.2 fixes this problem. Note, these changes are incompatible to previous versions. A number of GRANT commands might be needed after upgrade.
  - SHOW BINLOG EVENTS now requires the BINLOG MONITOR privilege (required REPLICATION SLAVE

prior to 10.5.2).
    - SHOW SLAVE HOSTS now requires the REPLICATION MASTER ADMIN privilege (required REPLICATION SLAVE prior to 10.5.2).
    - SHOW SLAVE STATUS now requires the REPLICATION SLAVE ADMIN or the SUPER privilege (required REPLICATION CLIENT or SUPER prior to 10.5.2).
    - SHOW RELAYLOG EVENTS now requires the REPLICATION SLAVE ADMIN privilege (required REPLICATION SLAVE prior to 10.5.2).
- In order to help the server understand which version a privilege record was written by, the mysql.global_priv.priv field contains a new JSON field, `version_id` (MDEV-21704)
- SHOW PRIVILEGES now correctly lists the `Delete history` privilege, rather than displaying it as `Delete versioning rows`. (MDEV-20382)

InnoDB

- An upgrade will only be possible after a clean shutdown. mariabackup --prepare will not work with backups taken before version 10.5.2.
- Efficient InnoDB redo log record format (MDEV-12353)
- Deprecate and ignore innodb_scrub_log and innodb_scrub_log_speed (MDEV-21870)
- Remove INFORMATION_SCHEMA.INNODB_TABLESPACES_SCRUBBING table and deprecate and ignore innodb-background-scrub-data-uncompressed, innodb-background-scrub-data-compressed, innodb-background-scrub-data-interval and innodb-background-scrub-data-check-interval (MDEV-15528)
- Deprecate and ignore innodb_log_files_in_group (MDEV-14425)
- Do not acquire InnoDB record locks when covering table locks exist (MDEV-14479)
- Issue a message on changing deprecated innodb_log_files_in_group (MDEV-21990)
- Optimize append only files for NVDIMM (MDEV-17084)
- Improve innodb redo log group commit performance (MDEV-21534)
- Punch holes when pages are freed (MDEV-15528)

Optimizer

- Allow packed sort keys in sort buffer (MDEV-21580)

Performance Schema

- Merge 5.7 P_S transaction instrumentation and tables (MDEV-16435)
- Merge 5.7 P_S memory instrumentation and tables (MDEV-16431)
- Merge 5.7 P_S mdl instrumentation and tables (MDEV-16432)
- Merge 5.7 P_S sxlocks instrumentation and tables (MDEV-16436)
- Merge 5.7 P_S user variables instrumentation and tables (MDEV-16439)
- Merge 5.7 P_S [show] status instrumentation and tables (MDEV-16438)
- Merge 5.7 P_S ps instrumentation and tables (MDEV-16433)
- Merge 5.7 P_S sp instrumentation and tables (MDEV-16434)

Replication

- `ENFORCE` option for slave_run_triggers_for_rbr (MDEV-21833)

ANALYZE FORMAT=JSON

- Add information about packed addon fields in ANALYZE FORMAT=JSON (MDEV-21838)

Binaries

- All binaries previously beginning with `mysql` now begin with `mariadb`, with symlinks for the corresponding `mysql` command. (MDEV-21303)

Galera

- Galera wsrep library updated to 26.4.4
- Galera Cluster Node During IST gets stuck going from "Synced" to "Joining:..." (MDEV-21002)

Other

- HeidiSQL updated to 11.0 (MDEV-22032)
- require_secure_transport system variable, for rejecting connections attempted using insecure transport (MDEV-13362)
- sql_if_exists session system variable, which adds an implicit IF EXISTS to ALTER, RENAME and DROP of TABLES, VIEWS, FUNCTIONS and PACKAGES. (MDEV-19964)
- XA PREPARE transactions must survive client disconnection (MDEV-742)
- Binary tarball size has been reduced (MDEV-21943)

Changelog

For a complete list of changes made in MariaDB 10.5.2, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.5.2, see the MariaDB Foundation release announcement.

Do not use *beta* releases in production!

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.7.2.23 MariaDB 10.5.1 Release Notes

The most recent release of MariaDB 10.5 is:
MariaDB 10.5.23 Stable (GA) Download Now
Alternate download from mariadb.org

Download | Release Notes | Changelog | Overview of 10.5

Release date: 14 Feb 2020

MariaDB 10.5 is the current *development* series of MariaDB. It is an evolution of MariaDB 10.4 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.5.1 is a ***Beta*** release.

Do not use *beta* releases in production!

For an overview of MariaDB 10.5 see the What is MariaDB 10.5? page.

Thanks, and enjoy MariaDB!

Notable Changes

Notable changes of this release include:

InnoDB

- Remove dummy tablespace for the redo log (MDEV-18115)

- Optimize access to InnoDB page header fields (MDEV-21133)
- Remove multiple InnoDB buffer pool instances (MDEV-15058)
  - Deprecate and ignore innodb_buffer_pool_instances and innodb_page_cleaners
  - Columns that indicated the buffer pool instance from the Information Schema innodb_buffer_page, innodb_buffer_page_lru, innodb_buffer_pool_stats, innodb_cmpmem and innodb_cmpmem_reset tables now return a dummy value of `0` .
- Deprecate and ignore innodb_log_optimize_ddl (MDEV-19747)
- Prefer MDL to dict_sys.latch for innodb background tasks (MDEV-16678)
- Use fdatasync() for redo log where appropriate (MDEV-21382)
- Replace recv_sys.heap with list of buf_block_t (MDEV-21351)
- Several fixes to server hangs (MDEV-16264)

Optimizer

- Allow packed values of non-sorted fields in the sort buffer (MDEV-21263)

Replication and Galera

- slave_parallel_mode now defaults to `optimistic` (MDEV-18648).
- Make REPLICA a synonym for SLAVE in SQL statements (MDEV-20601)
- Galera GTID support (commit)
- Add new mode to wsrep_OSU_method in which Galera checks storage engine of the effected table (MDEV-20051)
- Galera: Replicate MariaDB GTID to other nodes in the cluster (MDEV-20720)

PCRE

- Migrate to PCRE2 (MDEV-14024)

Compatibility

- Show internal type for TIMESTAMP, DATETIME, and TIME columns (MDEV-19906)

Variables

- Numerous deprecated variables removed (MDEV-18650)
  - multi_range_count
  - thread_concurrency
  - timed_mutexes

Changelog

For a complete list of changes made in MariaDB 10.5.1, with links to detailed information on each push, see the changelog .

Contributors

For a full list of contributors to MariaDB 10.5.1, see the MariaDB Foundation release announcement .

Do not use *beta* releases in production!

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.7.2.24 MariaDB 10.5.0 Release Notes

The most recent release of MariaDB 10.5 is:
MariaDB 10.5.23 Stable (GA) Download Now
Alternate download from mariadb.org

Download | Release Notes | Changelog | Overview of 10.5

Release date: 3 Dec 2019

MariaDB 10.5 is the current development series of MariaDB. It is an evolution of MariaDB 10.4 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.5.0 is an ***Alpha*** release.

Do not use *alpha* releases in production!

For an overview of MariaDB 10.5 see the What is MariaDB 10.5? page.

Thanks, and enjoy MariaDB!

Notable Changes

This is the first alpha release in the MariaDB 10.5 series.

Notable changes of this release include:

INET 6 Data Type

- New INET6 data type for storing IPv6 addresses (MDEV-274).

Syntax

- INSERT ... RETURNING (MDEV-10014)
- REPLACE ... RETURNING (MDEV-10014)
- EXCEPT ALL and INTERSECT ALL (MDEV-18844)
- Database comments in CREATE DATABASE and ALTER DATABASE statements (MDEV-307)
- Setup default partitions for system versioning (MDEV-19903)
- Fix REFERENCES constraint in column definition (MDEV-20729)

JSON

- JSON_ARRAYAGG
- JSON_OBJECTAGG
- Add information about packed addon fields in ANALYZE FORMAT=JSON (MDEV-21838)

S3 Storage Engine

- S3 Storage Engine, a read-only storage engine that stores its data in Amazon S3 (MDEV-17841)

Thread Pool

- Information Schema tables (THREADPOOL_GROUPS, THREADPOOL_QUEUES and THREADPOOL_STATS) for internals of generic threadpool (MDEV-19313)

InnoDB

- innodb_adaptive_hash_index now defaults to `OFF` (MDEV-20487)
- innodb_checksum_algorithm now defaults to `full_crc32` (MDEV-19534)
- innodb_checksums has been removed (MDEV-19534)

- innodb_log_checksums has been deprecated (MDEV-19543)
- innodb_locks_unsafe_for_binlog has been removed (MDEV-19544)
- innodb_stats_sample_pages has been removed (MDEV-19551)
- innodb_undo_logs has been deprecated (MDEV-19570)
- innodb_rollback_segments has been removed (MDEV-19570)
- Set innodb_log_files_in_group=1 by default (MDEV-20907)
- Extend SHOW STATUS LIKE 'Innodb_%' (MDEV-18582)
- Clean up INFORMATION_SCHEMA.INNODB_ tables (MDEV-19940)
- Doublewrite buffer is unnecessarily used for newly (re)initialized pages (MDEV-19738)
- Defer change buffer merge until pages are requested (MDEV-19514)

InnoDB Refactoring

- Remove buf_page_t::newest_modification (MDEV-21132)
- Replace recv_sys_t::addr_hash with a std::map (MDEV-19586)
- Obsolete internal parser for FK in InnoDB (MDEV-20480)
- InnoDB thread pool for background tasks (MDEV-16264)

Binary Log

- Extended binlog metadata (MDEV-20477)

Query Optimizer

- ANALYZE for statements is improved, now it also shows the time spent checking the WHERE clause and doing other auxiliary operations (MDEV-20854)
- Inferred IS NOT NULL predicates can be used by the range optimizer (MDEV-15777)

Galera

- Galera 4 Inconsistency voting (MDEV-17048)

General

- The Information Schema SYSTEM_VARIABLES Table has a new column showing from which config file a variable derives its value (MDEV-12684)
- Switch Perl DBI scripts from DBD::mysql to DBD::MariaDB driver (MDEV-19755)
- The Aria max key length is now 2000 bytes, compared to 1000 bytes in MyISAM.

Do not use *alpha* releases in production!

Changelog

For a complete list of changes made in MariaDB 10.5.0, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.5.0, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.8 MariaDB Server 10.4

Changes and Improvements in MariaDB 10.4
Current Version: 10.4.32 | Status: Stable (GA) | Release Date: 13 Nov 2023

Release Notes - MariaDB 10.4 Series
MariaDB 10.4 Series Release Notes

Changelogs - MariaDB 10.4 Series
MariaDB 10.4 changelogs.

There are 2 related questions.

7.0.8.1 Changes and Improvements in MariaDB 10.4

The most recent release of MariaDB 10.4 is:
MariaDB 10.4.32 Stable (GA) Download Now
Alternate download from mariadb.org

MariaDB 10.4 is a previous major stable version. The first stable release of 10.4 was in June 2019, and it will be maintained until June 2024.

Contents

Implemented Features

Authentication

- See Authentication from MariaDB 10.4 for an overview of the changes.
- The unix_socket authentication plugin is now default on Unix-like systems, which is a major change to authentication in MariaDB (MDEV-12484)
- User password expiry (MDEV-7597)
- Account Locking (MDEV-13095)
- The obsolete mysql.host table is no longer created (MDEV-15851)
- Much faster privilege checks for MariaDB setups with many user accounts or many database grants (MDEV-15649)
- mysql.user table is retired. User accounts and global privileges are now stored in the mysql.global_priv table (MDEV-17658)
- SET PASSWORD support for ed25519 and other authentication plugins (MDEV-12321)

InnoDB

- Added instant DROP COLUMN and changing of the order of columns (MDEV-15562)

- More Instant VARCHAR extension or ROW_FORMAT=DYNAMIC and ROW_FORMAT=COMPACT (MDEV-15563)
  - change CHAR(0) to any VARCHAR
  - change a CHAR that is longer than 255 bytes to VARCHAR or wider CHAR
  - change a VARCHAR that is shorter than 128 bytes into any longer VARCHAR
- Instant collation or charset changes for non-indexed columns (MDEV-15564)
- Reduced redo log volume for undo tablespace initialization (MDEV-17138)
- Removed crash-upgrade support for pre-10.2.19 TRUNCATE TABLE (MDEV-13564)
- Added key rotation for innodb_encrypt_log (MDEV-12041)
- Implement innodb_checksum_algorithm=full_crc32 (MDEV-12026)

Optimizer

- Implementation of the optimizer trace, one can enable the optimizer trace by enabling the system variable optimizer_trace (MDEV-6111)
- Engine Independent Table Statistics is now enabled by default; new default values are use_stat_tables=PREFERABLY_FOR_QUERIES and optimizer_use_condition_selectivity=4 (MDEV-15253)
  - Two new values for the variable use_stat_tables: `COMPLEMENTARY_FOR_QUERIES` and `PREFERABLY_FOR_QUERIES` (MDEV-17255)
  - Histograms are now collected by default (MDEV-18608).
  - analyze_sample_percentage variable added. The default value is 100 (`ANALYZE` will use the whole table), but one can also set analyze to only use a sample of table data to collect the statistics.
- Condition pushdown optimization now has bigger scope:
  - Conditions can be pushed into materialized IN-subqueries (MDEV-12387)
  - Conditions in `HAVING` clause can be pushed to WHERE. This behavior is controlled through optimizer switch flag `condition_pushdown_from_having` .
- The optimizer switch flag `optimize_join_buffer_size` now defaults to `on` (MDEV-17903)
- Rowid Filtering optimization added (MDEV-16188). It is controlled through optimizer switch flag `rowid_filter` .

Syntax

- Temporal tables extended with support for application-time periods (MDEV-16973 , MDEV-16974 , MDEV-16975 , MDEV-17082)
- Support of brackets (parentheses) for specifying precedence in UNION/EXCEPT/INTERSECT operations (MDEV-11953)
- New FLUSH SSL command to reload SSL certificates without server restart (MDEV-16266)
- New `CAST` target — `CAST(x AS INTERVAL DAY_SECOND(N))` (MDEV-17776)
- `IF NOT EXISTS` clause added to INSTALL PLUGIN and `IF EXISTS` clause added to UNINSTALL PLUGIN and UNINSTALL SONAME (MDEV-16294)
- Unique indexes can be created on BLOB or TEXT fields (MDEV-371)

Variables

For a list of all new variables, see System Variables Added in MariaDB 10.4 and Status Variables Added in MariaDB 10.4.

- Added to the tcp_nodelay system variable (MDEV-16277)
- Removed the Innodb_pages0_read status variable (MDEV-15705).
- New sql-mode setting, `TIME_ROUND_FRACTIONAL` (MDEV-16991)
- New variable gtid_cleanup_batch_size for determining how many old rows must accumulate in the mysql.gtid_slave_pos table before a background job will be run to delete them.
- The default for eq_range_index_dive_limit is now `200` (previously `0`) (MDEV-18551)
- core_file on Windows now defaults to `ON` (MDEV-18439)

Replication

- Speed up rotation of binary logs, `SHOW BINARY LOGS` etc with optimizing binary log index file locking (MDEV-19116 , MDEV-19117).
- A new server command, SHUTDOWN WAIT FOR ALL SLAVES, and a new mysqladmin shutdown --wait-for-all-slaves option, are added to instruct the server to wait for the last binlog event to be sent to all connected slaves before shutting down. (MDEV-18450).

Backup

- BACKUP STAGE allows one to implement very efficient backups with minimal locking. MDEV-5336 .

Galera 4

In MariaDB 10.4.2 and later, Galera has been upgraded from Galera 3 to Galera 4.

Galera 4 Versions

The following table lists each version of the Galera 4 wsrep provider, and it lists which version of MariaDB each one was first released in. If you would like to install Galera 4 using yum, apt, or zypper, then the package is called `galera-4`.

Galera Version	Released in MariaDB Version
26.4.14	10.10.3, 10.9.5, 10.8.7, 10.7.8, 10.6.12, 10.5.19, 10.4.28
26.4.13	10.10.2, 10.9.4, 10.8.6, 10.7.7, 10.6.11, 10.5.18, 10.4.27
26.4.12	10.10.1, 10.9.2, 10.8.4, 10.7.5, 10.6.9, 10.5.17, 10.4.26
26.4.11	10.8.1, 10.7.2, 10.6.6, 10.5.14, 10.4.22
26.4.9	10.6.4, 10.5.12, 10.4.21
26.4.8	10.6.1, 10.5.10, 10.4.19
26.4.7	10.5.9, 10.4.18
26.4.6	10.5.7, 10.4.16
26.4.5	10.5.4, 10.4.14
26.4.4	10.5.1, 10.4.13
26.4.3	10.5.0, 10.4.9
26.4.2	10.4.4
26.4.1	10.4.3
26.4.0	10.4.2

New Features in Galera 4

The `mysql` database contains new tables related to Galera replication:

- `wsrep_cluster`
- `wsrep_cluster_members`
- `wsrep_streaming_log`

End users may read but not modify these tables.

The new streaming replication feature allows replicating transactions of unlimited size. With streaming replication, a cluster is replicating a transaction in small fragments during transaction execution. This transaction fragmenting is controlled by two new configuration variables:

- `wsrep_trx_fragment_unit (bytes, rows, statements)` defines the metrics for how to measure transaction size limit for fragmenting. Possible values are:
  - `bytes` : transaction's binlog events buffer size in bytes
  - `rows` : number of rows affected by the transaction
  - `statements` : number of SQL statements executed in the multi-statement transaction
- `wsrep_trx_fragment_size` defines the limit for fragmenting. When a transaction's size, in terms of the configured fragment unit, has grown over this limit, a new fragment will be replicated.

Transactions replicated through galera replication will now process the commit phase using MariaDB's group commit logic. This will affect transaction throughput, especially when binary logging is enabled in the cluster.

Limitations in Galera 4

Rolling Upgrades from Galera 3 to Galera 4

Rolling upgrades from MariaDB 10.3 (or earlier) to MariaDB 10.4 also require an upgrade from Galera 3 to Galera 4. Galera 4 has a lot of changes and improvements that were not present in Galera 3.

Prior to the General Availability (GA) releases of MariaDB 10.4 and Galera 4, earlier versions of MariaDB 10.4 and Galera 4 had bugs that could lead to problems if Galera 4 nodes were in a cluster with Galera 3 nodes during a rolling upgrade. In these versions, rolling upgrades were not supported. This meant that, in order to upgrade a cluster, the cluster had to be completely stopped, and the nodes could only be restarted after the entire cluster had been upgraded to MariaDB 10.4 and

Galera 4.

These bugs have been fixed in more recent versions, and rolling upgrades from Galera 3 to Galera 4 are supported. In order to perform a rolling upgrade, it is recommended to upgrade to MariaDB 10.4.6 or later and Galera 26.4.2 or later. However, as a general rule, users should try to ensure that they are upgrading to the latest versions of MariaDB 10.4 and Galera 4.

For more detailed information on how to upgrade, see Upgrading from MariaDB 10.3 to MariaDB 10.4 with Galera Cluster.

General

- Crash safe Aria-based system tables (MDEV-16421)
- Added Linux abstract socket support (MDEV-15655)
- Enabled C++11 (MDEV-16410)
- Performance improvements in Unicode collations (MDEV-17534, MDEV-17511, MDEV-17502, MDEV-17474)
- User data type plugins (MDEV-4912, in progress)
- Improvements with SQL standard INTERVAL support to help functions TIMESTAMP() and ADDTIME() return more predictable results.
  - Historically, MariaDB uses the TIME data type for both "time of the day" values and "duration" values. In the first meaning the natural value range is from '00:00:00' to '23:59:59.999999', in the second meaning the range is from '-838:59:59.999999' to '+838:59:59.999999'.
  - To remove this ambiguity and for the SQL standard conformance we plan to introduce a dedicated data type INTERVAL that will be able to store values in the range at least from '-87649415:59:59.999999' to '+87649415:59:59.999999', which will be enough to represent the time difference between TIMESTAMP'0001-01-01 00:00:00' and TIMESTAMP'9999-12-31 23:59:59.999999'.
  - As a first step we support this range of values for intermediate calculations when TIME-alike string and numeric values are used in INTERVAL (i.e. duration) context, e.g. as the second argument of SQL functions TIMESTAMP(ts,interval) and ADDTIME(ts,interval), so the following can now be calculated:

```
SELECT ADDTIME(TIMESTAMP'0001-01-01 00:00:00', '87649415:59:59.999999');
-> '9999-12-31 23:59:59.999999'

SELECT TIMESTAMP(DATE'0001-01-01', '87649415:59:59.999999')
-> '9999-12-31 23:59:59.999999'

SELECT ADDTIME(TIME'-838:59:59.999999', '1677:59:59.999998');
-> '838:59:59.999999'
```

- Support for window UDF functions via the new method x_remove (MDEV-15073)
- Json service for plugins (MDEV-5313)
- Change in behavior for FLUSH TABLES (MDEV-5336).
- The JSON_VALID function is automatically used as a CHECK constraint for the JSON data type alias in order to ensure that a valid json document is inserted (MDEV-13916)
- MariaDB Named Commands (MDEV-17591)
- MariaDB systemd multi-instance service have changed. See systemd page for details.

Security Vulnerabilities Fixed in MariaDB 10.4

> For a complete list of security vulnerabilities (CVEs) fixed across all versions of MariaDB, see the Security Vulnerabilities Fixed in MariaDB page.

- CVE-2023-5157: MariaDB 10.4.26
- CVE-2023-22084: MariaDB 10.4.32
- CVE-2022-47015: MariaDB 10.4.29
- CVE-2022-38791: MariaDB 10.4.26
- CVE-2022-32091: MariaDB 10.4.26
- CVE-2022-32089: MariaDB 10.4.26
- CVE-2022-32088: MariaDB 10.4.25
- CVE-2022-32087: MariaDB 10.4.25
- CVE-2022-32086: MariaDB 10.4.25
- CVE-2022-32085: MariaDB 10.4.25
- CVE-2022-32084: MariaDB 10.4.26
- CVE-2022-32083: MariaDB 10.4.25
- CVE-2022-32081: MariaDB 10.4.26
- CVE-2022-31624: MariaDB 10.4.22

- CVE-2022-27458: MariaDB 10.4.25
- CVE-2022-27457: MariaDB 10.4.25
- CVE-2022-27456: MariaDB 10.4.25
- CVE-2022-27455: MariaDB 10.4.25
- CVE-2022-27452: MariaDB 10.4.25
- CVE-2022-27451: MariaDB 10.4.25
- CVE-2022-27449: MariaDB 10.4.25
- CVE-2022-27448: MariaDB 10.4.25
- CVE-2022-27447: MariaDB 10.4.25
- CVE-2022-27446: MariaDB 10.4.25
- CVE-2022-27445: MariaDB 10.4.25
- CVE-2022-27444: MariaDB 10.4.25
- CVE-2022-27387: MariaDB 10.4.25
- CVE-2022-27386: MariaDB 10.4.25
- CVE-2022-27385: MariaDB 10.4.22
- CVE-2022-27384: MariaDB 10.4.25
- CVE-2022-27383: MariaDB 10.4.25
- CVE-2022-27382: MariaDB 10.4.25
- CVE-2022-27381: MariaDB 10.4.25
- CVE-2022-27380: MariaDB 10.4.25
- CVE-2022-27379: MariaDB 10.4.25
- CVE-2022-27378: MariaDB 10.4.25
- CVE-2022-27377: MariaDB 10.4.25
- CVE-2022-27376: MariaDB 10.4.25
- CVE-2022-24052: MariaDB 10.4.23
- CVE-2022-24051: MariaDB 10.4.23
- CVE-2022-24050: MariaDB 10.4.23
- CVE-2022-24048: MariaDB 10.4.23
- CVE-2022-21595: MariaDB 10.4.23
- CVE-2022-21451: MariaDB 10.4.19
- CVE-2022-21427: MariaDB 10.4.25
- CVE-2022-0778: MariaDB 10.4.23
- CVE-2021-46669: MariaDB 10.4.25
- CVE-2021-46668: MariaDB 10.4.24
- CVE-2021-46667: MariaDB 10.4.22
- CVE-2021-46666: MariaDB 10.4.20
- CVE-2021-46665: MariaDB 10.4.24
- CVE-2021-46664: MariaDB 10.4.24
- CVE-2021-46663: MariaDB 10.4.24
- CVE-2021-46662: MariaDB 10.4.22
- CVE-2021-46661: MariaDB 10.4.24
- CVE-2021-46659: MariaDB 10.4.23
- CVE-2021-46658: MariaDB 10.4.21
- CVE-2021-46657: MariaDB 10.4.20
- CVE-2021-35604: MariaDB 10.4.22
- CVE-2021-27928: MariaDB 10.4.18
- CVE-2021-2389: MariaDB 10.4.21
- CVE-2021-2372: MariaDB 10.4.21
- CVE-2021-2194: MariaDB 10.4.16
- CVE-2021-2166: MariaDB 10.4.19
- CVE-2021-2154: MariaDB 10.4.19
- CVE-2021-2144: MariaDB 10.4.9
- CVE-2021-2022: MariaDB 10.4.14
- CVE-2021-2007: MariaDB 10.4.7
- CVE-2020-7221: MariaDB 10.4.12
- CVE-2020-2922: MariaDB 10.4.7
- CVE-2020-28912: MariaDB 10.4.16
- CVE-2020-2814: MariaDB 10.4.13
- CVE-2020-2812: MariaDB 10.4.13
- CVE-2020-2780: MariaDB 10.4.9
- CVE-2020-2760: MariaDB 10.4.13
- CVE-2020-2752: MariaDB 10.4.13
- CVE-2020-2574: MariaDB 10.4.12
- CVE-2020-15180: MariaDB 10.4.15
- CVE-2020-14812: MariaDB 10.4.16
- CVE-2020-14789: MariaDB 10.4.16
- CVE-2020-14776: MariaDB 10.4.16

- CVE-2020-14765: MariaDB 10.4.16
- CVE-2020-13249: MariaDB 10.4.13
- CVE-2019-2974: MariaDB 10.4.9
- CVE-2019-2938: MariaDB 10.4.9
- CVE-2019-2805: MariaDB 10.4.7
- CVE-2019-2758: MariaDB 10.4.7
- CVE-2019-2740: MariaDB 10.4.7
- CVE-2019-2739: MariaDB 10.4.7
- CVE-2019-2737: MariaDB 10.4.7
- CVE-2019-2628: MariaDB 10.4.5
- CVE-2019-2627: MariaDB 10.4.5
- CVE-2019-2614: MariaDB 10.4.5
- CVE-2018-25032: MariaDB 10.4.26

List of All MariaDB 10.4 Releases

Date	Release	Status	Release Notes	Changelog
13 Nov 2023	**MariaDB 10.4.32**	Stable (GA)	Release Notes	Changelog
14 Aug 2023	**MariaDB 10.4.31**	Stable (GA)	Release Notes	Changelog
7 Jun 2023	**MariaDB 10.4.30**	Stable (GA)	Release Notes	Changelog
10 May 2023	**MariaDB 10.4.29**	Stable (GA)	Release Notes	Changelog
6 Feb 2023	**MariaDB 10.4.28**	Stable (GA)	Release Notes	Changelog
7 Nov 2022	**MariaDB 10.4.27**	Stable (GA)	Release Notes	Changelog
15 Aug 2022	**MariaDB 10.4.26**	Stable (GA)	Release Notes	Changelog
20 May 2022	**MariaDB 10.4.25**	Stable (GA)	Release Notes	Changelog
12 Feb 2022	**MariaDB 10.4.24**	Stable (GA)	Release Notes	Changelog
9 Feb 2022	**MariaDB 10.4.23**	Stable (GA)	Release Notes	Changelog
8 Nov 2021	**MariaDB 10.4.22**	Stable (GA)	Release Notes	Changelog
6 Aug 2021	**MariaDB 10.4.21**	Stable (GA)	Release Notes	Changelog
23 Jun 2021	**MariaDB 10.4.20**	Stable (GA)	Release Notes	Changelog
7 May 2021	**MariaDB 10.4.19**	Stable (GA)	Release Notes	Changelog
22 Feb 2021	**MariaDB 10.4.18**	Stable (GA)	Release Notes	Changelog
11 Nov 2020	**MariaDB 10.4.17**	Stable (GA)	Release Notes	Changelog
3 Nov 2020	**MariaDB 10.4.16**	Stable (GA)	Release Notes	Changelog
7 Oct 2020	**MariaDB 10.4.15**	Stable (GA)	Release Notes	Changelog
10 Aug 2020	**MariaDB 10.4.14**	Stable (GA)	Release Notes	Changelog
12 May 2020	**MariaDB 10.4.13**	Stable (GA)	Release Notes	Changelog
28 Jan 2020	**MariaDB 10.4.12**	Stable (GA)	Release Notes	Changelog
11 Dec 2019	**MariaDB 10.4.11**	Stable (GA)	Release Notes	Changelog
8 Nov 2019	**MariaDB 10.4.10**	Stable (GA)	Release Notes	Changelog
5 Nov 2019	**MariaDB 10.4.9**	Stable (GA)	Release Notes	Changelog
11 Sep 2019	**MariaDB 10.4.8**	Stable (GA)	Release Notes	Changelog
31 Jul 2019	**MariaDB 10.4.7**	Stable (GA)	Release Notes	Changelog
18 Jun 2019	**MariaDB 10.4.6**	Stable (GA)	Release Notes	Changelog
21 May 2019	**MariaDB 10.4.5**	RC	Release Notes	Changelog
7 Apr 2019	**MariaDB 10.4.4**	RC	Release Notes	Changelog
25 Feb 2019	**MariaDB 10.4.3**	RC	Release Notes	Changelog
29 Jan 2019	**MariaDB 10.4.2**	Beta	Release Notes	Changelog
20 Dec 2018	**MariaDB 10.4.1**	Beta	Release Notes	Changelog

9 Nov 2018	**MariaDB 10.4.0**	Alpha	Release Notes	Changelog

7.0.8.2 Release Notes - MariaDB 10.4 Series

MariaDB 10.4.32 Release Notes

Status: Stable (GA) | Release Date: 13 Nov 2023

MariaDB 10.4.31 Release Notes

Status: Stable (GA) | Release Date: 14 Aug 2023

MariaDB 10.4.30 Release Notes

Status: Stable (GA) | Release Date: 7 Jun 2023

MariaDB 10.4.29 Release Notes

Status: Stable (GA) | Release Date: 10 May 2023

MariaDB 10.4.28 Release Notes

Status: Stable (GA) | Release Date: 6 Feb 2023

MariaDB 10.4.27 Release Notes

Status: Stable (GA) | Release Date: 7 Nov 2022

MariaDB 10.4.26 Release Notes

Status: Stable (GA) | Release Date: 15 Aug 2022

MariaDB 10.4.25 Release Notes

Status: Stable (GA) | Release Date: 20 May 2022

MariaDB 10.4.24 Release Notes

Status: Stable (GA) | Release Date: 12 Feb 2022

MariaDB 10.4.23 Release Notes

Status: Stable (GA) | Release Date: 9 Feb 2022

MariaDB 10.4.22 Release Notes

Status: Stable (GA) | Release Date: 8 Nov 2021

MariaDB 10.4.21 Release Notes

Status: Stable (GA) | Release Date: 6 Aug 2021

MariaDB 10.4.20 Release Notes

Status: Stable (GA) | Release Date: 23 Jun 2021

MariaDB 10.4.19 Release Notes

Status: Stable (GA) | Release Date: 7 May 2021

MariaDB 10.4.18 Release Notes

Status: Stable (GA) | Release Date: 22 Feb 2021

MariaDB 10.4.17 Release Notes

Status: Stable (GA) | Release Date: 11 Nov 2020

MariaDB 10.4.16 Release Notes

Status: Stable (GA) | Release Date: 3 Nov 2020

MariaDB 10.4.15 Release Notes

Status: Stable (GA) | Release Date: 7 Oct 2020

MariaDB 10.4.14 Release Notes

Status: Stable (GA) | Release Date: 10 Aug 2020

MariaDB 10.4.13 Release Notes
Status: Stable (GA) | Release Date: 12 May 2020

MariaDB 10.4.12 Release Notes
Status: Stable (GA) | Release Date: 28 Jan 2020

MariaDB 10.4.11 Release Notes
Status: Stable (GA) | Release Date: 11 Dec 2019

MariaDB 10.4.10 Release Notes
Status: Stable (GA) | Release Date: 8 Nov 2019

MariaDB 10.4.9 Release Notes
Status: Stable (GA) | Release Date: 5 Nov 2019

MariaDB 10.4.8 Release Notes
Status: Stable (GA) | Release Date: 11 Sep 2019

MariaDB 10.4.7 Release Notes
Status: Stable (GA) | Release Date: 31 Jul 2019

MariaDB 10.4.6 Release Notes
Status: Stable (GA) | Release Date: 18 Jun 2019

MariaDB 10.4.5 Release Notes
Status: Release Candidate (RC) | Release Date: 21 May 2019

MariaDB 10.4.4 Release Notes
Status: Release Candidate | Release Date: 7 Apr 2019

MariaDB 10.4.3 Release Notes
Status: Release Candidate | Release Date: 25 Feb 2019

MariaDB 10.4.2 Release Notes
Status: Beta | Release Date: 29 Jan 2019

MariaDB 10.4.1 Release Notes
Status: Beta | Release Date: 20 Dec 2018

MariaDB 10.4.0 Release Notes
Status: Alpha | Release Date: 9 Nov 2018

7.0.8.2.1 MariaDB 10.4.32 Release Notes

Download | Release Notes | Changelog | Overview of 10.4

Alternate download from mariadb.org

Release date: 13 Nov 2023

MariaDB 10.4 is a previous *stable* series of MariaDB, maintained until June 2024. It is an evolution of MariaDB 10.3 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.4.32 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.4 see the What is MariaDB 10.4? page.

Thanks, and enjoy MariaDB!

Notable Items

InnoDB

- DROP INDEX followed by CREATE INDEX may corrupt data (MDEV-32132)
- ROW_FORMAT=COMPRESSED table: InnoDB: 2048 bytes should have been read. Only 0 bytes read. (MDEV-31875)
- Server aborts during alter operation when table doesn't have foreign index (MDEV-32527)
- row_merge_fts_doc_tokenize() handles FTS plugin parser inconsistently (MDEV-32578)
- InnoDB: tried to purge non-delete-marked record of an index on a virtual column prefix (MDEV-30024)
- lock_row_lock_current_waits counter in information_schema.innodb_metrics may become negative (MDEV-30658)
- SET GLOBAL innodb_max_purge_lag_wait=… hangs if innodb_read_only=ON (MDEV-31813)
- Auto-increment no longer works for explicit FTS_DOC_ID (MDEV-32017)
- Assertion `pos < table->n_def' failed in dict_table_get_nth_col (MDEV-32337)
- innochecksum man pages seem to be inconsistent with the binary (10.2.25) (MDEV-20583)
- innodb_compression_algorithm=0 (none) increments Innodb_num_pages_page_compression_error (MDEV-30825)
- wrong table name in innodb's "row too big" errors (MDEV-32128)
- Optimize is_file_on_ssd() to speedup opening tablespaces on Windows (MDEV-32228)

Optimizer

- Crash when HAVING in a correlated subquery references columns in the outer query (MDEV-29731)
- Server crashes at TABLE::add_tmp_key (MDEV-32320)
- Server crashes inside filesort at my_decimal::to_binary (MDEV-32324)
- Assertion `bitmap_is_set(&m_part_info->read_partitions, m_part_spec.start_part)' failed in ha_partition::handle_ordered_index_scan (MDEV-24283)
- Crash when searching for the best split of derived table (MDEV-32064)
- Test case from opt_tvc.test fails with statement memory protection (MDEV-32225)
- Significant slowdown for query with many outer joins (MDEV-32351)
- test_if_skip_sort_order() should catch the join types JT_EQ_REF, JT_CONST and JT_SYSTEM and skip sort order for these (MDEV-32475)

Replication

- rpl.rpl_parallel_temptable failure due to incorrect commit optimization of temptables (MDEV-10356)
- Lock wait timeout with INSERT-SELECT, autoinc, and statement-based replication (MDEV-31482)
- strings/ctype-ucs2.c:2336: my_vsnprintf_utf32: Assertion `(n % 4) == 0' failed in my_vsnprintf_utf32 on INSERT (MDEV-32249)
- Assertion fails in MDL_context::acquire_lock upon parallel replication of CREATE SEQUENCE (MDEV-31792)
- SHOW SLAVE STATUS Last_SQL_Errno Race Condition on Errored Slave Restart (MDEV-31177)
- seconds_behind_master is inaccurate for Delayed replication (MDEV-32265)
- detailize the semisync replication magic number error (MDEV-32365)
- Parallel replication deadlock victim preference code errorneously removed (MDEV-31655)

Galera

- Assertion `state() == s_executing || state() == s_prepared || state() == s_committing || state() == s_must_abort || state() == s_replaying' failed. (MDEV-24912)
- Assertion `state() == s_executing || state() == s_preparing || state() == s_prepared || state() == s_must_abort || state() == s_aborting || state() == s_cert_failed || state() == s_must_replay' failed (MDEV-31285)
- wsrep_sst_mariabackup not working on FreeBSD (MDEV-31467)
- Galera library 26.4.16 fails with every server version (MDEV-32024)
- Galera node remains paused after interleaving FTWRLs (MDEV-32282)
- Failed to insert streaming client (MDEV-32051)
- When set at runtime, wsrep_sst_method accepts any value (MDEV-31470)
- galera needs packaging script changes to successfully build (MDEV-32642)

Data Definition

- MariaDB crash on calling function (MDEV-23902)
- ASAN errors in grn_obj_unlink / ha_mroonga::clear_indexes upon index operations (MDEV-31970)

Scripts and Clients

- mariadb-binlog -T/--table (mysqlbinlog) option (MDEV-25369)
- mariadb-admin (mysqladmin) wrong error with simple_password_check (MDEV-22418)
- mariadb-install-db shows warning on missing directory $pamtooldir/auth_pam_tool_dir (MDEV-32142)
- main.mysql_client_test, main.mysql_client_test_comp failed on ASAN build with error: 5888, status: 23, errno: 2

(MDEV-19369)
- mariadb-install-db (mysql_install_db) doesn't properly grant proxy privileges to all default root user accounts (MDEV-21194)

Tests

- main.events_stress or events.events_stress fails with view-protocol (MDEV-31455)
- main.delete_use_source fails (hangs) with view-protocol (MDEV-31457)
- main.sum_distinct-big and main.merge-big fail with timeout with view-protocol (MDEV-31465)
- main.secure_file_priv_win fails with 2nd execution PS protocol (MDEV-32023)
- Windows : mtr output on is messed up with large MTR_PARALLEL (MDEV-32387)
- main.mysql_client_test_comp failed in buildbot, error on exec (MDEV-16641)

MariaBackup

- MariaBackup full backup failed with InnoDB: Failing assertion: success in storage/innobase/fil/fil0fil.cc line 657 (MDEV-18200)
- mbstream breaks page compression on XFS (MDEV-25734)

Character Sets, Data Types, Collations

- Prefix keys for CHAR work differently for MyISAM vs InnoDB (MDEV-30048)
- Inconsistent results of DISTINCT with NOPAD (MDEV-30050)
- Assertion `(length % 4) == 0' failed in my_lengthsp_utf32 on INSERT (MDEV-28835)
- Compressed varchar values lost on joins when sorting on columns from joined table(s) (MDEV-31724)
- UBSAN shift exponent X is too large for 64-bit type 'long long int' in sql/field.cc (MDEV-32226)
- Wrong bit encoding using COALESCE (MDEV-32244)

Spider

- Spider UBSAN runtime error: applying non-zero offset x to null pointer in st_spider_param_string_parse::restore_delims (MDEV-31117)
- Segfault when setting spider_delete_all_rows to 0 and delete all rows of a spider table, ASAN heap-use-after-free in spider_db_delete_all_rows (MDEV-31996)
- ASAN errors in spider_fields::free_conn_holder or spider_create_group_by_handler (MDEV-28998)

General

- binlog_do_db option breaks importing sql dumps (MDEV-29989)
- Crashes in MDL_key::mdl_key_init with lower-case-table-names=2 (MDEV-32025)
- getting error 'Illegal parameter data types row and bigint for operation '+' ' when using ITERATE in a FOR..DO (MDEV-32275)
- Assertion `arena_for_set_stmt== 0' failed in LEX::set_arena_for_set_stmt upon SET STATEMENT (MDEV-17711)
- main.mysqlcheck fails on ARM with ASAN use-after-poison in my_mb_wc_filename (MDEV-26494)
- main.delayed fails with wrong error code or timeout when executed after main.deadlock_ftwrl (MDEV-27523)
- Assertion failed: !pfs->m_idle || (state == PSI_SOCKET_STATE_ACTIVE) (MDEV-28561)
- MyISAM wrong server status flags (MDEV-28820)
- Server crashes in check_sequence_fields upon CREATE TABLE .. SEQUENCE=1 AS SELECT .. (MDEV-29771)
- slow log Rows_examined out of range (MDEV-30820)
- " `rpm --setugids` " breaks PAM authentication (MDEV-30904)
- incorrect examined rows in case of stored function usage (MDEV-31742)
- Compilation failing on MacOS (unknown warning option -Wno-unused-but-set-variable) (MDEV-31890)
- Server crash upon inserting into Mroonga table with compressed column (MDEV-31966)
- hash unique corrupts index on virtual blobs (MDEV-32012)
- insert into an empty table fails with hash unique (MDEV-32015)
- Valgrind/MSAN warnings in dynamic_column_update_move_left (MDEV-32140)
- Memory leak showed in MDEV-6146 test suite (MDEV-32223)
- Test from subselect.test fails with statement memory protection (MDEV-32245)
- Memory leak when executing PS for query with IN subquery (MDEV-32369)
- Allow the setting of Auto_increment on FK referenced columns (MDEV-32018)
- mariadb-upgrade fails with sql_safe_updates = on (MDEV-29914)
- Assertion `!(thd->server_status & (1U | 8192U))' failed in MDL_context::release_transactional_locks (MDEV-32541)
- Information schema leaks table names and structure to unauthorized users (MDEV-32500)
- Missing CHACHA20-POLY1305 support in WolfSSL (MDEV-31653)
- incorrect error about cyclic reference about JSON type virtual column (MDEV-32586)

- Disable TLS v1.0 and 1.1 for MariaDB (MDEV-31369)
- Better indication of refusing to start because of ProtectHome (MDEV-25177)
- Database upgrade fails: slow_log table (MDEV-27757)
- myrocks_hotbackup.1 and test suite files installed when engine is disabled (MDEV-29993)
- client_ed25519.dll isn't inluded for HeidiSQL. (MDEV-31315)

Docker Official Images

- Invert single and double quotes for sql command definitions in healthcheck.sh due to failure under sql_mode=ANSI_QUOTES - contribution by Dominik Häckel
- healthcheck.sh --no-defaults behaviour was corrected - reported by Dominik Häckel
- Added /docker-entrypoint-init.d for tar{,compression} from mariadb-backup - instructions
- Refactor `docker_mariadb_init` in the entrypoint for extending the MariaDB image
- CIS failure due to world-writable directory /var/run/mysqld, added sticky bit - reported by @ollie1
- Add PROXY privileges for root@MARIADB_ROOT_HOST - reported by Matthieu Gusmini
- healthcheck.sh added --galera_online test, to match what the mariadb-operator does.

Security

- Fixes for the following security vulnerabilities:
  - CVE-2023-22084

Changelog

For a complete list of changes made in MariaDB 10.4.32, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.4.32, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.8.2.2 MariaDB 10.4.31 Release Notes

Download | Release Notes | Changelog | Overview of 10.4

Alternate download from mariadb.org

Release date: 14 Aug 2023

MariaDB 10.4 is a previous *stable* series of MariaDB, maintained until June 2024. It is an evolution of MariaDB 10.3 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.4.31 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.4 see the What is MariaDB 10.4? page.

Thanks, and enjoy MariaDB!

Notable Items

General

- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.4 for Ubuntu 18.04 LTS "Bionic"
- mysqldump --force doesn't ignore error as it should (MDEV-31092)
- ROW variables do not get assigned from subselects (MDEV-31250)
- Crash after setting global session_track_system_variables to an invalid value (MDEV-25237)
- ODKU of non-versioning column inserts history row (MDEV-23100)
- UPDATE not working properly on transaction precise system versioned table (MDEV-25644)
- Assertion `const_item_cache == true' failed in Item_func::fix_fields (MDEV-31319)
- ANALYZE doesn't work with pushed derived tables (MDEV-29284)
- get_partition_set is never executed in ha_partition::multi_range_key_create_key due to bitwise & with 0 constant (MDEV-24712)
- Client can crash the server with a mysql_list_fields("view") call (MDEV-30159)
- I_S.parameters not immediatly changed updated after procedure change (MDEV-31064)

Character Sets, Data Types

- UBSAN: null pointer passed as argument 1, which is declared to never be null in my_strnncoll_binary on SELECT ... COUNT or GROUP_CONCAT (MDEV-28384)
- Possibly wrong result or Assertion `0' failed in Item_func_round::native_op (MDEV-23838)
- Assertion `(length % 4) == 0' failed in my_lengthsp_utf32 on SELECT (MDEV-29019)
- UBSAN: negation of -X cannot be represented in type 'long long int'; cast to an unsigned type to negate this value to itself in Item_func_mul::int_op and Item_func_round::int_op (MDEV-30932)
- Assorted assertion failures in json_find_path with certain collations (MDEV-23187)

InnoDB

- innochecksum dies with Floating point exception (MDEV-31641)
- Deadlock with 3 concurrent DELETEs by unique key (MDEV-10962)
- Assertion `!strcmp(index->table->name.m_name, "SYS_FOREIGN") || !strcmp(index->table->name.m_name, "SYS_FOREIGN_COLS")' failed in btr_node_ptr_max_size (MDEV-19216)
- MODIFY COLUMN can break FK constraints, and lead to unrestorable dumps (MDEV-31086)

Aria

- Various crashes upon INSERT/UPDATE after changing Aria settings (MDEV-28054)
- Various crashes/asserts/corruptions when Aria encryption is enabled/used, but the encryption plugin is not loaded (MDEV-26258)

Spider

- SIGSEGV in spider_db_open_item_field and SIGSEGV in spider_db_print_item_type, on SELECT (MDEV-29447)
- Spider variables that double as table params overriding mechanism is buggy (MDEV-31524)

Optimizer

- Assertion `last_key_entry >= end_pos' failed in virtual bool JOIN_CACHE_HASHED::put_record() (MDEV-31348)
- Problem with open ranges on prefix blobs keys (MDEV-31800)
- Equal on two RANK window functions create wrong result (MDEV-20010)
- Recursive CTE execution is interrupted without errors or warnings (MDEV-31214)
- MAX_SEL_ARG memory exhaustion is not visible in the optimizer trace (MDEV-30964)
- SHOW TABLES not working properly with lower_case_table_names=2 (MDEV-30765)
- Segfault on select query using index for group-by and filesort (MDEV-30143)

Replication

- Parallel Slave SQL Thread Can Update Seconds_Behind_Master with Active Workers (MDEV-30619)
- ALTER SEQUENCE ends up in optimistic parallel slave binlog out-of-order (MDEV-31503)
- STOP SLAVE takes very long time on a busy system (MDEV-13915)
- rpl.rpl_manual_change_index_file occasionally fails in BB with Result length mismatch (MDEV-30214)

Galera

- Node has been dropped from the cluster on Startup / Shutdown with async replica (MDEV-31413)
- MariaDB stuck on starting commit state (waiting on commit order critical section) (MDEV-29293)
- Assertion `state() == s_aborting || state() == s_must_replay` failed in int wsrep::transaction::after_rollback() (MDEV-30013)
- Assertion `!wsrep_has_changes(thd) || (thd->lex->sql_command == SQLCOM_CREATE_TABLE && !thd->is_current_stmt_binlog_format_row()) || thd->wsrep_cs().transaction().state() == wsrep::transaction::s_aborted` failed (MDEV-30388)
- Server crashes when wsrep_sst_donor and wsrep_cluster_address set to NULL (MDEV-28433)
- Create temporary sequence can cause inconsistency (MDEV-31335)
- Galera 4 unable to query cluster state if not primary component (MDEV-21479)

Changelog

For a complete list of changes made in MariaDB 10.4.31, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.4.31, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.8.2.3 MariaDB 10.4.30 Release Notes

The most recent release of MariaDB 10.4 is:
MariaDB 10.4.32 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.4.30 | Release Notes | Changelog | Overview of 10.4

Release date: 7 Jun 2023

MariaDB 10.4 is a previous *stable* series of MariaDB, maintained until June 2024. It is an evolution of MariaDB 10.3 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.4.30 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.4 see the What is MariaDB 10.4? page.

Thanks, and enjoy MariaDB!

Notable Items

InnoDB

- Server crashes in st_join_table::choose_best_splitting (MDEV-31403)
- Crash with condition pushable into derived and containing outer reference (MDEV-31240)
- Revert "MDEV-30473: Do not allow GET_LOCK() / RELEASE_LOCK() in cluster"

Optimizer

- Crash with condition pushable into derived and containing outer reference (MDEV-31403 MDEV-31240)
- Crash with EXPLAIN EXTENDED for multi-table update of system table (MDEV-31224)

Changelog

For a complete list of changes made in MariaDB 10.4.30, with links to detailed information on each push, see the changelog .

Contributors

For a full list of contributors to MariaDB 10.4.30, see the MariaDB Foundation release announcement .

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.8.2.4 MariaDB 10.4.29 Release Notes

The most recent release of MariaDB 10.4 is:
MariaDB 10.4.32 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.4.29 | Release Notes | Changelog | Overview of 10.4

Release date: 10 May 2023

MariaDB 10.4 is a previous *stable* series of MariaDB, maintained until June 2024. It is an evolution of MariaDB 10.3 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.4.29 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.4 see the What is MariaDB 10.4? page.

Thanks, and enjoy MariaDB!

Notable Items

InnoDB

- Crash on ROLLBACK in a ROW_FORMAT=COMPRESSED table (MDEV-30882)
- UNIQUE USING HASH accepts duplicate entries for tricky collations (MDEV-30034)
- rec_get_offsets() is not optimal (MDEV-30567)

Backup

- mariadb-backup doesn't utilise innodb-undo-log-directory (if specified as a relative path) during copy-back operation (MDEV-28187)
- mariabackup issues error messages during InnoDB tablespaces export on partial backup preparing (MDEV-29050)
- mariadb-backup does not copy Aria logs if aria_log_dir_path is used (MDEV-30968)

Replication

- Fixed a deadlock on parallel slave involving full image Write event on the sequence engine (MDEV-29621)

- Fixed an attempted out-of-order binlogging error on slave involving ALTER on the sequence engine (MDEV-31077)
- Corrected non-versioned master to versioned slave replication on no-unique attribute table (MDEV-30430)
- Mended encrypted binlog master to error out to gtid-mode slave when master could not decrypt a binlog file (MDEV-28798)
- Refined optimistic parallel slave to error-exit without any hang (MDEV-30780)

Optimizer

- Split Materialized optimization is improved to re-fill the materialized table only if necessary. The fewer number of table refills is taken into account when choosing query plan, too (MDEV-26301).
- Queries using `SELECT DISTINCT some_expression(aggregate_function())` could produce wrong query result. (MDEV-20057)
- EXPLAIN could erroneously report that Rowid Filter optimization is used for partitioned tables. Partitioned tables do not support it. (MDEV-30596)
- A bug in selectivity computations for SINGLE/DOUBLE_PREC_HB histograms could cause wrong estimates to be produced. This could cause the optimizer to pick sub-optimal query plans (MDEV-31067).

Docker Official Images

- Add replication setup to containers contributed by Md Sahil (MDEV-29762)

Security

- Fixes for the following security vulnerabilities :
  - CVE-2022-47015

Changelog

For a complete list of changes made in MariaDB 10.4.29, with links to detailed information on each push, see the changelog .

Contributors

For a full list of contributors to MariaDB 10.4.29, see the MariaDB Foundation release announcement .

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.8.2.5 MariaDB 10.4.28 Release Notes

Download | Release Notes | Changelog | Overview of 10.4

Release date: 6 Feb 2023

MariaDB 10.4 is a previous *stable* series of MariaDB, maintained until June 2024. It is an evolution of MariaDB 10.3 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.4.28 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.4 see the What is MariaDB 10.4? page.

Thanks, and enjoy MariaDB!

Notable Items

- As mentioned in the 10.4.27 release notes, our Yum/DNF/Zypper repositories for Red Hat Enterprise Linux, CentOS, openSUSE, and SUSE are changing **with this release** to being signed with a new GPG key with SHA2 digest algorithms instead of SHA1. See this blog post and the GPG page for more details.

InnoDB

- Full-text index corruption with system versioning (MDEV-25004)

Galera

- Fixes for cluster wide write conflict resolving (MDEV-29684)

Replication

- Parallel slave applying in binlog order is corrected for admin class of commands including ANALYZE (MDEV-30323)
- Seconds_Behind_Master is now shown now more precisely at the slave applier start, including in the delayed mode (MDEV-29639)
- `mysqlbinlog --verbose` is made to show the type of compressed columns (MDEV-25277)

JSON

- JSON_PRETTY added as an alias for JSON_DETAILED (MDEV-19160)

General

- Infinite sequence of recursive calls when processing embedded CTE (MDEV-30248)
- Crash with a query containing nested WINDOW clauses (MDEV-30052)
- Major performance regression with 10.6.11 (MDEV-29988)

Changelog

For a complete list of changes made in MariaDB 10.4.28, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.4.28, see the MariaDB Foundation release announcement.

7.0.8.2.6 MariaDB 10.4.27 Release Notes

Download 10.4.27 | Release Notes | Changelog | Overview of 10.4

Release date: 7 Nov 2022

MariaDB 10.4 is a previous *stable* series of MariaDB, maintained until June 2024. It is an evolution of MariaDB 10.3 with

several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.4.27 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.4 see the What is MariaDB 10.4? page.

Thanks, and enjoy MariaDB!

Notable Items

SSL

- The server no longer tolerates incorrectly configured SSL (MDEV-29811). If you have enabled SSL in `my.cnf` but have not configured it properly (for example, a certificate file is missing), MariaDB used to silently disable SSL, leaving you under impression that everything was fine and connections were secure. Since this release, MariaDB will fail to start if SSL is enabled, but cannot be switched on.

Backup

- mariabackup --compress hangs (MDEV-29043)
- Assertion on info.page_size failed in xb_delta_open_matching_space (MDEV-18589)

InnoDB

- InnoDB unnecessarily extends data files (MDEV-13013)
- Adaptive hash index MDEV-27700, MDEV-29384
- MVCC and locking MDEV-29666, MDEV-27927
- Virtual columns MDEV-29299, MDEV-29753

Galera

- Galera updated to 26.4.13
- Galera server crashes after 10.3 > 10.4 upgrade (MDEV-29375)
- wsrep_incoming_addresses status variable prints 0 as port number if the port is not mentioned in wsrep_node_incoming_address system variable (MDEV-28868)

JSON

- JSON_VALUE() does not parse NULL properties properly (MDEV-27151)

Replication

- minor correction in unsafe warning message (MDEV-28827)
- False replication error-stop of REVOKE PRIVILEGES from a non-existing user on primary (MDEV-28530) in combination with a filtering replica is corrected
- SET DEFAULT ROLE replication is mended on a replica that filters system tables (MDEV-28294)

Repositories

- Beginning with the next release (Q1 2023), our Yum, DNF, and Zypper repositories for Red Hat, Fedora, and SUSE will be migrated to being signed with a new GPG key. The key we are migrating to is the same one we already use for our Debian and Ubuntu Repositories.
  - The short Key ID is: `0xC74CD1D8`
  - The long Key ID is: `0xF1656F24C74CD1D8`
  - The full fingerprint of the key is: `177F 4010 FE56 CA33 3630 0305 F165 6F24 C74C D1D8`
  - The key can be imported now in preparation for this change using the following command:

```
sudo rpm --import https://supplychain.mariadb.com/MariaDB-Server-GPG-KEY
```

Docker Official Image

The following changes have been made to the `docker.io/library/mariadb` container image.

- The number of gpg packages packages has been removed, leaving enough to `apt-get update`, but `dirmngr` that would fetch keys has been removed. (inspired by issue #469)
- The environment variable `LANG=C.UTF-8` has been added for those that exec into containers and copy paste UTF8 characters (fixes issue #468).
- Adds OCI labels to image (fixes issue 436 and users need for version)
- MariaDB config: skip-host-cache and skip-name-resolve moved to `/etc/mysql/mariadb.conf.d/05-skipcache.cnf`

Security

- Fixes for the following security vulnerabilities: <</style

Changelog

For a complete list of changes made in MariaDB 10.4.27, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.4.27, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.8.2.7 MariaDB 10.4.26 Release Notes

The most recent release of MariaDB 10.4 is:
MariaDB 10.4.32 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.4.26 | Release Notes | Changelog | Overview of 10.4

Release date: 15 Aug 2022

MariaDB 10.4 is a previous *stable* series of MariaDB. It is an evolution of MariaDB 10.3 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.4.26 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.4 see the What is MariaDB 10.4? page.

Thanks, and enjoy MariaDB!

Notable Items

InnoDB

- InnoDB corruption due to lack of file locking (MDEV-28495)
- FULLTEXT search with apostrophe, and mandatory words (MDEV-20797)
- ALTER TABLE IMPORT TABLESPACE corrupts an encrypted table (MDEV-28779)
- ALTER TABLE wrong-result fix (MDEV-26294)

Replication

- ER_SLAVE_INCIDENT error is specified now on slave to be seen with SHOW-SLAVE-STATUS (MDEV-21087)
- INCIDENT_EVENT is no longer binlogged when a being logged transaction can be safely rolledback (MDEV-21443)
- sequences related row-format events are made to correspond to binlog_row_image (MDEV-28487)

Galera

- Possible to write/update with read_only=ON and not a SUPER privilege (MDEV-28546)
- Node crashes with Transport endpoint is not connected mysqld got signal 6 (MDEV-25068)
- Galera4 not able to report proper wsrep_incoming_addresses (MDEV-20627)

Optimizer

- Server crash in JOIN_CACHE::free or in copy_fields (MDEV-23809)
  - Queries that use DISTINCT and an always-constant function like COLLATION(aggegate_func(...)) could cause a server crash. Note that COLLATION() is a special function - its value is constant even if its argument is not costant.
- Crash when using ANY predicand with redundant subquery in GROUP BY clause (MDEV-29139)
  - A query with a subuquery in this form could cause a crash:

```
... ANY (SELECT ... GROUP BY (SELECT redundant_subselect_here)) ...
```

- MariaDB Server SEGV on INSERT .. SELECT (MDEV-26427)
  - Certain queries in form "INSERT ... SELECT with_aggregate_or_window_func" could cause a crash.
- restore_prev_nj_state() doesn't update cur_sj_inner_tables correctly (MDEV-28749)
  - Subquery semi-join optimization could miss LooseScan or FirstMatch strategies for certain queries.
- Optimizer uses all partitions after upgrade to 10.3 (MDEV-28246)
  - For multi-table UPDATE or DELETE queries, the optimizer failed to apply Partition Pruning optimization for the table that is updated or deleted from.

CONNECT

- CONNECT Engine now supports INSERT IGNORE with Mysql Table type (MDEV-27766)

mysql Client

- New mysql client option, `-enable-cleartext-plugin` . Option does not do anything, and is for MySQL-compatibility purposes only.

General

- Crash in JSON_EXTRACT (MDEV-29188)
- ALTER TABLE ALGORITHM=NOCOPY does not work after upgrade (MDEV-28727)
- Server crash upon CREATE VIEW with unknown column in ON condition (MDEV-29088)
- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.4 for Debian 10 "Buster" for ppc64el

Security

- Fixes for the following security vulnerabilities :
  - CVE-2023-5157
  - CVE-2022-32089
  - CVE-2022-32081
  - CVE-2018-25032
  - CVE-2022-32091
  - CVE-2022-32084
  - CVE-2022-38791

Changelog

For a complete list of changes made in MariaDB 10.4.26, with links to detailed information on each push, see the changelog

.

Contributors

For a full list of contributors to MariaDB 10.4.26, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.8.2.8 MariaDB 10.4.25 Release Notes

The most recent release of MariaDB 10.4 is:
MariaDB 10.4.32 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.4.25 | Release Notes | Changelog | Overview of 10.4

Release date: 20 May 2022

MariaDB 10.4 is a previous *stable* series of MariaDB. It is an evolution of MariaDB 10.3 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.4.25 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.4 see the What is MariaDB 10.4? page.

Thanks, and enjoy MariaDB!

Notable Items

InnoDB

- innodb_disallow_writes removed (MDEV-25975)
- InnoDB gap locking fixes (MDEV-20605, MDEV-28422)

Replication

- Server initialization time gtid_slave_pos purge related reason of crashing in binlog background thread is removed (MDEV-26473)
- Shutdown of the semisync master can't produce inconsistent state anymore (MDEV-11853)
- Binlogs disappear after rsync IST (MDEV-28583)
- master crash is eliminated in compressed semisync replication protocol with packet counting amendment (MDEV-25580)
- OPTIMIZE on a sequence does not cause counterfactual ER_BINLOG_UNSAFE_STATEMENT anymore (MDEV-24617)
- Automatically generated Gtid_log_list_event is made to recognize within replication event group as a formal member (MDEV-28550)
- Replication unsafe INSERT .. ON DUPLICATE KEY UPDATE using two or more unique key values at a time with MIXED format binlogging is corrected (MDEV-28310)
- Replication unsafe INSERT .. ON DUPLICATE KEY UPDATE stops issuing unnecessary "Unsafe statement" with MIXED binlog format (MDEV-21810)
- Incomplete replication event groups are detected to error out by the slave IO thread (MDEV-27697)
- mysqlbinlog --stop-never --raw now flushes the result file to disk after each processed event so the file can be listed

with the actual bytes (MDEV-14608)

Backup

- Incorrect binlogs after Galera SST using rsync and mariabackup (MDEV-27524)
- mariabackup does not detect multi-source replication slave (MDEV-21037)
- Useless warning "InnoDB: Allocated tablespace ID <id> for <tablename>, old maximum was 0" during backup stage (MDEV-27343)
- mariabackup prepare fails for incrementals if a new schema is created after full backup is taken (MDEV-28446)

Optimizer

- A SEGV in Item_field::used_tables/update_depend_map_for_order... (MDEV-26402)
- ANALYZE FORMAT=JSON fields are incorrect for UNION ALL queries (MDEV-27699)
- Subquery in an UPDATE query uses full scan instead of range (MDEV-22377)
- Assertion `item1->type() == Item::FIELD_ITEM ... (MDEV-19398)
- Server crashes in Expression_cache_tracker::fetch_current_stats (MDEV-28268)
- MariaDB server crash at Item_subselect::init_expr_cache_tracker (MDEV-26164, MDEV-26047)
- Crash with union of my_decimal type in ORDER BY clause (MDEV-25994)
- SIGSEGV in st_join_table::cleanup (MDEV-24560)
- Assertion `!eliminated' failed in Item_subselect::exec (MDEV-28437)

General

- Server error messages are now available in Chinese (MDEV-28227)
- For RHEL/CentOS 7, non x86_64 architectures are no longer supported upstream and so our support will also be dropped with this release
- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.4 for Debian 9 "Stretch"

Security

- Fixes for the following security vulnerabilities:
  - CVE-2021-46669
  - CVE-2022-21427
  - CVE-2022-27376
  - CVE-2022-27377
  - CVE-2022-27378
  - CVE-2022-27379
  - CVE-2022-27380
  - CVE-2022-27381
  - CVE-2022-27382
  - CVE-2022-27383
  - CVE-2022-27384
  - CVE-2022-27386
  - CVE-2022-27387
  - CVE-2022-27444
  - CVE-2022-27445
  - CVE-2022-27446
  - CVE-2022-27447
  - CVE-2022-27448
  - CVE-2022-27449
  - CVE-2022-27451
  - CVE-2022-27452
  - CVE-2022-27455
  - CVE-2022-27456
  - CVE-2022-27457
  - CVE-2022-27458
  - CVE-2022-32087
  - CVE-2022-32086
  - CVE-2022-32085
  - CVE-2022-32083
  - CVE-2022-32088

Changelog

For a complete list of changes made in MariaDB 10.4.25, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.4.25, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.8.2.9 MariaDB 10.4.24 Release Notes

The most recent release of MariaDB 10.4 is:
MariaDB 10.4.32 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.4.24 | Release Notes | Changelog | Overview of 10.4

Release date: 12 Feb 2022

MariaDB 10.4 is a previous *stable* series of MariaDB. It is an evolution of MariaDB 10.3 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.4.24 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.4 see the What is MariaDB 10.4? page.

Thanks, and enjoy MariaDB!

Notable Items

- This release fixes a blocking problem with the MariaDB 10.4.23 release when manually running mariadb-upgrade. (MDEV-27789)
- See MariaDB 10.4.23 for other changes since the previous release.

Security

- Fixes for the following security vulnerabilities:
  - CVE-2021-46665
  - CVE-2021-46664
  - CVE-2021-46661
  - CVE-2021-46668
  - CVE-2021-46663

Changelog

For a complete list of changes made in MariaDB 10.4.24, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.4.24, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.8.2.10 MariaDB 10.4.23 Release Notes

The most recent release of MariaDB 10.4 is:
MariaDB 10.4.32 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.4.23 | Release Notes | Changelog | Overview of 10.4

Release date: 9 Feb 2022

This release is no longer available for download after a problem was noticed when manually running mariadb-upgrade. See MDEV-27789 for more details.

Please use a later release.

MariaDB 10.4 is a previous *stable* series of MariaDB. It is an evolution of MariaDB 10.3 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.4.23 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.4 see the What is MariaDB 10.4? page.

Thanks, and enjoy MariaDB!

Notable Items

InnoDB

- --skip-symbolic-links does not disallow `.isl` file creation (MDEV-26870)
- Indexed CHAR columns are broken with NO_PAD collations (MDEV-25440)

Galera

- Galera updated to 26.4.11
- Galera SST scripts should use ssl_capath (not ssl_ca) for CA directory (MDEV-27181)
- Alter Sequence do not replicate to another nodes with in Galera Cluster (MDEV-19353)
- Galera crash - Assertion. Possible parallel writeset problem (MDEV-26803)
- CREATE TABLE with FOREIGN KEY constraint fails to apply in parallel (MDEV-27276)

Replication

- Seconds behind master corrected from artificial spikes at relay-log rotation (MDEV-16091)
- Statement rollback in binlog when transaction creates or drop temporary table is set right (MDEV-26833)
- CREATE-or-REPLACE SEQUENCE is made to binlog with the DDL flag to stabilize its parallel execution on slave (MDEV-27365)

Packaging & Misc

- prohibition running two upgrades in parallel (MDEV-27068, MDEV-27107, MDEV-27279)

- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.4 for Ubuntu 21.04 Hirsute, CentOS 8, and Fedora 33
- mariadb_repo_setup script updated to version 2022-02-08, with the following fixes and enhancements:
  - Default location of the script has been moved to: https://r.mariadb.com/downloads/mariadb_repo_setup (old location is deprecated, but still works)
  - The GPG keyring file, used with Debian and Ubuntu repositories, has moved to: https://supplychain.mariadb.com/mariadb-keyring-2019.gpg and the checksum for the file can be found at: https://supplychain.mariadb.com/mariadb-keyring-2019.gpg.sha256
  - Support for RHEL and SLES aarch64 repositories added
  - New function added to verify that the MariaDB Server version, if specified on the command line, follows the correct naming and that a corresponding repository actually exists.
  - Fixed repository pinning for Ubuntu and Debian repositories
  - MariaDB Server 10.7 is now the default server version

Docker Library

- Faster initialization by disabling binary logging during initialization (MDEV-27074)
- mysql_upgrade can be run if needed using the environment variable MARIADB_AUTO_UPGRADE=1 (MDEV-25670)
- A healthcheck script /usr/local/bin/healthcheck.sh is installed in the container with various checking options (MDEV-25434)
- mysql@localhost user is created with the environment variable MARIADB_MYSQL_LOCALHOST_USER=1 and additional grants (beyond USAGE) with MARIADB_MYSQL_LOCALHOST_GRANTS={global grant list} (MDEV-27732)
- skip innodb buffer pool loads/dumps on temporary startup/shutdown for faster startup/initialization, and accurate "healthcheck.sh --innodb_buffer_pool_loaded"
- change group ownership on datadir/socket dir (issue #401)
- log note about note on Securing system users, mysql_secure_installation not required (reddit suggestion)
- speed up Docker Library initialization of timezones (MDEV-27608, MDEV-23326)

Security

- Fixes for the following security vulnerabilities:
  - CVE-2022-24052
  - CVE-2022-24051
  - CVE-2022-24050
  - CVE-2022-24048
  - CVE-2021-46659
  - CVE-2022-0778
  - CVE-2022-21595

Changelog

For a complete list of changes made in MariaDB 10.4.23, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.4.23, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.8.2.11 MariaDB 10.4.22 Release Notes

The most recent release of MariaDB 10.4 is:
MariaDB 10.4.32 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.4.22 | Release Notes | Changelog | Overview of 10.4

Release date: 8 Nov 2021

MariaDB 10.4 is a previous *stable* series of MariaDB. It is an evolution of MariaDB 10.3 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.4.22 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.4 see the What is MariaDB 10.4? page.

Thanks, and enjoy MariaDB!

Notable Items

Galera

- Fix for `WSREP: invalid state ROLLED_BACK (FATAL)` (MDEV-25114)

InnoDB

- `ALTER TABLE…IMPORT TABLESPACE` fixes (MDEV-18543, MDEV-20931, MDEV-26131, MDEV-26621)
- `innodb_undo_log_truncate` fixes (MDEV-26450, MDEV-26672, MDEV-26864)

Replication

- Memory hogging on slave by ROW event applier is eliminated (MDEV-26712)
- `mysql --binary-mode` now properly handles `\\0` in data (MDEV-25444)
- Fixes race condition between `SHOW BINARY LOGS` and `RESET MASTER` (MDEV-20215)
- Missed statement rollback in case transaction drops or create temporary table is corrected (MDEV-26833)

Audit Plugin

- The `QUERY_DDL` server_audit_events setting now logs `CREATE/DROP [PROCEDURE / FUNCTION / USER]` statements. See MariaDB Audit Plugin - Log Settings. (MDEV-23457)

Packaging & Misc

- Session tracking flag in OK_PACKET (MDEV-26868)
- Some views force server (and mysqldump) to generate invalid SQL for their definitions (MDEV-26299)

Security

- Fixes for the following security vulnerabilities:
  - CVE-2021-35604
  - CVE-2021-46667
  - CVE-2021-46662
  - CVE-2022-27385
  - CVE-2022-27385
  - CVE-2022-31624

Changelog

For a complete list of changes made in MariaDB 10.4.22, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.4.22, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.8.2.12 MariaDB 10.4.21 Release Notes

The most recent release of MariaDB 10.4 is:
MariaDB 10.4.32 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.4.21 | Release Notes | Changelog | Overview of 10.4

Release date: 6 Aug 2021

Warning: This version can cause InnoDB file corruption on FreeBSD and on AIX. If you are using AIX, please, stick to an earlier release, or upgrade to a more recent release. If you are using FreeBSD, upgrade to the bugfix release (the version ends with _1) of the mariadb-server from the Ports Collection. See MDEV-26537.

MariaDB 10.4 is a previous *stable* series of MariaDB. It is an evolution of MariaDB 10.3 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.4.21 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.4 see the What is MariaDB 10.4? page.

Thanks, and enjoy MariaDB!

Notable Items

InnoDB

- InnoDB no longer acquires advisory file locks by default (MDEV-24393)
- Encryption: Automatically disable key rotation checks for file_key_management plugin (MDEV-14180)
- Some fixes from MySQL 5.7.35 (MDEV-26205)

Optimizer

- A query that uses ORDER BY .. LIMIT clause and "Range checked for each record optimization" could produce incorrect results under some circumstances (MDEV-25858)
- Queries that have more than 32 equality conditions comparing columns of different tables ("tableX.colX=tableY.colY) could cause a stack overrun in the query optimizer (MDEV-17783, MDEV-23937)
- "Condition pushdown into derived table" optimization cannot be applied if the expression being pushed refers to a derived table column which is computed from expression that has a stored function call, @session variable reference, or other similar construct. The fix for MDEV-25969 makes it so that only the problematic part of the condition is not pushed. The rest of the condition is now pushed. (MDEV-25969)
- A query with window function on the left side of the subquery could cause a crash. (MDEV-25630)

Packaging & Misc

- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.4 for Ubuntu 20.10 Groovy
- Galera updated to 26.4.9

Security

- Fixes for the following security vulnerabilities:
  - CVE-2021-2372
  - CVE-2021-2389
  - CVE-2021-46658

MongoDB protocol support files for the CONNECT engine are missing in this release.
If you want to use CONNECT engine with MongoDB, you need to download
Mongo2.jar or Mongo3.jar and put a path to this file into the `connect_class_path` in the `my.cnf`.

Changelog

For a complete list of changes made in MariaDB 10.4.21, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.4.21, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.8.2.13 MariaDB 10.4.20 Release Notes

The most recent release of MariaDB 10.4 is:
MariaDB 10.4.32 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.4.20 | Release Notes | Changelog | Overview of 10.4

Release date: 23 Jun 2021

MariaDB 10.4 is a previous *stable* series of MariaDB. It is an evolution of MariaDB 10.3 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.4.20 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.4 see the What is MariaDB 10.4? page.

Thanks, and enjoy MariaDB!

Notable Items

This version of MariaDB is being released now to fix the following two regressions:

- Table alias from previous statement interferes later commands (MDEV-25672)
- Join using derived with aggregation returns incorrect results (MDEV-25714)

In addition to the above, this release also contains the following fixes:

InnoDB

- InnoDB spatial indexes miss large geometry fields after MDEV-25459 (MDEV-25758)
- Double free of transaction during truncate operation (MDEV-25663)
- Double free of table when inplace alter FTS add index fails (MDEV-25721)
- Potential hang in purge for virtual columns (MDEV-25664)
- Change buffer entries for secondary indexes are lost on InnoDB restart (MDEV-25869)

Replication

- Do not replicate killed multi-table OPTIMIZE TABLE when the signal arrives before any table has been processed (MDEV-22530)
- Fix optistic parallel applier to not deadlock on admin commands OPTIMIZE, REPAIR, and ANALYZE (MDEV-17515)
- Backport MDEV-20821 parallel slave server shutdown hang (MDEV-22370)

Security

- Fixes for the following security vulnerabilities:
  - CVE-2021-46666
  - CVE-2021-46657

MongoDB protocol support files for the CONNECT engine are missing in this release.
If you want to use CONNECT engine with MongoDB, you need to download
Mongo2.jar or Mongo3.jar and put a path to this file into the `connect_class_path` in the `my.cnf`.

Changelog

For a complete list of changes made in MariaDB 10.4.20, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.4.20, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.8.2.14 MariaDB 10.4.19 Release Notes

The most recent release of MariaDB 10.4 is:
MariaDB 10.4.32 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.4.19 | Release Notes | Changelog | Overview of 10.4

Release date: 7 May 2021

MariaDB 10.4 is the previous *stable* series of MariaDB. It is an evolution of MariaDB 10.3 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.4.19 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.4 see the What is MariaDB 10.4? page.

Thanks, and enjoy MariaDB!

Notable Changes

- ST_DISTANCE_SPHERE for calculating the spherical distance between two geometries (point or multipoint) on a sphere (MDEV-13467)
- Crash with invalid multi-table update of view in 2nd execution of SP (MDEV-24823)
- Incorrect name resolution for subqueries in ON expressions (MDEV-25362)
- Complex query in Store procedure corrupts results (MDEV-25182)
- `DELETE HISTORY` may delete current data on system-versioned table (MDEV-25468)
- Crashes with nested table value constructors (MDEV-22786)
- Server crashes in `thd_clear_errors()` (MDEV-23542)
- The statement set `password=password('')` executed in PS mode fails in case it is run by a user with expired password (MDEV-25197)

mariabackup

- RENAME TABLE causes "Ignoring data file" messages (MDEV-25568)

InnoDB

- Deprecated the `*innodb` and `*none` options in innodb_checksum_algorithm (MDEV-25106)
- MVCC read from index on CHAR or VARCHAR wrongly omits rows (MDEV-25459)
- Race conditions in persistent statistics (MDEV-10682, MDEV-18802, MDEV-25051)
- Sequence created by one connection remains invisible to another (MDEV-24545)
- innodb_flush_method=O_DIRECT fails on compressed tables (MDEV-25121)

Replication

- Replication Heartbeat event was uncapable to cary 4GB+ offsets (MDEV-16146)
- FLUSH LOGS race against Binlog checkpoint event creation (MDEV-24526)
- slave_compressed_protocol did not work correctly with semi-sync (MDEV-24773)

Galera

- Galera updated to 26.4.8
- SET PASSWORD command fail with wsrep api (MDEV-25258)
- Long BF log wait turns on InnoDB Monitor output without telling, never turns it off (MDEV-25319)
- Assertion `` `state_ == s_exec' `` failed in `wsrep::client_state::start_transaction` (MDEV-22227)
- Frequently Crashing Mariadb Cluster 10.4.18 (MDEV-24980)
- Signal 11 on `TABLE_LIST::placeholder()` (MDEV-24878)
- `ALTER TABLE` not replicated with Galera in MariaDB 10.5.9 (MDEV-24956)
- "Flush SSL" command doesn't reload wsrep cert (MDEV-22668)
- Avoid unnecessary rollbacks with SR (MDEV-25553)

Packaging & Misc

- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.4 for Ubuntu 16.04 Xenial

Security

- Fixes for the following security vulnerabilities:
  - CVE-2021-2166
  - CVE-2021-2154
  - CVE-2022-21451

MongoDB protocol support files for the CONNECT engine are missing in this release.
If you want to use CONNECT engine with MongoDB, you need to download
Mongo2.jar or Mongo3.jar and put a path to this file into the `connect_class_path` in the `my.cnf`.

Changelog

For a complete list of changes made in MariaDB 10.4.19, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.4.19, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.8.2.15 MariaDB 10.4.18 Release Notes

The most recent release of MariaDB 10.4 is:
MariaDB 10.4.32 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.4.18 | Release Notes | Changelog | Overview of 10.4

Release date: 22 Feb 2021

Last month long-time MariaDB VP of Engineering, Rasmus Johansson, passed due to complications from cancer. His loss has been felt keenly by the whole MariaDB team. Our thoughts are with his family during this difficult time and this release is dedicated to his memory.

MariaDB 10.4 is the previous *stable* series of MariaDB. It is an evolution of MariaDB 10.3 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.4.18 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.4 see the What is MariaDB 10.4? page.

Thanks, and enjoy MariaDB!

Notable Changes

InnoDB

- MDEV-24188 - Hang in buf_page_create() after reusing a previously freed page
- MDEV-24275 - InnoDB persistent stats analyze forces full scan forcing lock crash
- MDEV-24449 - Corruption of system tablespace or last recovered page

Galera

- Galera updated to 26.4.7
- MDEV-23328 - Server hang due to Galera lock conflict resolution
- MDEV-23851 - BF-BF Conflict issue because of UK GAP locks
- MDEV-20717 - Plugin system variables and activation options can break mysqld --wsrep_recover
- MDEV-24469 - Assertion `active() == false` failed with "XA START.."

- MDEV-23647 - Garbd can't initiate SST anymore in 10.5
- MDEV-25179 - wsrep_provider and wsrep_notify_cmd system variables are now read-only

Replication

- MDEV-8134 - relay-log is corrected to rotate past 999999
- MDEV-23033 - fixed slave applier for row-based events with FK constraints on virtual columns
- MDEV-4633 - Relay_Log_Space of Show-Slave-Status is made thread-safe
- MDEV-10272 - add master host/port info to slave thread exit messages
- MDEV-23846 - improves mysqlbinlog error message issuing

Misc

- MDEV-24122 - anomalies in mysql.user tables on previously 5.7 MySQL versions corrected
- Binary tarballs now use WolfSSL v4.6.0
- MDEV-23630 - mysqldump --system option
- Fixes for the following security vulnerabilities:
  - CVE-2021-27928

Changelog

For a complete list of changes made in MariaDB 10.4.18, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.4.18, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.8.2.16 MariaDB 10.4.17 Release Notes

The most recent release of MariaDB 10.4 is:
MariaDB 10.4.32 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.4.17 | Release Notes | Changelog | Overview of 10.4

Release date: 11 Nov 2020

MariaDB 10.4 is the previous *stable* series of MariaDB. It is an evolution of MariaDB 10.3 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.4.17 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.4 see the What is MariaDB 10.4? page.

Thanks, and enjoy MariaDB!

Notable Changes

- Out-of-cycle release to fix regressions in MariaDB 10.4.16
- Follow up to MDEV-19838 to alter protocol checks to support the following implementations (which add garbage to the end of some packets):
  - PHP PDO (all versions) (MDEV-24121)
  - mysqlnd (from PHP < 7.3) (MDEV-24121)
  - mysql-connector-python (all versions) (MDEV-24134)
  - and mysql-connector-java (all versions)
- Arbitrary InnoDB buffer pool and data file corruption (MDEV-24096)
- The query optimizer consumed a lot of memory when handling construct in form of `key_column [NOT] IN (large-list-of constants)` (MDEV-24117)

Changelog

For a complete list of changes made in MariaDB 10.4.17, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.4.17, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.8.2.17 MariaDB 10.4.16 Release Notes

The most recent release of MariaDB 10.4 is:
MariaDB 10.4.32 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.4.16 | Release Notes | Changelog | Overview of 10.4

Release date: 3 Nov 2020

MariaDB 10.4 is the previous *stable* series of MariaDB. It is an evolution of MariaDB 10.3 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.4.16 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.4 see the What is MariaDB 10.4? page.

Thanks, and enjoy MariaDB!

Notable Changes

This release introduced an InnoDB data corruption bug (MDEV-24096). If any InnoDB tables contain indexed virtual columns or unique indexes on BLOB or TEXT columns, any InnoDB tables or tablespaces may become irrepairably corrupted.

- Set the default value of innodb_log_optimize_ddl to `OFF` by default (MDEV-23720)
- BLACKHOLE Storage Engine maximum index size increased from 1000 to 3500 bytes (MDEV-24017)
- Calculating (auto rounding) issue (MDEV-23702)
- Temporary tables can no longer overwrite existing files. Instead an error is returned should a conflict occur (MDEV-

23569)
- Binlog checksum verification at recovery time (MDEV-23832)
- Verbose print-out of Geometry types by mysqlbinlog (MDEV-22330)
- SHOW BINLOG EVENTS from <pos> validates <pos> when binlog checksummed (MDEV-21839)
- Freeing memory of replicate_do_table (MDEV-23534)
- Corrected verbose mysqlbinlog output for multi-record Rows-log-event (MDEV-16372)
- SET GLOBAL replicate_do_db = DEFAULT no longer causes crash (MDEV-20744)
- User killed queries that were running an index condition pushdown in InnoDB will now return an error (MDEV-23938)
- Wrong direxec param data caused crash; Numerous fixes about Mac builds (by Dmitri Shulga) (MDEV-19838)
- server_audit plugin now logs proxy users (MDEV-19443)
- Crash on `SELECT` on a table with indexed virtual columns (MDEV-18366)
- InnoDB updated to 5.7.32 (MDEV-23989)
- Bug fixes related to adaptive hash index (MDEV-23452, MDEV-23370)
- Fixed a bug in the recovery of encrypted tables (MDEV-23456)
- Fixed a race condition in MVCC reads (MDEV-22924)
- `ALTER TABLE` fixes (MDEV-22277, MDEV-22939, MDEV-23199, MDEV-23356, MDEV-23499, MDEV-23672, MDEV-23685, MDEV-23722)
- Diskspace not reused for BLOB in data file (MDEV-23072)
- InnoDB: Failing assertion: `!space->referenced()` (MDEV-23651)
- `SIGSEGV` in `maria_create()` because of double free (MDEV-23222)
- `CREATE TEMPORARY TABLE .. LIKE` (system versioned table) returns error if unique index is defined in the table (MDEV-23968)
- Error upon querying the view, that selecting from versioned table with partitions (MDEV-23779)
- `CREATE .. SELECT` wrong result on join versioned table (MDEV-23799)
- `SIGSEGV` in `check_fields` on `UPDATE` (MDEV-22805)
- Parser fix (MDEV-23094)
- Fixed crash in Innodb when rowid_filter query is killed (MDEV-22761)
- Fixed a crash with the NTH_VALUE function (MDEV-15180)
- Computing certain window functions on a server started with --encrypt-tmp_files=ON could cause a wrong query result or crash (MDEV-23867)
- A query with a certain form of WHERE clause over a table with multiple indexes could pick a less efficient range plan (MDEV-23811)
- Fixed a memory leak for correlated subqueries with ROLLUP (MDEV-17066)

Galera

- Galera wsrep library updated to 26.4.6
- Fixed assertion failure on before_commit (MDEV-22681)
- Fixed assertion after ROLLBACK AND CHAIN (MDEV-22055)
- IPv6 SST handling improved (MDEV-21770, MDEV-23576, MDEV-23580, MDEV-23581, MDEV-23574)
- Fixed SIGSEGV in lock_rec_unlock (MDEV-23101)
- Fixed replication of timezone if only 1 timezone is loaded (MDEV-22626)
- Fixed replication of CREATE OR REPLACE TRIGGER (MDEV-21578)
- Fixed SST FLUSH TABLES WITH READ LOCK timeout (MDEV-22543)

Notes

- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.4 for CentOS/RHEL 6 and Fedora 31
- Packages for Ubuntu 20.10 "Groovy Gorilla" added
- Packages for Debian 10 "buster" arm64 and ppc64el added
- Packages for Debian 9 "stretch" arm64 added

- Fixes for the following security vulnerabilities:
  - CVE-2020-14812
  - CVE-2020-14765
  - CVE-2020-14776
  - CVE-2020-14789
  - CVE-2020-28912 (MDEV-24040)
  - CVE-2021-2194 (MDEV-18366)

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the

Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.8.2.18 MariaDB 10.4.15 Release Notes

The most recent release of MariaDB 10.4 is:
MariaDB 10.4.32 Stable (GA) Download Now
Alternate download from mariadb.org

Download 10.4.15 | Release Notes | Changelog | Overview of 10.4

Release date: 7 Oct 2020

MariaDB 10.4 is the previous *stable* series of MariaDB. It is an evolution of MariaDB 10.3 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.4.15 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.4 see the What is MariaDB 10.4? page.

Thanks, and enjoy MariaDB!

Notable Changes

- Fixes for the following security vulnerabilities:
  - CVE-2020-15180

Changelog

For a complete list of changes made in MariaDB 10.4.15, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.4.15, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.8.2.19 MariaDB 10.4.14 Release Notes

The most recent release of MariaDB 10.4 is:
MariaDB 10.4.32 Stable (GA) Download Now
Alternate download from mariadb.org

Download | Release Notes | Changelog | Overview of 10.4

Release date: 10 Aug 2020

MariaDB 10.4 is the current *stable* series of MariaDB. It is an evolution of MariaDB 10.3 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.4.14 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.4 see the What is MariaDB 10.4? page.

Thanks, and enjoy MariaDB!

Notable Changes

Variables

- Limit innodb_encryption_threads to 255 (MDEV-22258).
- Minimum value of max_sort_length raised to 8 (previously 4) so fixed size data types like DOUBLE and BIGINT are not truncated for lower values of max_sort_length (MDEV-22715).

InnoDB

- Fixed corruption in delete buffering (MDEV-22497)
- Fixed a deadlock in FLUSH TABLES…FOR EXPORT (MDEV-22890)
- InnoDB data file extension is not crash-safe (MDEV-23190)
- Minor fixes related to encryption and FULLTEXT INDEX
- Dropping the adaptive hash index may cause DDL to lock up InnoDB (MDEV-22456)
- `innodb_log_optimize_ddl=OFF` is not crash safe (MDEV-21347)
- Mariadb service won't shutdown when it's running and the OS datetime updated backwards (MDEV-17481)
- Doublewrite recovery can corrupt data pages (MDEV-11799)
- Fixed race conditions related to buffer pool resizing
- ALTER TABLE fixes (MDEV-22637, MDEV-23244, MDEV-22988, MDEV-23295, MDEV-22771)
- Slow InnoDB shutdown on large instance (MDEV-22778)
- Performance improvements (MDEV-22778)
- Crash recovery fixes (MDEV-21347, MDEV-23190, MDEV-11799)

Replication

- Make the binlog dump thread to log into errorlog a requested GTID position (MDEV-20428)
- Fix stop of the optimistic parallel slave at requested START-SLAVE-UNTIL position (MDEV-15152)
- Properly handle `RESET MASTER TO` value, when the value exceeds the max allowed `2147483647` (MDEV-22451)
- Correct 'relay-log.info' updating by concurrent parallel workers (MDEV-22806)
- Eliminate deadlock involving parallel workers, `STOP SLAVE` and `FLUSH TABLES WITH READ LOCK` (MDEV-23089)
- Correct master-slave automatic reconnection by slave to always pass through all steps of the initial connect. Specifically, do not skip master notification about slave binlog checksum awareness (MDEV-14203)
- Refine mysqlbinlog output to print out `START TRANSACTION` at `Gtid_log_event` processing which satisfies clients that submit the output with `sql_mode=oracle` (MDEV-23108)
- Replication aborts with `ER_SLAVE_CONVERSION_FAILED` upon `CREATE ... SELECT` in ORACLE mode (MDEV-19632)

Optimizer

- ALTER TABLE ... ANALYZE PARTITION ... with EITS reads and locks all rows ... (MDEV-21472)
- Print ranges in the optimizer trace created for non-indexed columns when `optimizer_use_condition_selectivity >2` Now the optimizer trace shows the ranges constructed while getting estimates from EITS (MDEV-22665)
- `LATERAL DERIVED` is not clearly visible in `EXPLAIN FORMAT=JSON`, make `LATERAL DERIVED` tables visible in `EXPLAIN FORMAT=JSON` output (MDEV-17568)
- Crash on `WITH RECURSIVE` large query (MDEV-22748)
- Crash with Prepared Statement with a '?' parameter inside a re-used CTE (MDEV-22779)

Other

- div_precision_increment is now taken into account for all intermediate calculations. Previously results could be unpredictable. Note that this means results will have a lower precision in some cases - see div_precision_increment (MDEV-19232)
- mariadb_schema data type qualifier allowing MariaDB native date types in an SQL_MODE that has conflicting data type translations.
- MariaDB could crash after changing the query_cache size (MDEV-5924)
- Errors and SIGSEGV on CREATE TABLE w/ various charsets (MDEV-22111)
- Crash in `CREATE TABLE AS SELECT` when the precision of returning type = 0 (MDEV-22502)
- XA: Reject DDL operations between PREPARE and COMMIT (MDEV-22420)
- Stop `mariabackup --prepare` on errors during innodb redo log applying (MDEV-22354)
- Server crashes in `mysql_alter_table` upon adding a non-null date column under `NO_ZERO_DATE` with `ALGORITHM=INPLACE` (MDEV-18042)
- Can't uninstall plugin if the library file doesn't exist (MDEV-21258)
- Mariabackup parameter cleanup (MDEV-18215, MDEV-21298, MDEV-21301, MDEV-22894)
- Rounding functions return wrong datatype (MDEV-23366, MDEV-23367, MDEV-23368, MDEV-23350, MDEV-23351, MDEV-23337, MDEV-23323)
- Create `mariadb.sys` user on each update even is the user is not needed (MDEV-23102)
- Galera wsrep library updated to 26.4.5
- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.4 for Ubuntu 19.10 Eoan and Fedora 30

- Fixes for the following security vulnerabilities:
  - CVE-2021-2022

Changelog

For a complete list of changes made in MariaDB 10.4.14, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.5.5, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.8.2.20 MariaDB 10.4.13 Release Notes

The most recent release of MariaDB 10.4 is:
MariaDB 10.4.32 Stable (GA) Download Now
Alternate download from mariadb.org

Download | Release Notes | Changelog | Overview of 10.4

Release date: 12 May 2020

MariaDB 10.4 is the current *stable* series of MariaDB. It is an evolution of MariaDB 10.3 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.4.13 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.4 see the What is MariaDB 10.4? page.

Thanks, and enjoy MariaDB!

Notable Changes

Events

- Fixed issue that, from MariaDB 10.3.19, disabled all events created by a server with a different `server_id`. Note that the fix does not re-enable affected events. (MDEV-21758)

Privileges

- SHOW PRIVILEGES now correctly lists the `Delete history` privilege, rather than displaying it as `Delete versioning rows`. (MDEV-20382)

Performance

- Optimizer flag rowid_filter leads to long query (MDEV-21794)
- WSREP_ON is unnecessarily expensive to evaluate (MDEV-22203)
- Misc wsrep performance optimization (MDEV-7962)

Security

- Added system user for user view which allows to remove root (MDEV-19650)
- WolfSSL updated
- ALTER USER doesn't remove excess authentication plugins from mysql.global_priv (MDEV-21928)
- mysql_upgrade creating empty global_priv table (MDEV-21244)

Aria

- Updated `aria_pack` to support transactional tables and added options: `--datadir`, `--ignore-control-file`, `--require-control-file`, more details here

ALTER TABLE

- Error on online ADD PRIMARY KEY after instant DROP/reorder (MDEV-21658)
- Assertion failure in file data0type.cc (MDEV-20726)
- Server aborts upon attempt to create foreign key on spatial field (MDEV-21792)
- DROP COLUMN, DROP INDEX is wrongly claimed to be ALGORITHM=INSTANT (MDEV-22465)
- Introduce a file format constraint to ALTER TABLE. See innodb_instant_alter_column_allowed (MDEV-20590)
- FORCE all partition to rebuild if any one of the partition does rebuild (MDEV-21832)
- InnoDB aborts while adding instant column for discarded tablespace (MDEV-22446)

FULLTEXT INDEX

- Assertion `` `!table->fts->in_queue' `` failed in `fts_optimize_remove_table` (MDEV-21550)
- FTS thread aborts during shutdown (MDEV-21563)

Optimizer

- Optimizer, Wrong query results with `optimizer_switch="split_materialized=on"` (MDEV-21614)
- SHOW GRANTS does not quote role names properly (MDEV-20076)
- Paritioning INSERT chooses wrong partition for RANGE partitioning by DECIMAL column (MDEV-21195)

Mariabackup

- Mariabackup does not honor ignore_db_dirs from server config (MDEV-19347)
- Mariabackup `--ftwrl-wait-timeout` never times out on explicit lock (MDEV-20230)

Crash Recovery

- Running out of file descriptors and eventual crash (MDEV-18027)

Galera

- Galera wsrep library updated to 26.4.4
- Galera Cluster Node During IST gets stuck going from "Synced" to "Joining:..." (MDEV-21002)

Other

- HeidiSQL updated to 11.0 (MDEV-22032)
- Wrong estimate of affected BLOB columns in update of PRIMARY KEY (MDEV-22384)
- Duplicate key value is silently truncated to 64 characters in `print_keydup_error` (MDEV-20604)
- Session tracking returns incorrectly long tracking data (MDEV-22504)
- Add `pam_user_map.so` file to binary tarball package (MDEV-21913)
- mysql_upgrade is made aware of the upstream slave tables to issue warnings when that takes place (MDEV-10047)
- Corruption for `SET GLOBAL innodb_` string variables (MDEV-22393)
- mysqldump parameter, `--ignore-table-data`, added (MDEV-22037)
- Server can fail while replicating conditional comments (Bug#28388217)
- Added the `xml-report` option to mysql-test-run (MDEV-22176)
- Packages and repositories for Ubuntu 20.04 "focal" added
- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.4 for Debian 8 "Jessie"
- Fixes for the following security vulnerabilities:
  - CVE-2020-2752
  - CVE-2020-2812
  - CVE-2020-2814
  - CVE-2020-2760
  - CVE-2020-13249

Changelog

For a complete list of changes made in MariaDB 10.4.13, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.4.13, see the MariaDB Foundation release announcement. Thanks, and enjoy MariaDB!

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.8.2.21 MariaDB 10.4.12 Release Notes

The most recent release of MariaDB 10.4 is:
MariaDB 10.4.32 Stable (GA) Download Now
Alternate download from mariadb.org

Note that this version contains an issue that disabled all events created by a server with a different server_id. See MDEV-21758 for details.

Download | Release Notes | Changelog | Overview of 10.4

Release date: 28 Jan 2020

MariaDB 10.4 is the current *stable* series of MariaDB. It is an evolution of MariaDB 10.3 with several entirely new features

not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.4.12 will be a ***Stable (GA)*** release.

For an overview of MariaDB 10.4 see the What is MariaDB 10.4? page.

Thanks, and enjoy MariaDB!

Notable Changes

General

- MDEV-21337: fix aligned_malloc()
- MDEV-21343: Threadpool/Unix- wait_begin() function does not wake/create threa ds, when it should
- wolfssl updated to v4.3.0-stable
- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.4 for Ubuntu 19.04 Disco

Mariabackup

- MDEV-21255: Deadlock of parallel slave and mariabackup (with failed log copy thread)

InnoDB

- MDEV-20950: Reduce size of record offsets
- MDEV-19176: Reduce the memory usage during recovery
- MDEV-21429: TRUNCATE and OPTIMIZE are being refused due to "row size too large"
- MDEV-21500: Server hang when using simulated AIO
- MDEV-21509: Possible hang during purge of history, or rollback
- MDEV-21511: Wrong estimate of affected BLOB columns in update
- MDEV-21512: InnoDB may hang due to SPATIAL INDEX
- MDEV-21513: Avoid some crashes in ALTER TABLE...IMPORT TABLESPACE
- MDEV-18865: Assertion `t->first->versioned_by_id()' failed in innodb_prepare_commit_versioned
- MDEV-21260: InnoDB does not report error when trying open volumes on UNC paths on Windows

Aria

- MDEV-14183: aria_pack segfaults in compress_maria_file

Optimizer

- MDEV-21318: Wrong results with window functions and implicit grouping
- MDEV-16579: Wrong result of query using DISTINCT COUNT(*) OVER (*)
- MDEV-21383: Possible range plan is not used under certain conditions

Replication

- MDEV-18046: Crashes caused by random values to the offset option of SHOW BINLOG EVENT offset command
- MDEV-19376: Semisync Master could crash when it executed RESET MASTER and a replica reconnects using GTID protocol
- MDEV-20821: parallel slave server shutdown may hang when slave workers were online

Security

- Fixes for the following security vulnerabilities:
  - CVE-2020-2574
  - CVE-2020-7221

Changelog

For a complete list of changes made in MariaDB 10.4.12, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.4.12, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.8.2.22 MariaDB 10.4.11 Release Notes

The most recent release of MariaDB 10.4 is:
MariaDB 10.4.32 Stable (GA) Download Now
Alternate download from mariadb.org

Note that this version contains an issue that disabled all events created by a server with a different server_id. See MDEV-21758 for details.

Download | Release Notes | Changelog | Overview of 10.4

Release date: 11 Dec 2019

MariaDB 10.4 is the current *stable* series of MariaDB. It is an evolution of MariaDB 10.3 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.4.11 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.4 see the What is MariaDB 10.4? page.

Thanks, and enjoy MariaDB!

Notable Changes

General

- MDEV-13492: SEC_E_INVALID_TOKEN when server sends large message during SSL handshake

Mariabackup

- MDEV-18310: Aria engine: Undo phase failed from incremental backup

InnoDB

- MDEV-20949: Stop issuing '`row size`' error on DML
- MDEV-20832: Don't print "`row size too large`" warnings in error log if `innodb_strict_mode=OFF` and `log_warnings<=2`
- MDEV-21024: Remove redundant writes to the redo log
- MDEV-21069: Crash on `DROP TABLE` if the data file is corrupted
- some cleanup of AIO code, to better report errors
- MDEV-20611: MRR scan over partitioned InnoDB table produces "`Out of memory`" error
- MDEV-21088: Table cannot be loaded after instant `ADD/DROP COLUMN`
- MDEV-21045: heap-use-after-poison in `ADD PRIMARY KEY` after instant `ADD COLUMN`
- MDEV-21172: Memory leak after failed `ADD PRIMARY KEY`

- MDEV-21158: `trx_undo_seg_free()` is never redo-logged
- MDEV-20190: Instant operation fails when add column and collation change on non-indexed column

Optimizer

- MDEV-21044: Wrong result when using a smaller size for sort buffer
- MDEV-20611: MRR scan over partitioned InnoDB table produces "Out of memory" error
- MDEV-20407: mysqld got `signal 11 ; rowid_filter`

Replication

- MDEV-19376: `Repl_semi_sync_master::commit_trx` assertion failure
- MDEV-20707: Missing memory barrier in parallel replication error handler in `wait_for_prior_commit()`

Versioning

- MDEV-18929: 2nd execution of SP does not detect `ER_VERS_NOT_VERSIONED`
- MDEV-21011: Table corruption reported for versioned partitioned table after `DELETE`

Misc

- Packages for Fedora 31 have been added in this release
- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.4 for Fedora 29

Changelog

For a complete list of changes made in MariaDB 10.4.11, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.4.11, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.8.2.23 MariaDB 10.4.10 Release Notes

The most recent release of MariaDB 10.4 is:
MariaDB 10.4.32 Stable (GA) Download Now
Alternate download from mariadb.org

Note that this version contains an issue that disabled all events created by a server with a different server_id. See MDEV-21758 for details.

Download | Release Notes | Changelog | Overview of 10.4

Release date: 8 Nov 2019

MariaDB 10.4 is the current *stable* series of MariaDB. It is an evolution of MariaDB 10.3 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.4.10 will be a ***Stable (GA)*** release.

For an overview of MariaDB 10.4 see the What is MariaDB 10.4? page.

Thanks, and enjoy MariaDB!

Notable Changes

- MDEV-20987: InnoDB fails to start when FTS table has FK relation
- See also the release notes for MariaDB 10.4.9 for additional items of note

Changelog

For a complete list of changes made in MariaDB 10.4.10, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.4.10, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.8.2.24 MariaDB 10.4.9 Release Notes

The most recent release of MariaDB 10.4 is:
MariaDB 10.4.32 Stable (GA) Download Now
Alternate download from mariadb.org

After an upgrade MariaDB Server can crash if InnoDB tables exist with a `FULLTEXT INDEX` and a `FOREIGN KEY` constraint attached to them. We got reports that the crash already will be encountered on startup, but a crash is also possible at a later stage. See MDEV-20987 for more details.

Do not download or use this release.

Download | Release Notes | Changelog | Overview of 10.4

Release date: 5 Nov 2019

MariaDB 10.4 is the current *stable* series of MariaDB. It is an evolution of MariaDB 10.3 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.4.9 will be a ***Stable (GA)*** release.

For an overview of MariaDB 10.4 see the What is MariaDB 10.4? page.

Thanks, and enjoy MariaDB!

Notable Changes

Notable changes of this release include:

- MDEV-20864: Debug-only option innodb_change_buffer_dump for dumping the contents of the InnoDB change buffer to the server error log at startup.
- MariaBackup:
  - MDEV-18438: mbstream recreates xtrabackup_info on same directory as backup file
  - MDEV-20703: mariabackup creates binlog files in server binlog directory on --prepare --export step
- FULLTEXT INDEX:
  - MDEV-19647: Server hangs after dropping full text indexes and restart
  - MDEV-19529: InnoDB hang on `DROP FULLTEXT INDEX`
  - MDEV-19073: FTS row mismatch after crash recovery
  - MDEV-20621: FULLTEXT INDEX activity causes InnoDB hang
- MDEV-20927: Duplicate key with auto increment
- ALTER TABLE:
  - MDEV-20799: DROP Virtual Column crash
  - MDEV-20852: BtrBulk is unnecessarily holding `dict_index_t::lock`
- System-Versioned Tables:
  - MDEV-16210: FK constraints on versioned tables use historical rows, which may cause constraint violation
  - MDEV-20812: Unexpected `ER_ROW_IS_REFERENCED_2` or server crash in `row_ins_foreign_report_err` upon DELETE from versioned table with FK
- MDEV-20117: corruption after instant DROP/reorder COLUMN
- Galera wsrep library updated to 26.4.3
- Packages for Ubuntu 19.10 Eoan have been added in this release
- Fixes for the following security vulnerabilities:
  - CVE-2019-2974
  - CVE-2019-2938
  - CVE-2020-2780
  - CVE-2021-2144

Changelog

For a complete list of changes made in MariaDB 10.4.9, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.4.9, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.8.2.25 MariaDB 10.4.8 Release Notes

The most recent release of MariaDB 10.4 is:
MariaDB 10.4.32 Stable (GA) Download Now
Alternate download from mariadb.org

Download | Release Notes | Changelog | Overview of 10.4

Release date: 11 Sep 2019

MariaDB 10.4 is the current *stable* series of MariaDB. It is an evolution of MariaDB 10.3 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.4.8 will be a ***Stable (GA)*** release.

For an overview of MariaDB 10.4 see the What is MariaDB 10.4? page.

Thanks, and enjoy MariaDB!

Notable Changes

Notable changes of this release include:

- Connect updated to Connect 1.06.0010
- MDEV-20231: Update server HELP
- MDEV-20066: This bug could cause a table to become corrupt if a column was added instantly
- MDEV-15326: A race condition in InnoDB transaction commit that affects record locking was fixed
- MDEV-17187: Table doesn't exist in engine after ALTER of FOREIGN KEY
- MDEV-20301: InnoDB's MVCC has `O(N^2)` behaviors
- MDEV-18128: Simplify .ibd file creation
- MDEV-20060: Failing assertion: `srv_log_file_size <= 512ULL << 30` while preparing backup
- MDEV-20247: Replication hangs with "preparing" and never starts
- MDEV-17614: Remove unnecessary locking for `INSERT...ON DUPLICATE KEY UPDATE`
- MDEV-20311: row_ins_step accesses uninitialized memory
- MDEV-20479: Assertion failure in `dict_table_get_nth_col()` after `INSTANT DROP COLUMN`
- MDEV-20340: Encrypted temporary tables cannot be read with `innodb_checksum_algorithm=full_crc32`
- MDEV-19947: Repositories for RHEL 8 ppc64le added

Changelog

For a complete list of changes made in MariaDB 10.4.8, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.4.8, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.8.2.26 MariaDB 10.4.7 Release Notes

The most recent release of MariaDB 10.4 is:
MariaDB 10.4.32 Stable (GA) Download Now
Alternate download from mariadb.org

Download | Release Notes | Changelog | Overview of 10.4

Release date: 31 Jul 2019

MariaDB 10.4 is the current *stable* series of MariaDB. It is an evolution of MariaDB 10.3 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.4.7 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.4 see the What is MariaDB 10.4? page.

Thanks, and enjoy MariaDB!

Notable Changes

Notable changes of this release include:

- MDEV-19922: HeidiSQL updated to 10.2
- MDEV-19795: Merge upstream MyRocks.
- MDEV-17228: Encrypted temporary tables are not encrypted.
- MDEV-18328: Disks Plugin is now stable and requires the FILE privilege.
- MDEV-16508: Spider - sql_mode not maintained between spider node and data nodes.
- Merge relevant InnoDB changes from MySQL 5.7.27
- Adjust spin loops to the x86 PAUSE instruction latency (MDEV-19845)
- MDEV-20102: When the ctas from a big table is interrupted ,then you can't drop or recreate the table
- MDEV-19292: InnoDB's row size calculations were fixed, which might result in "Row size too large" errors when creating or altering tables with lots columns. This can occur even if previous MariaDB releases did not throw errors for the same tables. Some workarounds are listed at InnoDB Row Formats Overview: Upgrading Causes Row Size Too Large Errors.
- ALTER TABLE: MDEV-15641, MDEV-19630, MDEV-19916, MDEV-19974, MDEV-17301, MDEV-18266
- Indexed virtual columns: MDEV-16222, MDEV-17005, MDEV-19870
- FULLTEXT INDEX: MDEV-14154
- Encryption: MDEV-17228, MDEV-19914
- Galera + FOREIGN KEY: MDEV-19660
- Recovery & Mariabackup: MDEV-19978
- MDEV-20091: DROP TEMPORARY table is logged despite no CREATE was logged
- MDEV-19871: Add page id matching check in innochecksum tool
- MDEV-20179: Server hangs on shutdown during installation of Spider
- As per the MariaDB Deprecation Policy, this will be the last release of MariaDB 10.4 for OpenSUSE 42.3 and Ubuntu 18.10 "Cosmic"
- Fixes for the following security vulnerabilities:
  - CVE-2019-2805
  - CVE-2019-2740
  - CVE-2019-2739
  - CVE-2019-2737
  - CVE-2019-2758
  - CVE-2020-2922
  - CVE-2021-2007

Changelog

For a complete list of changes made in MariaDB 10.4.7, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.4.7, see the MariaDB Foundation release announcement.

7.0.8.2.27 MariaDB 10.4.6 Release Notes

Download | Release Notes | Changelog | Overview of 10.4

Release date: 18 Jun 2019

With this release, MariaDB 10.4 is now the current *stable* series of MariaDB. It is an evolution of MariaDB 10.3 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.4.6 is a ***Stable (GA)*** release.

For an overview of MariaDB 10.4 see the What is MariaDB 10.4? page.

Thanks, and enjoy MariaDB!

Notable Changes

Notable changes of this release include:

- MariaDB Server is now statically linked with the bundled wolfSSL library in MSI and ZIP packages on Windows, as well as in .deb packages provided by Debian's and Ubuntu's default repositories (MDEV-18531).
  - See TLS and Cryptography Libraries Used by MariaDB for more details.
- MariaDB Named Commands (MDEV-17591)
- System-versioned tables: MDEV-19486
- Galera: MDEV-17458
- Virtual columns: MDEV-19027, MDEV-19602
- Recovery: MDEV-19541, MDEV-19587, MDEV-19435
- Encryption: MDEV-19509, MDEV-19695
- Other:
  - MDEV-19614 - SET GLOBAL innodb_ deadlock due to LOCK_global_system_variables
  - MDEV-19725 - Incorrect error handling in ALTER TABLE

Changelog

For a complete list of changes made in MariaDB 10.4.6, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.4.6, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.8.2.28 MariaDB 10.4.5 Release Notes

The most recent release of MariaDB 10.4 is:
MariaDB 10.4.32 Stable (GA) Download Now
Alternate download from mariadb.org

Download | Release Notes | Changelog | Overview of 10.4

Release date: 21 May 2019

MariaDB 10.4 is the current *development* series of MariaDB. It is an evolution of MariaDB 10.3 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.4.5 is a ***Release Candidate (RC)*** release.

Do not use non-stable (non-GA) releases in production!

For an overview of MariaDB 10.4 see the What is MariaDB 10.4? page.

Thanks, and enjoy MariaDB!

Notable Changes

General Server

- New mysqlimport option, `--ignore-foreign-keys` (MDEV-788).
- Setting sql_mode to `MSSQL` implements a limited subset of Microsoft SQL Server's language. See SQL_MODE=MSSQL (MDEV-19142).
- Add CAST(expr AS FLOAT) (MDEV-16872).
- List of slave transaction errors that will result in a retry rather than a halt (slave_transaction_retry_errors) have been increased by default, assisting Spider setups to be more robust (MDEV-16543).
- MDEV-15458 - Segfault in heap_scan() upon UPDATE after ADD SYSTEM VERSIONING
- MDEV-19235 - MariaDB Server compiled for 128 Indexes crashes at startup

JSON

- JSON_MERGE_PATCH and JSON_MERGE_PRESERVE (MDEV-13992)

InnoDB

- Merge InnoDB changes from MySQL 5.6.44 and 5.7.26
- Fixes of corruption or crashes: MDEV-19241, MDEV-13942, MDEV-19385, MDEV-16060, MDEV-18220, MDEV-17540
- InnoDB recovery fixes and speedup: MDEV-12699, MDEV-19356

Encryption

- MDEV-14398 - innodb_encrypt_tables will work even with innodb_encryption_rotate_key_age=0

System-Versioned Tables

- MDEV-15966 - System-versioned tables are now protected from TRUNCATE TABLE statements.

Information schema

- MDEV-19490 show tables fails when selecting the information_schema database

Statistics

- MDEV-19407 - Assertion `field->table->stats_is_read' failed in is_eits_usable
- New status variable, Aborted_connects_preauth, that records the number of connection attempts that were aborted prior to authentication (MDEV-19277).

Packaging

- As per the MariaDB Deprecation Policy, this is the last release of MariaDB 10.4 for Fedora 28
- Packages and a repository for Fedora 30 and Ubuntu 19.04 "disco" have been added with this release, visit the Repository Configuration Tool for instructions on adding the repository

Security

- Fixes for the following security vulnerabilities:
  - CVE-2019-2614

- CVE-2019-2627
- CVE-2019-2628

Changelog

For a complete list of changes made in MariaDB 10.4.5, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.4.5, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.8.2.29 MariaDB 10.4.4 Release Notes

The most recent release of MariaDB 10.4 is:
MariaDB 10.4.32 Stable (GA) Download Now

Alternate download from mariadb.org

Download | Release Notes | Changelog | Overview of 10.4

Release date: 7 Apr 2019

MariaDB 10.4 is the current *development* series of MariaDB. It is an evolution of MariaDB 10.3 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.4.4 is a ***Release Candidate*** release.

Do not use non-stable (non-GA) releases in production!

For an overview of MariaDB 10.4 see the What is MariaDB 10.4? page.

Thanks, and enjoy MariaDB!

Notable Changes

- Enhancements:
  - MDEV-12026/MDEV-18644: innodb_checksum_algorithm=full_crc32 (more robust file format)
  - MDEV-13301: Optimize DROP INDEX, ADD INDEX into RENAME INDEX
  - MDEV-17380: innodb_flush_neighbors=ON should be ignored on SSD
- InnoDB data corruption fixes: MDEV-14126, MDEV-18981, MDEV-18879, MDEV-18972, MDEV-18272
- Performance fixes to purge, startup and shutdown: MDEV-18936, MDEV-18878, MDEV-18733
- Various fixes to ALTER TABLE
- Replication:
  - MDEV-18450: wait for all slaves shutdown
  - MDEV-19116, MDEV-19117: speed up rotation of binary logs, `SHOW BINARY LOGS` etc with optimizing binary log index file locking
- Includes Connector/C 3.1.0
- Repositories for CentOS 7, RHEL 7 & 8, Fedora 28 & 29, and SLES 12 & 15 now include a src.rpm file that you can

use to build MariaDB. Instructions for doing so are found on the Building MariaDB from a Source RPM page
- The Galera library in the repositories has been updated to version **26.4.2**
- As per the MariaDB Deprecation Policy, this is the last release of MariaDB 10.4 for Ubuntu 14.04 Trusty

Changelog

For a complete list of changes made in MariaDB 10.4.4, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.4.4, see the MariaDB Foundation release announcement.

7.0.8.2.30 MariaDB 10.4.3 Release Notes

The most recent release of MariaDB 10.4 is:
MariaDB 10.4.32 Stable (GA) Download Now
Alternate download from mariadb.org

Download | Release Notes | Changelog | Overview of 10.4

Release date: 25 Feb 2019

MariaDB 10.4 is the current *development* series of MariaDB. It is an evolution of MariaDB 10.3 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.4.3 is a ***Release Candidate*** release.

Do not use non-stable (non-GA) releases in production!

For an overview of MariaDB 10.4 see the What is MariaDB 10.4? page.

Thanks, and enjoy MariaDB!

Notable Changes

Notable changes of this release include:

- MDEV-12484: The unix_socket authentication plugin is now default on Unix-like systems, which is a major change to authentication in MariaDB
- MDEV-11340: Allow multiple alternative authentication methods for the same user
- MDEV-7597: User password expiry
- MDEV-6111: Implementation of the optimizer trace, one can enable the optimizer trace by enabling the system variable optimizer_trace
- Temporal tables extended with support for application-time periods (MDEV-16973, MDEV-16974, MDEV-16975, MDEV-17082)
- MDEV-18551: The default for eq_range_index_dive_limit is now `200` (previously `0`)
- MDEV-17903: The optimizer switch flag `optimize_join_buffer_size` now defaults to `on`
- New optimizer switch flags `rowid_filter` and `condition_pushdown_from_having`
- MDEV-18439: core_file on Windows now defaults to `ON`
- MDEV-18608: Histograms are now collected by default.
- MDEV-13916: The JSON_VALID function is automatically used as a CHECK constraint for the JSON data type alias in order to ensure that a valid json document is inserted
- Spider updated to 3.3.14
- Unique indexes can be created on BLOB or TEXT fields (MDEV-371)
- MDEV-18564: wsrep_load_data_splitting is deprecated and now set to `OFF` by default
- analyze_sample_percentage system variable
- InnoDB ALTER TABLE fixes: MDEV-18222, MDEV-18256, MDEV-18016, MDEV-18295, MDEV-16849, MDEV-18219
- Mariabackup fixes: MDEV-18194, MDEV-18415, MDEV-18611

- New InnoDB features:
  - MDEV-12026: Implement innodb_checksum_algorithm=full_crc32
  - MDEV-15563: More Instant VARCHAR extension
  - MDEV-15564: Instant collation or charset changes for non-indexed columns
  - MDEV-16188: Use in-memory PK filters built from range index scans
- Debian has stopped supporting the ppc64el architecture for Debian 8 Jessie and so this is the last release of MariaDB 10.4 on Jessie for that architecture

Changelog

For a complete list of changes made in MariaDB 10.4.3, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.4.3, see the MariaDB Foundation release announcement.

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.8.2.31 MariaDB 10.4.2 Release Notes

The most recent release of MariaDB 10.4 is:
MariaDB 10.4.32 Stable (GA) Download Now
Alternate download from mariadb.org

Download | Release Notes | Changelog | Overview of 10.4

Release date: 29 Jan 2019

MariaDB 10.4 is the current *development* series of MariaDB. It is an evolution of MariaDB 10.3 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.4.2 is a ***Beta*** release.

Do not use *beta* releases in production!

For an overview of MariaDB 10.4 see the What is MariaDB 10.4? page.

Thanks, and enjoy MariaDB!

Notable Changes

Notable changes of this release include:

- Galera 4 version 26.4.0 has been added in this release, see the **Galera 4 Notes** section for details
- Account Locking (MDEV-13095)
- a number of bugs related to MDEV-15562 Instant DROP COLUMN have been fixed
- New variable, max_password_errors for limiting the number of failed connection attempts by a user.
- MDEV-17475: Maximum value of table_definition_cache is now `2097152` .

Galera 4 Notes

Upgrading to Galera 4 version 26.4.0

Rolling upgrades from earlier 10.3 (or older) MariaDB releases are not supported in this release. For upgrading a 10.3-based cluster, any applications accessing the cluster should be stopped and the cluster shut down. Then for each cluster node the following procedure should be carried out:

- Install MariaDB 10.4.2 and Galera 4 version 26.4.0
- Start MariaDB server, but make sure it is not trying to connect to the cluster by configuring `wsrep_provider=none`
- While MariaDB server is running, run `mysql_upgrade` for the server
- Stop MariaDB server

After that, you can bootstrap the cluster. If there was ongoing application load on the cluster during the initial cluster shutdown phase, you should make sure to bootstrap the cluster with the node which was shutdown last.

We are working on rolling upgrade support for the final GA version of MariaDB 10.4. With a rolling upgrade, a live cluster can be upgraded node by node, and the cluster is able to process application load when having a hybrid setup of 10.3 and 10.4 nodes.

New Features in Galera 26.4.0

The 'mysql' schema contains new Galera replication related tables:

- `wsrep_cluster`
- `wsrep_cluster_members`
- `wsrep_streaming_log`

End users may read but not modify these tables.

The new streaming replication feature allows replicating transactions of unlimited size. With streaming replication, a cluster is replicating a transaction in small fragments during transaction execution. This transaction fragmenting is controlled by two new configuration variables:

- `wsrep_trx_fragment_unit (bytes, rows, statements)` defines the metrics for how to measure transaction size limit for fragmenting. Possible values are:
  - `bytes` : transaction's binlog events buffer size in bytes
  - `rows` : number of rows affected by the transaction
  - `statements` : number of SQL statements executed in the multi-statement transaction
- `wsrep_trx_fragment_size` defines the limit for fragmenting. When a transaction's size, in terms of the configured fragment unit, has grown over this limit, a new fragment will be replicated.

Transactions replicated through galera replication will now process the commit phase using MariaDB's group commit logic. This will affect transaction throughput, especially when binary logging is enabled in the cluster.

Limitations in Galera 26.4.0

- Upgrading from 10.3 version 25.3.25 to 10.4.2 version 26.4.0 must happen on a stopped cluster. Only after all nodes have been upgraded to MariaDB 10.4.2 and Galera 26.4.0 can the cluster be started up
- Splitting transactions of `LOAD DATA` execution will conflict with streaming replication, and should not be used if streaming replication is configured

Changelog

For a complete list of changes made in MariaDB 10.4.2, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.4.2, see the MariaDB Foundation release announcement.

Do not use *beta* releases in production!

---

announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.8.2.32 MariaDB 10.4.1 Release Notes

The most recent release of MariaDB 10.4 is:
MariaDB 10.4.32 Stable (GA) Download Now
Alternate download from mariadb.org

Download | Release Notes | Changelog | Overview of 10.4

Release date: 20 Dec 2018

MariaDB 10.4 is the current *development* series of MariaDB. It is an evolution of MariaDB 10.3 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.4.1 is a ***Beta*** release.

Do not use *beta* releases in production!

For an overview of MariaDB 10.4 see the What is MariaDB 10.4? page.

Thanks, and enjoy MariaDB!

Notable Changes

This will be the first beta release in the MariaDB 10.4 series.

Notable changes of this release include:

Syntax

- New FLUSH SSL command to reload SSL certificates without server restart (MDEV-16266)
- New `CAST` target — `CAST(x AS INTERVAL DAY_SECOND(N))` (MDEV-17776)

Variables

- New sql-mode setting, `TIME_ROUND_FRACTIONAL` (MDEV-16991)
- Two new values for the variable use_stat_tables: `COMPLEMENTARY_FOR_QUERIES` and `PREFERABLY_FOR_QUERIES` (MDEV-17255)
- Engine Independent Table Statistics is now enabled by default; new default values are use_stat_tables=PREFERABLY_FOR_QUERIES and optimizer_use_condition_selectivity=4 (MDEV-15253)
- New variable gtid_cleanup_batch_size for determining how many old rows must accumulate in the mysql.gtid_slave_pos table before a background job will be run to delete them.

Other Features

- Support for window UDF functions via the new method x_remove (MDEV-15073)
- Json service for plugins (MDEV-5313)
- Much faster privilege checks for MariaDB setups with many user accounts or many database grants (MDEV-15649)
- mysql.user table is retired. User accounts and global privileges are now stored in the mysql.global_priv table (MDEV-17658)
- Change in behavior for FLUSH TABLES (MDEV-5336).

Bug Fixes

Lots of miscellaneous fixes, including:

- Bug fixes for MDEV-15562 instant DROP COLUMN

Changelog

For a complete list of changes made in MariaDB 10.4.1, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.4.1, see the MariaDB Foundation release announcement.

Do not use *beta* releases in production!

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions which Include MariaDB page.

7.0.8.2.33 MariaDB 10.4.0 Release Notes

The most recent release of MariaDB 10.4 is:
MariaDB 10.4.32 Stable (GA) Download Now
Alternate download from mariadb.org

Download | Release Notes | Changelog | Overview of 10.4

Release date: 9 Nov 2018

MariaDB 10.4 is the current *development* series of MariaDB. It is an evolution of MariaDB 10.3 with several entirely new features not found anywhere else and with backported and reimplemented features from MySQL.

MariaDB 10.4.0 is an ***Alpha*** release.

Do not use *alpha* releases in production!

For an overview of MariaDB 10.4 see the What is MariaDB 10.4? page.

Thanks, and enjoy MariaDB!

Notable Changes

This will be the first alpha release in the MariaDB 10.4 series.

Notable changes of this release include:

InnoDB

- Added instant DROP COLUMN and changing of the order of columns (MDEV-15562)
- Reduced redo log volume for undo tablespace initialization (MDEV-17138)
- Removed crash-upgrade support for pre-10.2.19 TRUNCATE TABLE (MDEV-13564)
- Added key rotation for innodb_encrypt_log (MDEV-12041)

Optimizer

- Push conditions into materialized IN subqueries (MDEV-12387)

Variables

- Added the tcp_nodelay system variable (MDEV-16277)
- Removed the global Innodb_pages0_read status variable (MDEV-15705).

General

- `IF NOT EXISTS` clause added to INSTALL PLUGIN and `IF EXISTS` clause added to UNINSTALL PLUGIN and UNINSTALL SONAME (MDEV-16294)
- The obsolete mysql.host table is no longer created (MDEV-15851)
- Support of brackets (parentheses) for specifying precedence in UNION/EXCEPT/INTERSECT operations (MDEV-11953)
- Crash safe Aria-based system tables (MDEV-16421)
- Added Linux abstract socket support (MDEV-15655)
- Enabled C++11 (MDEV-16410)
- SET PASSWORD support for ed25519 and other authentication plugins (MDEV-12321)
- Performance improvements in Unicode collations (MDEV-17534, MDEV-17511, MDEV-17502, MDEV-17474)
- User data type plugins (MDEV-4912, in progress)
- Improvements with SQL standard INTERVAL support to help functions TIMESTAMP() and ADDTIME() return more predictable results.
    - Historically, MariaDB uses the TIME data type for both "time of the day" values and "duration" values. In the first meaning the natural value range is from '00:00:00' to '23:59:59.999999', in the second meaning the range is from '-838:59:59.999999' to '+838:59:59.999999'.
    - To remove this ambiguity and for the SQL standard conformance we plan to introduce a dedicated data type INTERVAL that will be able to store values in the range at least from '-87649415:59:59.999999' to '+87649415:59:59.999999', which will be enough to represent the time difference between TIMESTAMP'0001-01-01 00:00:00' and TIMESTAMP'9999-12-31 23:59:59.999999'.
    - As a first step we support this range of values for intermediate calculations when TIME-alike string and numeric values are used in INTERVAL (i.e. duration) context, e.g. as the second argument of SQL functions TIMESTAMP(ts,interval) and ADDTIME(ts,interval), so the following can now be calculated:

```
SELECT ADDTIME(TIMESTAMP'0001-01-01 00:00:00', '87649415:59:59.999999');
-> '9999-12-31 23:59:59.999999'

SELECT TIMESTAMP(DATE'0001-01-01', '87649415:59:59.999999')
-> '9999-12-31 23:59:59.999999'

SELECT ADDTIME(TIME'-838:59:59.999999', '1677:59:59.999998');
-> '838:59:59.999999'
```

Changelog

For a complete list of changes made in MariaDB 10.4.0, with links to detailed information on each push, see the changelog.

Contributors

For a full list of contributors to MariaDB 10.4.0, see the MariaDB Foundation release announcement.

Do not use *alpha* releases in production!

Be notified of new MariaDB Server releases automatically by subscribing to the MariaDB Foundation community announce 'at' lists.mariadb.org announcement list (this is a low traffic, announce-only list). MariaDB plc customers will be notified for all new releases, security issues and critical bug fixes for all MariaDB plc products thanks to the Notification Services.

MariaDB may already be included in your favorite OS distribution. More information can be found on the Distributions

which Include MariaDB page.

7.0.8.3 Changes & Improvements in MariaDB 10.3

MariaDB 10.3 is no longer maintained. Please use a more recent release.

The most recent release of MariaDB 10.3 is:
MariaDB 10.3.39 Stable (GA) Download Now

Contents

MariaDB 10.3 is a previous major stable version. The first stable release was in May 2018, and it was maintained until May 2023.

For details on upgrading from MariaDB 10.2, see Upgrading from MariaDB 10.2 to 10.3.

MariaDB Server 10.3 is included in MariaDB TX 3.0. Watch the webinar recording to learn more about the new features included in this release.

The following lists the major new features in MariaDB 10.3:

Implemented Features

Syntax / General Features

- System-versioned tables (also known as AS OF) (MDEV-12894)
- Table Value Constructors (MDEV-12172) — GSoC 2017 project by Galina Shalygina
- Transform [NOT] IN predicate with long list of values INTO [NOT] IN subquery (MDEV-12176) — GSoC 2017 project by Galina Shalygina
- ROW TYPE OF now supports local SP variables (MDEV-14139)
- Aggregate stored functions (MDEV-7773) — GSoC 2016 project by Varun Gupta
- Support for LIMIT clause in GROUP_CONCAT() (MDEV-11297)
- PERCENTILE_CONT, PERCENTILE_DISC, and MEDIAN window functions (MDEV-12985)
- FOR ... END FOR statement (MDEV-14415)
- XA RECOVER FORMAT='SQL' (MDEV-14593)
- Oracle compatible SUBSTR() function (MDEV-14012) — contribution by Jérôme Brauge
- INVISIBLE columns (MDEV-10177) — GSoC 2016 project by Sachin Setiya

- Various scalability improvements (MDEV-14529, MDEV-14505)
- Sequences can now be used with DEFAULT.
- Semi-sync plugin merged into the server (MDEV-13073) — contribution by Alibaba
- CREATE SEQUENCE (MDEV-10139)
- SHOW CREATE SEQUENCE
- ALTER SEQUENCE
- DROP SEQUENCE
- NEXT VALUE FOR
- PREVIOUS VALUE FOR
- SETVAL()
- INTERSECT and EXCEPT. These are both now reserved words and can no longer be used as an identifier without being quoted (MDEV-10141)
- ROW data type for stored routine variables (MDEV-10914, MDEV-12007, MDEV-12291)
- TYPE OF and ROW TYPE OF anchored data types for stored routine variables (MDEV-12461)
- Cursors with parameters (MDEV-12457)
- DDL Fast Fail - WAIT/NOWAIT (MDEV-11379, MDEV-11388)
- CHR() function (MDEV-12685)
- DELETE statement can delete from the table that is used in a subquery in the `WHERE` clause (MDEV-12137)
- Stored routine parameters can use ROW TYPE OF (MDEV-13581)
- The server now supports the PROXY protocol - see also the new proxy_protocol_networks system variable (MDEV-11159)
- Instant ADD COLUMN (MDEV-11369) — *Tencent Game DBA Team, developed by vinchen.*
- UPDATE statements with the same source and target (MDEV-12874) — *from Jerome Brauge.*
- ORDER BY and LIMIT in multi-table update (MDEV-13911)
- DATE_FORMAT(date, format, locale) - 3 argument form of DATE_FORMAT (MDEV-11553)
- The MariaDB SQL/PL stored procedure dialect (enabled with sql_mode=ORACLE) now supports Oracle style packages. Support for the following statements has been added (MDEV-10591):
  - CREATE PACKAGE
  - CREATE PACKAGE BODY
  - DROP PACKAGE
  - DROP PACKAGE BODY
  - SHOW CREATE PACKAGE
  - SHOW CREATE PACKAGE BODY
- New sql_mode `SIMULTANEOUS_ASSIGNMENT` to make the SET part of the UPDATE statement evaluate all assignments simultaneously, not left-to-right.
- Correctness improvement - TRUNCATE honors transactional locks (MDEV-15061)
- Windows binaries now use high-precision timer when available (MDEV-15694). This makes much less probable for two queries to have the same `CURRENT_TIMESTAMP(6)` value, for example.
- Two new ALTER TABLE ... ALGORITHM options, INSTANT and NOCOPY, which allow operations that would require any data files to be modified, or that would require rebuilding the clustered index respectively, to be refused rather than potentially perform slowly (MDEV-13134)
- mysqldump `--ignore-database` option (MDEV-13336)

Compatibility

- As a result of implementing Table Value Constructors, the VALUES function has been renamed to VALUE() (MDEV-12172)
- When running with sql_mode=ORACLE, the server now understands a subset of Oracle's PL/SQL language instead of the traditional MariaDB syntax for stored routines. See MDEV-10142, MDEV-10764 and SQL_MODE=ORACLE From MariaDB 10.3 to know the current status.
- New sql_mode, EMPTY_STRING_IS_NULL.
- INTERSECT and EXCEPT are both now reserved words and can no longer be used as an identifier without being quoted (MDEV-10141)
- Functions that used to only return 64-bit now can return 32-bit results (MDEV-12619).

Compression

- Storage-engine Independent Column Compression (MDEV-11371) — *Tencent Game DBA Team, developed by willhan, also thanks to AliSQL.*
- On Linux, shrink the core dumps by omitting the InnoDB buffer pool (MDEV-10814)

Encryption

- Temporary files created by merge sort and row log are encrypted if innodb_encrypt_log is set to `1`, regardless of whether the table encrypted or not (MDEV-12634).

Optimizer/Performance

- Condition pushdown through `PARTITION BY` clause of window functions (MDEV-10855)
- New Lateral Derived optimization was introduced.
- Numerous performance improvements for high-concurrency load
- Numerous scalability and performance improvements to global data structures, including MDEV-14756, MDEV-15019, MDEV-14482, MDEV-15059, MDEV-15104
- Performance improvements to persistent data structures: MDEV-15090, MDEV-15132

Storage Engines

InnoDB

- innodb_fast_shutdown now has a new mode, `3`, which skips the rollback of connected transactions (MDEV-15832)

Spider

The Spider storage engine has been updated to 3.3.13. The partitioning storage engine has been updated to support all the new Spider features including:

- Direct join support. This allows Spider to do JOINS and GROUP BYs internally.
- Direct update and delete.
- Direct aggregates.
- slave_transaction_retry_errors and slave-transaction-retry-interval allow more control over handling delays or conflicts when applying binary logs.

Most of the features were done as part of MDEV-7698.

OQGRAPH

- OQGraph now supports the "leaves" algorithm (MDEV-11271) — contribution by Heinz Wiesinger

Partition Engine

- Numerous improvements for the partition engine (MDEV-7698) — contribution by Kentoku Shiba
  - Full text support.
  - Multi-range-read (Gives better performance when handling multiple ranges).
  - Support for condition pushdown.
  - HANDLER support
- Aggregate pushdown
- Bulk update/delete

Information Schema

- The Information Schema is optimized to use much less memory when selecting from INFORMATION_SCHEMA.TABLES or any other table with many VARCHAR or TEXT columns (MDEV-14275)
- The Information Schema Columns table now displays system versioning info in the EXTRA column - MDEV-15062

Logging

- Disable logging of certain statements to the general log or the slow query log with the log_disabled_statements and log_slow_disabled_statements system variables.
- A new option to log_slow_filter, `filsort_priority_queue`.

Replication

- Per-engine mysql.gtid_slave_pos tables (MDEV-12179) *— Implemented by Kristian Nielsen funded by Booking.com.*

Data Type API

10.3 continues refactoring for the data type API started in 10.2, which will make it possible to have user data type plugins. This work is still in progress (see MDEV-4912 for the current status and subtasks). Most of the task in this category do not change the server behavior. Some tasks do have a visible effect.

Idle Transactions

Connections with idle transactions can be automatically killed after a specified time period by means of the idle_transaction_timeout, idle_readonly_transaction_timeout and idle_write_transaction_timeout system variables.

System Variables

For a list of all new variables, see System Variables Added in MariaDB 10.3 and Status Variables Added in MariaDB 10.3.

- New system variable gtid_pos_auto_engines.
- New system variable secure_timestamp for restricting the direct setting of a session timestamp (MDEV-15923)
- session variables tracking is enabled by default (MDEV-11825)
- Remove deprecated variables innodb_file_format, innodb_file_format_check, innodb_file_format_max and innodb_large_prefix.
- version_source_revision - permits seeing which version of the source was used for the build (MDEV-12583).
- Added bind_address as a system variable (MDEV-12542).
- The max value of the max_prepared_stmt_count system variable has been increased from 1048576 to 4294967295
- The proxy_protocol_networks variable can now be modified without restarting the server (MDEV-15501)

Security Vulnerabilities Fixed in MariaDB 10.3

> For a complete list of security vulnerabilities (CVEs) fixed across all versions of MariaDB, see the Security Vulnerabilities Fixed in MariaDB page.

- CVE-2022-47015: MariaDB 10.3.39
- CVE-2022-38791: MariaDB 10.3.36
- CVE-2022-32091: MariaDB 10.3.36
- CVE-2022-32088: MariaDB 10.3.35
- CVE-2022-32087: MariaDB 10.3.35
- CVE-2022-32085: MariaDB 10.3.35
- CVE-2022-32084: MariaDB 10.3.36
- CVE-2022-32083: MariaDB 10.3.35
- CVE-2022-31624: MariaDB 10.3.32
- CVE-2022-27458: MariaDB 10.3.35
- CVE-2022-27456: MariaDB 10.3.35
- CVE-2022-27452: MariaDB 10.3.35
- CVE-2022-27449: MariaDB 10.3.35
- CVE-2022-27448: MariaDB 10.3.35
- CVE-2022-27447: MariaDB 10.3.35
- CVE-2022-27445: MariaDB 10.3.35
- CVE-2022-27387: MariaDB 10.3.35
- CVE-2022-27386: MariaDB 10.3.35
- CVE-2022-27385: MariaDB 10.3.32
- CVE-2022-27384: MariaDB 10.3.35
- CVE-2022-27383: MariaDB 10.3.35
- CVE-2022-27381: MariaDB 10.3.35
- CVE-2022-27380: MariaDB 10.3.35
- CVE-2022-27379: MariaDB 10.3.35
- CVE-2022-27378: MariaDB 10.3.35
- CVE-2022-27377: MariaDB 10.3.35
- CVE-2022-27376: MariaDB 10.3.35
- CVE-2022-24052: MariaDB 10.3.33
- CVE-2022-24051: MariaDB 10.3.33
- CVE-2022-24050: MariaDB 10.3.33
- CVE-2022-24048: MariaDB 10.3.33
- CVE-2022-21595: MariaDB 10.3.33
- CVE-2022-21451: MariaDB 10.3.29
- CVE-2022-21427: MariaDB 10.3.35
- CVE-2022-0778: MariaDB 10.3.33
- CVE-2021-46669: MariaDB 10.3.35
- CVE-2021-46668: MariaDB 10.3.34
- CVE-2021-46667: MariaDB 10.3.32
- CVE-2021-46666: MariaDB 10.3.30
- CVE-2021-46665: MariaDB 10.3.34

- CVE-2021-46664: MariaDB 10.3.34
- CVE-2021-46663: MariaDB 10.3.34
- CVE-2021-46662: MariaDB 10.3.32
- CVE-2021-46661: MariaDB 10.3.34
- CVE-2021-46659: MariaDB 10.3.33
- CVE-2021-46658: MariaDB 10.3.31
- CVE-2021-46657: MariaDB 10.3.30
- CVE-2021-35604: MariaDB 10.3.32
- CVE-2021-27928: MariaDB 10.3.28
- CVE-2021-2389: MariaDB 10.3.31
- CVE-2021-2372: MariaDB 10.3.31
- CVE-2021-2194: MariaDB 10.3.26
- CVE-2021-2166: MariaDB 10.3.29
- CVE-2021-2154: MariaDB 10.3.29
- CVE-2021-2144: MariaDB 10.3.19
- CVE-2021-2022: MariaDB 10.3.24
- CVE-2021-2007: MariaDB 10.3.17
- CVE-2020-2922: MariaDB 10.3.17
- CVE-2020-28912: MariaDB 10.3.26
- CVE-2020-2814: MariaDB 10.3.23
- CVE-2020-2812: MariaDB 10.3.23
- CVE-2020-2780: MariaDB 10.3.19
- CVE-2020-2760: MariaDB 10.3.23
- CVE-2020-2752: MariaDB 10.3.23
- CVE-2020-2574: MariaDB 10.3.22
- CVE-2020-15180: MariaDB 10.3.25
- CVE-2020-14812: MariaDB 10.3.26
- CVE-2020-14789: MariaDB 10.3.26
- CVE-2020-14776: MariaDB 10.3.26
- CVE-2020-14765: MariaDB 10.3.26
- CVE-2020-13249: MariaDB 10.3.23
- CVE-2019-2974: MariaDB 10.3.19
- CVE-2019-2938: MariaDB 10.3.19
- CVE-2019-2805: MariaDB 10.3.17
- CVE-2019-2758: MariaDB 10.3.17
- CVE-2019-2740: MariaDB 10.3.17
- CVE-2019-2739: MariaDB 10.3.17
- CVE-2019-2737: MariaDB 10.3.17
- CVE-2019-2628: MariaDB 10.3.15
- CVE-2019-2627: MariaDB 10.3.15
- CVE-2019-2614: MariaDB 10.3.15
- CVE-2019-2537: MariaDB 10.3.13
- CVE-2019-2510: MariaDB 10.3.13
- CVE-2019-2503: MariaDB 10.3.10
- CVE-2018-3284: MariaDB 10.3.11
- CVE-2018-3282: MariaDB 10.3.11
- CVE-2018-3277: MariaDB 10.3.11
- CVE-2018-3251: MariaDB 10.3.11
- CVE-2018-3200: MariaDB 10.3.11
- CVE-2018-3185: MariaDB 10.3.11
- CVE-2018-3174: MariaDB 10.3.11
- CVE-2018-3173: MariaDB 10.3.11
- CVE-2018-3162: MariaDB 10.3.11
- CVE-2018-3156: MariaDB 10.3.11
- CVE-2018-3143: MariaDB 10.3.11
- CVE-2018-3066: MariaDB 10.3.9
- CVE-2018-3064: MariaDB 10.3.9
- CVE-2018-3063: MariaDB 10.3.9
- CVE-2018-3060: MariaDB 10.3.9
- CVE-2018-3058: MariaDB 10.3.9
- CVE-2018-25032: MariaDB 10.3.36
- CVE-2016-9843: MariaDB 10.3.11

Comparison with MySQL

- System Variable Differences Between MariaDB 10.3 and MySQL 8.0

- Function Differences Between MariaDB 10.3 and MySQL 8.0
- System Variable Differences Between MariaDB 10.3 and MySQL 5.7
- Function Differences Between MariaDB 10.3 and MySQL 5.7

List of All MariaDB 10.3 Releases

Date	Release	Status	Release Notes	Changelog
10 May 2023	**MariaDB 10.3.39**	Stable (GA)	Release Notes	Changelog
6 Feb 2023	**MariaDB 10.3.38**	Stable (GA)	Release Notes	Changelog
7 Nov 2022	**MariaDB 10.3.37**	Stable (GA)	Release Notes	Changelog
15 Aug 2022	**MariaDB 10.3.36**	Stable (GA)	Release Notes	Changelog
20 May 2022	**MariaDB 10.3.35**	Stable (GA)	Release Notes	Changelog
12 Feb 2022	**MariaDB 10.3.34**	Stable (GA)	Release Notes	Changelog
9 Feb 2022	**MariaDB 10.3.33**	Stable (GA)	Release Notes	Changelog
8 Nov 2021	**MariaDB 10.3.32**	Stable (GA)	Release Notes	Changelog
6 Aug 2021	**MariaDB 10.3.31**	Stable (GA)	Release Notes	Changelog
23 Jun 2021	**MariaDB 10.3.30**	Stable (GA)	Release Notes	Changelog
7 May 2021	**MariaDB 10.3.29**	Stable (GA)	Release Notes	Changelog
22 Feb 2021	**MariaDB 10.3.28**	Stable (GA)	Release Notes	Changelog
11 Nov 2020	**MariaDB 10.3.27**	Stable (GA)	Release Notes	Changelog
3 Nov 2020	**MariaDB 10.3.26**	Stable (GA)	Release Notes	Changelog
7 Oct 2020	**MariaDB 10.3.25**	Stable (GA)	Release Notes	Changelog
10 Aug 2020	**MariaDB 10.3.24**	Stable (GA)	Release Notes	Changelog
12 May 2020	**MariaDB 10.3.23**	Stable (GA)	Release Notes	Changelog
28 Jan 2020	**MariaDB 10.3.22**	Stable (GA)	Release Notes	Changelog
11 Dec 2019	**MariaDB 10.3.21**	Stable (GA)	Release Notes	Changelog
8 Nov 2019	**MariaDB 10.3.20**	Stable (GA)	Release Notes	Changelog
5 Nov 2019	**MariaDB 10.3.19**	Stable (GA)	Release Notes	Changelog
11 Sep 2019	**MariaDB 10.3.18**	Stable (GA)	Release Notes	Changelog
31 Jul 2019	**MariaDB 10.3.17**	Stable (GA)	Release Notes	Changelog
17 Jun 2019	**MariaDB 10.3.16**	Stable (GA)	Release Notes	Changelog
14 May 2019	**MariaDB 10.3.15**	Stable (GA)	Release Notes	Changelog
2 Apr 2019	**MariaDB 10.3.14**	Stable (GA)	Release Notes	Changelog
21 Feb 2019	**MariaDB 10.3.13**	Stable (GA)	Release Notes	Changelog
7 Jan 2019	**MariaDB 10.3.12**	Stable (GA)	Release Notes	Changelog
20 Nov 2018	**MariaDB 10.3.11**	Stable (GA)	Release Notes	Changelog
4 Oct 2018	**MariaDB 10.3.10**	Stable (GA)	Release Notes	Changelog
15 Aug 2018	**MariaDB 10.3.9**	Stable (GA)	Release Notes	Changelog
2 Jul 2018	**MariaDB 10.3.8**	Stable (GA)	Release Notes	Changelog
25 May 2018	**MariaDB 10.3.7**	Stable (GA)	Release Notes	Changelog
16 Apr 2018	**MariaDB 10.3.6**	Release Candidate (RC)	Release Notes	Changelog
26 Feb 2018	**MariaDB 10.3.5**	Release Candidate (RC)	Release Notes	Changelog
18 Jan 2018	**MariaDB 10.3.4**	Beta	Release Notes	Changelog
23 Dec 2017	**MariaDB 10.3.3**	Beta	Release Notes	Changelog
9 Oct 2017	**MariaDB 10.3.2**	Alpha	Release Notes	Changelog

29 Aug 2017	**MariaDB 10.3.1**	Alpha	Release Notes	Changelog
16 Apr 2017	**MariaDB 10.3.0**	Alpha	Release Notes	Changelog

7.0.9 MariaDB Server 10.2

Changes & Improvements in MariaDB 10.2
Current Version: 10.2.44 | Status: Stable (GA) | Release Date: 20 May 2022

Release Notes - MariaDB 10.2 Series
MariaDB 10.2 Series Release Notes

Changelogs - MariaDB 10.2 Series
MariaDB 10.2 changelogs.

7.0.9.1 Changes & Improvements in MariaDB 10.2

MariaDB 10.2 is no longer maintained. Please use a more recent release.

The most recent release of MariaDB 10.2 is:
MariaDB 10.2.44 Stable (GA) Download Now

Contents

MariaDB 10.2 is a previous major stable version. The first stable release was in May 2017.

For details on upgrading from MariaDB 10.1, see Upgrading from MariaDB 10.1 to 10.2.

The following lists the major new features in MariaDB 10.2:

Implemented Features

InnoDB as Default

- InnoDB is now the default storage engine. Until MariaDB 10.1, MariaDB used the XtraDB storage engine as default. XtraDB in 10.2 is not up to date with the latest features of InnoDB and cannot be used. As the InnoDB on disk format is identical to XtraDB's this will not cause any problems when upgrading to MariaDB 10.2. See Why does MariaDB 10.2 use InnoDB instead of XtraDB?

Syntax / General Features

- MyRocks storage engine added. (It has its own maturity level. In MariaDB 10.2.14, it is considered Gamma) (MDEV-9658)
- Window functions have been introduced.
- The SHOW CREATE USER statement was introduced
- New CREATE USER options for limiting resource usage and tls/ssl
- New ALTER USER statement
- Non-recursive Common Table Expressions
- Recursive Common Table Expressions (MDEV-9864)
- New WITH statement. `WITH` is a common table expression that allows you to refer to a subquery expression many times in a query (MDEV-8308 & MDEV-9864) — *Original code from Galina Shalygina*
- Support for CHECK CONSTRAINT (MDEV-7563)
- Support for DEFAULT with expressions (MDEV-10134)
- BLOB and TEXT fields can now have a DEFAULT value
- Lots of restrictions lifted for Virtual computed columns
- Number of supported decimals in DECIMAL has increased from `30` to `38` (MDEV-10138)
- Added catchall for list partitions (MDEV-8348)
- Oracle-style EXECUTE IMMEDIATE statement (MDEV-10585)
- PREPARE Statement/Dynamic SQL now understand most expressions (MDEV-10866, MDEV-10709).
- InnoDB tables now support spatial indexes
- ed25519 authentication plugin (MDEV-12160)
- Better InnoDB crash recovery progress reporting (MDEV-11027)
- Improvements to InnoDB startup/shutdown to make it more robust
- AWS Key Management plugin added for Windows, CentOS, RHEL, and Fedora packages
- Atomic writes support made more general. Shannon system SSD cards are now supported.

Incompatible Changes

- TokuDB has been split into a separate package, mariadb-plugin-tokudb.
- SQL_MODE has been changed; in particular, NOT NULL fields with no default will no longer fall back to a dummy value for inserts which do not specify a value for that field.
- Replication from legacy MySQL servers may require setting binlog_checksum to NONE.
- New reserved words: OVER, RECURSIVE, and ROWS.

Triggers

- Multiple triggers for the same event (MDEV-6112)
- The FOLLOWS/PRECEDES clauses have been added to the CREATE TRIGGER statement
- Multiple triggers are now counted in the Executed_triggers status variable (MDEV-10915)
- SHOW TRIGGERS and SHOW CREATE TRIGGER now include the date and time the trigger was created

Replication / Binary Log

- DML_only flashback can rollback instances/databases/tables to an old snapshot (MDEV-10570)
- New variable read_binlog_speed_limit permits restricting the speed at which the slave reads the binlog from the master (MDEV-11064) — *Original code from Tencent Game DBA Team, developed by chouryzhou.*
- Delayed replication is supported (MDEV-7145) — *Backported from MySQL 5.6 by Kristian Nielsen funded by Booking.com.*
- Compression of events in the binary log is supported (MDEV-11065) — *Original code from Tencent Game DBA Team, developed by vinchen.*
- Default binary log format changed to mixed (MDEV-7635)
- Default value of replicate_annotate_row_events changed to `ON` (MDEV-7635)
- Default value of slave_net_timeout reduced to 60 seconds (MDEV-7635)
- Default server_id changed from `0` to `1`

GeoJSON / JSON

- The JSON data type (an alias for LONGTEXT) was introduced.
- JSON functions added (MDEV-9143)
- Implement ST_AsGeoJSON and ST_GeomFromGeoJSON functions so the spatial features can be imported/exported using GeoJSON format (MDEV-11042)

Information Schema

- An information schema plugin to report all user variables, which creates the Information Schema USER_VARIABLES

Table (MDEV-7331)
- Changes to the Information Schema COLUMNS table. Literals are now quoted in the `COLUMN_DEFAULT` column to distinguish them from expressions (MDEV-13132), and two new columns added providing information about generated (virtual, or computed) columns (MDEV-9255).

EXPLAIN

- EXPLAIN FORMAT=JSON now shows `outer_ref_condition` field which contains the condition that the(?) SELECT checks on each re-execution (MDEV-9652)
- EXPLAIN FORMAT=JSON now shows `sort_key` field which shows the sort criteria used by `filesort` operation. (commit 2078392)
- EXPLAIN used to show incorrect information about how the optimizer resolved `ORDER BY` clause or `Distinct`. This was a long-standing problem with roots back in MySQL. Now, after MDEV-8646 and related fixes, the problem doesn't exist anymore. (For test cases, see MDEV-7982, MDEV-8857, MDEV-7885, MDEV-326)

Optimizations

- Connection setup was made faster by moving creation of THD to new thread (MDEV-6150)
- Pushdown conditions into non-mergeable views/derived tables (MDEV-9197, condition-pushdown-into-derived-table-optimization) — *Original code from Galina Shalygina*
- ANALYZE TABLE has been re-implemented so as not to lock the entire table when collecting engine independent statistics (MDEV-7901)
- Internal CRC32 routines use the optimized implementation on Power8 — MDEV-9872
- Table cache can automatically partition itself as needed to reduce the contention (MDEV-10296)

Compatibility

- 88 new NO PAD collations added. In `NO PAD` collations, end spaces are significant in comparisons (MDEV-9711) — *Original code from Daniil Medvedev*
- MariaDB now works when started with a MySQL 5.7.6+ data directory (MDEV-11170)

CONNECT

- Zipped File Tables for the CONNECT storage engine (MDEV-11295)
- The CONNECT engine now supports the JDBC Table type (MDEV-9765)

System Variables

For a list of all new system variables, see System Variables Added in MariaDB 10.2. Variable changes include:

- New variable to disable deadlock detection innodb_deadlock_detect
- aria_recover has been renamed to aria_recover_options (MDEV-8542)
- Default values of the aria_recover and myisam_recover_options system variables changed to `BACKUP,QUICK`
- The server version can now be faked to work around dated applications that require a particular version string (MDEV-7780)
- slave_parallel_workers is now an alias for slave_parallel_threads
- New status variables com_alter_user, com_multi and com_show_create_user
- New variable for setting a directory for storing temporary non-tablespace InnoDB files, innodb_tmpdir
- New variable read_binlog_speed_limit permits restricting the speed at which the slave reads the binlog from the master (MDEV-11064)
- innodb_log_files_in_group can now be set to `1` (MDEV-12061)
- The thread pool now gives higher priority to connections that have an active transaction. This can be controlled with the new thread_pool_prio_kickup_timer and thread_pool_priority system variables. (MDEV-10297)
- Default value of group_concat_max_len changed to 1M (MDEV-7635)
- Default value of sql_mode changed to STRICT_TRANS_TABLES,ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITU (MDEV-7635) (MariaDB 10.2.4 and later)
- Default value of innodb_compression_algorithm changed to `zlib` - this does not mean pages are now compressed by default, see compression (MDEV-11838)
- Default value of innodb_log_compressed_pages changed to `ON` from MariaDB 10.1.2 to MariaDB 10.1.25 (MDEV-7635 and MDEV-13247)
- Default value of innodb_use_atomic_writes and innodb_use_trim changed to `ON`
- The unused innodb_api_* variables have been removed (MDEV-12050)
- tmp_disk_table_size was added to allow one to limit the size of temporary disk tables stored in tmpdir. At the same time tmp_memory_table_size was added an alias for tmp_table_size. At some point we plan to deprecate

`tmp_table_size`. (MariaDB 10.2.7 and later).

Status Variables

For a list of all new status variables, see Status Variables Added in MariaDB 10.2.

Scripts

- Continuous binary log backup has been added to mysqlbinlog (MDEV-8713)
- mysql_zap and mysqlbug have been removed (MDEV-7376, MDEV-8654)

Other Changes

- Added support for OpenSSL 1.1 and LibreSSL (MDEV-10332)
- Persistent AUTO_INCREMENT for InnoDB (MDEV-6076)
- Support COM_RESET_CONNECTION (MDEV-10340)
- "fast mutexes" have been removed. These aren't faster than normal mutexes, and have been disabled by default for years (MDEV-8111)
- Old GPL client library is gone; now MariaDB Server comes with the LGPL Connector/C client library (MDEV-9055)
- MariaDB is no longer compiled with jemalloc
- TokuDB is now a separate package, not part of the server RPM (because TokuDB still needs jemalloc).
- Upgrading to a new major release no longer requires setting innodb_fast_shutdown to `0`. Omitting it can make the upgrade process a lot faster. (MDEV-12289)

Security Vulnerabilities Fixed in MariaDB 10.2

> For a complete list of security vulnerabilities (CVEs) fixed across all versions of MariaDB, see the Security Vulnerabilities Fixed in MariaDB page.

- CVE-2022-32088: MariaDB 10.2.44
- CVE-2022-32083: MariaDB 10.2.44
- CVE-2022-31624: MariaDB 10.2.41
- CVE-2022-27445: MariaDB 10.2.44
- CVE-2022-27387: MariaDB 10.2.44
- CVE-2022-27386: MariaDB 10.2.44
- CVE-2022-27384: MariaDB 10.2.44
- CVE-2022-27383: MariaDB 10.2.44
- CVE-2022-27381: MariaDB 10.2.44
- CVE-2022-27380: MariaDB 10.2.44
- CVE-2022-27378: MariaDB 10.2.44
- CVE-2022-27377: MariaDB 10.2.44
- CVE-2022-24052: MariaDB 10.2.42
- CVE-2022-24051: MariaDB 10.2.42
- CVE-2022-24050: MariaDB 10.2.42
- CVE-2022-24048: MariaDB 10.2.42
- CVE-2022-21595: MariaDB 10.2.42
- CVE-2022-21451: MariaDB 10.2.38
- CVE-2022-21427: MariaDB 10.2.44
- CVE-2022-0778: MariaDB 10.2.42
- CVE-2021-46669: MariaDB 10.2.44
- CVE-2021-46668: MariaDB 10.2.43
- CVE-2021-46667: MariaDB 10.2.41
- CVE-2021-46666: MariaDB 10.2.39
- CVE-2021-46665: MariaDB 10.2.43
- CVE-2021-46664: MariaDB 10.2.43
- CVE-2021-46663: MariaDB 10.2.43
- CVE-2021-46661: MariaDB 10.2.43
- CVE-2021-46659: MariaDB 10.2.42
- CVE-2021-46658: MariaDB 10.2.40
- CVE-2021-46657: MariaDB 10.2.39
- CVE-2021-35604: MariaDB 10.2.41
- CVE-2021-27928: MariaDB 10.2.37
- CVE-2021-2389: MariaDB 10.2.40
- CVE-2021-2372: MariaDB 10.2.40

- CVE-2021-2194: MariaDB 10.2.35
- CVE-2021-2180: MariaDB 10.2.38
- CVE-2021-2174: MariaDB 10.2.18
- CVE-2021-2166: MariaDB 10.2.38
- CVE-2021-2154: MariaDB 10.2.38
- CVE-2021-2144: MariaDB 10.2.28
- CVE-2021-2022: MariaDB 10.2.33
- CVE-2021-2011: MariaDB 10.2.15
- CVE-2021-2007: MariaDB 10.2.26
- CVE-2020-2922: MariaDB 10.2.26
- CVE-2020-28912: MariaDB 10.2.35
- CVE-2020-2814: MariaDB 10.2.32
- CVE-2020-2812: MariaDB 10.2.32
- CVE-2020-2780: MariaDB 10.2.28
- CVE-2020-2760: MariaDB 10.2.32
- CVE-2020-2752: MariaDB 10.2.32
- CVE-2020-2574: MariaDB 10.2.31
- CVE-2020-15180: MariaDB 10.2.34
- CVE-2020-14812: MariaDB 10.2.35
- CVE-2020-14789: MariaDB 10.2.35
- CVE-2020-14776: MariaDB 10.2.35
- CVE-2020-14765: MariaDB 10.2.35
- CVE-2020-14550: MariaDB 10.2.15
- CVE-2020-13249: MariaDB 10.2.32
- CVE-2019-2974: MariaDB 10.2.28
- CVE-2019-2938: MariaDB 10.2.28
- CVE-2019-2805: MariaDB 10.2.26
- CVE-2019-2758: MariaDB 10.2.26
- CVE-2019-2740: MariaDB 10.2.26
- CVE-2019-2739: MariaDB 10.2.26
- CVE-2019-2737: MariaDB 10.2.26
- CVE-2019-2628: MariaDB 10.2.24
- CVE-2019-2627: MariaDB 10.2.24
- CVE-2019-2614: MariaDB 10.2.24
- CVE-2019-2537: MariaDB 10.2.22
- CVE-2019-2510: MariaDB 10.2.22
- CVE-2019-2503: MariaDB 10.2.18
- CVE-2019-2455: MariaDB 10.2.15
- CVE-2018-3284: MariaDB 10.2.19
- CVE-2018-3282: MariaDB 10.2.19
- CVE-2018-3277: MariaDB 10.2.19
- CVE-2018-3251: MariaDB 10.2.19
- CVE-2018-3200: MariaDB 10.2.19
- CVE-2018-3185: MariaDB 10.2.19
- CVE-2018-3174: MariaDB 10.2.19
- CVE-2018-3173: MariaDB 10.2.19
- CVE-2018-3162: MariaDB 10.2.19
- CVE-2018-3156: MariaDB 10.2.19
- CVE-2018-3143: MariaDB 10.2.19
- CVE-2018-3133: MariaDB 10.2.12
- CVE-2018-3081: MariaDB 10.2.15
- CVE-2018-3066: MariaDB 10.2.17
- CVE-2018-3064: MariaDB 10.2.17
- CVE-2018-3063: MariaDB 10.2.17
- CVE-2018-3060: MariaDB 10.2.17
- CVE-2018-3058: MariaDB 10.2.17
- CVE-2018-2819: MariaDB 10.2.15
- CVE-2018-2817: MariaDB 10.2.15
- CVE-2018-2813: MariaDB 10.2.15
- CVE-2018-2810: MariaDB 10.2.15
- CVE-2018-2787: MariaDB 10.2.15
- CVE-2018-2786: MariaDB 10.2.15
- CVE-2018-2784: MariaDB 10.2.15
- CVE-2018-2782: MariaDB 10.2.15
- CVE-2018-2781: MariaDB 10.2.15
- CVE-2018-2777: MariaDB 10.2.15
- CVE-2018-2771: MariaDB 10.2.15

- CVE-2018-2767: MariaDB 10.2.15
- CVE-2018-2766: MariaDB 10.2.15
- CVE-2018-2761: MariaDB 10.2.15
- CVE-2018-2759: MariaDB 10.2.15
- CVE-2018-2755: MariaDB 10.2.15
- CVE-2018-2668: MariaDB 10.2.13
- CVE-2018-2665: MariaDB 10.2.13
- CVE-2018-2640: MariaDB 10.2.13
- CVE-2018-2622: MariaDB 10.2.13
- CVE-2018-2612: MariaDB 10.2.13
- CVE-2018-2562: MariaDB 10.2.13
- CVE-2017-3653: MariaDB 10.2.8
- CVE-2017-3641: MariaDB 10.2.8
- CVE-2017-3636: MariaDB 10.2.8
- CVE-2017-3464: MariaDB 10.2.6
- CVE-2017-3456: MariaDB 10.2.6
- CVE-2017-3453: MariaDB 10.2.6
- CVE-2017-3313: MariaDB 10.2.5
- CVE-2017-3309: MariaDB 10.2.6
- CVE-2017-3308: MariaDB 10.2.6
- CVE-2017-3302: MariaDB 10.2.5
- CVE-2017-3257: MariaDB 10.2.8
- CVE-2017-15365: MariaDB 10.2.10
- CVE-2017-10384: MariaDB 10.2.8
- CVE-2017-10379: MariaDB 10.2.8
- CVE-2017-10378: MariaDB 10.2.10
- CVE-2017-10365: MariaDB 10.2.8
- CVE-2017-10320: MariaDB 10.2.8
- CVE-2017-10286: MariaDB 10.2.8
- CVE-2017-10268: MariaDB 10.2.10
- CVE-2016-9843: MariaDB 10.2.19

Comparison with MySQL

- Incompatibilities and Feature Differences Between MariaDB 10.2 and MySQL 5.7
- System Variable Differences Between MariaDB 10.2 and MySQL 5.7
- Function Differences Between MariaDB 10.2 and MySQL 5.7
- System Variable Differences Between MariaDB 10.2 and MySQL 5.6
- Function Differences Between MariaDB 10.2 and MySQL 5.6

List of All MariaDB 10.2 Releases

Date	Release	Status	Release Notes	Changelog
20 May 2022	**MariaDB 10.2.44**	Stable (GA)	Release Notes	Changelog
12 Feb 2022	**MariaDB 10.2.43**	Stable (GA)	Release Notes	Changelog
9 Feb 2022	**MariaDB 10.2.42**	Stable (GA)	Release Notes	Changelog
8 Nov 2021	**MariaDB 10.2.41**	Stable (GA)	Release Notes	Changelog
6 Aug 2021	**MariaDB 10.2.40**	Stable (GA)	Release Notes	Changelog
23 Jun 2021	**MariaDB 10.2.39**	Stable (GA)	Release Notes	Changelog
7 May 2021	**MariaDB 10.2.38**	Stable (GA)	Release Notes	Changelog
22 Feb 2021	**MariaDB 10.2.37**	Stable (GA)	Release Notes	Changelog
11 Nov 2020	**MariaDB 10.2.36**	Stable (GA)	Release Notes	Changelog
3 Nov 2020	**MariaDB 10.2.35**	Stable (GA)	Release Notes	Changelog
7 Oct 2020	**MariaDB 10.2.34**	Stable (GA)	Release Notes	Changelog
10 Aug 2020	**MariaDB 10.2.33**	Stable (GA)	Release Notes	Changelog
12 May 2020	**MariaDB 10.2.32**	Stable (GA)	Release Notes	Changelog
28 Jan 2020	**MariaDB 10.2.31**	Stable (GA)	Release Notes	Changelog

11 Dec 2019	**MariaDB 10.2.30**	Stable (GA)	Release Notes	Changelog
8 Nov 2019	**MariaDB 10.2.29**	Stable (GA)	Release Notes	Changelog
5 Nov 2019	**MariaDB 10.2.28**	Stable (GA)	Release Notes	Changelog
11 Sep 2019	**MariaDB 10.2.27**	Stable (GA)	Release Notes	Changelog
31 Jul 2019	**MariaDB 10.2.26**	Stable (GA)	Release Notes	Changelog
17 Jun 2019	**MariaDB 10.2.25**	Stable (GA)	Release Notes	Changelog
9 May 2019	**MariaDB 10.2.24**	Stable (GA)	Release Notes	Changelog
25 Mar 2019	**MariaDB 10.2.23**	Stable (GA)	Release Notes	Changelog
11 Feb 2019	**MariaDB 10.2.22**	Stable (GA)	Release Notes	Changelog
2 Jan 2019	**MariaDB 10.2.21**	Stable (GA)	Release Notes	Changelog
24 Dec 2018	**MariaDB 10.2.20**	Stable (GA)	Release Notes	Changelog
19 Nov 2018	**MariaDB 10.2.19**	Stable (GA)	Release Notes	Changelog
25 Sep 2018	**MariaDB 10.2.18**	Stable (GA)	Release Notes	Changelog
14 Aug 2018	**MariaDB 10.2.17**	Stable (GA)	Release Notes	Changelog
26 Jun 2018	**MariaDB 10.2.16**	Stable (GA)	Release Notes	Changelog
17 May 2018	**MariaDB 10.2.15**	Stable (GA)	Release Notes	Changelog
27 Mar 2018	**MariaDB 10.2.14**	Stable (GA)	Release Notes	Changelog
13 Feb 2018	**MariaDB 10.2.13**	Stable (GA)	Release Notes	Changelog
4 Jan 2018	**MariaDB 10.2.12**	Stable (GA)	Release Notes	Changelog
28 Nov 2017	**MariaDB 10.2.11**	Stable (GA)	Release Notes	Changelog
31 Oct 2017	**MariaDB 10.2.10**	Stable (GA)	Release Notes	Changelog
27 Sep 2017	**MariaDB 10.2.9**	Stable (GA)	Release Notes	Changelog
18 Aug 2017	**MariaDB 10.2.8**	Stable (GA)	Release Notes	Changelog
12 Jul 2017	**MariaDB 10.2.7**	Stable (GA)	Release Notes	Changelog
23 May 2017	**MariaDB 10.2.6**	Stable (GA)	Release Notes	Changelog
5 Apr 2017	**MariaDB 10.2.5**	RC	Release Notes	Changelog
17 Feb 2017	**MariaDB 10.2.4**	RC	Release Notes	Changelog
24 Dec 2016	**MariaDB 10.2.3**	Beta	Release Notes	Changelog
27 Sep 2016	**MariaDB 10.2.2**	Beta	Release Notes	Changelog
4 Jul 2016	**MariaDB 10.2.1**	Alpha	Release Notes	Changelog
18 Apr 2016	**MariaDB 10.2.0**	Alpha	Release Notes	Changelog

7.0.10 MariaDB Server 10.1

Changes & Improvements in MariaDB 10.1

Current Version: 10.1.48 | Status: Stable (GA) | Release Date: 3 Nov 2020

Release Notes - MariaDB 10.1 Series

MariaDB 10.1 Series Release Notes

Changelogs - MariaDB 10.1 Series

MariaDB 10.1 changelogs.

There are 1 related questions.

7.0.10.1 Changes & Improvements in MariaDB 10.1

MariaDB 10.1 is no longer supported. Please use a more recent release.

The most recent release of MariaDB 10.1 is:
MariaDB 10.1.48 Stable (GA) Download Now

Contents

MariaDB 10.1 is a previous stable version of MariaDB. The first stable release was in October 2014.

For details on upgrading from MariaDB 10.0, see Upgrading from MariaDB 10.0 to 10.1.

The following lists the major new features in MariaDB 10.1:

Implemented Features

Galera

- Galera, a true multi-master solution, is a standard part of MariaDB 10.1.
- Two new Information Schema tables for examining wsrep information, WSREP_MEMBERSHIP and WSREP_STATUS (MDEV-7053)

Encryption

- Table, Tablespace and log Encryption.

Page Compression

- InnoDB/XtraDB Page Compression
- Page compression for FusionIO.

Replication

- Optimistic mode of in-order parallel replication (MDEV-6676)
- domain_id based replication filters - see CHANGE MASTER TO (MDEV-6593)
- Enhanced semisync replication; Wait for at least one slave to acknowledge transaction before committing (MDEV-162).
- Triggers can now be run on the slave for row-based events.
- Dump Thread Enhancements from Google. Makes multiple slave setups faster by allowing concurrent reading of binary log. (MDEV-7257)
- Commits in certain instances in parallel replication complete immediately, avoiding losing throughput when many

transactions need conflicting locks. See binlog_commit_wait_count.
- RESET_MASTER is extended with `TO #` which allows one to specify the number of the first binary log. (MDEV-8469)
- Due to the implementation of SQL standards-compliant behavior when dealing with Primary Keys with Nullable Columns, in certain edge cases, there may be replication issues when replicating from a MariaDB 10.0 master to a MariaDB 10.1 slave using statement-based replication. See MDEV-12248 .

Roles

- SET DEFAULT ROLE (MDEV-5210).
- New columns for the INFORMATION_SCHEMA.APPLICABLE_ROLES table.

Optimization

- ORDER BY optimization is improved by several fixes for real-world cases.
- Don't create frm files for temporary tables (MDEV-4260).
- MAX_STATEMENT_TIME can be used to automatically abort long running queries. (MDEV-4427).
- UNION ALL works without usage of a temporary table (MDEV-334). The feature was backported from MySQL 5.7
- Scalability fixes (MDEV-7004). Up to 60% higher throughput in sysbench benchmarks on Power8.
- Make simple queries faster as we call malloc() fewer times.
- Automatic discovery of performance schema tables (simpler mysql_install_db code). (MDEV-4262), Performance Schema tables no longer use `.frm` files.
- Other Webscale patches (MDEV-6039)
- MDEV-7728 xid cache scalability was significantly improved (by using lock-free hash)

GIS

- Support for Spatial Reference systems for the GIS data (MDEV-60), new `REF_SYSTEM_ID` column attribute can be used to specify Spatial Reference System ID for columns of spatial data types:
- More functions from the OGC standard added (MDEV-4045):
  - ST_Boundary
  - ST_ConvexHull
  - ST_IsRing
  - ST_PointOnSurface
  - ST_Relate
- INFORMATION_SCHEMA.GEOMETRY_COLUMNS table.
- INFORMATION_SCHEMA.SPATIAL_REF_SYS table.

Syntax

- Consistent support for `IF EXISTS` , `IF NOT EXISTS` , and `OR REPLACE` clauses:
  - CREATE DATABASE (MDEV-7280)
  - CREATE FUNCTION UDF (MDEV-7283)
  - CREATE ROLE (MDEV-7288)
  - CREATE SERVER (MDEV-7285)
  - CREATE USER (MDEV-7288)
  - CREATE VIEW (MDEV-7283)
  - DROP ROLE (MDEV-7288)
  - DROP USER (MDEV-7288)
  - CREATE EVENT and DROP EVENT (MDEV-7281)
  - CREATE INDEX and DROP INDEX (MDEV-7284)
  - CREATE TRIGGER and DROP TRIGGER (MDEV-7286)
- Information Schema plugins can now support SHOW and FLUSH statements . New statements include:
  - SHOW QUERY_RESPONSE_TIME
  - FLUSH QUERY_RESPONSE_TIME
  - SHOW LOCALES
- New columns for the INFORMATION_SCHEMA.APPLICABLE_ROLES and INFORMATION_SCHEMA.VIEWS tables.
- ANALYZE statement provides output that looks like EXPLAIN output, but also includes data from the query execution (how many rows were actually read, etc).
- EXPLAIN FORMAT=JSON is a re-implementation of similar feature in MySQL 5.6
- ANALYZE FORMAT=JSON produces detailed information about the statement execution
- GET_LOCK() now supports microseconds in the timeout, no longer rounding fractions to the nearest integer (MDEV-4018)
- Compound statements can be used outside of stored programs.

- The number of rows affected by a slow UPDATE or DELETE is now recorded in the slow query log - see also mysql.slow_log Table. (MDEV-4412)
- SQL standards-compliant behavior when dealing with Primary Keys with Nullable Columns. Note that this could cause replication issues in certain edge cases when replicating from a MariaDB 10.0 master to a MariaDB 10.1 slave. using statement-based replication. See MDEV-12248.
- Explicit or implicit casts from MAX(string) to INT, DOUBLE or DECIMAL now produce warnings (MDEV-8852).

XtraDB / InnoDB

- Allow up to 64K pages in InnoDB (old limit was 16K) (MDEV-6075).
- The Facebook/Kakao defragmentation patch (see Defragmenting InnoDB Tablespaces) which uses OPTIMIZE TABLE to defragment InnoDB tablespaces).

Collations

- Added the `utf8_thai_520_w2`, `utf8mb4_thai_520_w2`, `ucs2_thai_520_w2`, `utf16_thai_520_w2` and `utf32_thai_520_w2` collations.

Variables

For a list of all new variables, see System Variables Added in MariaDB 10.1 and Status Variables Added in MariaDB 10.1. Some of these, and other variable-related changes, include:

- INFORMATION_SCHEMA.SYSTEM_VARIABLES gives information, like description and value origin, for system variables (MDEV-6138).
- MDEV-6858 New server variable enforce_storage_engine
- New status variables to show the number of grants on different objects (see Status Variables Added in MariaDB 10.1)
- Default size of query_alloc_block_size changed from `8192` to `16384` and query_prealloc_size from `8192` to `24576` to avoid the need for simple queries with one join to call `my_malloc`.
- Added variable default-tmp-storage-engine (From MySQL 5.6) (MDEV-6107).
- SET STATEMENT - set variables for the duration of the query (MDEV-5231). This is a backport of Per-query Variable Statement feature of Percona Server 5.6 (which, in turn, is based in MySQL GSoC 2009 project by Joseph Lukas), with many bugs fixed.
- --mysql56-temporal-format option to use the MySQL-5.6 low level formats to store TIME, DATETIME and TIMESTAMP types. (MDEV-5528)
- Backport innodb_default_row_format (MDEV-14904)
- `mysqld --help --verbose` now shows valid variables for ENUM variables. (MDEV-6137)
- MDEV-6981 New status variables to track MASTER_GTID_WAIT time.
- MDEV-7198 New status variable Slave_skipped_errors.
- `--silent-startup` mysqld option. If specified, mysqld does not print Notes to the error log during startup.

Plugins

- Password validation plugin API (MDEV-6431).
- simple_password_check password validation plugin. It can enforce a minimum password length and guarantee that a password contains at least a specified number of uppercase and lowercase letters, digits, and punctuation characters.
- cracklib_password_check password validation plugin. It only allows passwords that are strong enough to pass CrackLib test. This is the same test that `pam_cracklib.so` does, installed by default on many Linux distributions.
- ed25519 authentication plugin for traditional password-based authentication. A new, secure alternative to the old `mysql_native_password` plugin.

Security

- Enhance security using special compilation options - MariaDB is now compiled with security hardening options by default. It is an additional protection layer that makes new, yet unknown, security vulnerabilities more difficult to exploit. (MDEV-5730)

Security Vulnerabilities Fixed in MariaDB 10.1

> For a complete list of security vulnerabilities (CVEs) fixed across all versions of MariaDB, see the Security Vulnerabilities Fixed in MariaDB page.

- CVE-2021-2144: MariaDB 10.1.42
- CVE-2021-2022: MariaDB 10.1.46
- CVE-2021-2011: MariaDB 10.1.33
- CVE-2021-2007: MariaDB 10.1.41
- CVE-2020-2922: MariaDB 10.1.41
- CVE-2020-28912: MariaDB 10.1.48
- CVE-2020-2814: MariaDB 10.1.45
- CVE-2020-2812: MariaDB 10.1.45
- CVE-2020-2780: MariaDB 10.1.42
- CVE-2020-2752: MariaDB 10.1.45
- CVE-2020-2574: MariaDB 10.1.44
- CVE-2020-15180: MariaDB 10.1.47
- CVE-2020-14812: MariaDB 10.1.48
- CVE-2020-14765: MariaDB 10.1.48
- CVE-2020-14550: MariaDB 10.1.33
- CVE-2019-2974: MariaDB 10.1.42
- CVE-2019-2805: MariaDB 10.1.41
- CVE-2019-2740: MariaDB 10.1.41
- CVE-2019-2739: MariaDB 10.1.41
- CVE-2019-2737: MariaDB 10.1.41
- CVE-2019-2627: MariaDB 10.1.39
- CVE-2019-2614: MariaDB 10.1.39
- CVE-2019-2537: MariaDB 10.1.38
- CVE-2019-2529: MariaDB 10.1.38
- CVE-2019-2503: MariaDB 10.1.36
- CVE-2019-2455: MariaDB 10.1.33
- CVE-2018-3282: MariaDB 10.1.37
- CVE-2018-3251: MariaDB 10.1.37
- CVE-2018-3174: MariaDB 10.1.37
- CVE-2018-3156: MariaDB 10.1.37
- CVE-2018-3143: MariaDB 10.1.37
- CVE-2018-3133: MariaDB 10.1.30
- CVE-2018-3081: MariaDB 10.1.33
- CVE-2018-3066: MariaDB 10.1.35
- CVE-2018-3064: MariaDB 10.1.35
- CVE-2018-3063: MariaDB 10.1.35
- CVE-2018-3058: MariaDB 10.1.35
- CVE-2018-2819: MariaDB 10.1.33
- CVE-2018-2817: MariaDB 10.1.33
- CVE-2018-2813: MariaDB 10.1.33
- CVE-2018-2787: MariaDB 10.1.33
- CVE-2018-2784: MariaDB 10.1.33
- CVE-2018-2782: MariaDB 10.1.33
- CVE-2018-2781: MariaDB 10.1.33
- CVE-2018-2771: MariaDB 10.1.33
- CVE-2018-2767: MariaDB 10.1.33
- CVE-2018-2766: MariaDB 10.1.33
- CVE-2018-2761: MariaDB 10.1.33
- CVE-2018-2755: MariaDB 10.1.33
- CVE-2018-2668: MariaDB 10.1.31
- CVE-2018-2665: MariaDB 10.1.31
- CVE-2018-2640: MariaDB 10.1.31
- CVE-2018-2622: MariaDB 10.1.31
- CVE-2018-2612: MariaDB 10.1.31
- CVE-2018-2562: MariaDB 10.1.31
- CVE-2017-3653: MariaDB 10.1.26
- CVE-2017-3651: MariaDB 10.1.19
- CVE-2017-3641: MariaDB 10.1.26
- CVE-2017-3636: MariaDB 10.1.26
- CVE-2017-3600: MariaDB 10.1.19
- CVE-2017-3464: MariaDB 10.1.23
- CVE-2017-3456: MariaDB 10.1.23
- CVE-2017-3453: MariaDB 10.1.23
- CVE-2017-3318: MariaDB 10.1.21
- CVE-2017-3317: MariaDB 10.1.21
- CVE-2017-3313: MariaDB 10.1.22
- CVE-2017-3312: MariaDB 10.1.21

- CVE-2017-3309: MariaDB 10.1.23
- CVE-2017-3308: MariaDB 10.1.23
- CVE-2017-3302: MariaDB 10.1.22
- CVE-2017-3291: MariaDB 10.1.21
- CVE-2017-3265: MariaDB 10.1.21
- CVE-2017-3258: MariaDB 10.1.21
- CVE-2017-3257: MariaDB 10.1.21
- CVE-2017-3244: MariaDB 10.1.21
- CVE-2017-3243: MariaDB 10.1.21
- CVE-2017-3238: MariaDB 10.1.21
- CVE-2017-15365: MariaDB 10.1.30
- CVE-2017-10384: MariaDB 10.1.26
- CVE-2017-10379: MariaDB 10.1.26
- CVE-2017-10378: MariaDB 10.1.29
- CVE-2017-10286: MariaDB 10.1.26
- CVE-2017-10268: MariaDB 10.1.29
- CVE-2016-9843: MariaDB 10.1.37
- CVE-2016-8283: MariaDB 10.1.18
- CVE-2016-7440: MariaDB 10.1.19
- CVE-2016-6664: MariaDB 10.1.21
- CVE-2016-6663: MariaDB 10.1.18
- CVE-2016-6662: MariaDB 10.1.17
- CVE-2016-5629: MariaDB 10.1.18
- CVE-2016-5626: MariaDB 10.1.18
- CVE-2016-5624: MariaDB 10.1.18
- CVE-2016-5616: MariaDB 10.1.18
- CVE-2016-5584: MariaDB 10.1.19
- CVE-2016-5483: MariaDB 10.1.19
- CVE-2016-5444: MariaDB 10.1.14
- CVE-2016-5440: MariaDB 10.1.15
- CVE-2016-3615: MariaDB 10.1.15
- CVE-2016-3521: MariaDB 10.1.15
- CVE-2016-3492: MariaDB 10.1.18
- CVE-2016-3477: MariaDB 10.1.15
- CVE-2016-3471: MariaDB 10.1.9
- CVE-2016-3459: MariaDB 10.1.14
- CVE-2016-3452: MariaDB 10.1.14
- CVE-2016-2047: MariaDB 10.1.10
- CVE-2016-0668: MariaDB 10.1.12
- CVE-2016-0666: MariaDB 10.1.14
- CVE-2016-0655: MariaDB 10.1.14
- CVE-2016-0651: MariaDB 10.1.10
- CVE-2016-0650: MariaDB 10.1.12
- CVE-2016-0649: MariaDB 10.1.12
- CVE-2016-0648: MariaDB 10.1.14
- CVE-2016-0647: MariaDB 10.1.14
- CVE-2016-0646: MariaDB 10.1.12
- CVE-2016-0644: MariaDB 10.1.12
- CVE-2016-0643: MariaDB 10.1.14
- CVE-2016-0642: MariaDB 10.1.10
- CVE-2016-0641: MariaDB 10.1.12
- CVE-2016-0640: MariaDB 10.1.12
- CVE-2016-0616: MariaDB 10.1.10
- CVE-2016-0610: MariaDB 10.1.9
- CVE-2016-0609: MariaDB 10.1.10
- CVE-2016-0608: MariaDB 10.1.10
- CVE-2016-0606: MariaDB 10.1.10
- CVE-2016-0600: MariaDB 10.1.10
- CVE-2016-0598: MariaDB 10.1.10
- CVE-2016-0597: MariaDB 10.1.10
- CVE-2016-0596: MariaDB 10.1.10
- CVE-2016-0546: MariaDB 10.1.10
- CVE-2016-0505: MariaDB 10.1.10
- CVE-2015-7744: MariaDB 10.1.9
- CVE-2015-4913: MariaDB 10.1.8
- CVE-2015-4895: MariaDB 10.1.8
- CVE-2015-4879: MariaDB 10.1.8

- CVE-2015-4870: MariaDB 10.1.8
- CVE-2015-4866: MariaDB 10.1.8
- CVE-2015-4864: MariaDB 10.1.8
- CVE-2015-4861: MariaDB 10.1.8
- CVE-2015-4858: MariaDB 10.1.8
- CVE-2015-4836: MariaDB 10.1.8
- CVE-2015-4830: MariaDB 10.1.8
- CVE-2015-4826: MariaDB 10.1.8
- CVE-2015-4819: MariaDB 10.1.8
- CVE-2015-4816: MariaDB 10.1.8
- CVE-2015-4815: MariaDB 10.1.8
- CVE-2015-4807: MariaDB 10.1.8
- CVE-2015-4802: MariaDB 10.1.8
- CVE-2015-4792: MariaDB 10.1.8

Comparison with MySQL

- System Variable Differences Between MariaDB 10.1 and MySQL 5.6
- System Variable Differences Between MariaDB 10.1 and MySQL 5.7

List of all MariaDB 10.1 releases

Date	Release	Status	Release Notes	Changelog
3 Nov 2020	**MariaDB 10.1.48**	Stable (GA)	Release Notes	Changelog
7 Oct 2020	**MariaDB 10.1.47**	Stable (GA)	Release Notes	Changelog
10 Aug 2020	**MariaDB 10.1.46**	Stable (GA)	Release Notes	Changelog
12 May 2020	**MariaDB 10.1.45**	Stable (GA)	Release Notes	Changelog
28 Jan 2020	**MariaDB 10.1.44**	Stable (GA)	Release Notes	Changelog
8 Nov 2019	**MariaDB 10.1.43**	Stable (GA)	Release Notes	Changelog
5 Nov 2019	**MariaDB 10.1.42**	Stable (GA)	Release Notes	Changelog
31 Jul 2019	**MariaDB 10.1.41**	Stable (GA)	Release Notes	Changelog
8 May 2019	**MariaDB 10.1.40**	Stable (GA)	Release Notes	Changelog
2 May 2019	**MariaDB 10.1.39**	Stable (GA)	Release Notes	Changelog
6 Feb 2019	**MariaDB 10.1.38**	Stable (GA)	Release Notes	Changelog
2 Nov 2018	**MariaDB 10.1.37**	Stable (GA)	Release Notes	Changelog
8 Sep 2018	**MariaDB 10.1.36**	Stable (GA)	Release Notes	Changelog
7 Aug 2018	**MariaDB 10.1.35**	Stable (GA)	Release Notes	Changelog
18 Jun 2018	**MariaDB 10.1.34**	Stable (GA)	Release Notes	Changelog
9 May 2018	**MariaDB 10.1.33**	Stable (GA)	Release Notes	Changelog
27 Mar 2018	**MariaDB 10.1.32**	Stable (GA)	Release Notes	Changelog
6 Feb 2018	**MariaDB 10.1.31**	Stable (GA)	Release Notes	Changelog
22 Dec 2017	**MariaDB 10.1.30**	Stable (GA)	Release Notes	Changelog
14 Nov 2017	**MariaDB 10.1.29**	Stable (GA)	Release Notes	Changelog
28 Sep 2017	**MariaDB 10.1.28**	Stable (GA)	Release Notes	Changelog
25 Sep 2017	**MariaDB 10.1.27**	Stable (GA)	Release Notes	Changelog
10 Aug 2017	**MariaDB 10.1.26**	Stable (GA)	Release Notes	Changelog
4 Jul 2017	**MariaDB 10.1.25**	Stable (GA)	Release Notes	Changelog
31 May 2017	**MariaDB 10.1.24**	Stable (GA)	Release Notes	Changelog

3 May 2017	**MariaDB 10.1.23**	Stable (GA)	Release Notes	Changelog
14 Mar 2017	**MariaDB 10.1.22**	Stable (GA)	Release Notes	Changelog
18 Jan 2017	**MariaDB 10.1.21**	Stable (GA)	Release Notes	Changelog
15 Dec 2016	**MariaDB 10.1.20**	Stable (GA)	Release Notes	Changelog
7 Nov 2016	**MariaDB 10.1.19**	Stable (GA)	Release Notes	Changelog
30 Sep 2016	**MariaDB 10.1.18**	Stable (GA)	Release Notes	Changelog
30 Aug 2016	**MariaDB 10.1.17**	Stable (GA)	Release Notes	Changelog
18 Jul 2016	**MariaDB 10.1.16**	Stable (GA)	Release Notes	Changelog
1 Jul 2016	**MariaDB 10.1.15**	Stable (GA)	Release Notes	Changelog
10 May 2016	**MariaDB 10.1.14**	Stable (GA)	Release Notes	Changelog
25 Mar 2016	**MariaDB 10.1.13**	Stable (GA)	Release Notes	Changelog
25 Feb 2016	**MariaDB 10.1.12**	Stable (GA)	Release Notes	Changelog
29 Jan 2016	**MariaDB 10.1.11**	Stable (GA)	Release Notes	Changelog
24 Dec 2015	**MariaDB 10.1.10**	Stable (GA)	Release Notes	Changelog
23 Nov 2015	**MariaDB 10.1.9**	Stable (GA)	Release Notes	Changelog
17 Oct 2015	**MariaDB 10.1.8**	Stable (GA)	Release Notes	Changelog
9 Sep 2015	**MariaDB 10.1.7**	Release Candidate (RC)	Release Notes	Changelog
27 Jul 2015	**MariaDB 10.1.6**	Beta	Release Notes	Changelog
4 Jun 2015	**MariaDB 10.1.5**	Beta	Release Notes	Changelog
13 Apr 2015	**MariaDB 10.1.4**	Beta	Release Notes	Changelog
2 Mar 2015	**MariaDB 10.1.3**	Beta	Release Notes	Changelog
7 Dec 2014	**MariaDB 10.1.2**	Alpha	Release Notes	Changelog
17 Oct 2014	**MariaDB 10.1.1**	Alpha	Release Notes	Changelog
30 Jun 2014	**MariaDB 10.1.0**	Alpha	Release Notes	Changelog

7.0.11 MariaDB Server 10.0

Changes & Improvements in MariaDB 10.0

Current Version: 10.0.38 | Status: Stable (GA) | Release Date: 31 Jan 2019

Release Notes - MariaDB 10.0 Series

MariaDB 10.0 Series Release Notes

Changelogs - MariaDB 10.0 Series

MariaDB 10.0 changelogs.

7.0.11.1 Changes & Improvements in MariaDB 10.0

MariaDB 10.0 is no longer maintained. Please use a more recent release.

The most recent release in the MariaDB 10.0 series is:

MariaDB 10.0.38 Download Now

MariaDB 10.0 is a previous stable series of MariaDB. It is built on the MariaDB 5.5 series with backported features from MySQL 5.6 and entirely new features not found anywhere else. The first stable release was in March 2014, and the final release was in January 2019.

For details on upgrading from MariaDB 5.5, see Upgrading from MariaDB 5.5 to MariaDB 10.0.

Blog posts with details of the reasoning behind calling this version MariaDB 10:

- http://blog.mariadb.org/mariadb-10-0-and-mysql-5-6/
- http://blog.mariadb.org/what-comes-in-between-mariadb-now-and-mysql-5-6/
- http://blog.mariadb.org/explanation-on-mariadb-10-0/

Contents

Implemented Features

Features that are in a release.

New Features

- For a list of all new variables, see System Variables Added in MariaDB 10.0

- Parallel Replication
- Global Transaction ID (MDEV-26)
- Multi source replication (MDEV-253) — *Original code from Taobao, developed by Peng Lixun*.
- Slave started with `--binlog-format=STATEMENT` can replicate from master with any type of `--binlog-format` Starting from MariaDB 10.0.22.

- Cassandra storage engine (MDEV-4695)
- CONNECT storage engine (MDEV-4146)
- Better table discovery. Sequence storage engine. Assisted discovery in FederatedX (MDEV-3808)
- Spider storage engine (MDEV-4438)
- TokuDB storage engine (MDEV-4507)
- Mroonga full-text search storage engine

- QUERY_RESPONSE_TIME plugin

- Engine independent table statistics (MDEV-3806)
- Subquery optimizations: EXISTS-to-IN optimization, (MDEV-38 - NOT EXISTS to IN), MDEV-537, MDEV-3862
- Histogram-based statistics for non-indexed columns (MDEV-4145)

- SHOW EXPLAIN command (MDEV-165)
- EXPLAIN in the slow query log (MDEV-407)
- Per thread memory usage (MDEV-4011). — *Original code from Taobao, developed by Peng Lixun*.
  - information_schema.processlist has two new columns: `MEMORY_USAGE` and `EXAMINED_ROWS`.
  - SHOW STATUS has a new variable: Memory_used.
- SHOW PLUGINS SONAME 'XXX' (MDEV-3807)
- SHUTDOWN statement (MDEV-4660)
- Killing a query by query id, not thread id (MDEV-4911)

- Faster UNIQUE key generation with ALTER TABLE (MDEV-539)
- Implement async commit checkpoint in InnoDB and XtraDB (MDEV-532)
- Support for atomic writes on FusionIO DirectFS (MDEV-4338)

- DELETE ... RETURNING (MDEV-3814)
- IF (NOT) EXISTS clauses for ALTER TABLE (MDEV-318)
- CREATE OR REPLACE TABLE (MDEV-5491)
  - slave-ddl-exec-mode variable to specify how `CREATE TABLE` and `DROP TABLE` is replicated.
- Dynamic columns now support names (MDEV-377, summary of changes)
- multiple use locks (GET_LOCK) in one connection (MDEV-3917)
- Better error messages (all error numbers now include descriptive text explaining what the number means)
- table attributes with sysvar as a default value (MDEV-4022)

- regular expression enhancements
  - new regular expression library with modern features (PCRE)
  - new functions REGEXP_REPLACE, REGEXP_INSTR, REGEXP_SUBSTR.
- Roles (MDEV-4397)
- metadata_lock_info information schema. Shows you which meta data locks are active.
- Adjustable hash size for MyISAM and Aria. This can greatly improve shutdown time (from hours to minutes) if you are using a lot of MyISAM/Aria tables with delayed keys.
- FLUSH TABLES ... FOR EXPORT
- The Extended Keys optimization is enabled by default
- MariaDB audit plugin
- filesort-with-small-limit-optimization is now visible through the slow query log and a new status variable, sort_priority_queue_sorts
- Error log flood protection

New Features Re-implemented From a Similar MySQL Feature

- CURRENT_TIMESTAMP as DEFAULT for DATETIME columns (MDEV-452)
- EXPLAIN for INSERT/UPDATE/DELETE (MDEV-3798, MWL#51)

New Features Backported from MySQL 5.6

- New InnoDB *— from MySQL 5.6.14 in MariaDB 10.0.8 onwards*
- New Performance schema *— from MySQL 5.6.10 in MariaDB 10.0.4 onwards*
- New Information Schema tables, updates and defaults.
- Optimized read only transaction (for InnoDB). This includes support for TRANSACTION READ ONLY.
- Filesort optimization for queries using the `ORDER BY ... LIMIT` optimization - A useful optimization for showing only a few rows of a bigger result set. (MDEV-4026)
- backport --plugin-load-add (MDEV-3860)
- Online ALTER TABLE (MDEV-3933) (ALGORITHM=INPLACE, etc)
- InnoDB persistent statistics.
- privileges on temporary tables
- character set related extenstions
- GET DIAGNOSTICS
- EXCHANGE PARTITION
- Partition selection
- Temporal literals (such as `TIME'12:34:56'`)
- WEIGHT_STRING() function
- Collation customization improvements (see Supported Characters Sets and Collations)
- TO_BASE64() and FROM_BASE64() functions

Incompatible Changes

- New reserved word: RETURNING. This can no longer be used as an identifier without being quoted.

Security Vulnerabilities Fixed in MariaDB 10.0

> For a complete list of security vulnerabilities (CVEs) fixed across all versions of MariaDB, see the Security Vulnerabilities Fixed in MariaDB page.

- CVE-2021-2032: MariaDB 10.0.11
- CVE-2021-2011: MariaDB 10.0.35
- CVE-2020-14550: MariaDB 10.0.35
- CVE-2019-2537: MariaDB 10.0.38
- CVE-2019-2529: MariaDB 10.0.38
- CVE-2019-2503: MariaDB 10.0.37
- CVE-2019-2481: MariaDB 10.0.11
- CVE-2019-2455: MariaDB 10.0.35
- CVE-2018-3282: MariaDB 10.0.37
- CVE-2018-3251: MariaDB 10.0.37
- CVE-2018-3174: MariaDB 10.0.37
- CVE-2018-3156: MariaDB 10.0.37
- CVE-2018-3143: MariaDB 10.0.37
- CVE-2018-3133: MariaDB 10.0.34

- CVE-2018-3081: MariaDB 10.0.35
- CVE-2018-3066: MariaDB 10.0.36
- CVE-2018-3064: MariaDB 10.0.36
- CVE-2018-3063: MariaDB 10.0.36
- CVE-2018-3058: MariaDB 10.0.36
- CVE-2018-2819: MariaDB 10.0.35
- CVE-2018-2817: MariaDB 10.0.35
- CVE-2018-2813: MariaDB 10.0.35
- CVE-2018-2787: MariaDB 10.0.35
- CVE-2018-2784: MariaDB 10.0.35
- CVE-2018-2782: MariaDB 10.0.35
- CVE-2018-2781: MariaDB 10.0.35
- CVE-2018-2771: MariaDB 10.0.35
- CVE-2018-2767: MariaDB 10.0.35
- CVE-2018-2766: MariaDB 10.0.35
- CVE-2018-2761: MariaDB 10.0.35
- CVE-2018-2755: MariaDB 10.0.35
- CVE-2018-2668: MariaDB 10.0.34
- CVE-2018-2665: MariaDB 10.0.34
- CVE-2018-2640: MariaDB 10.0.34
- CVE-2018-2622: MariaDB 10.0.34
- CVE-2018-2612: MariaDB 10.0.34
- CVE-2018-2562: MariaDB 10.0.34
- CVE-2017-3653: MariaDB 10.0.32
- CVE-2017-3651: MariaDB 10.0.28
- CVE-2017-3641: MariaDB 10.0.32
- CVE-2017-3636: MariaDB 10.0.32
- CVE-2017-3600: MariaDB 10.0.28
- CVE-2017-3464: MariaDB 10.0.31
- CVE-2017-3456: MariaDB 10.0.31
- CVE-2017-3453: MariaDB 10.0.31
- CVE-2017-3318: MariaDB 10.0.29
- CVE-2017-3317: MariaDB 10.0.29
- CVE-2017-3313: MariaDB 10.0.30
- CVE-2017-3312: MariaDB 10.0.29
- CVE-2017-3309: MariaDB 10.0.31
- CVE-2017-3308: MariaDB 10.0.31
- CVE-2017-3302: MariaDB 10.0.30
- CVE-2017-3291: MariaDB 10.0.29
- CVE-2017-3265: MariaDB 10.0.29
- CVE-2017-3258: MariaDB 10.0.29
- CVE-2017-3257: MariaDB 10.0.29
- CVE-2017-3244: MariaDB 10.0.29
- CVE-2017-3243: MariaDB 10.0.29
- CVE-2017-3238: MariaDB 10.0.29
- CVE-2017-10384: MariaDB 10.0.32
- CVE-2017-10379: MariaDB 10.0.32
- CVE-2017-10378: MariaDB 10.0.33
- CVE-2017-10286: MariaDB 10.0.32
- CVE-2017-10268: MariaDB 10.0.33
- CVE-2016-9843: MariaDB 10.0.37
- CVE-2016-8283: MariaDB 10.0.28
- CVE-2016-7440: MariaDB 10.0.28
- CVE-2016-6664: MariaDB 10.0.29
- CVE-2016-6663: MariaDB 10.0.28
- CVE-2016-6662: MariaDB 10.0.27
- CVE-2016-5630: MariaDB 10.0.27
- CVE-2016-5629: MariaDB 10.0.28
- CVE-2016-5626: MariaDB 10.0.28
- CVE-2016-5624: MariaDB 10.0.28
- CVE-2016-5616: MariaDB 10.0.28
- CVE-2016-5612: MariaDB 10.0.27
- CVE-2016-5584: MariaDB 10.0.28
- CVE-2016-5483: MariaDB 10.0.28
- CVE-2016-5444: MariaDB 10.0.25
- CVE-2016-5440: MariaDB 10.0.26
- CVE-2016-3615: MariaDB 10.0.26

- CVE-2016-3521: MariaDB 10.0.26
- CVE-2016-3492: MariaDB 10.0.28
- CVE-2016-3477: MariaDB 10.0.26
- CVE-2016-3471: MariaDB 10.0.22
- CVE-2016-3459: MariaDB 10.0.25
- CVE-2016-3452: MariaDB 10.0.25
- CVE-2016-2047: MariaDB 10.0.23
- CVE-2016-0668: MariaDB 10.0.24
- CVE-2016-0666: MariaDB 10.0.25
- CVE-2016-0655: MariaDB 10.0.25
- CVE-2016-0651: MariaDB 10.0.23
- CVE-2016-0650: MariaDB 10.0.24
- CVE-2016-0649: MariaDB 10.0.24
- CVE-2016-0648: MariaDB 10.0.25
- CVE-2016-0647: MariaDB 10.0.25
- CVE-2016-0646: MariaDB 10.0.24
- CVE-2016-0644: MariaDB 10.0.24
- CVE-2016-0643: MariaDB 10.0.25
- CVE-2016-0642: MariaDB 10.0.23
- CVE-2016-0641: MariaDB 10.0.24
- CVE-2016-0640: MariaDB 10.0.24
- CVE-2016-0616: MariaDB 10.0.23
- CVE-2016-0610: MariaDB 10.0.22
- CVE-2016-0609: MariaDB 10.0.23
- CVE-2016-0608: MariaDB 10.0.23
- CVE-2016-0606: MariaDB 10.0.23
- CVE-2016-0600: MariaDB 10.0.23
- CVE-2016-0598: MariaDB 10.0.23
- CVE-2016-0597: MariaDB 10.0.23
- CVE-2016-0596: MariaDB 10.0.23
- CVE-2016-0546: MariaDB 10.0.23
- CVE-2016-0505: MariaDB 10.0.23
- CVE-2016-0502: MariaDB 10.0.4
- CVE-2015-7744: MariaDB 10.0.22
- CVE-2015-4913: MariaDB 10.0.22
- CVE-2015-4895: MariaDB 10.0.21
- CVE-2015-4879: MariaDB 10.0.21
- CVE-2015-4870: MariaDB 10.0.22
- CVE-2015-4866: MariaDB 10.0.18
- CVE-2015-4864: MariaDB 10.0.20
- CVE-2015-4861: MariaDB 10.0.22
- CVE-2015-4858: MariaDB 10.0.22
- CVE-2015-4836: MariaDB 10.0.22
- CVE-2015-4830: MariaDB 10.0.22
- CVE-2015-4826: MariaDB 10.0.22
- CVE-2015-4819: MariaDB 10.0.21
- CVE-2015-4816: MariaDB 10.0.21
- CVE-2015-4815: MariaDB 10.0.22
- CVE-2015-4807: MariaDB 10.0.22
- CVE-2015-4802: MariaDB 10.0.22
- CVE-2015-4792: MariaDB 10.0.22
- CVE-2015-4757: MariaDB 10.0.18
- CVE-2015-4752: MariaDB 10.0.20
- CVE-2015-3152: MariaDB 10.0.20
- CVE-2015-2648: MariaDB 10.0.20
- CVE-2015-2643: MariaDB 10.0.20
- CVE-2015-2620: MariaDB 10.0.20
- CVE-2015-2582: MariaDB 10.0.20
- CVE-2015-2573: MariaDB 10.0.17
- CVE-2015-2571: MariaDB 10.0.18
- CVE-2015-2568: MariaDB 10.0.17
- CVE-2015-2326: MariaDB 10.0.18
- CVE-2015-2325: MariaDB 10.0.18
- CVE-2015-0505: MariaDB 10.0.18
- CVE-2015-0501: MariaDB 10.0.18
- CVE-2015-0499: MariaDB 10.0.18
- CVE-2015-0441: MariaDB 10.0.17

- CVE-2015-0433: MariaDB 10.0.17
- CVE-2015-0432: MariaDB 10.0.16
- CVE-2015-0411: MariaDB 10.0.16
- CVE-2015-0391: MariaDB 10.0.13
- CVE-2015-0382: MariaDB 10.0.16
- CVE-2015-0381: MariaDB 10.0.16
- CVE-2015-0374: MariaDB 10.0.16
- CVE-2014-8964: MariaDB 10.0.18
- CVE-2014-6568: MariaDB 10.0.16
- CVE-2014-6564: MariaDB 10.0.13
- CVE-2014-6559: MariaDB 10.0.15
- CVE-2014-6555: MariaDB 10.0.15
- CVE-2014-6551: MariaDB 10.0.13
- CVE-2014-6530: MariaDB 10.0.13
- CVE-2014-6520: MariaDB 10.0.13
- CVE-2014-6507: MariaDB 10.0.15
- CVE-2014-6505: MariaDB 10.0.13
- CVE-2014-6500: MariaDB 10.0.15
- CVE-2014-6496: MariaDB 10.0.15
- CVE-2014-6495: MariaDB 10.0.13
- CVE-2014-6494: MariaDB 10.0.15
- CVE-2014-6491: MariaDB 10.0.15
- CVE-2014-6489: MariaDB 10.0.13
- CVE-2014-6484: MariaDB 10.0.13
- CVE-2014-6478: MariaDB 10.0.13
- CVE-2014-6474: MariaDB 10.0.13
- CVE-2014-6469: MariaDB 10.0.15
- CVE-2014-6464: MariaDB 10.0.15
- CVE-2014-6463: MariaDB 10.0.13
- CVE-2014-4287: MariaDB 10.0.13
- CVE-2014-4274: MariaDB 10.0.13
- CVE-2014-4260: MariaDB 10.0.12
- CVE-2014-4258: MariaDB 10.0.12
- CVE-2014-4243: MariaDB 10.0.9
- CVE-2014-4207: MariaDB 10.0.12
- CVE-2014-3470: MariaDB 10.0.13
- CVE-2014-2494: MariaDB 10.0.12
- CVE-2014-2440: MariaDB 10.0.11
- CVE-2014-2438: MariaDB 10.0.9
- CVE-2014-2436: MariaDB 10.0.11
- CVE-2014-2432: MariaDB 10.0.9
- CVE-2014-2431: MariaDB 10.0.11
- CVE-2014-2430: MariaDB 10.0.11
- CVE-2014-2419: MariaDB 10.0.9
- CVE-2014-0437: MariaDB 10.0.8
- CVE-2014-0420: MariaDB 10.0.8
- CVE-2014-0412: MariaDB 10.0.8
- CVE-2014-0402: MariaDB 10.0.7
- CVE-2014-0401: MariaDB 10.0.8
- CVE-2014-0393: MariaDB 10.0.7
- CVE-2014-0386: MariaDB 10.0.7
- CVE-2014-0384: MariaDB 10.0.9
- CVE-2014-0224: MariaDB 10.0.13
- CVE-2014-0221: MariaDB 10.0.13
- CVE-2014-0198: MariaDB 10.0.13
- CVE-2014-0195: MariaDB 10.0.13
- CVE-2013-5908: MariaDB 10.0.8
- CVE-2013-5891: MariaDB 10.0.7
- CVE-2013-5807: MariaDB 10.0.5
- CVE-2013-3839: MariaDB 10.0.5
- CVE-2013-3812: MariaDB 10.0.4
- CVE-2013-3809: MariaDB 10.0.4
- CVE-2013-3808: MariaDB 10.0.3
- CVE-2013-3805: MariaDB 10.0.3
- CVE-2013-3804: MariaDB 10.0.4
- CVE-2013-3802: MariaDB 10.0.4
- CVE-2013-3801: MariaDB 10.0.3

- CVE-2013-3794: MariaDB 10.0.3
- CVE-2013-3793: MariaDB 10.0.4
- CVE-2013-3783: MariaDB 10.0.4
- CVE-2013-2392: MariaDB 10.0.3
- CVE-2013-2391: MariaDB 10.0.3
- CVE-2013-2389: MariaDB 10.0.3
- CVE-2013-2378: MariaDB 10.0.2
- CVE-2013-2376: MariaDB 10.0.3
- CVE-2013-2375: MariaDB 10.0.3
- CVE-2013-1861: MariaDB 10.0.4
- CVE-2013-1555: MariaDB 10.0.2
- CVE-2013-1552: MariaDB 10.0.2
- CVE-2013-1544: MariaDB 10.0.3
- CVE-2013-1532: MariaDB 10.0.3
- CVE-2013-1531: MariaDB 10.0.1
- CVE-2013-1526: MariaDB 10.0.2
- CVE-2013-1523: MariaDB 10.0.2
- CVE-2013-1521: MariaDB 10.0.2
- CVE-2013-1512: MariaDB 10.0.2
- CVE-2013-1511: MariaDB 10.0.3
- CVE-2013-1506: MariaDB 10.0.2
- CVE-2013-1502: MariaDB 10.0.3
- CVE-2013-0389: MariaDB 10.0.1
- CVE-2013-0386: MariaDB 10.0.1
- CVE-2013-0385: MariaDB 10.0.1
- CVE-2013-0384: MariaDB 10.0.1
- CVE-2013-0383: MariaDB 10.0.1
- CVE-2013-0371: MariaDB 10.0.1
- CVE-2013-0368: MariaDB 10.0.1
- CVE-2013-0367: MariaDB 10.0.1
- CVE-2012-5627: MariaDB 10.0.1
- CVE-2012-5615: MariaDB 10.0.13, MariaDB 10.0.1
- CVE-2012-5614: MariaDB 10.0.2
- CVE-2012-5612: MariaDB 10.0.1
- CVE-2012-5611: MariaDB 10.0.1
- CVE-2012-5096: MariaDB 10.0.1
- CVE-2012-4414: MariaDB 10.0.0 [2]
- CVE-2012-1705: MariaDB 10.0.1
- CVE-2012-1702: MariaDB 10.0.1
- CVE-2012-0578: MariaDB 10.0.1
- CVE-2012-0574: MariaDB 10.0.1
- CVE-2012-0572: MariaDB 10.0.1
- CVE-2010-5298: MariaDB 10.0.13

Comparison with MySQL

See System Variable Differences Between MariaDB 10.0 and MySQL 5.6

List of All MariaDB 10.0 Releases

Date	Release	Status	Release Notes	Changelog
31 Jan 2019	**MariaDB 10.0.38**	Stable (GA)	Release Notes	Changelog
1 Nov 2018	**MariaDB 10.0.37**	Stable (GA)	Release Notes	Changelog
1 Aug 2018	**MariaDB 10.0.36**	Stable (GA)	Release Notes	Changelog
3 May 2018	**MariaDB 10.0.35**	Stable (GA)	Release Notes	Changelog
30 Jan 2018	**MariaDB 10.0.34**	Stable (GA)	Release Notes	Changelog
30 Oct 2017	**MariaDB 10.0.33**	Stable (GA)	Release Notes	Changelog
7 Aug 2017	**MariaDB 10.0.32**	Stable (GA)	Release Notes	Changelog
23 May 2017	**MariaDB 10.0.31**	Stable (GA)	Release Notes	Changelog

8 Mar 2017	**MariaDB 10.0.30**	Stable (GA)	Release Notes	Changelog
13 Jan 2017	**MariaDB 10.0.29**	Stable (GA)	Release Notes	Changelog
28 Oct 2016	**MariaDB 10.0.28**	Stable (GA)	Release Notes	Changelog
25 Aug 2016	**MariaDB 10.0.27**	Stable (GA)	Release Notes	Changelog
24 Jun 2016	**MariaDB 10.0.26**	Stable (GA)	Release Notes	Changelog
30 Apr 2016	**MariaDB 10.0.25**	Stable (GA)	Release Notes	Changelog
19 Feb 2016	**MariaDB 10.0.24**	Stable (GA)	Release Notes	Changelog
18 Dec 2015	**MariaDB 10.0.23**	Stable (GA)	Release Notes	Changelog
29 Oct 2015	**MariaDB 10.0.22**	Stable (GA)	Release Notes	Changelog
6 Aug 2015	**MariaDB 10.0.21**	Stable (GA)	Release Notes	Changelog
18 Jun 2015	**MariaDB 10.0.20**	Stable (GA)	Release Notes	Changelog
9 May 2015	**MariaDB 10.0.19**	Stable (GA)	Release Notes	Changelog
7 May 2015	**MariaDB 10.0.18**	Stable (GA)	Release Notes	Changelog
27 Feb 2015	**MariaDB 10.0.17**	Stable (GA)	Release Notes	Changelog
27 Jan 2015	**MariaDB 10.0.16**	Stable (GA)	Release Notes	Changelog
25 Nov 2014	**MariaDB 10.0.15**	Stable (GA)	Release Notes	Changelog
26 Sep 2014	**MariaDB 10.0.14**	Stable (GA)	Release Notes	Changelog
11 Aug 2014	**MariaDB 10.0.13**	Stable (GA)	Release Notes	Changelog
16 Jun 2014	**MariaDB 10.0.12**	Stable (GA)	Release Notes	Changelog
12 May 2014	**MariaDB 10.0.11**	Stable (GA)	Release Notes	Changelog
31 Mar 2014	**MariaDB 10.0.10**	Stable (GA)	Release Notes	Changelog
10 Mar 2014	**MariaDB 10.0.9**	Release Candidate	Release Notes	Changelog
10 Feb 2014	**MariaDB 10.0.8**	Release Candidate	Release Notes	Changelog
27 Dec 2013	**MariaDB 10.0.7**	Beta	Release Notes	Changelog
18 Nov 2013	**MariaDB 10.0.6**	Beta	Release Notes	Changelog
7 Nov 2013	**MariaDB 10.0.5**	Beta	Release Notes	Changelog
16 Aug 2013	**MariaDB 10.0.4**	Alpha	Release Notes	Changelog
11 Jun 2013	**MariaDB 10.0.3**	Alpha	Release Notes	Changelog
24 Apr 2013	**MariaDB 10.0.2**	Alpha	Release Notes	Changelog
6 Feb 2013	**MariaDB 10.0.1**	Alpha	Release Notes	Changelog
12 Nov 2012	**MariaDB 10.0.0**	Alpha	Release Notes	Changelog

7.0.12 MariaDB Server 5.5

Changes & Improvements in MariaDB 5.5

Current Version: 5.5.68 | Status: Stable (GA) | Release Date: 12 May 2020

Release Notes - MariaDB 5.5 Series

MariaDB 5.5 Series Release Notes

Changelogs - MariaDB 5.5 Series

MariaDB 5.5 changelogs.

There are 2 related questions.

7.0.12.1 Changes & Improvements in MariaDB 5.5

MariaDB 5.5 is no longer supported. Please use a more recent release.

The most recent release in the MariaDB 5.5 series is:
MariaDB 5.5.68 Download Now

Contents

MariaDB 5.5 is MariaDB 5.3 + MySQL 5.5, with added features. The first stable release was in April 2012, and the final release in May 2020.

For upgrading to MariaDB 10.0, the more recent stable release, see Upgrading from MariaDB 5.5 to MariaDB 10.0.

Feature Comparison Matrix

We have created an Optimizer Feature Comparison Matrix showing the new optimizer features in MariaDB 5.5 and 5.3 compared to MySQL 5.5 and 5.6.

See also a detailed breakdown of System variable differences between MariaDB 5.5 and MySQL 5.5.

New Features

- Significantly more efficient thread pool, comparable in functionality to the closed source feature in MySQL Enterprise.
- Non-blocking client API Library (MWL#192)
- @@skip_replication option (MWL#234)
- SphinxSE updated to version 2.0.4.
- Extended Keys support for XtraDB and InnoDB
- New INSTALL SONAME statement (MWL#77)
- New LIMIT ROWS EXAMINED optimization (MDEV-28)
- `mysql_real_connect()` Changes
  - In MySQL, and in MariaDB versions before 5.5.21, `mysql_real_connect()` removes from the MYSQL object any options set with `mysql_option()` when it fails. Beginning with MariaDB 5.5.21, options are preserved by a failing `mysql_real_connect();` use `mysql_close()`, as normal, to clear them.
  - This only has effect if the MYSQL object is reused after a `mysql_real_connect()` failure (which is unusual). No real-life incompatibilities are expected from this change (it is unlikely that an application would rely on options being automatically removed between connection attempts).
- The variables `replicate_do_*`, `replicate_ignore_*`, and `replicate_wild_*` have been made dynamic, so they can be changed without requiring a server restart. See Dynamic Replication Variables for more information.
- New status variables for checking if features are used. These are very useful in user feedback to tell developers how much a feature is used:
  - Feature_dynamic_columns
  - Feature_fulltext
  - Feature_gis
  - Feature_locale
  - Feature_subquery
  - Feature_timezone
  - Feature_trigger
  - Feature_xml
- New status variables to see what's going on:
  - Opened_views
  - Executed_triggers

    - Executed_events
- New plugin to log SQL level errors. SQL_ERROR_LOG
- New variable OLD_MODE to set compatibility behavior with older MySQL or MariaDB versions.

Information Schema

There are a number of new INFORMATION SCHEMA tables:

- INNODB_SYS_COLUMNS
- INNODB_SYS_FIELDS
- INNODB_SYS_FOREIGN
- INNODB_SYS_FOREIGN_COLS
- PARAMETERS
- TABLESPACES

Minor Extensions

- Updates to performance schema tables are not stored in the binary log and thus not replicated to slaves. This is to ensure that monitoring of the master will not cause a slower performance on all slaves. This also fixes a crash on the slaves.

New features are added to MariaDB 10.0.

Deprecated / Disabled Features

- PBXT is no longer in the binary builds/distributions. It's however still in the source distributions and in the source tree. The reason is that PBXT is no longer actively maintained, has a few bugs that are not fixed and is not in widespread use.

Switching Between InnoDB and XtraDB

MariaDB 5.5 comes with both XtraDB (compiled in) and InnoDB (as a plugin). By default MariaDB 5.5 uses XtraDB. If you want to switch to use InnoDB you can do:

```
mysqld --ignore-builtin-innodb --plugin-load=innodb=ha_innodb.so \
--plugin_dir=/usr/local/mysql/lib/mysql/plugin
```

(plugin_dir should point to where ha_innodb.so is installed)

The above options can of course also be added to your my.cnf file:

```
[mysqld]
ignore-builtin-innodb
plugin-load=innodb=ha_innodb.so
plugin_dir=/usr/local/mysql/lib/mysql/plugin
```

If you want you can also compile MariaDB with InnoDB as default.

Security Vulnerabilities Fixed in MariaDB 5.5

> For a complete list of security vulnerabilities (CVEs) fixed across all versions of MariaDB, see the Security Vulnerabilities Fixed in MariaDB page.

- CVE-2021-2144: MariaDB 5.5.66
- CVE-2021-2011: MariaDB 5.5.61
- CVE-2021-2007: MariaDB 5.5.65
- CVE-2020-2922: MariaDB 5.5.65
- CVE-2020-2812: MariaDB 5.5.68
- CVE-2020-2780: MariaDB 5.5.66
- CVE-2020-2752: MariaDB 5.5.68
- CVE-2020-2574: MariaDB 5.5.67
- CVE-2020-14550: MariaDB 5.5.61
- CVE-2019-2974: MariaDB 5.5.66

- CVE-2019-2805: MariaDB 5.5.65
- CVE-2019-2740: MariaDB 5.5.65
- CVE-2019-2739: MariaDB 5.5.65
- CVE-2019-2737: MariaDB 5.5.65
- CVE-2019-2627: MariaDB 5.5.64
- CVE-2019-2614: MariaDB 5.5.64
- CVE-2019-2529: MariaDB 5.5.63
- CVE-2019-2503: MariaDB 5.5.62
- CVE-2019-2481: MariaDB 5.5.37
- CVE-2019-2455: MariaDB 5.5.60
- CVE-2018-3282: MariaDB 5.5.62
- CVE-2018-3174: MariaDB 5.5.62
- CVE-2018-3133: MariaDB 5.5.59
- CVE-2018-3081: MariaDB 5.5.61
- CVE-2018-3066: MariaDB 5.5.61
- CVE-2018-3063: MariaDB 5.5.61
- CVE-2018-3058: MariaDB 5.5.61
- CVE-2018-2819: MariaDB 5.5.60
- CVE-2018-2817: MariaDB 5.5.60
- CVE-2018-2813: MariaDB 5.5.60
- CVE-2018-2781: MariaDB 5.5.60
- CVE-2018-2771: MariaDB 5.5.60
- CVE-2018-2767: MariaDB 5.5.60
- CVE-2018-2761: MariaDB 5.5.60
- CVE-2018-2755: MariaDB 5.5.60
- CVE-2018-2668: MariaDB 5.5.59
- CVE-2018-2665: MariaDB 5.5.59
- CVE-2018-2640: MariaDB 5.5.59
- CVE-2018-2622: MariaDB 5.5.59
- CVE-2018-2562: MariaDB 5.5.59
- CVE-2017-3653: MariaDB 5.5.57
- CVE-2017-3651: MariaDB 5.5.53
- CVE-2017-3641: MariaDB 5.5.57
- CVE-2017-3636: MariaDB 5.5.57
- CVE-2017-3600: MariaDB 5.5.53
- CVE-2017-3464: MariaDB 5.5.55
- CVE-2017-3456: MariaDB 5.5.55
- CVE-2017-3453: MariaDB 5.5.55
- CVE-2017-3318: MariaDB 5.5.54
- CVE-2017-3317: MariaDB 5.5.54
- CVE-2017-3313: MariaDB 5.5.55
- CVE-2017-3312: MariaDB 5.5.54
- CVE-2017-3309: MariaDB 5.5.55
- CVE-2017-3308: MariaDB 5.5.55
- CVE-2017-3302: MariaDB 5.5.55
- CVE-2017-3291: MariaDB 5.5.54
- CVE-2017-3265: MariaDB 5.5.54
- CVE-2017-3258: MariaDB 5.5.54
- CVE-2017-3244: MariaDB 5.5.54
- CVE-2017-3243: MariaDB 5.5.54
- CVE-2017-3238: MariaDB 5.5.54
- CVE-2017-10384: MariaDB 5.5.57
- CVE-2017-10379: MariaDB 5.5.57
- CVE-2017-10378: MariaDB 5.5.58
- CVE-2017-10268: MariaDB 5.5.58
- CVE-2016-9843: MariaDB 5.5.62
- CVE-2016-8283: MariaDB 5.5.52
- CVE-2016-7440: MariaDB 5.5.53
- CVE-2016-6664: MariaDB 5.5.54
- CVE-2016-6663: MariaDB 5.5.52
- CVE-2016-6662: MariaDB 5.5.51
- CVE-2016-5629: MariaDB 5.5.52
- CVE-2016-5626: MariaDB 5.5.52
- CVE-2016-5624: MariaDB 5.5.52
- CVE-2016-5616: MariaDB 5.5.52
- CVE-2016-5612: MariaDB 5.5.51
- CVE-2016-5584: MariaDB 5.5.53

- CVE-2016-5483: MariaDB 5.5.53
- CVE-2016-5444: MariaDB 5.5.49
- CVE-2016-5440: MariaDB 5.5.50
- CVE-2016-3615: MariaDB 5.5.50
- CVE-2016-3521: MariaDB 5.5.50
- CVE-2016-3492: MariaDB 5.5.52
- CVE-2016-3477: MariaDB 5.5.50
- CVE-2016-3471: MariaDB 5.5.46
- CVE-2016-3452: MariaDB 5.5.49
- CVE-2016-2047: MariaDB 5.5.47
- CVE-2016-0666: MariaDB 5.5.49
- CVE-2016-0651: MariaDB 5.5.47
- CVE-2016-0650: MariaDB 5.5.48
- CVE-2016-0649: MariaDB 5.5.48
- CVE-2016-0648: MariaDB 5.5.49
- CVE-2016-0647: MariaDB 5.5.49
- CVE-2016-0646: MariaDB 5.5.48
- CVE-2016-0644: MariaDB 5.5.48
- CVE-2016-0643: MariaDB 5.5.49
- CVE-2016-0642: MariaDB 5.5.47
- CVE-2016-0641: MariaDB 5.5.48
- CVE-2016-0640: MariaDB 5.5.48
- CVE-2016-0616: MariaDB 5.5.47
- CVE-2016-0609: MariaDB 5.5.47
- CVE-2016-0608: MariaDB 5.5.47
- CVE-2016-0606: MariaDB 5.5.47
- CVE-2016-0600: MariaDB 5.5.47
- CVE-2016-0598: MariaDB 5.5.47
- CVE-2016-0597: MariaDB 5.5.47
- CVE-2016-0596: MariaDB 5.5.47
- CVE-2016-0546: MariaDB 5.5.47
- CVE-2016-0505: MariaDB 5.5.47
- CVE-2016-0502: MariaDB 5.5.32
- CVE-2015-7744: MariaDB 5.5.46
- CVE-2015-4913: MariaDB 5.5.46
- CVE-2015-4879: MariaDB 5.5.45
- CVE-2015-4870: MariaDB 5.5.46
- CVE-2015-4864: MariaDB 5.5.44
- CVE-2015-4861: MariaDB 5.5.46
- CVE-2015-4858: MariaDB 5.5.46
- CVE-2015-4836: MariaDB 5.5.46
- CVE-2015-4830: MariaDB 5.5.46
- CVE-2015-4826: MariaDB 5.5.46
- CVE-2015-4819: MariaDB 5.5.45
- CVE-2015-4816: MariaDB 5.5.45
- CVE-2015-4815: MariaDB 5.5.46
- CVE-2015-4807: MariaDB 5.5.46
- CVE-2015-4802: MariaDB 5.5.46
- CVE-2015-4792: MariaDB 5.5.46
- CVE-2015-4757: MariaDB 5.5.43
- CVE-2015-4752: MariaDB 5.5.44
- CVE-2015-3152: MariaDB 5.5.44
- CVE-2015-2648: MariaDB 5.5.44
- CVE-2015-2643: MariaDB 5.5.44
- CVE-2015-2620: MariaDB 5.5.44
- CVE-2015-2582: MariaDB 5.5.44
- CVE-2015-2573: MariaDB 5.5.42
- CVE-2015-2571: MariaDB 5.5.43
- CVE-2015-2568: MariaDB 5.5.42
- CVE-2015-0505: MariaDB 5.5.43
- CVE-2015-0501: MariaDB 5.5.43
- CVE-2015-0499: MariaDB 5.5.43
- CVE-2015-0441: MariaDB 5.5.42
- CVE-2015-0433: MariaDB 5.5.42
- CVE-2015-0432: MariaDB 5.5.41
- CVE-2015-0411: MariaDB 5.5.41
- CVE-2015-0391: MariaDB 5.5.39

- CVE-2015-0382: MariaDB 5.5.41
- CVE-2015-0381: MariaDB 5.5.41
- CVE-2015-0374: MariaDB 5.5.41
- CVE-2014-6568: MariaDB 5.5.41
- CVE-2014-6559: MariaDB 5.5.40
- CVE-2014-6555: MariaDB 5.5.40
- CVE-2014-6551: MariaDB 5.5.39
- CVE-2014-6530: MariaDB 5.5.39
- CVE-2014-6520: MariaDB 5.5.39
- CVE-2014-6507: MariaDB 5.5.40
- CVE-2014-6505: MariaDB 5.5.39
- CVE-2014-6500: MariaDB 5.5.40
- CVE-2014-6496: MariaDB 5.5.40
- CVE-2014-6495: MariaDB 5.5.39
- CVE-2014-6494: MariaDB 5.5.40
- CVE-2014-6491: MariaDB 5.5.40
- CVE-2014-6484: MariaDB 5.5.39
- CVE-2014-6478: MariaDB 5.5.39
- CVE-2014-6469: MariaDB 5.5.40
- CVE-2014-6464: MariaDB 5.5.40
- CVE-2014-6463: MariaDB 5.5.39
- CVE-2014-4287: MariaDB 5.5.39
- CVE-2014-4274: MariaDB 5.5.39
- CVE-2014-4260: MariaDB 5.5.38
- CVE-2014-4258: MariaDB 5.5.38
- CVE-2014-4243: MariaDB 5.5.36
- CVE-2014-4207: MariaDB 5.5.38
- CVE-2014-2494: MariaDB 5.5.38
- CVE-2014-2440: MariaDB 5.5.37
- CVE-2014-2438: MariaDB 5.5.36
- CVE-2014-2436: MariaDB 5.5.37
- CVE-2014-2432: MariaDB 5.5.36
- CVE-2014-2431: MariaDB 5.5.37
- CVE-2014-2430: MariaDB 5.5.37
- CVE-2014-2419: MariaDB 5.5.36
- CVE-2014-0437: MariaDB 5.5.35
- CVE-2014-0420: MariaDB 5.5.35
- CVE-2014-0412: MariaDB 5.5.35
- CVE-2014-0402: MariaDB 5.5.34
- CVE-2014-0401: MariaDB 5.5.35
- CVE-2014-0393: MariaDB 5.5.34
- CVE-2014-0386: MariaDB 5.5.34
- CVE-2014-0384: MariaDB 5.5.36
- CVE-2013-5908: MariaDB 5.5.35
- CVE-2013-5891: MariaDB 5.5.34
- CVE-2013-5807: MariaDB 5.5.33
- CVE-2013-3839: MariaDB 5.5.33
- CVE-2013-3812: MariaDB 5.5.32
- CVE-2013-3809: MariaDB 5.5.32
- CVE-2013-3808: MariaDB 5.5.31
- CVE-2013-3805: MariaDB 5.5.31
- CVE-2013-3804: MariaDB 5.5.32
- CVE-2013-3802: MariaDB 5.5.32
- CVE-2013-3801: MariaDB 5.5.31
- CVE-2013-3794: MariaDB 5.5.31
- CVE-2013-3793: MariaDB 5.5.32
- CVE-2013-3783: MariaDB 5.5.32
- CVE-2013-2392: MariaDB 5.5.31
- CVE-2013-2391: MariaDB 5.5.31
- CVE-2013-2389: MariaDB 5.5.31
- CVE-2013-2378: MariaDB 5.5.30
- CVE-2013-2376: MariaDB 5.5.31
- CVE-2013-2375: MariaDB 5.5.31
- CVE-2013-1861: MariaDB 5.5.32
- CVE-2013-1555: MariaDB 5.5.30
- CVE-2013-1552: MariaDB 5.5.30
- CVE-2013-1548: MariaDB 5.5.27

- CVE-2013-1544: MariaDB 5.5.31
- CVE-2013-1532: MariaDB 5.5.31
- CVE-2013-1531: MariaDB 5.5.29
- CVE-2013-1526: MariaDB 5.5.30
- CVE-2013-1523: MariaDB 5.5.30
- CVE-2013-1521: MariaDB 5.5.30
- CVE-2013-1512: MariaDB 5.5.30
- CVE-2013-1511: MariaDB 5.5.31
- CVE-2013-1506: MariaDB 5.5.30
- CVE-2013-1502: MariaDB 5.5.31
- CVE-2013-0389: MariaDB 5.5.29
- CVE-2013-0386: MariaDB 5.5.29
- CVE-2013-0385: MariaDB 5.5.29
- CVE-2013-0384: MariaDB 5.5.29
- CVE-2013-0383: MariaDB 5.5.29
- CVE-2013-0371: MariaDB 5.5.29
- CVE-2013-0368: MariaDB 5.5.29
- CVE-2013-0367: MariaDB 5.5.29
- CVE-2012-5627: MariaDB 5.5.29 [2]
- CVE-2012-5615: MariaDB 5.5.29 [2]
- CVE-2012-5614: MariaDB 5.5.30
- CVE-2012-5612: MariaDB 5.5.29 [2]
- CVE-2012-5611: MariaDB 5.5.29, MariaDB 5.5.28
- CVE-2012-5096: MariaDB 5.5.29
- CVE-2012-5060: MariaDB 5.5.28
- CVE-2012-4414: MariaDB 5.5.27 [2]
- CVE-2012-3197: MariaDB 5.5.27
- CVE-2012-3180: MariaDB 5.5.28
- CVE-2012-3177: MariaDB 5.5.28, MariaDB 5.5.27
- CVE-2012-3173: MariaDB 5.5.27
- CVE-2012-3167: MariaDB 5.5.27
- CVE-2012-3166: MariaDB 5.5.27
- CVE-2012-3163: MariaDB 5.5.27
- CVE-2012-3160: MariaDB 5.5.28
- CVE-2012-3158: MariaDB 5.5.27
- CVE-2012-3150: MariaDB 5.5.27
- CVE-2012-2750: MariaDB 5.5.23
- CVE-2012-1757: MariaDB 5.5.24
- CVE-2012-1756: MariaDB 5.5.24
- CVE-2012-1735: MariaDB 5.5.24
- CVE-2012-1734: MariaDB 5.5.24
- CVE-2012-1705: MariaDB 5.5.29
- CVE-2012-1703: MariaDB 5.5.22
- CVE-2012-1702: MariaDB 5.5.29
- CVE-2012-1697: MariaDB 5.5.22
- CVE-2012-1690: MariaDB 5.5.22
- CVE-2012-1689: MariaDB 5.5.23
- CVE-2012-1688: MariaDB 5.5.22
- CVE-2012-0578: MariaDB 5.5.29
- CVE-2012-0574: MariaDB 5.5.29
- CVE-2012-0572: MariaDB 5.5.29
- CVE-2012-0540: MariaDB 5.5.24
- CVE-2005-0004: MariaDB 5.5.66

The following CVEs are also fixed in MariaDB 5.5 but the fix is not tied to a specific version number:

- CVE-2012-0113
- CVE-2011-2262
- CVE-2012-0116
- CVE-2012-0118
- CVE-2012-0496
- CVE-2012-0115
- CVE-2012-0119
- CVE-2012-0120
- CVE-2012-0484
- CVE-2012-0485
- CVE-2012-0486
- CVE-2012-0487

- CVE-2012-0488
- CVE-2012-0489
- CVE-2012-0490
- CVE-2012-0491
- CVE-2012-0495
- CVE-2012-0112
- CVE-2012-0117
- CVE-2012-0114
- CVE-2012-0492
- CVE-2012-0493
- CVE-2012-0075
- CVE-2012-0494

List of All MariaDB 5.5 Releases

Date	Release	Status	Release Notes	Changelog
12 May 2020	**MariaDB 5.5.68**	Stable (GA)	Release Notes	Changelog
28 Jan 2020	**MariaDB 5.5.67**	Stable (GA)	Release Notes	Changelog
5 Nov 2019	**MariaDB 5.5.66**	Stable (GA)	Release Notes	Changelog
31 Jul 2019	**MariaDB 5.5.65**	Stable (GA)	Release Notes	Changelog
29 Apr 2019	**MariaDB 5.5.64**	Stable (GA)	Release Notes	Changelog
30 Jan 2019	**MariaDB 5.5.63**	Stable (GA)	Release Notes	Changelog
26 Oct 2018	**MariaDB 5.5.62**	Stable (GA)	Release Notes	Changelog
31 Jul 2018	**MariaDB 5.5.61**	Stable (GA)	Release Notes	Changelog
23 Apr 2018	**MariaDB 5.5.60**	Stable (GA)	Release Notes	Changelog
19 Jan 2018	**MariaDB 5.5.59**	Stable (GA)	Release Notes	Changelog
18 Oct 2017	**MariaDB 5.5.58**	Stable (GA)	Release Notes	Changelog
19 Jul 2017	**MariaDB 5.5.57**	Stable (GA)	Release Notes	Changelog
3 May 2017	**MariaDB 5.5.56**	Stable (GA)	Release Notes	Changelog
13 Apr 2017	**MariaDB 5.5.55**	Stable (GA)	Release Notes	Changelog
24 Dec 2016	**MariaDB 5.5.54**	Stable (GA)	Release Notes	Changelog
17 Oct 2016	**MariaDB 5.5.53**	Stable (GA)	Release Notes	Changelog
13 Sep 2016	**MariaDB 5.5.52**	Stable (GA)	Release Notes	Changelog
10 Aug 2016	**MariaDB 5.5.51**	Stable (GA)	Release Notes	Changelog
17 Jun 2016	**MariaDB 5.5.50**	Stable (GA)	Release Notes	Changelog
22 Apr 2016	**MariaDB 5.5.49**	Stable (GA)	Release Notes	Changelog
11 Feb 2016	**MariaDB 5.5.48**	Stable (GA)	Release Notes	Changelog
10 Dec 2015	**MariaDB 5.5.47**	Stable (GA)	Release Notes	Changelog
12 Oct 2015	**MariaDB 5.5.46**	Stable (GA)	Release Notes	Changelog
6 Aug 2015	**MariaDB 5.5.45**	Stable (GA)	Release Notes	Changelog
11 Jun 2015	**MariaDB 5.5.44**	Stable (GA)	Release Notes	Changelog
1 May 2015	**MariaDB 5.5.43**	Stable (GA)	Release Notes	Changelog
19 Feb 2015	**MariaDB 5.5.42**	Stable (GA)	Release Notes	Changelog
21 Dec 2014	**MariaDB 5.5.41**	Stable (GA)	Release Notes	Changelog
9 Oct 2014	**MariaDB 5.5.40**	Stable (GA)	Release Notes	Changelog
5 Aug 2014	**MariaDB 5.5.39**	Stable (GA)	Release Notes	Changelog
9 Jun 2014	**MariaDB 5.5.38**	Stable (GA)	Release Notes	Changelog
17 Apr 2014	**MariaDB 5.5.37**	Stable (GA)	Release Notes	Changelog

25 Feb 2014	**MariaDB 5.5.36**	Stable (GA)	Release Notes	Changelog
29 Jan 2014	**MariaDB 5.5.35**	Stable (GA)	Release Notes	Changelog
21 Nov 2013	**MariaDB 5.5.34**	Stable (GA)	Release Notes	Changelog
20 Sep 2013	**MariaDB 5.5.33a**	Stable (GA)	Release Notes	Changelog
17 Sep 2013	**MariaDB 5.5.33**	Stable (GA)	Release Notes	Changelog
18 Jul 2013	**MariaDB 5.5.32**	Stable (GA)	Release Notes	Changelog
23 May 2013	**MariaDB 5.5.31**	Stable (GA)	Release Notes	Changelog
12 Mar 2013	**MariaDB 5.5.30**	Stable (GA)	Release Notes	Changelog
30 Jan 2013	**MariaDB 5.5.29**	Stable (GA)	Release Notes	Changelog
29 Nov 2012	**MariaDB 5.5.28a**	Stable (GA)	Release Notes	Changelog
22 Oct 2012	**MariaDB 5.5.28**	Stable (GA)	Release Notes	Changelog
7 Sep 2012	**MariaDB 5.5.27**	Stable (GA)	Release Notes	Changelog
22 Jun 2012	**MariaDB 5.5.25**	Stable (GA)	Release Notes	Changelog
31 May 2012	**MariaDB 5.5.24**	Stable (GA)	Release Notes	Changelog
11 Apr 2012	**MariaDB 5.5.23**	Stable (GA)	Release Notes	Changelog
29 Mar 2012	**MariaDB 5.5.22**	Release Candidate	Release Notes	Changelog
16 Mar 2012	**MariaDB 5.5.21**	Beta	Release Notes	Changelog
25 Feb 2012	**MariaDB 5.5.20**	Alpha	Release Notes	Changelog

7.0.13 MariaDB Server 5.3

Changes & Improvements in MariaDB 5.3

MariaDB 5.3 is no longer supported. Last release: 5.3.12 | Release Date: 30 Jan 2013

Release Notes - MariaDB 5.3 Series

Release Notes - MariaDB 5.3 Series

Changelogs - MariaDB 5.3 Series

MariaDB 5.3 changelogs.

7.0.13.1 Changes & Improvements in MariaDB 5.3

MariaDB 5.3 is no longer supported. Please use a more recent release.

Download MariaDB 5.3

Date	Release	Status	Release Notes	Changelog
30 Jan 2013	**MariaDB 5.3.12**	Stable (GA)	Release Notes	Changelog
29 Nov 2012	**MariaDB 5.3.11**	Stable (GA)	Release Notes	Changelog
13 Nov 2012	**MariaDB 5.3.10**	Stable (GA)	Release Notes	Changelog
02 Oct 2012	**MariaDB 5.3.9**	Stable (GA)	Release Notes	Changelog
28 Aug 2012	**MariaDB 5.3.8**	Stable (GA)	Release Notes	Changelog
4 May 2012	**MariaDB 5.3.7**	Stable (GA)	Release Notes	Changelog

9 Apr 2012	**MariaDB 5.3.6**	Stable (GA)	Release Notes	Changelog
29 Feb 2012	**MariaDB 5.3.5**	Stable (GA)	Release Notes	Changelog
15 Feb 2012	**MariaDB 5.3.4**	Release Candidate	Release Notes	Changelog
21 Dec 2011	**MariaDB 5.3.3**	Release Candidate	Release Notes	Changelog
14 Oct 2011	**MariaDB 5.3.2**	Beta	Release Notes	Changelog
10 Sep 2011	**MariaDB 5.3.1**	Beta	Release Notes	Changelog
26 July 2011	**MariaDB 5.3.0**	Beta	Release Notes	Changelog

Contents

The focus for MariaDB 5.3 is to radically improve performance for subqueries, as well as for joins and single-table queries over large data sets.

MariaDB 5.3 is based on MariaDB 5.2 and thus on MariaDB 5.1 and MySQL 5.1. It is no longer being supported.

Some of the code was backported from MySQL 6.0 (a MySQL version that was never released as GA by Oracle), some was re-engineered and enriched by new features, and some code was written from scratch.

Any new feature or combination of features can be switched on/off dynamically via the optimizer_switch system variable.

The first stable (GA) release of MariaDB 5.3 was MariaDB 5.3.5, which was released on 29 Feb 2012.

You can download the latest binaries of MariaDB 5.3 here, or get the latest source code from launchpad.

Feature Comparison Matrix

We have created an Optimizer Feature Comparison Matrix showing the new optimizer features in MariaDB 5.5 and 5.3 compared to MySQL 5.5 and 5.6.

Query optimizer

Subquery optimizations

Subqueries are finally usable in practice. It is no longer necessary to rewrite subqueries manually into joins or into separate queries. MariaDB 5.3 aims to provide reasonably efficient handling for all kinds of subqueries. All problems with `EXPLAIN` taking a long time have also been resolved.

- Semi-join subquery optimizations
  These transform subqueries into 'semi-joins', entities similar to inner joins, and then use join optimizer to pick the best semi-join execution strategy. Overall the process is similar to how joins are processed in MySQL,MariaDB and other database systems.
  - Table pullout optimization
  - FirstMatch execution strategy
  - Semi-join Materialization execution strategy
  - LooseScan execution strategy
  - DuplicateWeedout execution strategy
- Non-semi-join optimizations
  If a subquery is not a semi-join, MariaDB 5.3 will make a cost-based choice between these two strategies:

    - Materialization for non-correlated subqueries, with efficient NULL-aware execution
    - IN-to-EXISTS transformation (the only optimization inherited from MariaDB 5.2 and MySQL 5.1/5.5)

- Subquery Cache
  The subquery cache makes sure that subqueries are re-executed as few times as possible, improving performance of already optimized subqueries.
- Subqueries are never executed during `EXPLAIN`, thus resulting in almost instant `EXPLAIN`.
- DISTINCT and GROUP BY without HAVING are optimized away from subqueries.

The Subquery Optimizations Map shows new subqueries optimizations graphically.

Optimizations for derived tables and views

- No early materialization of derived tables (e.g. subqueries in a `FROM` clause) and materialized views (`EXPLAIN` is always instantaneous)
- Thanks to Derived Table Merge optimization, mergeable derived tables are now processed like mergeable VIEWs.
- Derived Table with Keys optimization gives the optimizer an option to create indexes over materialized derived tables
- Fields of merge-able views and derived tables are involved now in all optimizations employing equalities

Disk access optimization

- Index Condition Pushdown
- Multi-Range-Read optimization (MRR)
    - Key-ordered retrieval

Join optimizations

- Block-based Join Algorithms
- Block Nested Loop algorithm can be used for outer joins
- Block Hash Join (classic algorithm) is implemented and can be used for any equi-joins
- Block Index Join (Batch Key Access Join) is supported and can exploit the benefits of ordered retrievals for primary and secondary keys provided by the new implementation of MRR
- All block based algorithms for joins can use the benefits of new incremental join buffers
- All block based algorithms fully support outer joins including nested outer joins
- All block based algorithms can use the benefits of the first match optimization for semi-joins and the non-exist optimization for outer joins
- All block based algorithms for joins can exploit the benefits of index condition push-down.
- The total memory space used by the query for join buffers can be limited now, and block based algorithms can allocate join buffers up to their needs (not exceeding the set limits).
- Condition over outer tables extracted from ON expressions of outer joins are evaluated before inner tables are accessed (supported for both regular index join and block index join)
- Early checks for nulls for the fields from any null-rejecting conditions are performed

Index Merge improvements

- Correct optimization of index_merge vs range access: Fair choice between range and index_merge optimizations
- index_merge/sort_intersection strategy

Optimizer control

- @@optimizer_switch variable can be used to turn on/off all new optimizations.

NoSQL-style interfaces

- HandlerSocket plugin included.
- Faster HANDLER commands; HANDLER READ now also work with prepared statements.
- Dynamic Columns support.

Replication and binary logging

- Group commit for the binary log — MariaDB 5.3 implements group commit which works when using XtraDB with the binary log enabled. (In previous MariaDB releases, and all MySQL releases at the time of writing, group commit works in InnoDB/XtraDB when the binary log is disabled, but stops working when the binary log is enabled).

- Annotation of row-based replication events with the original SQL statement — When using row-based replication, the binary log does not contain SQL statements, only discrete single-row insert/update/delete *events*. This can make it harder to read mysqlbinlog output and understand where in an application a given event may have originated, complicating analysis and debugging.This feature adds an option to include the original SQL statement as a comment in the binary log (and shown in mysqlbinlog output) for row-based replication events.
- Checksums for binlog events. This is a backport of the same feature in MySQL 5.6. It was implemented in MWL#180.
- Enhancements for START TRANSACTION WITH CONSISTENT SNAPSHOT — In MariaDB 5.3, `START TRANSACTION WITH CONSISTENT SNAPSHOT` now also works with the binary log. This means it is possible to obtain the binlog position corresponding to a transactional snapshot of the database without blocking any other queries. This is used by the command "`mysqldump --single-transaction --master-data`" to do a fully non-blocking backup which can be used to provision a new slave. "`START TRANSACTION WITH CONSISTENT SNAPSHOT`" now also works consistently between transactions involving more than one storage engine (currently XTraDB and PBXT support this).
- Row-based replication for tables with no primary key — This feature can improve the performance of row-based replication on tables that do not have a primary key (or other unique key), but which do have another index that can help locate rows to update or delete. With this feature, index cardinality information from `ANALYZE TABLE` is considered when selecting the index to use (before this feature is implemented, the first index was selected unconditionally).
- `mysqlbinlog` will now omit redundant `use` statements around `BEGIN`, `SAVEPOINT`, `COMMIT`, and `ROLLBACK` events when reading MySQL 5.0 binlogs.

Datatypes

- Microsecond support for NOW() and timestamp, time, and datetime columns.
- CAST() now supports AS DECIMAL[(M,D)] and AS INT.
- CAST() and all other datetime/time functions now supports microsecond fully.

Windows performance improvements

- Backported Windows performance patches from MySQL 5.5.
- Asynchronous IO in XtraDB is redesigned and is now faster, due to the use of IO completion ports.
- Additional durability option for XtraDB : `innodb_flush_method` can now be `O_DSYNC`, like on Unixes. The effect of using this option is that the log file is opened with `FILE_FLAG_WRITETHROUGH`, and `FlushFileBuffers()` is not done. This may improve speed in write-heavy scenarios.
- A new Windows MSI installer.
- Includes a GUI-tool, HeidiSQL.

Miscellaneous

- GIS precise operations
- New status variables: `Rows_tmp_read`, `Handler_tmp_write`, and `Handler_tmp_update` which count what happens with internal temporary tables. `Rows_read`, `Handler_write` and `Handler_update` no longer count operations on internal temporary tables.
- New status variable `Handler_read_rnd_deleted`, which is number of deleted rows found and skipped while scanning a table. Before this was part of `Handler_read_rnd_next`.
- New variable 'in_transaction' that is 1 if you are in a transaction, 0 otherwise.
- Progress reports for `ALTER TABLE` and `LOAD DATA INFILE`. In addition Aria tables gives progress reports for `REPAIR TABLE` and `CHECK TABLE`. The progress can be seen in `SHOW PROCESSLIST`, `INFORMATION_SCHEMA.PROCESSLIST` and is sent to MariaDB clients that calls `mysql_real_connect()` with the new `CLIENT_PROGRESS` flag. `mysql` command line client supports the new progress indications.
- PBXT consistent commit ordering — This feature implements the new commit ordering storage engine API in PBXT. With this feature, it is possible to use "`START TRANSACTION WITH CONSISTENT SNAPSHOT`" and get consistency among transactions which involve both XtraDB and InnoDB. (Without this feature, there is no such consistency guarantee. For example, even after running "`START TRANSACTION WITH CONSISTENT SNAPSHOT`" it was still possible for the InnoDB/XtraDB part of some transaction *T* to be visible and the PBXT part of the same transaction *T* to not be visible.)
- MariaDB unique error numbers now start from `1900` to not clash with MySQL error numbers.
- `/*M!##### */` new executed comment syntax that can be used when you want use new MariaDB syntax but still want your program to be compatible with MySQL.
- A MariaDB optimized version of mytop is included in the MariaDB distribution.
- Enhanced KILL syntax:

```
KILL [HARD | SOFT] [CONNECTION | QUERY] [thread_id | USER user_name]
```

- `max_user_connections` (both the global variable and the `GRANT` option) can be set to `-1` to stop users from connecting to the server. The global `max_user_connections` variable does not affect users with the `SUPER` privilege.
- The IGNORE directive does not ignore all errors (like fatal errors), only things that are safe to ignore.

You can access the MariaDB 5.3 tree from launchpad.

Security Vulnerabilities Fixed in MariaDB 5.3

For a complete list of security vulnerabilities (CVEs) fixed across all versions of MariaDB, see the Security Vulnerabilities Fixed in MariaDB page.

- CVE-2013-1531: MariaDB 5.3.12
- CVE-2013-0389: MariaDB 5.3.12
- CVE-2013-0385: MariaDB 5.3.12
- CVE-2013-0384: MariaDB 5.3.12
- CVE-2013-0383: MariaDB 5.3.12
- CVE-2013-0375: MariaDB 5.3.12
- CVE-2012-5627: MariaDB 5.3.12 [2]
- CVE-2012-5615: MariaDB 5.3.12 [2]
- CVE-2012-5612: MariaDB 5.3.12
- CVE-2012-5611: MariaDB 5.3.12, MariaDB 5.3.11
- CVE-2012-4414: MariaDB 5.3.8 [2]
- CVE-2012-1705: MariaDB 5.3.12
- CVE-2012-1702: MariaDB 5.3.12
- CVE-2012-0574: MariaDB 5.3.12
- CVE-2012-0572: MariaDB 5.3.12

7.0.14 MariaDB Server 5.2

Changes & Improvements in MariaDB 5.2

MariaDB 5.2 is no longer supported. Last release: 5.2.14 | Release Date: 30 Jan 2013

Release Notes - MariaDB 5.2 Series

MariaDB 5.2 Series Release Notes

Changelogs - MariaDB 5.2 Series

MariaDB 5.2 changelogs.

7.0.14.1 Changes & Improvements in MariaDB 5.2

MariaDB 5.2 is no longer supported. Please use a more recent release.

Download MariaDB 5.2

Date	Release	Status	Release Notes	Changelog
30 Jan 2013	**MariaDB 5.2.14**	Stable (GA)	Release Notes	Changelog
29 Nov 2012	**MariaDB 5.2.13**	Stable (GA)	Release Notes	Changelog
6 Apr 2012	**MariaDB 5.2.12**	Stable (GA)	Release Notes	Changelog
2 Apr 2012	**MariaDB 5.2.11**	Stable (GA)	Release Notes	Changelog
5 Dec 2011	**MariaDB 5.2.10**	Stable (GA)	Release Notes	Changelog
22 Sep 2011	**MariaDB 5.2.9**	Stable (GA)	Release Notes	Changelog

18 Aug 2011	**MariaDB 5.2.8**	Stable (GA)	Release Notes Changelog
14 Jun 2011	**MariaDB 5.2.7**	Stable (GA)	Release Notes Changelog
12 May 2011	**MariaDB 5.2.6**	Stable (GA)	Release Notes Changelog
3 Mar 2011	**MariaDB 5.2.5**	Stable (GA)	Release Notes Changelog
6 Dec 2010	**MariaDB 5.2.4**	Stable (GA)	Release Notes Changelog
10 Nov 2010	**MariaDB 5.2.3**	Stable (GA)	Release Notes Changelog
28 Sep 2010	**MariaDB 5.2.2**	Release Candidate	Release Notes Changelog
18 Jun 2010	**MariaDB 5.2.1**	Beta	Release Notes Changelog
10 Apr 2010	**MariaDB 5.2.0**	Beta	Release Notes Changelog

Contents

MariaDB 5.2 contains features that didn't have time to go into MariaDB 5.1. For all practical purposes it's a drop in replacement for MariaDB 5.1 (and thus MySQL 5.1).

MariaDB 5.2 is based on MariaDB 5.1 and thus MySQL 5.1.

The new features in 5.2 are quite isolated and as most have been in use by members in the MySQL community for a long time. Current versions of MariaDB 5.2 are stable and can be downloaded from http://downloads.askmonty.org.

New storage engines

- OQGRAPH
  - Allows you to handle hierarchies (tree structures) and complex graphs (nodes having many connections in several directions)
- SphinxSE: Text search within MariaDB.
  - A built-in Sphinx client which allows MariaDB to talk to searchd, run search queries, and obtain search results.

New features

- Virtual columns
  - Columns that are an expression and are calculated on retrieval.
- Extended User Statistics
  - Client, User, Index and Table statistics.
- Segmented MyISAM key cache
  - The key cache's global mutex is split into several mutex which gives a notable speed improvement under multi user load. We have registered up to 250% more performance thanks to this.
- Pluggable Authentication
  - Authentication is done via an extensible plugin, which makes it easy to add any kind of authentication to MariaDB.
- Storage-engine-specific CREATE TABLE
  - Allows one to specify additional attributes per field, index or table to the storage engine.
- Enhancements to INFORMATION SCHEMA.PLUGINS table
  - We expose more information about the plugins, like maturity levels.
- Group commit for the Aria engine.
  - Speeds up multi user inserts.

Other things

We have also done several smaller speed improvements, bug fixes and code cleanups.

Security Vulnerabilities Fixed in MariaDB 5.2

For a complete list of security vulnerabilities (CVEs) fixed across all versions of MariaDB, see the Security Vulnerabilities Fixed in MariaDB page.

- CVE-2013-1531: MariaDB 5.2.14
- CVE-2013-0389: MariaDB 5.2.14
- CVE-2013-0385: MariaDB 5.2.14
- CVE-2013-0384: MariaDB 5.2.14
- CVE-2013-0383: MariaDB 5.2.14
- CVE-2013-0375: MariaDB 5.2.14
- CVE-2012-5627: MariaDB 5.2.14 [2]
- CVE-2012-5615: MariaDB 5.2.14 [2]
- CVE-2012-5612: MariaDB 5.2.14
- CVE-2012-5611: MariaDB 5.2.14, MariaDB 5.2.13
- CVE-2012-4414: MariaDB 5.2.13 [2]
- CVE-2012-1705: MariaDB 5.2.14
- CVE-2012-1702: MariaDB 5.2.14
- CVE-2012-0574: MariaDB 5.2.14
- CVE-2012-0572: MariaDB 5.2.14

7.0.15 MariaDB Server 5.1

Changes & Improvements in MariaDB 5.1

MariaDB 5.1 is no longer supported. Last release: 5.1.67 | Release Date: 30 Jan 2013

Release Notes - MariaDB 5.1 Series

MariaDB 5.1 Series Release Notes

Changelogs - MariaDB 5.1 Series

MariaDB 5.1 changelogs.

7.0.15.1 Changes & Improvements in MariaDB 5.1

MariaDB 5.1 is no longer supported. Please use a more recent release.

Download MariaDB 5.1

Date	Release	Status	Release Notes	Changelog
30 Jan 2013	**MariaDB 5.1.67**	Stable (GA)	Release Notes	Changelog
29 Nov 2012	**MariaDB 5.1.66**	Stable (GA)	Release Notes	Changelog
6 Apr 2012	**MariaDB 5.1.62**	Stable (GA)	Release Notes	Changelog
2 Apr 2012	**MariaDB 5.1.61**	Stable (GA)	Release Notes	Changelog
5 Dec 2011	**MariaDB 5.1.69**	Stable (GA)	Release Notes	Changelog
1 Mar 2011	**MariaDB 5.1.55**	Stable (GA)	Release Notes	Changelog
6 Dec 2010	**MariaDB 5.1.53**	Stable (GA)	Release Notes	Changelog
19 Nov 2010	**MariaDB 5.1.51**	Stable (GA)	Release Notes	Changelog
9 Sep 2010	**MariaDB 5.1.50**	Stable (GA)	Release Notes	Changelog
9 Aug 2010	**MariaDB 5.1.49**	Stable (GA)	Release Notes	Changelog
1 Jun 2010	**MariaDB 5.1.47**	Stable (GA)	Release Notes	Changelog
10 May 2010	**MariaDB 5.1.44**b	Stable (GA)	Release Notes	Changelog
24 Mar 2010	**MariaDB 5.1.44**	Stable (GA)	Release Notes	Changelog
1 Feb 2010	**MariaDB 5.1.42**	Stable (GA)	Release Notes	Changelog
13 Jan 2010	**MariaDB 5.1.41**	Release Candidate	Release Notes	Changelog
15 Nov 2009	**MariaDB 5.1.39**	Beta	Release Notes	Changelog

29 Oct 2009	**MariaDB 5.1.38**	Beta	Release Notes Changelog

Contents

In short, MariaDB 5.1 is a binary drop in replacement of MySQL 5.1, but with performance like MySQL 5.5 (thanks to XtraDB), fewer bugs, and more features. Thanks to the extended and improved `mysql_upgrade` program it's also easier to upgrade from MySQL 5.0 to MariaDB 5.1 than to MySQL 5.1.

MariaDB 5.1 is based on MySQL 5.1. We do a merge once a month with MySQL5.1 to ensure all MySQL bug fixes get into MariaDB.

See also:

- MariaDB versus MySQL - Compatibility

New storage engines:

- Aria: A crash-safe storage engine based on MyISAM.
- XtraDB: Drop-in replacement for InnoDB based on the InnoDB plugin.
- PBXT: A transactional storage engine with a lot of nice features.
- FederatedX: Drop-in replacement for Federated.

Speed improvements

- CHECKSUM TABLE is faster.
- We have eliminated/improved some not needed character set conversions. Overall speed improvements is 1-5 % (according to sql-bench) but can be higher for big result sets with all characters between 0x00-0x7f.
- Our use of the Aria storage engine enables faster complex queries (queries which normally use disk-based temporary tables). The Aria storage engine is used for internal temporary tables, which should give you a speedup when doing complex selects. Aria is usually faster for temporary tables when compared to MyISAM because Aria caches row data in memory and normally doesn't have to write the temporary rows to disk.
- There are some improvements to DBUG code to make its execution faster when debug is compiled in but not used.

Extensions & new features

- Table Elimination (New optimization) (MWL#17)
- Pool of Threads (Allows you to have 200,000+ connections to MariadB)
- MariaDB can handle up to 32 key segments per key (up from 16)
- Added `--abort-source-on-error` to the mysql client.
- Microsecond Precision in Processlist
- mysqltest extensions

Easier to upgrade

We have made it easy to upgrade from MySQL 5.0 to MariaDB 5.1

- InnoDB and Archive tables are now upgraded properly.
- More options to mysql_upgrade and mysqlcheck to find out what's going on.
- Cleaned up wrong warnings from mysqlcheck.

(Upgrading from MySQL 5.1 to MariaDB 5.1 is a trivial 1 min exercise as MariaDB is a binary drop in replacement of MySQL. Just remove MySQL and install MariaDB and things will ***just work***)

Better Testing

- More tests in the test suite.
- All tests runs now clean without having to restart test. (Oracle re-runs tests 3 times and assumes things are ok if one tests works)

- Test builds with different configure options to get better feature testing.
- Remove invalid tests. (e.g. Don't test feature **"X"** if that feature is not in the build you are testing.)

Fewer warnings and bugs

- A build is not regarded ok if there are any errors or compiler warnings.
- We have fixed a lot of bugs in the MySQL code which we have found while merging the MySQL code into MariaDB and by running the extended test suite.

Security Vulnerabilities Fixed in MariaDB 5.1

> For a complete list of security vulnerabilities (CVEs) fixed across all versions of MariaDB, see the Security Vulnerabilities Fixed in MariaDB page.

- CVE-2013-1548: MariaDB 5.1.66
- CVE-2013-1531: MariaDB 5.1.67
- CVE-2013-0389: MariaDB 5.1.67
- CVE-2013-0385: MariaDB 5.1.67
- CVE-2013-0384: MariaDB 5.1.67
- CVE-2013-0383: MariaDB 5.1.67
- CVE-2013-0375: MariaDB 5.1.67
- CVE-2012-5612: MariaDB 5.1.67
- CVE-2012-5611: MariaDB 5.1.67, MariaDB 5.1.66
- CVE-2012-5060: MariaDB 5.1.66
- CVE-2012-4414: MariaDB 5.1.66 [2]
- CVE-2012-3197: MariaDB 5.1.66
- CVE-2012-3180: MariaDB 5.1.66
- CVE-2012-3177: MariaDB 5.1.66
- CVE-2012-3173: MariaDB 5.1.66
- CVE-2012-3167: MariaDB 5.1.66
- CVE-2012-3166: MariaDB 5.1.66
- CVE-2012-3163: MariaDB 5.1.66
- CVE-2012-3160: MariaDB 5.1.66
- CVE-2012-3158: MariaDB 5.1.66
- CVE-2012-3150: MariaDB 5.1.66
- CVE-2012-1734: MariaDB 5.1.66
- CVE-2012-1705: MariaDB 5.1.67
- CVE-2012-1703: MariaDB 5.1.62
- CVE-2012-1702: MariaDB 5.1.67
- CVE-2012-1690: MariaDB 5.1.62
- CVE-2012-1689: MariaDB 5.1.66
- CVE-2012-1688: MariaDB 5.1.62
- CVE-2012-0574: MariaDB 5.1.67
- CVE-2012-0572: MariaDB 5.1.67
- CVE-2012-0540: MariaDB 5.1.66
- CVE-2009-4484: MariaDB 5.1.42

The following CVEs are also fixed in MariaDB 5.1 but the fix is not tied to a specific version number:

- CVE-2012-0113
- CVE-2011-2262
- CVE-2012-0116
- CVE-2012-0118
- CVE-2012-0087
- CVE-2012-0101
- CVE-2012-0102
- CVE-2012-0115
- CVE-2012-0119
- CVE-2012-0120
- CVE-2012-0484
- CVE-2012-0485
- CVE-2012-0490
- CVE-2012-0112
- CVE-2012-0114
- CVE-2012-0492
- CVE-2012-0075

8 The Community

MariaDB is a project developed by the open source community. The MariaDB Foundation is the main steward for the project. However, anyone can participate in the development.

The following links provides information to help you participate in making MariaDB a success

News & Information

News, web logs and other published information related to MariaDB.

Bug Tracking

How to Report Bugs and Make Feature Requests.

Contributing & Participating

Getting Help With MariaDB

Review of resources available when you need help.

Reporting Documentation Bugs

Reporting a bug or feature request in the MariaDB documentation.

MariaDB Community Bug Reporting

Guidelines for reporting bugs in MariaDB software.

Contributing to the MariaDB Project

How to contribute to the MariaDB project: code, documentation, feedback, etc.

Contributing Code

Guidelines and procedures for contributing code to MariaDB.

Project Suggestions

List of MariaDB Projects in which you might Assist.

Log of MariaDB Contributions

List of contributions and contributors to MariaDB and how to log them.

Donate to the Foundation

Donating to the MariaDB Foundation.

Sponsoring the MariaDB Project

Ways to sponsor the MariaDB project (e.g. with labor or funds).

Web Hosting Providers Offering MariaDB

List of known web hosting providers that offer MariaDB to their clients.

Using Git with MariaDB

How to use git to troubleshoot the source code or contribute code to MariaDB.

Google Summers of Code

Information on Google Summer of Code for each year since 2013.

Google Season of Docs

Information on each Google Season of Docs.

Friends of MariaDB

People Behind MariaDB

The founders, the MariaDB Foundation and the MariaDB Corporation.

MariaDB Users & Developers

Connect and interact with other MariaDB users and developers.

Social Media & Networking

List of MariaDB-related social media links and pages.

Applications Supporting MariaDB

Projects which support or work with MariaDB software.

Legal Documents

Legal Matters

Information on Legal Matters and Licenses.

MariaDB Contributor Agreement

The full text of the MariaDB Contributor Agreement, including signature lines.

MariaDB Contributor Agreement FAQ

Frequently asked questions about the MariaDB Contributor Agreement.

MariaDB License

List of licenses for the MariaDB server and clients.

Branding Guidelines

Guidelines for using the MariaDB logo.

MariaDB Trademark Policy

Policies and guidelines on using the MariaDB name and logos, the trademarks of MariaDB.

There are 450 related questions.

8.1 Bug Tracking

How and where the community can report bugs and file feature requests, and how bug reports are processed for community users. MariaDB Corporation provides SLA for customer issues (see Technical Support Services).

MariaDB Community Bug Reporting

Guidelines for reporting bugs in MariaDB software.

Reporting Documentation Bugs

Reporting a bug or feature request in the MariaDB documentation.

MariaDB Community Bug Processing

The Bug Reporting and Tracking Process.

MariaDB Security Bug Fixing Policy

Bug fixing policy and how security issues are handled.

Building MariaDB Server for Debugging

How to compile the mysqld daemon with debugging enabled.

Extracting Entries from the Binary Log

Using mariadb-binlog to extract entries from the binary log for debugging.

There are 18 related questions.

8.1.1 MariaDB Community Bug Reporting

Contents

For reporting documentation bugs specifically, see Reporting Documentation Bugs.

MariaDB's bug and feature tracker is found at https://jira.mariadb.org.

This page contains general guidelines for the community for reporting bugs in MariaDB products. If you want to discuss a problem or a new feature with other MariaDB developers, you can find the email lists and forums here.

Known Bugs

First, check that the bug isn't already filed in the MariaDB bugs database.

For the MariaDB bugs database, use JIRA search to check if a report you are going to submit already exists. You are not expected to be a JIRA search guru, but please at least make some effort.

- Choose `Issues => Search for issues` ;
- If the form opens for you with a long blank line at top, press `Basic` on the right to switch to a simpler mode;
- In the `Project` field, choose the related project, (`MDEV` for generic MariaDB server and clients);
- In the `Contains text` text field, enter the most significant key words from your future report;
- Press `Enter` or the magnifying glass icon to search.

If you see bug reports which are already closed, pay attention to the 'Fix version/s' field -- it is possible that they were fixed in the *upcoming* release. If they are said to be fixed in the release that you are currently using or earlier, you can ignore them and file a new one (although please mention in your bug report that you found them, it might be useful).

If you find an open bug report, please vote/add a comment that the bug also affects you along with any additional information you have that may help us to find and fix the bug.

If the bug is not in the MariaDB bugs database yet, then it's time to file a bug report. If you're filing a bug report about a bug that's already in the MySQL bugs database, please indicate so at the start of the report. Filing bug reports from MySQL in the MariaDB bugs database makes sense, because:

- It shows the MariaDB team that there is interest in having this bug fixed in MariaDB.
- It allows work to start on fixing the bug in MariaDB - assigning versions, assigning MariaDB developers to the bug, etc.

Reporting a Bug

Bugs and feature requests are reported to the MariaDB bugs database.

JIRA Privacy

Please note that our JIRA entries are public, and JIRA is very good at keeping a record of everything that has been done. What this means is that if you ever include confidential information in the description there will be a log containing it, even after you've deleted it. The only way to get rid of it will be removing the JIRA entry completely.

Attachments in JIRA are also public.

Access to a comment can be restricted to a certain group (e.g. Developers only), but the existing groups are rather wide, so you should not rely on it either.

If you have private information -- SQL fragments, logs, database dumps, etc. -- that you are willing to share with MariaDB team, but not with the entire world, put it into a file, compress if necessary, upload to the mariadb-ftp-server, and just mention it in the JIRA description. This way only the MariaDB team will have access to it.

Reporting Security Vulnerabilities

As explained above, all JIRA issues are public. If you believe you have found a security vulnerability, send an email to security@mariadb.org, please, do not use JIRA for that. We will enter it in JIRA ourselves, following the responsible disclosure practices.

Contents of a Good Bug Report

Below is the information we need to be able to fix bugs. The more information we get and the easier we can repeat the bug, the faster it will be fixed.

A good bug report consists of:

a. The environment (Operating system, hardware and MariaDB version) where the bug happened.
b. Any related errors or warnings from the server error log file. Normally it is `hostname.err` file in your database directory, but it can be different depending on the distribution and version; if you cannot find it, run `SELECT @@log_error` on the running server. If either the variable or the file it points at is empty, the error log most likely goes to your system log. If this is systemd you can get the last 50 lines of the MariaDB log with `journalctl -n 50 -u mariadb.service`. If possible, attach the full unabridged error log at least from the last server restart and till the end of the log.,
c. If the problem is related to MariaDB updates, or otherwise changing the version of the server, recovery from a previous crash, and such, then include the previous versions used, and the error log from previous server sessions.
d. The content of your my.cnf file or alternatively the output from `mariadbd --print-defaults` or `SHOW VARIABLES`.
e. Any background information you can provide (stack trace, tables, table definitions (`show-create-table SHOW CREATE TABLE {tablename}`), data dumps, query logs).
f. If the bug is about server producing wrong query results: the actual result (what you are getting), the expected result (what you think should be produced instead), and, unless it is obvious, the reason why you think the current result is wrong.
g. If the bug about a performance problem, e.g. a certain query is slower on one version than on another, output of `EXPLAIN EXTENDED <query>` on both servers. If its a `SELECT` query use analyze-format-json ANALYZE FORMAT=JSON.
h. A test case or some other way to repeat the bug. This should preferably be in plain SQL or in mysqltest format. See **mysqltest/README** for information about this.
i. If it's impossible to do a test case, then providing us with a backtrace information would be of great help.

JIRA Fields

The section below describes which JIRA fields need to be populated while filing reports, and what should be put there. Apart from what's mentioned below, you don't have to fill or change any fields while creating a new bug report.

Project

If you are filing a report for MariaDB server, client programs, or MariaDB Galera cluster, the target project is `MDEV`. Connectors and MaxScale have separate projects with corresponding names. If you choose a wrong project, bug processing can be delayed, but there is no reason to panic -- we'll correct it. If you inform us about the mistake, we'll change it faster.

Some project names include:

- CONC - MariaDB Connector/C
- CONJ - MariaDB Connector/J
- CONJS - MariaDB Connector/node.js
- CONPY - MariaDB Connector/Python
- MCOL - ColumnStore
- MDBF - MariaDB Foundation Development (anything related to the mariadb.org domain)
- MDEV - MariaDB server, client programs, or MariaDB Galera Cluster
- MXS - MaxScale
- ODBC - MariaDB Connector/ODBC

Type

Feature requests are not the same as bug reports. Specify a `Task` type for feature requests in Jira, and a `Bug` type for bug reports. Like with the project field, choosing a wrong type will put the request to the wrong queue and can delay its processing, but eventually it will be noticed and amended.

See also plans for next release for things that we are considering to have in the next MariaDB release.

Summary

Please make sure the summary line is informative and distinctive. It should always be easy to recognize your report among other similar ones, otherwise a reasonable question arises -- why are they not duplicates?

Examples:

- good summary: *Server crash with insert statement containing DEFAULT into view*
- not a good summary: *mariadbd crash*

Generally, we try not to change the original summary without a good reason to do it, so that you can always recognize your own reports easily.

Priority

We do not have separate Severity/Priority fields in JIRA, so this Priority field serves a double purpose. For original reports, it indicates the importance of the problem from the reporter's point of view. The default is 'Major'; there are two lower and two higher values. Please set the value accurately. While we do take it into account during initial processing, increasing the value above reasonable won't do any good, the only effect will be the waste of time while somebody will be trying to understand why a trivial problem got such a high priority. After that, the value will be changed, and the report will be processed in its due time anyway.

Affected Versions

Put everything you know about which versions are affected. There are both major versions (10.6, 10.5 etc.) and minor versions (10.5.9, 10.4.12, etc.) available for choosing. Please always specify there the exact version(s) (X.Y.Z) which you are working with, and where you experience the problem.

Additionally, If you know the exact version where the problem appeared, please put it as well. If the problem has been present, as far as you know, in all previous releases, you can also put there the major version, e.g. 10.0. Alternatively, you can mention all of it in the description or comments.

Please also note in the description or comments which versions you know as *not* affected. This information will help to shorten further processing.

Environment

Put here environment-related information that might be important for reproducing or analyzing the problem: operating system, hardware, related 3rd-party applications, compilers, etc.

Description

The most important part of the description are steps to reproduce the problem. See more details about bug report contents above in the section Contents of a good bug report.

If in the process of reproducing, you executed some SQL, don't describe it in words such as "I created a table with text columns and date columns and populated it with some rows" -- instead, whenever possible, put the exact SQL queries that you ran. The same goes for problems that you encountered: instead of saying "it did not work, the query failed, I got an error", always paste the exact output that you received.

Use {noformat}...{noformat} and {code}...{code} blocks for code and console output in the description.

Attachments

If you have SQL code, a database dump, a log etc. of a reasonable size, attach them to the report (archive them first if necessary). If they are too big, you can upload them to ftp.askmonty.org/private. It is always a good idea to attach your cnf file(s), unless it is absolutely clear from the nature of the report that configuration is irrelevant.

Links

If you found or filed a bug report either in MariaDB or MySQL or Percona bug base which you think is related to yours, you

can put them in the `Links` section; same for any external links to 3rd-party resources which you find important to mention. Alternatively, you can just mention them in the description or comments.

Tags

You don't have to set any tags, but if you want to use any for your convenience, feel free to do so. However, please don't put too generic values -- for example, the tag `mariadb` is meaningless, because everything there is `mariadb` . Don't be surprised if some tags are removed later during report processing.

Bugs that also Affect MySQL or Percona

Our normal practice is to report a bug upstream if it's applicable to their version. While we can do it on your behalf, it is always better if you do it yourself -- it will be easier for you to track it further.

If the bug affects MySQL, it should also be reported at MySQL bugs database. If the bug affects Percona server and not MySQL, it should go to Percona Launchpad.

Collecting Additional Information for a Bug Report

Getting a Stack Trace with Details

See the article How to produce a stack trace from a core file.

Extracting a Portion of a Binary Log

See the article here.

Getting Help with your Servers

If you require personalized assistance, want to ensure that the bug is fixed with high priority, or want someone to login to your server to find out what's wrong, you can always purchase a Support contract from MariaDB plc or use their consulting services.

8.1.2 Reporting Documentation Bugs

Contents

Documentation bugs and feature requests should be reported at https://jira.mariadb.org.

This page contains general guidelines for the community for reporting documentation bugs.

Known Bugs

First, check that the bug isn't already filed in the MariaDB bugs database.

For the MariaDB bugs database, use JIRA search to check if a report you are going to submit already exists. You are not expected to be a JIRA search guru, but please at least make some effort.

- Choose `Issues => Search for issues` ;
- If the form opens for you with a long blank line at top, press `Basic` on the right to switch to a simpler mode;
- In the `Project` field, choose the related project, (`MDEV` for generic MariaDB server and clients);

- In the `Contains text` text field, enter the most significant key words from your future report;
- Press `Enter` or the magnifying glass icon to search.

If you see bug reports which are already closed, but you still see the error, please confirm that the issue still exists in the Knowledge Base.

If you find an open bug report, please vote/add a comment that the bug also affects you along with any additional information you have that may help us to find and fix the bug.

If the bug is not in the MariaDB bugs database yet, then it's time to file a bug report.

Reporting a Bug

Bugs and feature requests are reported to the MariaDB bugs database.

JIRA Privacy

Please note that our JIRA entries are public, and JIRA is very good at keeping a record of everything that has been done. What this means is that if you ever include confidential information in the description there will be a log containing it, even after you've deleted it. The only way to get rid of it will be removing the JIRA entry completely.

Attachments in JIRA are also public.

Access to a comment can be restricted to a certain group (e.g. Developers only), but the existing groups are rather wide, so you should not rely on it either.

JIRA Fields

The section below describes which JIRA fields need to be populated while filing reports, and what should be put there. Apart from what's mentioned below, you don't have to fill or change any fields while creating a new bug report.

Project

If you are filing a report for documentation about MariaDB server, client programs, or MariaDB Galera Cluster, the target project is `MDEV`. Connectors and MaxScale have separate projects with corresponding names. If you choose a wrong project, bug processing can be delayed, but there is no reason to panic -- we'll correct it. If you inform us about the mistake, we'll change it faster.

Some project names include:

- CONC - MariaDB Connector/C
- CONJ - MariaDB Connector/J
- CONJS - MariaDB Connector/node.js
- CONPY - MariaDB Connector/Python
- MCOL - ColumnStore
- MDBF - MariaDB Foundation Development (anything related to the mariadb.org domain)
- MDEV - MariaDB server, client programs, or MariaDB Galera Cluster
- MXS - MaxScale
- ODBC - MariaDB Connector/ODBC

Type

Feature requests are not the same as bug reports. Specify a `Task` type for feature requests in Jira, and a `Bug` type for bug reports. Like with the project field, choosing a wrong type will put the request to the wrong queue and can delay its processing, but eventually it will be noticed and amended.

Summary

Please make sure the summary line is informative and distinctive. It should always be easy to recognize your report among other similar ones, otherwise a reasonable question arises -- why are they not duplicates?

Examples:

- good summary: *SELECT max_statement_time clause example gives incorrect results*
- not a good summary: *code example doesn't work*

Generally, we try not to change the original summary without a good reason to do it, so that you can always recognize your own reports easily.

Priority

We do not have separate Severity/Priority fields in JIRA, so this Priority field serves a double purpose. For original reports, it indicates the importance of the problem from the reporter's point of view. The default is 'Major'; there are two lower and two higher values. Please set the value accurately. While we do take it into account during initial processing, increasing the value above reasonable won't do any good, the only effect will be the waste of time while somebody will be trying to understand why a trivial problem got such a high priority. After that, the value will be changed, and the report will be processed in its due time anyway.

Component/s

Documentation bugs should have "Documentation" added as a component in order to be correctly assigned.

Affected Version/s

Since the documentation is not version-dependent, you can put N/A in this field.

Environment

Usually this can be left empty, but if applicable, put any environment-related information that might be important for reproducing or analyzing the problem.

Description

The most important part of the description are the steps to reproduce the problem. Link to the page on the Knowledge Base with the error/s. Where applicable, provide a sample structure and results clearly demonstrating the problem.

Use {noformat}...{noformat} and {code}...{code} blocks for code and console output in the description.

Links

If the documentation error relates to an existing bug/feature request in JIRA (for example an undocumented new feature), you should link it here. Alternatively, you can just mention them in the description or comments.

Tags

You don't have to set any tags, but if you want to use any for your convenience, feel free to do so. However, please don't put too generic values -- for example, the tag `mariadb` is meaningless, because everything there is `mariadb` . Don't be surprised if some tags are removed later during report processing.

8.1.3 MariaDB Community Bug Processing

Contents

This page describes how community bug reports are processed among our products and explains what you need to notice while tracking bugs.

Commitments

MariaDB does not have any SLA or guaranteed reaction times on bugs in Jira. While we are taking bugs reported by the community very seriously, and aim to provide response and to handle issues as fast as possible, MariaDB does not have a dedicated bug verification team, this activity is performed on the best-effort basis.

To make sure your bug report will be confirmed and moved forward faster, please follow **the guidelines** about creating bug reports.

Bug Verification Routine

As of today, initial bug processing routine in MariaDB is not strictly formalized. This section describes the de-facto status rather than any policy.

The process is different for bug reports (*Bug* type in JIRA) vs feature requests (*Task* type). The process described below is related to bug reports.

Incoming Queue

All new bug reports go to the waiting list, to be reproduced and confirmed by a member of the team. The bug stays in the queue until one or more of the conditions below are met:

- Bug report is assigned to a developer;
- Bug report gets status 'Confirmed';
- Bug report gets the label 'upstream';
- Bug report is closed (on whatever reason).

With other things equal, bug reports in the queue are initially handled in the FIFO manner; however, there are various factors that make things not equal.

Bug Processing Order

First thing that is taken into account is **Priority**. It does not mean that everything needs to be filed as Critical; on the contrary, it means that Priority should be chosen wisely. Although a report with higher Priority will be looked at sooner, as soon as it becomes clear that the Priority is set to a higher value than the problem deserves, it will be amended and put back to the queue. However, if the high priority is justified, we will try to process the report as fast as possible.

Another important factor is **the quality of the report** itself.

- If the report is written well and **has all information**, including a reproducible test case, it can be verified and moved forward quickly.
- If the report is written clearly enough, but does not have enough information, it will get fast enough first response where we will request the missing details; but the further exchange can take a lot of time until we get everything we need to work on the issue.
- Finally, reports which are written in a tangled and incomprehensible manner get the slowest response time, because even if eventually it turns out that they do have all required information, it is difficult and time-consuming to extract and process, so they can be put aside for some time.

First Response

Complete processing of a reported bug can be complicated and time-consuming, especially the reproducing part. We do not want our users to wait for long not knowing if their bug report has even been noticed, we try to provide first response quicker than that.

First response to the bug, which we are trying to provide as quickly as possible, is one of these:

- If we can reproduce the problem based on the information that was provided in the initial description, the report gets the status Confirmed.
- If it is obvious from the initial description that the bug report is a **duplicate** of an existing one, or the problem has already been fixed in later releases or in the upcoming release, or the described behavior is not a bug, or, in very rare cases, it is admitted to be a bug, but it is not going to be fixed, the report gets closed with the corresponding Resolution value and a comment with the explanation.
- If the bug report at least appears to describe a real bug, but we do not have enough information to proceed, we will request the information from the reporter, and the report will go to the **Need feedback** list.
- If on some reason it is clear from the bug report that it will be very difficult to reproduce based on the information from the user, but there is a reason to believe that the problem can be analyzed by code inspection, the bug report can be assigned to a developer who is an expert in the corresponding area for the analysis.

We realize that "as quickly as possible" is a relative term. The dream scenario is that all reports are responded to in a matter of hours; however, more realistically, it can take a few days, and in some cases, when the team is overly busy with a big upcoming release or some other extraordinary events, it can even be weeks.

Need Feedback

When a report does not have all the information to reproduce the problem right away (which is quite often the case), we will ask the reporter to provide the missing information. Usually it takes more than one iteration to get everything right, so it is important that you respond to the questions as precisely as you can. Please make sure that you answered **all** questions (or, if you cannot answer some of them, please say so, otherwise we will have to ask again, and more time will be wasted on it).

There is no status "Need Feedback" in our JIRA; instead, we are using the label `need_feedback`. As long as the report has this label, it remains on the "Waiting for feedback" list. The label is set and removed manually by whoever asks for the feedback and receives it; so it can happen that the reporter has provided the response, but it remained unnoticed and the bug keeps waiting. It will be our fault, but human errors happen; it would help a lot if the reporter removed the label along with providing the feedback.

E-mail Notifications

This question arises fairly often, so it deserves mentioning.

As already said before, the `need_feedback` label is set and *removed* manually. JIRA e-mail updates about it can be confusing when you look at them quickly. For example, when someone *removes* the label, the email looks like this:

```
Elena Stepanova updated MDEV-9791:
----------------------------------
    Labels:   (was: need_feedback)
```

What it says that the `Labels` field has become empty, while before it had been `need_feedback`. People often misread it and ask "What else do you need from me? I've answered your questions". This update means that at the moment we don't need anything, your report is back to the incoming queue, and your feedback will be analyzed as soon as possible. Then, we will possibly ask more questions and set the label again, and the notification will look like this:

```
Elena Stepanova updated MDEV-9801:
----------------------------------
    Labels: need_feedback  (was: )
```

Successful Outcome

If the feedback exchange was fruitful and we received enough information to proceed, the bug report will go through the normal **verification steps**.

Incomplete Reports

Reports do not stay open on the "Need Feedback" list forever. After a month of waiting, if we do not get a response from the reporter, and still cannot proceed without it, we close the report as **Incomplete** with the corresponding comment. This state is not irreversible: you can still add comments and provide the information even when the report is closed as Incomplete, and it will be re-opened.

Worst Case Scenario

Sometimes it happens that after iterations of feedback requests we run out of ideas what else to ask from the reporter, and still could not verify the bug, or that the reporter is willing to collaborate with us, but cannot provide the necessary information on objective reasons (usually when the problem happens on a production instance). In some cases we might close the report as "Cannot reproduce", which we consider our loss; but more often we want to keep it open, in hope that more information arrives, maybe from a different source, and together with this report they will help us get to the bottom of the problem; if it happens so, the report gets assigned to somebody without being confirmed, just so it remains at least on somebody's radar, and it will stay open for a long time. It does not mean it is forgotten, it means that for the time being we hit the wall. You are very welcome to comment on such reports, whenever you think you might have something to add, because this is exactly what we are waiting for.

Bug Verification

Normally the bug report has to go through the following steps before it is moved forward to fixing:

- the described problem needs to be reproduced;
- it needs to be checked against all active post-Beta versions of MariaDB where it is theoretically applicable (as of the moment of writing this article, it is 5.5, 10.0, 10.1);
- in case it is a relatively recent regression, the guilty change needs to be found;
- the component or functional area should be determined, so that the bug gets assigned to the right person.

After that the bug is ready for fixing.

Bug Fixing Routine

Sometimes it seems hard to understand from the outside how MariaDB development team chooses which bugs to fix in a particular release, or why some bugs are fixed faster than others, or why critical bugs stay untouched for a long time.

Sprint Model for Bug Fixing

MariaDB currently uses 1- or 2-week sprint model for server development and bugfixing. It needs a separate article to describe it in more detail, but for bugfixing, in a nutshell it means the following.

- one or two weeks before a scheduled release the team creates a new sprint and evaluates existing bugs which affect this release;
- the selected bugs are added to the new sprint;
- during the active sprint, the developer is supposed to work on the tasks which are part of the sprint, unless a true emergency arises.

There are two important consequences of this model which sometimes cause a confusion:

1) If the current sprint is for one version, e.g. 10.0, and you file a bug for another version, e.g. 10.1, then, even if the bug is really critical, it won't be jumped on right away: it makes no sense, because the 10.1 is not going to be released next week anyway, while 10.0 will be. When the 10.0 sprint finishes, and 10.1 sprint starts, your bug will be picked up for that sprint and fixed then.

2) If the current sprint for 10.1 is already in progress, newly created 10.1 reports normally won't be included into it, unless they are considered so extremely important that the developer is allowed to ignore the sprint plan.

Bugs Chosen for a Sprint

When a new sprint is created, bugs which affect the scheduled release are evaluated.

- from all such bugs assigned to a developer, each developer chooses bugs he is able to work on during the given time interval;
- bug priority plays the most significant role in this process, but this is not the only factor.
    - `Blocker` bugs must be either fixed or degraded before the release goes out;

- `Critical` bugs should be chosen above other bugs, except for `Blocker` s;
- among `Major` bugs,
  - bugs with patches, either external, or upstream, or internal, are usually prioritized above ordinary bug reports;
  - external reports (community reports) are ranked higher than bugs reported by the development team;
  - bugs which can realistically be fixed in the given time interval are chosen more frequently than those that are likelly to take several cycles;
  - bugs which affect the reporter in a worse matter get more attention than those that have viable workarounds;
- `Minor` bugs are usually fixed when there are no more urgent tasks.

Tracking Progress

If a bug report has passed through verification stage, either being confirmed, or pushed forward to the development-level analysis as is, there can be various updates on it. It is important to understand what they mean.

JIRA Fields to Watch

All JIRA fields are public, but some of them are mainly used for internal development process, while others are more user-facing. **This article** describes which fields should be populated during the initial report submission. There is a different set of fields important for tracking purposes.

Resolution vs. Status

It might come as counter-intuitive, but in the existing JIRA structure, the `Status` field does not mean much for the user, it is mainly used for development and management purposes. On the contrary, the `Resoluton` field is entirely user-facing: it does not participate in planning or development. It remains the same 'Unresolved' for the whole life of the report, and is only changed when the bug gets closed, demonstrating the reason why it was closed.

Resolution

- `Unresolved` - the bug report remains open, the work has not been finished.
- `Fixed` - the bug has been fixed, see **Fix version/s** and possibly comments to the report for more information. This is almost always a terminal state, we do not re-open fixed bugs even if they later re-appear; please create a new one instead. The only case when it can be re-opened is when the 'Fix version/s' have not been released yet.
- `Duplicate` - the bug report is identical to an already existing open (or recently fixed) report, which will be quoted in the comments and/or links. It is usually a terminal state, unless it is proven later that the report was not a duplicate after all.
- `Not a bug` - the described behavior is not a bug, there will be a comment explaining why. It is usually a terminal state, unless you object and show why it is a bug. If the report is in fact a feature request, then rather than closing it as 'Not a bug', we will switch the type to 'Task'.
- `Incomplete` - we had requested feedback from the user and waited for 1 month, but did not receive it. It is a pseudo-terminal state, the report can be re-opened any time when the requested information is provided.
- `Cannot reproduce` - rather rarely used "resolution", which means we could not find the way to confirm the problem described by the reporter, and ran out of ideas what other information to request from the reporter in order to reproduce it.
- `Won't fix` - another rarely used "resolution", which means that the bug is admitted, but we have no intention to fix it. Usually it happens when the bug only affects old versions, and is not important enough to fix in the old versions; or, when it is related to systems or architectures we don't officially support.

Status

- `Open`, `Confirmed` - this distinction is used in our internal queues, but from the user's perspective the difference is slim: setting the bug report to 'Confirmed' does mean that we are satisfied with the information provided in the report, but the user will also know about it from our comments and other updates. Otherwise, bugs in both statuses can be considered for fixing.
- `In Progress`, `Stalled` - different intermediate states of bugs which help developers to filter their lists and management to gather a picture of the current activity. For the user, there is no important difference -- despite the negative semantics, 'Stalled' does not mean that something is wrong with the bug report, only that the developer is not working on it actively at the moment.
- `In review` - means, literally, that a peer review has been requested.
- `Closed` - means that the bug report is closed, on whatever reason. The real reason is in the 'Resolution' field.

Fix Versions

This is an important field for progress tracking. After the bug is confirmed or otherwise acknowledged, this field is populated with a set of major versions where we intend to fix it. E.g. if the field is set to `10.0 10.1`, it means that at the moment we consider it for fixing in some future 10.0 release (not necessarily the next one), and the bugfix will be merged into the next 10.1 release after that; but we do not consider it for fixing in 5.5, even if it is affected to.

To some extent, you can influence the initial plans: if you see that the fix is not targeted for versions where you think it should be, you can comment on the report, and if you provide convincing arguments and make your case, it can be reconsidered.

The value of the field is not a promise to fix the bug in the mentioned releases. It can be changed both ways: during further analysis, the developer can find out that it can be safely fixed in an earlier release, or, on the contrary, that it cannot be safely fixed in the GA release, and the fix can only go to the next versions which are currently under development.

After the bug is fixed, the value of the field is changed to the exact versions, e.g. `10.0.25 10.1.14`. It means that the patch has been pushed into the 10.0 branch, and will be released with 10.0.25 release; it also means that the patch *will be* merged to 10.1 tree and released with 10.1.14 release, but it does not mean that it is already in the 10.1 branch.

Priority

As the other article says, the **Priority** field serves two purposes. During the initial bug creation, it indicates the importance of the bug report from the user's perspective (in other bug tracking systems it is called 'Severity' or alike). After the bug has been confirmed, the same field is used for development purposes, to prioritize bug fixing (real 'Priority'). While we take into account the reporter's view on the matter, we can change the initial priority both ways, depending on the information we revealed during the problem analysis, versions affected, etc.

The value of the field normally means the following:

- `Blocker` - we currently think that the bug must be fixed before the next release(s) set in the 'Fix version/s' field;
- `Critical` - the bug should be picked up for fixing earlier than any other bugs apart from blockers;
- `Major` - the bug will be present in the main queue for fixing in the upcoming 'Fix version/s', although only a part of such bugs will be fixed in every release;
- `Minor`, `Trivial` - the bugs will be picked up when the assignee does not have more pressing issues for the upcoming release.

Please note that the Priority field only demonstrates our intentions at the moment, it does not guarantee that things will happen according to these intentions.

Labels

Labels are mostly used for more convenient filtering and don't carry much importance otherwise. However, there are a few that affect the processing of a bug report:

- `need_feedback` - its role during the initial bug processing was already described **above**. However, after a bug is confirmed and queued for fixing, it should not appear anymore; and even if it's left by mistake, it won't affect the progress.
- `upstream` - the label means that the bug also exists in the upstream version of the corresponding component - normally, in MySQL server or a client program, but can also be in Percona's XtraDB or TokuDB. Normally there should also be a link to the upstream bug report. Setting this label means that we might want to take for a while and see whether the bug is fixed in the upstream version before we fix it in MariaDB directly. It was usual for 5.5, less usual for 10.x where bugfixes, apart from InnoDB, are not merged automatically. The label is still set, but it is more for informational purposes than to affect the priority.
- `upstream-fixed` - the label means that the bug used to exist in the upstream version, but not anymore. It means that there is nothing more to wait; moreover, it might be worth picking up the bug soon and at least evaluating the upstream bugfix.

Bug Reports with Patches

MariaDB encourages contributors to provide bug fixes; so, bug reports which come with the fixes in general have a quicker turnaround. The bug fix can come in a form of Git pull request, or, in simple cases, as a diff pasted in or attached to the bug report itself.

Principles for External Bug Reports

There are some basic rules for bugs, particularly for setting the **Resolution** value, which we want to stick to and which might be different from procedures you came across in other projects. It mainly concerns *external* bugs (those that come from the community), for internal ones we can cut corners more freely.

This all is easier to understand if one remembers that the **Resolution** or its analogues in other bug-tracking systems is a *user-facing* field, as already mentioned above, and that it relates more to the report, than to the bug itself.

Duplicate

An older bug report cannot be a duplicate of a newer one, it is nonsensical. The only possible exception is when an older bug has no useful information whatsoever *and* the reporter does not provide any helpful feedback, while a newer report was not closed as a duplicate right away and got some useful updates. The common example of such exception is when the first report is just an optimized stack trace, no query, no data, nothing to work with, while the second report has a test case. But if the first reporter at least makes an effort to collaborate, the report deserves to be treated with respect.

Bug reports which have essentially different descriptions and/or test cases should not be duplicates. The common example is this: a developer creates a bug saying something like "this and that pieces of code are wrong, it should be so and so"; and then a user files a bug saying "this SQL produces a wrong result on this data set". Even if they are about *the same error in the code* at the end, they are not duplicate *bug reports*.

Obviously, a report can never be a duplicate of anything private (luckily it does not concern MariaDB server so far, as the bug reports are public).

In general, a bug report is a duplicate of another one if, and only if, the new reporter could find the existing report just by a reasonable **JIRA search**.

Cannot Reproduce

A bug report should not be closed as "cannot reproduce" if it was once verified/confirmed, but disappeared in later versions. It's unfair to the reporter, and also dangerous to the product. We should know why a bug stopped being reproducible -- either we find when and how it was fixed (and close the report as "Fixed in version X by a patch for Y"), or we discover that it wasn't in fact fixed, but just masked. The simplest example is a change of execution plan in optimizer: server would crash on a particular query, then due to a change in optimizer it started using a different plan for the same query, so it wouldn't go through the crashing path anymore. The crash is still there, though.

In general, the "cannot reproduce" resolution is a last resort. Usually if we can't reproduce something, it means that either the reporter did not provide required information (and then the resolution should be "Incomplete"), or we don't know what to request from the reporter, and then we should keep thinking, rather than close it. Of course, it happens that the bug is genuinely not reproducible, but it shouldn't be decided lightly.

8.1.4 MariaDB Security Bug Fixing Policy

See MariaDB Security Policy.

8.1.5 Building MariaDB Server for Debugging

Contents

Instructions on how to build a mysqld that contains all the information we need to fix the problem: (A more detailed explanation can be found here.)

Note: this text has been extracted into a separate article from Reporting bugs, see its full history there.

- Add the --core-file option to your /.my.cnf or /etc/my.cnf file under the [mysqld] tag.
- Get the latest MariaDB code from GitHub.
- Compile MariaDB with the **-g** compiler flag (Unix).
- Optionally: with more checking Compile MariaDB for debugging - will cause slowdown.
- Shut down your old mysqld server.
- Install the new compiled `mysqld` binary. Note that if you are compiling same version of MariaDB that you have already installed it's enough to just copy this one binary!
- Restart mysqld.

Compiling with **-g** should not cause any notable slowdown of the server.

You can of course also do **make install**, but the above way allows you to go back to your old binary if needed.

If you get any errors about a wrong number of error messages, you can fix that by copying the corresponding language file from **sql/share** over your old ones (this should be reasonably safe to do).

```
cp sql/share/english/* mariadb-install-dir/share/mysql/english
```

What to Do When You Get a Crash After Installing a Debug Binary

Now when you get a crash do the following:

- Create a README file that describes the problem. You can use the mysqlbug script to generate a template for this.
- Create a tar file containing the core, the mysqld binary and README. If possible, also add any database files that could help us repeat the problem!

```
sh> tar cvfz /tmp/mariadb-bug-'short-description'.tgz mariadb-data-dir/core* mariadb-
install-dir/libexec/mysqld README
```

- Send it to our secure ftp server:

```
sh> ftp -a ftp.askmonty.org
ftp> cd private
ftp> binary
ftp> put /tmp/mariadb-bug-'short-description'.tgz
ftp> quit
```

- To be able to follow the progress, create a bug report in JIRA about this. This should be easy to do based on the information you have in your README file.

8.1.6 Extracting Entries from the Binary Log

This article is relevant if the problem is on a replication slave.

Note: this text has been extracted into a separate article from Reporting bugs, see its full history there.

Sometimes a binary log event causes an error of some sort. A whole binary log file is sometimes impractical due to size or sensitivity reasons.

Step 1: Copy the binary log locally

This is just in case you don't quite extract the right information first. If the binlog expired off and you haven't got the right information, your bug report may not easily be reproducible.

```
sudo cp /var/lib/mysql/mysql-bin.000687 ~/
sudo chown $USER: ~/mysql-bin.000687
```

Step 2: Create an extract header

Binary logs have a header portion. Without the header mariadb-binlog won't be able to read it. The header also contains valuable session information

We look at the binary log to see how big the header and session information is:

```
mariadb-binlog --base64-output=decode-rows --verbose mysql-bin.000687 | more
/*!50530 SET @@SESSION.PSEUDO_SLAVE_MODE=1*/;
/*!40019 SET @@session.max_insert_delayed_threads=0*/;
/*!50003 SET @OLD_COMPLETION_TYPE=@@COMPLETION_TYPE,COMPLETION_TYPE=0*/;
DELIMITER /*!*/;
# at 4
#150323 22:45:58 server id 76  end_log_pos 245  Start: binlog v 4, server v 5.5.39-MariaDB-log cr
# at 245
#150323 22:45:58 server id 76  end_log_pos 328  Query   thread_id=9709067       exec_time=0     e
SET TIMESTAMP=1427116558.923924/*!*/;
SET @@session.pseudo_thread_id=9709067/*!*/;
SET @@session.foreign_key_checks=1, @@session.sql_auto_is_null=0, @@session.unique_checks=1, @@se
SET @@session.sql_mode=0/*!*/;
SET @@session.auto_increment_increment=1, @@session.auto_increment_offset=1/*!*/;
/*!\C utf8 *//*!*/;
SET @@session.character_set_client=33,@@session.collation_connection=33,@@session.collation_serve
SET @@session.time_zone='SYSTEM'/*!*/;
SET @@session.lc_time_names=0/*!*/;
SET @@session.collation_database=DEFAULT/*!*/;
BEGIN
/*!*/;
# at 328
```

We see that the session information ends at 328 because of the last line, so we extract to that point.

```
dd if=mysql-bin.000687 of=mysql-bin.000687-extract-offset-129619 bs=1 count=328
```

We need to find out at what offset the entry at 129619 ends and it might be useful to extract some previous entries as well.

```
mariadb-binlog --base64-output=decode-rows --verbose mysql-bin.000687 | grep  '^# at ' |  grep -C
# at 127602
# at 127690
# at 128201
# at 128290
# at 128378
# at 128829
# at 128918
# at 129006
# at 129459
# at 129548
# at 129619
# at 129647
# at 130070
# at 130097
# at 130168
# at 130196
# at 130738
# at 130942
# at 130969
# at 131040
# at 131244
```

Take a look at those entries with:

```
mariadb-binlog --base64-output=decode-rows --verbose --start-position 129006  --stop-position 130
```

Now let's assume we want to start at our original 129619 and finish before 130168

```
dd if=mysql-bin.000687 bs=1 skip=129619 count=$(( 130168 - 129619 ))  >> mysql-bin.000687-extract
```

Check the extract:

```
mariadb-binlog mysql-bin.000687-extract-offset-129619
```

Upload this to the private uploads or attach to the public bug report if nothing sensitive there.

8.2 Contributing & Participating

There are many ways to contribute to MariaDB.

Getting Help With MariaDB

Review of resources available when you need help.

Reporting Documentation Bugs

Reporting a bug or feature request in the MariaDB documentation.

MariaDB Community Bug Reporting

Guidelines for reporting bugs in MariaDB software.

Contributing to the MariaDB Project

How to contribute to the MariaDB project: code, documentation, feedback, etc.

Contributing Code

Guidelines and procedures for contributing code to MariaDB.

Project Suggestions

List of MariaDB Projects in which you might Assist.

Log of MariaDB Contributions

List of contributions and contributors to MariaDB and how to log them.

Donate to the Foundation

Donating to the MariaDB Foundation.

Sponsoring the MariaDB Project

Ways to sponsor the MariaDB project (e.g. with labor or funds).

Web Hosting Providers Offering MariaDB

List of known web hosting providers that offer MariaDB to their clients.

Using Git with MariaDB

How to use git to troubleshoot the source code or contribute code to MariaDB.

Google Summers of Code

Information on Google Summer of Code for each year since 2013.

Google Season of Docs

Information on each Google Season of Docs.

8.2.1 Getting Help With MariaDB

If you can't find help in the MariaDB documentation, you can also subscribe to the mailing lists or Zulip to communicate with MariaDB users and developers.

There's an active help community on DBA StackExchange.

If you have a question about a feature that is not properly documented or something that is not working as expected, go to the corresponding Knowledge Base page and ask your question there.

You can report and check on bugs which are unique to MariaDB in JIRA.

In some cases the documentation for MySQL can be used. Note that the MySQL documentation will not mention new features added in MariaDB, or other changes since introduced in MariaDB. See MariaDB versus MySQL and in greater detail under the MariaDB category.

MariaDB plc offers commercial support.

8.2.2 Contributing to the MariaDB Project

Contents

The success of MariaDB depends on the participation of the community. MariaDB would not be as good as it is today without the participation of the entire MariaDB community.

Getting Started

There are several ways to contribute besides Contributing Code, and not all of them require strong C/C++ skills. Areas for non-developers include:

- Reporting Bugs
  - To report a bug you'll need to sign up for an account by clicking on the Create an account link below the login fields
- Suggesting Features
- Answering questions in the Knowledge Base
- Helping other people answering problems or even fixing their bugs on IRC in the **#maria** channel on Libera.Chat or on MariaDB's Zulip instance at https://mariadb.zulipchat.com.
- Testing and Benchmarking
  - Bug tests can be uploaded to the 'private' directory of our FTP server.
- Creating documentation and KB articles for MariaDB, or translating existing documentation.
- Advocating MariaDB in your area.
- Participate in open source events and talk about MariaDB.
- Running a BuildBot on a platform which is not currently represented.
- Donate time or money to the MariaDB project.
- Ask your company to sponsor a feature.

- MariaDB Foundation page on getting involved

> **Contributing to the Knowledge Base** is a great way to help improve MariaDB.

MariaDB Email Lists

A great way to get started in MariaDB is to participate in e-mail discussions via our mailing lists (whichever list best matches your interests):

- developers
- discuss
- docs

Sensitive security issues can be sent directly to the persons responsible for MariaDB security: security [AT] mariadb (dot) org.

You can find additional email address, email archives and ways to connect with MariaDB people here.

All MariaDB contributors are expected to follow the Ubuntu Code Of Conduct.

Getting Started for Developers

- Contributing Code

See also the pages for new developers on the MariaDB Foundation website:

- Getting Started For Developers
- Get the Code, Build It, Test It
- Writing Good Test Cases for MariaDB Server
- Submitting a Pull Request

8.2.3 Contributing Code

Contents

This page contains general guidelines and procedures for contributing code. If you have any questions we invite you to ask on the maria-developers mailing list, on our Zulip instance at https://mariadb.zulipchat.com, or on the #maria IRC channel on https://libera.chat/. Other email lists and places to find MariaDB can be found here.

General information about contributing to MariaDB (for developers and non-developers) is found on the Contributing to the MariaDB Project page.

Finding Development Projects to Work on

There are many open development projects for MariaDB which you can contribute to (in addition to any ideas you may have yourself).

- We are using JIRA to manage the MariaDB project. Go to https://jira.mariadb.org and click on "Projects" to get to the MariaDB project. Browse around the unresolved and unassigned issues to see if there is something that interests you. Some issues have sponsors and you can be paid for doing them!
- A list of beginner friendly tasks is also available.
- Check the development plans for the next MariaDB version.
- Join maria-developers and ask for suggestions of tasks you could do. Please include your programming experience and your knowledge of the MariaDB source and how much you know about using MySQL/MariaDB with the email so

that we know which tasks we can suggest to you.

- If this is your first project, check out the Suggested Development page. It lists projects that will make a good start.
- Join MariaDB's Zulip instance at https://mariadb.zulipchat.com and ask for suggestions.
- Join #/maria on https://libera.chat/ IRC and ask for suggestions.

If you have your own ideas, please submit them to JIRA so other MariaDB developers can comment on them and suggest how to implement them. You can of course also use the maria-developers list for this.

What to Expect From a MariaDB Server Developer

This section is mainly directed to developers with commit rights to the MariaDB git repository. However, we hope it's also useful for anyone wanting to contribute code to MariaDB to know what a reviewer will expect from them.

This is not about coding style or if one should prefer C instead of C++. That would be a separate topic that should be created sooner or later.

The Basics

When coding, try to create code that 'never has to be changed again'. Try to make the code as performant as possible. In general it is acceptable to spend 50% more time to make the code 15% faster than what you originally intended. Take that into account when you plan your time estimates! That said, don't try to add classes or functionality that is not yet used.

The code should be easy to read and follow the coding standards of the project. Patches that are smaller and simpler are often better than complex solutions. Don't make the server depend on new external libraries without first checking with Sergei or Monty!

Add code comments for anything that is not obvious. When possible, use assertions within the code to document expectations of arguments etc. In general, if the code requires complex comments, think if there is a better way to structure the logic. Simpler is often better and with fewer bugs.

What to Have in a Commit Comment

- Jira issue number and summary ex: MDEV-23839 innodb_fast_shutdown=0 hang on change buffer merge
- An empty line
- A short description of the problem
- A description of the solution
- Any extra information needed to understand the patch
- The commit message should be self contained and the reviewer shouldn't preferably have to look at the Jira at all to understand the commit. This doesn't mean that the commit message should include all background and different design options considered, as the Jira should contain.
- Name of all reviewers and authors should be clear from the commit message. The preferred way would be (one line per person)
  - Reviewed-by: email
  - Co-authored-by: email
  - See https://docs.github.com/en/free-pro-team@latest/github/committing-changes-to-your-project/creating-a-commit-with-multiple-authors for details
- The default is that all code should be reviewed. Only in really extraordinary cases, like merge (where the original code was already reviewed) then it can be self-reviewed, which should clear from the commit. In this case the code should of course be tested extra carefully both locally and in buildbot before pushing.

Testing

- All code should have a test case that shows that the new code works or, in case of a bug fix, that the problem is fixed! It should fail with an unpatched server and work with the new version. In the extreme case that a test case is practically impossible to do, there needs to be documentation (in the commit message, optionally also in Jira) how the code was tested.
- The test case should have a reference to the Jira issue, if such one exists.
- Patches related to performance should be tested either by the developer (for simple commits) or by performance testers. The result should be put in Jira with a summary in the commit.
- Complex patches and should be tested by QA in a bb- branch before pushing. The Jira entry should include information that this has been done and what kind of test has been run.
- For anything not trivial, one should run either Valgrind or ASAN/MSAN on the new code. (Buildbot will do this for you if you can't get valgrind or ASAN to work). At least the test case added should be tested by either. If the developer cannot do that for some reason, he should check the buildbot builders that do this and ensure that at least his test case doesn't give any warnings about using not initialized memory or other failures.
- For complex code the developer should preferably use gcov or some similar tool to ensure that at least not all not-

error branches are executed. “mtr --gcov” or “dgcov.pl” can help you with this.

Getting Your Code into the Main MariaDB Tree

All code in MariaDB comes from one of the following sources:

- MySQL
- Code developed by people employed by the MariaDB Foundation.
- Code developed by people employed by MariaDB Corporation.
- Code shared with the MariaDB Foundation under the MCA.
- Code with a known origin that is under a permissive license (BSD or public domain).

If you want the code to be part of the main MariaDB tree, you also have to give the MariaDB Foundation a shared copyright to your code. This is needed so that the foundation can offer the code to other projects (like MySQL).

You do this by either:

1. Signing the MariaDB Contributor Agreement (MCA) and then scanning and sending it to the foundation.
2. Sending an email to maria-developers where you say that your patch and all fixes to it are provided to the MariaDB Foundation under the MCA.
3. Licensing your code using the BSD license.

We need shared copyright for the following reasons:

1. to defend the copyright or GPL if someone breaks it (this is the same reason why the Free Software Foundation also requires copyright assignment for its code)
2. to be able to donate code to MySQL (for example to fix security bugs or new features)
3. to allow people who have a non-free license to the MySQL code to also use MariaDB (the MCA/BSD allows us to give those companies the rights to all changes between MySQL and MariaDB so they can use MariaDB instead of MySQL)

More information about the MCA can be found on the MCA FAQ page.

Before Pushing Code to a Stable Branch

- Ensure that you have compiled everything for your new code, in a debug server (configured with cmake -DCMAKE_BUILD_TYPE=Debug) including embedded and all plugins that may be affected by your code change..
- Run the mysql-test-run (mtr) test suite locally with your debug server.
- For anything complex the full test suite should be run.
- For something absolutely trivial, at least the main suite must be run.
- Always push first to a bb- branch to test the code. When the bb- branch is green in buildbot you can push to the main branch. Take care of checking that Windows builds compiles (take extra care of checking this as this often fails) and that valgrind and msan builds doesn’t show any problems with your new test cases.
- If you have to do a rebase before pushing, you have to start from the beginning again.
- When porting code from third parties (such as MySQL), make sure to attribute copyright to the right owner, in the header of each modified file.
- For example: Copyright © 2000, 2018, Oracle and/or its affiliates. Copyright © 2009, 2020, MariaDB
- The only exception is that if the changes are trivial and the rebase was trivial and the local mysql-test-run worked, then you can push directly to the main branch. Only do this if you are 99% sure there are no issues! * Please don't make us regret that we have made this one exception! When we have protected git branches, then the above rule will be enforced automatically as the protection will take care of this.

Working on a New Project

- First create a Jira entry that explains the problems and the different solutions that can be used to solve the problem. If there is a new syntax include examples of queries and results.
- After getting an agreement of the to-be-used solution, update the Jira entry with the detailed architecture of the suggested solution.
- When the architecture is reviewed, the assigned developer can start coding.
- When the code is ready, the Jira entry should be updated with the reviewer.
- The reviewer checks the code and either approves it to be pushed or gives comments to the developers that should be fixed. In the later case the developer updates the code and gives it back to the reviewer. This continues until the code is approved.
- If the design changes during the project, the design in Jira needs to be updated.

Working on a Bug Fix

- Ensure that the Jira issue is up to date.
- For complex bugs that require redesign, follow the process in "Working on a new project"

- For simpler bugs, one can skip the listing of different solutions and architecture. However one should still document the reason for the bug and how it's fixed or to-be-fixed, in a JIRA comment.

Making Things Easier for Reviewers

- Ensure that code compiles, all MTR test works before asking for a review
- Try to split a bigger project into smaller, self-contained change sets.
- Automatic things, like renames of classes, variables, functions etc is better to be done in a separate commit.

When Reviewing Code

Remember that the stability and security of any project hangs a lot on the reviewers. If there is something wrong with an accepted patch, it's usually the reviewer who is to be blamed for it, as the reviewer was the one who allowed it to go in!

- Ensure that the code is licensed under New BSD or another approved license for MariaDB (basically any open source license not conflicting with GPL) or that the contributor has signed the MCA.
- GPL is only allowed for code from MySQL (as MariaDB is already depending on MySQL code).
- Ensure that commits are not too large. If the code is very large, give suggestions how to split it into smaller pieces. Merge commits, when rebasing is possible, are not allowed, to keep history linear.
- Check that the commits message describes the commit properly. For code that improves performance, ensure that Jira and the commit message contains information about the improvements.
- Check that there are no unexplained changes in old tests.
- Check the quality of the code (no obvious bugs, right algorithms used)
- Check if any code can be simplified or optimized. Using already existing functions, are loops optimal, are mutexes used correctly etc.
- Check that there is an appropriate test case for the code. See ‘testing’ for what is required!
- Ensuring the code follows the coding standard for MariaDB. This document should be created shortly, but in the meantime ask an old MySQL/MariaDB developer if you are unsure.
- Ensuring that the code follows the architecture agreed for in Jira (if it's in Jira).
- Code should be easy to understand (good code comments, good function and variable names etc).
- Ensure you understand every single line of code that is reviewed. If not, ask the developer to add more comments to get things clear or ask help from another reviewer.
- No performance degradations for all common cases.
- Any code that touches any sensitive area (files, communication, login, encryption or security) needs to have another reviewer that is an expert in this area.

8.2.4 Donate to the Foundation

To donate, please visit the MariaDB Foundation donations page.

8.2.5 Sponsoring the MariaDB Project

Sponsorships are crucial for ongoing and future development of the MariaDB project! There are a number of easy ways you for you to help the project:

- Fund the development of a specific feature. You can find a list of suggested features to sponsor here or in JIRA. Feel free to sign in and add more projects to either place!
- Contribute with developer time. If your organization has talented developers familiar with MariaDB or MySQL codebase they can become part of the MariaDB team and contribute to the development of the MariaDB project.
- Hire a developer that you dedicate to work on the MariaDB project.
- A pure donation with no strings attached

To get going with the sponsorship please visit the MariaDB Foundation sponsorships page.

8.2.6 Using Git with MariaDB

Tricks and tips on how to use Git, the source control system MariaDB uses.

MariaDB Source Code

How to get the source code for MariaDB from GitHub.

Using Git

Working with the git repository for the source code of MariaDB on GitHub.

2.1.2.8.2 MariaDB Source Code

8.2.6.2 Using Git

Contents

Just Getting the Source

If you just want to get the latest source and you don't require the ability to push changes to the MariaDB repository, you can use the following command to check out the latest 10.5 branch:

```
git clone -b 10.5 https://github.com/MariaDB/server.git
```

Setup up git for MariaDB

- Set your name with git

```
git config --global user.name "Ivan Ivanov"
git config --global user.email "ivan@mariadb.com"
```

- Go to https://github.com/settings/ssh and add your public SSH key (GitHub Help).
- Clone the repository

```
git clone git@github.com:MariaDB/server.git
cd server
git checkout 10.5
```

- Config repository pull and alias for fast forward:

```
git config pull.ff only
git config --global alias.ff "merge --ff-only"
```

Commit comments

In git commit messages are normally formatted as

```
subject

body
more body
...
```

That is, the first line is considered a *subject*, much like email subject. Many git commands and pages on github only show the commit subject, not the complete comment. After the subject goes an empty line, then the detailed description of the comment. Please, structure your commit comments this way, don't forget the empty line.

Branches

This is an important concept, and git branches do not have equivalents in bzr.

In Bazaar, we all used to have one shared repository, within which there were many branches. This seems to be impossible with git?

In Git, each repository has only one branch that is currently checked out.

```
git branch
```

List Existing Branches

To see which branches exists locally and remotely:

```
git branch --all
```

To Move to Work on a Specific Branch

```
git checkout branch-name
```

Note that if the output from `git branch --all` is `remotes/origin/XXX` you should just use `XXX` as branch name.

Making a Local Copy of a Branch (Like bzr clone)

```
branch clone old_directory new_directory
cd new_directory
git remote set-url origin git@github.com:MariaDB/server.git
git pull
```

Remove Last Commit From a Branch

```
git reset --hard HEAD^
```

Fetch a Branch Someone Has Done a Rebase on

If you get the following error on pull:

```
shell>  git pull
X11 forwarding request failed on channel 0
fatal: Not possible to fast-forward, aborting.
```

Instead of removing your copy and then clone, you can do:

```
git reset --hard origin/##branch-name##
```

Other Things About Branches

- Note: branches whose names start with `bb-` are automatically put into the buildbot.

Equivalents For Some bzr Commands

CAVEAT UTILITOR. Check the manual before running!

- `bzr status` is `git status`
- `bzr diff` is `git diff`
- `bzr log` is `git log`
- `bzr revert` is `git reset --hard`
- `bzr revert filename` is `git checkout filename`
- `bzr parent` is `git remote -v` (but there are more detailed commands)
- `bzr parent to-default-mariadb-repo` git remote set-url origin git@github.com:MariaDB/server.git
- `bzr push` is `git push REMOTENAME BRANCHNAME` . REMOTENAME is typically "origin", for example: `git push origin HEAD:10.3-new-feature` . The HEAD: stands for "from current branch".
- `bzr clean-tree --ignored` is `git clean -Xdf` (note the capital X!)
- `bzr root` is `git rev-parse --show-toplevel`
- `bzr missing --mine-only` is `git cherry -v origin` (kind-of).

GUIs

- `bzr gcommit` is `git citool`
- `bzr viz` is `gitk`
- `bzr gannotate` is `git gui blame`

Commit Emails

In the MariaDB project, it is a good idea (and a long tradition since MySQL Ab) to have all your commits sent to a commits@mariadb.org mailing list. It allows others to follow the progress, comment, etc.

A script and instructions on how to setup commit triggers in Git are here: http://bazaar.launchpad.net/~maria-captains/mariadb-tools/trunk/files/head:/git_template/ . Jira task for commit trigger was MDEV-6278.

Attributing Code to Someone Else

If you add code on behalf of someone else, please attribute the code to the original author:

- Run `git citool` and move changed files to staged.
- Don't `commit` , abort instead
- run `git commit --author="Original author name <email_address>"`

The above is needed as `git citool` can't handle the `--author` option.

Applying a Pull Request

At the end of the pull request page there is a button "Merge pull request" and next to it a link "command line instructions". Click the link, you'll see something like

Step 1: From your project repository, check out a new branch and test the changes.

```
git checkout -b mariadb-server-joeuser-cool-feature 10.3
git pull https://github.com/joeuser/mariadb-server cool-feature
```

Step 2: Merge the changes and update on GitHub.

```
git checkout 10.3
git merge --no-ff mariadb-server-joeuser-cool-feature
git push origin 10.3
```

Note where to pull from — **https://github.com/joeuser/mariadb-server cool-feature**.

Now, checkout the branch you want to merge it to, say, bb-10.3-stage, and do the following

```
git fetch https://github.com/joeuser/mariadb-server cool-feature
git checkout FETCH_HEAD
git rebase @{-1}
```

Now's the time to compile the code, test it, fix, if necessary. Then

```
git checkout @{-1}
git ff @{-1}
```

If you want to do small changes to the pull request, do it in a separate commit, after `git rebase @{-1}` above. If you want to do *big* changes to the pull request, perhaps you shouldn't merge it in the first place, but ask the contributor to fix it?

Examples

Diff For Last Commit

```
git show
```

Applying New Code From Main Tree to a Branch

You are working on a branch (`NEW_FEATURE`) and would like to have into that branch all changes from the main branch (`10.1`).

```
git checkout 10.1
git pull
git checkout NEW_FEATURE
git rebase 10.1
```

Applying a Bugfix in the Main Branch

You've just fixed and committed a bug in the main 10.1 branch and want to merge it with the latest 10.1. Often a rebase is better in this case. Assuming your current checked out branch is 10.1:

```
git fetch origin
git rebase origin/10.1
```

This will work even if you have done multiple commits in your local 10.1 tree.

Dealing With Conflicts When One Tries to Push

What to do when you have fixed a bug in the main tree but notices that someone has changed the tree since you pulled last time. This approach ensures that your patch is done in one block and not spread out over several change sets.

```
git clone 10.1
cd 10.1
< fix a bug here>
git citool
git push
# ^ and the above fails, because someone has pushed to 10.1 in between
git branch tmp
# ^ copy our work to branch named 'tmp'
get checkout 10.1
git reset --hard HEAD^
# ^ remove our work from '10.1' local branch'
git pull
# ^ get changes from remote
git checkout tmp
git rebase 10.1
# ^ switch to 'tmp' and try to rebase 'tmp' branch on top of 10.1 branch.
#   here you will be asked to merge if necessary
git checkout 10.1
git pull --ff . tmp
# ^ switch back to the '10.1' branch, and pull from 'tmp' branch.
git branch -D tmp
#^ this removes the tmp. branch
git push
```

Finding in Which MariaDB Version a Commit Exists

```
sh> git tag --contain e65f667bb60244610512efd7491fc77eccceb9db
mariadb-10.0.30
mariadb-10.1.22
mariadb-10.1.23
mariadb-10.2.5
mariadb-10.3.0
mariadb-galera-10.0.30
```

8.2.6.3 Configuring Git to Send Commit Notices

Commit emails for MariaDB are sent to `commits@mariab.org`. You can find the archive here.

To allow others to see what you are working on in your MariaDB tree, you should:

1. subscribe to the email list
2. configure git to send your commits to `commits@mariab.org`.

Download the post-commit git trigger script. Configure as

```
git config --global hooks.postcommitrecipients "commits@mariadb.org"
git config --global hooks.postcommitbranches "*"
```

Also you might want to check the README for the post-commit trigger.

The post-commit git trigger uses *sendmail* for sending emails. Some platforms don't have *sendmail* and then you'll need to modify to make use of something that is supported.

Also, the post-commit trigger is just one approach. You can also use git-email on at least Debian and Fedora to send commit emails to the MariaDB commits email list.

8.3 Legal Matters

GNU General Public License, Version 2

Full text of the GNU General Public License, Version 2

Legal Notices for the Knowledge Base

List of licenses and trademarks associated with the content in the MariaDB Knowledge Base.

8.3.1 GNU General Public License, Version 2

```
      GNU GENERAL PUBLIC LICENSE
         Version 2, June 1991

 Copyright (C) 1989, 1991 Free Software Foundation, Inc.,
 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

       Preamble

  The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General Public
License is intended to guarantee your freedom to share and change free
software--to make sure the software is free for all its users.  This
General Public License applies to most of the Free Software
Foundation's software and to any other program whose authors commit to
using it.  (Some other Free Software Foundation software is covered by
the GNU Lesser General Public License instead.)  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
this service if you wish), that you receive source code or can get it
if you want it, that you can change the software or use pieces of it
```

in new free programs; and that you know you can do these things.

  To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

  For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

  We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

  Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

  Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

  The precise terms and conditions for copying, distribution and modification follow.

      GNU GENERAL PUBLIC LICENSE
   TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

  0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

  1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

  2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

    a) You must cause the modified files to carry prominent notices
    stating that you changed the files and the date of any change.

    b) You must cause any work that you distribute or publish, that in
    whole or in part contains or is derived from the Program or any
    part thereof, to be licensed as a whole at no charge to all third

part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is

void, and will automatically terminate your rights under this License.
However, parties who have received copies, or rights, from you under
this License will not have their licenses terminated so long as such
parties remain in full compliance.

  5. You are not required to accept this License, since you have not
signed it. However, nothing else grants you permission to modify or
distribute the Program or its derivative works. These actions are
prohibited by law if you do not accept this License. Therefore, by
modifying or distributing the Program (or any work based on the
Program), you indicate your acceptance of this License to do so, and
all its terms and conditions for copying, distributing or modifying
the Program or works based on it.

  6. Each time you redistribute the Program (or any work based on the
Program), the recipient automatically receives a license from the
original licensor to copy, distribute or modify the Program subject to
these terms and conditions. You may not impose any further
restrictions on the recipients' exercise of the rights granted herein.
You are not responsible for enforcing compliance by third parties to
this License.

  7. If, as a consequence of a court judgment or allegation of patent
infringement or for any other reason (not limited to patent issues),
conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License. If you cannot
distribute so as to satisfy simultaneously your obligations under this
License and any other pertinent obligations, then as a consequence you
may not distribute the Program at all. For example, if a patent
license would not permit royalty-free redistribution of the Program by
all those who receive copies directly or indirectly through you, then
the only way you could satisfy both it and this License would be to
refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under
any particular circumstance, the balance of the section is intended to
apply and the section as a whole is intended to apply in other
circumstances.

It is not the purpose of this section to induce you to infringe any
patents or other property right claims or to contest validity of any
such claims; this section has the sole purpose of protecting the
integrity of the free software distribution system, which is
implemented by public license practices. Many people have made
generous contributions to the wide range of software distributed
through that system in reliance on consistent application of that
system; it is up to the author/donor to decide if he or she is willing
to distribute software through any other system and a licensee cannot
impose that choice.

This section is intended to make thoroughly clear what is believed to
be a consequence of the rest of this License.

  8. If the distribution and/or use of the Program is restricted in
certain countries either by patents or by copyrighted interfaces, the
original copyright holder who places the Program under this License
may add an explicit geographical distribution limitation excluding
those countries, so that distribution is permitted only in or among
countries not thus excluded. In such case, this License incorporates
the limitation as if written in the body of this License.

  9. The Free Software Foundation may publish revised and/or new versions
of the General Public License from time to time. Such new versions will
be similar in spirit to the present version, but may differ in detail to
address new problems or concerns.

Each version is given a distinguishing version number. If the Program
specifies a version number of this License which applies to it and "any
later version", you have the option of following the terms and conditions
either of that version or of any later version published by the Free
Software Foundation. If the Program does not specify a version number of
this License, you may choose any version ever published by the Free Software
Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year>  <name of author>

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License along
with this program; if not, write to the Free Software Foundation, Inc.,
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) year name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
```

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items--whatever suits your program.

```
You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the program, if
necessary.  Here is a sample; alter the names:

  Yoyodyne, Inc., hereby disclaims all copyright interest in the program
  `Gnomovision' (which makes passes at compilers) written by James Hacker.

  <signature of Ty Coon>, 1 April 1989
  Ty Coon, President of Vice

This General Public License does not permit incorporating your program into
proprietary programs.  If your program is a subroutine library, you may
consider it more useful to permit linking proprietary applications with the
library.  If this is what you want to do, use the GNU Lesser General
Public License instead of this License.
```

8.3.2 Legal Notices for the Knowledge Base